Intel® C++ Compiler User and Reference Guides

Document number: 304968-023US
Contents

Legal Information.................................................................................................41
Getting Help and Support......................................................................................43

Chapter 1: Introduction
   Introducing the Intel(R) C++ Compiler..........................................................45
   Notational Conventions....................................................................................45
   Related Information.........................................................................................48

Part I: Building Applications

Chapter 2: Overview: Building Applications
   Introduction to the Compiler...........................................................................53
   Compilation Phases.........................................................................................54
   Default Behavior of the Compiler....................................................................54
   Default Output Files.......................................................................................54
   Using Compiler Options..................................................................................55
   Saving Compiler Information in Your Executable............................................59

Chapter 3: Building Applications with Eclipse*
   Overview: Eclipse Integration...........................................................................61
   Multi-version Compiler Support........................................................................62
   Starting Eclipse...............................................................................................62
   Creating a Simple Project................................................................................63
       Creating a New Project..................................................................................63
       Adding a C Source File.................................................................................65
Chapter 4: Building Applications from the Command Line
Invoking the Compiler from the Command Line.......................85
Invoking the Compiler from the Command Line with make........86
Passing Options to the Linker.............................................87
Compiler Input Files..........................................................88
Output Files.......................................................................89
Specifying Compilation Output............................................89
    Specifying Executable Files.............................................89
    Specifying Object Files................................................90
    Specifying Assembly Files..............................................90
Specifying Alternate Tools and Paths.................................90
Using Precompiled Header Files.......................................91
Compiler Option Mapping Tool........................................94
Open Source Tools............................................................96

Chapter 5: Using Preprocessor Options
About Preprocessor Options.............................................97
Using Options for Preprocessing.......................................98
Using Options to Define Macros......................................99
Chapter 6: Modifying the Compilation Environment

About Modifying the Compilation Environment......................101
Setting Environment Variables............................................101
Using Configuration Files....................................................104
Specifying Include Files.....................................................105
Using Response Files..........................................................106

Chapter 7: Debugging

Using the Debugger..........................................................107
Preparing for Debugging....................................................107
Symbolic Debugging and Optimizations..............................107
Using Options for Debug Information.................................108

Chapter 8: Creating and Using Libraries

Overview: Using Libraries..................................................111
Default Libraries...............................................................111
Managing Libraries............................................................112
Creating Libraries.............................................................113
Using Intel Shared Libraries...............................................115
Compiling for Non-shared Libraries.....................................115
  Overview: Compiling for Non-shared Libraries..............115
  Global Symbols and Visibility Attributes......................116
  Symbol Preemption...................................................117
  Specifying Symbol Visibility Explicitly........................118
  Other Visibility-related Command-line Options...............119

Chapter 9: gcc* Compatibility

gcc Compatibility............................................................121
gcc* Interoperability.......................................................125
gcc Interoperability........................................................125
Compiler Options for Interoperability...............................125
Predefined Macros for Interoperability..............................128
gcc Built-in Functions................................................129
Thread-local Storage..................................................130

Chapter 10: Language Conformance
Conformance to the C Standard.......................................133
Conformance to the C++ Standard..................................133
Exported Templates...................................................133
Template Instantiation..............................................136

Chapter 11: Porting Applications
Overview: Porting Applications....................................137
Modifying Your makefile..............................................138
Equivalent Macros....................................................140
Equivalent Environment Variables................................144
Other Considerations................................................144
Porting from GNU gcc* to Microsoft Visual C++*.............147

Chapter 12: Error Handling
Remarks, Warnings, and Errors...................................149
Using Source Code Verification....................................152
Source Checker Overview..........................................152
Interprocedural Analysis..........................................159
Local Program Analysis..........................................161
C/C++ Specific Analysis..........................................162
Fortran-Specific Analysis........................................163
OpenMP* Analysis................................................164

Chapter 13: Reference
ANSI Standard Predefined Macros..............................169
Additional Predefined Macros..................................169

Part II: Compiler Options
Overview: Compiler Options......................................176
Chapter 14: Alphabetical Compiler Options

Compiler Option Descriptions and General Rules

A

A, QA...........................................................................204
A-, QA-......................................................................205
alias-args..................................................................206
alias-const, Qalias-const...........................................207
align.................................................................209
ansi.................................................................210
ansi-alias, Qansi-alias............................................211
Ap64......................................................................212
arch........................................................................213
As..........................................................................215
auto-ilp32, Qauto-ilp32.............................................216
ax, Qax...............................................................217

B

B............................................................................220
Bdynamic..................................................................221
bigobj.................................................................223
Bstatic....................................................................224

C

c............................................................................225
C............................................................................226
c99, Qc99...............................................................227
check-uninit..........................................................228
complex-limited-range, Qcomplex-limited-range.........229
cxxlib.....................................................................230

D

D.............................................................................232
dD, QdD.................................................................................233
debug (Linux* OS and Mac OS* X)........................................234
debug (Windows* OS).........................................................237
diag, Qdiag.................................................................240
diag-dump, Qdiag-dump....................................................245
diag, Qdiag.................................................................246
diag-enable sc-include, Qdiag-enable:sc-include..............251
diag-enable sc-parallel, Qdiag-enable:sc-parallel............252
diag-error-limit, Qdiag-error-limit.................................254
diag-file, Qdiag-file.......................................................255
diag-file-append, Qdiag-file-append.................................257
diag-id-numbers, Qdiag-id-numbers...............................258
diag-once, Qdiag-once.....................................................260
dM, QdM.............................................................................261
dN, QdN.............................................................................262
dryrun..............................................................................263
dumpmachine.................................................................264
dumpversion....................................................................265
dynamic-linker...............................................................266
dynamiclib.......................................................................267

E

E.......................................................................................268
early-template-check........................................................269
EH.....................................................................................270
EP.....................................................................................271
export................................................................................272
export-dir.......................................................................273

F

F (Mac OS* X)......................................................................274
F (Windows*)....................................................................275
Fa.....................................................................................276
FA.....................................................................................277
fno-gnu-keywords ..................................................... 315
fno-implicit-inline-templates ...................................... 316
fno-implicit-templates ............................................... 317
fno-operator-names .................................................. 318
fno-rtti .................................................................... 318
fnon-call-exceptions .................................................. 319
fnon-lvalue-assign .................................................... 320
fnsplit, Qfnsplit ....................................................... 321
Fo ........................................................................... 323
fomit-frame-pointer, Oy ............................................. 324
fp-model, fp ............................................................. 325
Fp .......................................................................... 333
fp-model, fp ............................................................. 334
fp-port, Qfp-port ....................................................... 341
fp-relaxed, Qfp-relaxed ............................................. 342
fp-speculation, Qfp-speculation .................................. 343
fp-stack-check, Qfp-stack-check ................................. 345
fpack-struct ............................................................. 346
fpascal-strings .......................................................... 347
fpermissive ............................................................... 348
fpic ........................................................................ 348
fpie ........................................................................ 350
Fr ........................................................................... 351
FR .......................................................................... 352
fr32 ........................................................................ 353
freg-struct-return ..................................................... 354
fshort Enums ............................................................ 355
fsyntax-only ............................................................. 356
fstack-security-check, GS ........................................... 357
fstack-security-check, GS ........................................... 358
fsyntax-only ............................................................. 359
ftemplate-depth, Qtemplate-depth .................. 360
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fts-model</td>
<td>361</td>
</tr>
<tr>
<td>ftrapuv, Qtrapuv</td>
<td>362</td>
</tr>
<tr>
<td>ftz, Qftz</td>
<td>363</td>
</tr>
<tr>
<td>func-groups</td>
<td>366</td>
</tr>
<tr>
<td>funroll-loops</td>
<td>366</td>
</tr>
<tr>
<td>funroll-all-loops</td>
<td>366</td>
</tr>
<tr>
<td>funsigned-bitfields</td>
<td>367</td>
</tr>
<tr>
<td>funsigned-char</td>
<td>368</td>
</tr>
<tr>
<td>fverbose-asm</td>
<td>369</td>
</tr>
<tr>
<td>fvisibility</td>
<td>370</td>
</tr>
<tr>
<td>fvisibility-inlines-hidden</td>
<td>373</td>
</tr>
<tr>
<td>g, Zi, Z7</td>
<td>374</td>
</tr>
<tr>
<td>g0</td>
<td>375</td>
</tr>
<tr>
<td>G2, G2-p9000</td>
<td>376</td>
</tr>
<tr>
<td>G5, G6, G7</td>
<td>378</td>
</tr>
<tr>
<td>GA</td>
<td>380</td>
</tr>
<tr>
<td>gcc</td>
<td>381</td>
</tr>
<tr>
<td>gcc</td>
<td>382</td>
</tr>
<tr>
<td>gcc-name</td>
<td>383</td>
</tr>
<tr>
<td>gcc-version</td>
<td>385</td>
</tr>
<tr>
<td>Gd</td>
<td>386</td>
</tr>
<tr>
<td>gdwarf-2</td>
<td>387</td>
</tr>
<tr>
<td>Ge</td>
<td>388</td>
</tr>
<tr>
<td>Gf</td>
<td>389</td>
</tr>
<tr>
<td>GF</td>
<td>390</td>
</tr>
<tr>
<td>Gh</td>
<td>391</td>
</tr>
<tr>
<td>GH</td>
<td>392</td>
</tr>
<tr>
<td>Gm</td>
<td>393</td>
</tr>
<tr>
<td>global-hoist, Qglobal-hoist</td>
<td>394</td>
</tr>
<tr>
<td>Gr</td>
<td>395</td>
</tr>
<tr>
<td>GR</td>
<td>396</td>
</tr>
</tbody>
</table>
Gs........................................................................................................397
fstack-security-check, GS..................................................................398
GT........................................................................................................399
GX........................................................................................................400
gxx-name............................................................................................401
GY........................................................................................................402
Gz........................................................................................................403
GZ........................................................................................................403
H
H, QH....................................................................................................404
H (Windows*).....................................................................................405
help........................................................................................................406
helppragma, Qhelppragma..................................................................408
homeparams......................................................................................409
hotpatch.............................................................................................410
I
I..............................................................................................................411
i-dynamic............................................................................................412
i-static.................................................................................................412
icc.........................................................................................................412
idirafter...............................................................................................413
imacros...............................................................................................414
inline-calloc, Qinline-calloc..............................................................415
inline-debug-info, Qinline-debug-info..............................................416
inline-factor, Qinline-factor..............................................................417
inline-forceinline, Qinline-forceinline..............................................419
inline-level, Ob..................................................................................420
inline-max-per-compile, Qinline-max-per-compile..........................422
inline-max-per-routine, Qinline-max-per-routine............................423
inline-max-size, Qinline-max-size......................................................425
inline-max-total-size, Qinline-max-total-size....................................427
inline-min-size, Qinline-min-size......................................................428
ip, Qip.................................................................430
ip-no-inlining, Qip-no-inlining.................................431
ip-no-pinlining, Qip-no-pinlining..............................432
IPF-flt-eval-method0, QIPF-flt-eval-method0..............433
IPF-fltacc, QIPF-fltacc...........................................435
IPF-fma, QIPF-fma..................................................436
IPF-fp-relaxed, QIPF-fp-relaxed...............................436
ipo, Qipo..................................................................436
ipo-c, Qipo-c..........................................................438
ipo-jobs, Qipo-jobs..................................................439
ipo-S, Qipo-S..........................................................440
ipo-separate, Qipo-separate......................................441
ipp, Qipp..................................................................442
iprefx.....................................................................444
iquote.....................................................................445
isystem.................................................................446
ivdep-parallel, Qivdep-parallel.................................447
iwidthprefix............................................................448
iwidthprefixbefore...................................................449
J
J............................................................................450
K
Kc++, TP..................................................................451
kernel......................................................................452
L
I.............................................................................453
L.............................................................................454
LD..........................................................................455
link.........................................................................456
M
m...........................................................................457
M, QM....................................................................459
<table>
<thead>
<tr>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>multibyte-chars, Qmultibyte-chars</td>
</tr>
<tr>
<td>multiple-processes, MP</td>
</tr>
<tr>
<td>noBool</td>
</tr>
<tr>
<td>no-bss-init, Qnobss-init</td>
</tr>
<tr>
<td>nodefaultlibs</td>
</tr>
<tr>
<td>nolib-inline</td>
</tr>
<tr>
<td>logo</td>
</tr>
<tr>
<td>nostartfiles</td>
</tr>
<tr>
<td>nostdinc++</td>
</tr>
<tr>
<td>nostdlib</td>
</tr>
<tr>
<td>o</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>Oa</td>
</tr>
<tr>
<td>inline-level, Ob</td>
</tr>
<tr>
<td>Od</td>
</tr>
<tr>
<td>Og</td>
</tr>
<tr>
<td>fbuiltin, Oi</td>
</tr>
<tr>
<td>Op</td>
</tr>
<tr>
<td>openmp, Qopenmp</td>
</tr>
<tr>
<td>openmp-lib, Qopenmp-lib</td>
</tr>
<tr>
<td>openmp-link, Qopenmp-link</td>
</tr>
<tr>
<td>openmp-profile, Qopenmp-profile</td>
</tr>
<tr>
<td>openmp-report, Qopenmp-report</td>
</tr>
<tr>
<td>openmp-stubs, Qopenmp-stubs</td>
</tr>
<tr>
<td>openmp-task, Qopenmp-task</td>
</tr>
<tr>
<td>openmp-threadprivate, Qopenmp-threadprivate</td>
</tr>
<tr>
<td>opt-block-factor, Qopt-block-factor</td>
</tr>
<tr>
<td>opt-calloc</td>
</tr>
<tr>
<td>opt-class-analysis, Qopt-class-analysis</td>
</tr>
<tr>
<td>opt-jump-tables, Qopt-jump-tables</td>
</tr>
</tbody>
</table>
opt-loadpair, Qopt-loadpair ........................................... 535
opt-malloc-options ...................................................... 536
opt-mem-bandwidth, Qopt-mem-bandwidth ..................... 538
opt-mod-versioning, Qopt-mod-versioning ..................... 539
opt-multi-version-aggressive,
    Qopt-multi-version-aggressive ................................... 541
opt-prefetch, Qopt-prefetch ......................................... 542
opt-prefetch-initial-values,
    Qopt-prefetch-initial-values .................................... 543
opt-prefetch-issue-excl-hint,
    Qopt-prefetch-issue-excl-hint .................................. 545
opt-prefetch-next-iteration,
    Qopt-prefetch-next-iteration .................................. 546
opt-ra-region-strategy, Qopt-ra-region-strategy ............ 547
opt-report, Qopt-report ............................................... 549
opt-report-file, Qopt-report-file .................................. 550
opt-report-help, Qopt-report-help ............................... 551
opt-report-phase, Qopt-report-phase ............................ 552
opt-report-routine, Qopt-report-routine ....................... 554
opt-streaming-stores, Qopt-streaming-stores ................. 555
opt-subscript-in-range, Qopt-subscript-in-range ............ 557
Os ............................................................................ 558
Ot ............................................................................ 559
Ow ......................................................................... 560
Ox ............................................................................ 561
fomit-frame-pointer, Oy ............................................. 562
p ............................................................................. 564
P ............................................................................. 565
par-affinity, Qpar-affinity .......................................... 566
par-num-threads, Qpar-num-threads .............................. 568
par-report, Qpar-report .............................................. 569
Contents

par-runtime-control, Qpar-runtime-control.......................570
par-schedule, Qpar-schedule........................................572
par-threshold, Qpar-threshold....................................575
parallel, Qparallel..................................................577
pc, Qpc....................................................................578
pch..........................................................................579
pch-create, Yc..........................................................581
pch-dir.....................................................................583
pch-use....................................................................584
pie...........................................................................586
pragma-optimization-level.........................................587
prec-div, Qprec-div...................................................588
prec-sqrt, Qprec-sqrt................................................589
print-multi-lib..........................................................590
prof-data-order, Qprof-data-order...............................591
prof-dir, Qprof-dir....................................................592
prof-file, Qprof-file...................................................593
prof-func-groups......................................................595
prof-func-order, Qprof-func-order................................596
prof-gen, Qprof-gen...................................................598
prof-hotness-threshold, Qprof-hotness-threshold..........600
prof-src-dir, Qprof-src-dir.........................................601
prof-src-root, Qprof-src-root.....................................603
prof-src-root-cwd, Qprof-src-root-cwd.........................605
prof-use, Qprof-use..................................................606
pthread.................................................................608

A, QA.......................................................................609
A-, QA-....................................................................610
alias-args..................................................................611
alias-const, Qalias-const..........................................613
ansi-alias, Qansi-alias.............................................614
auto-ilp32, Qauto-ilp32.............................................615
ax, Qax..................................................................616
c99, Qc99...............................................................620
Qchkstk.................................................................621
complex-limited-range, Qcomplex-limited-range........622
Qcov-dir..................................................................623
Qcov-file...............................................................624
Qcov-gen...............................................................625
Qcxx-features.......................................................626
diag, Qdiag............................................................627
diag-dump, Qdiag-dump..........................................632
diag, Qdiag............................................................633
diag-enable sc-include, Qdiag-enable:sc-include........638
diag-enable sc-parallel, Qdiag-enable:sc-parallel.........640
diag-error-limit, Qdiag-error-limit..........................642
diag-file, Qdiag-file.................................................643
diag-file-append, Qdiag-file-append........................644
diag-id-numbers, Qdiag-id-numbers.......................646
diag-once, Qdiag-once.............................................647
dD, QdD.................................................................648
dM, QdM.................................................................649
dN, QdN..................................................................650
Weffc++, Qeffc++..................................................651
fast-transcendentals, Qfast-transcendentals..............653
fma, Qfma.............................................................654
falign-functions, Qfnalign.......................................655
fnsplit, Qfnsplit.....................................................656
fp-port, Qfp-port....................................................658
fp-relaxed, Qfp-relaxed..........................................659
fp-speculation, Qfp-speculation.............................660
fp-stack-check, Qfp-stack-check.............................662
ffreestanding, Qffreestanding...............................663
ftz, Qftz..................................................................664
global-hoist, Qglobal-hoist.........................................666
H, QH.....................................................................667
help-pragma, Qhelp-pragma........................................668
QIA64-fr32................................................................669
QIfist......................................................................670
inline-calloc, Qinline-calloc.......................................670
inline-debug-info, Qinline-debug-info...........................671
Qinline-dllimport.....................................................672
inline-factor, Qinline-factor.......................................673
inline-forceinline, Qinline-forceinline..........................675
inline-max-per-compile, Qinline-max-per-compile.........676
inline-max-per-routine, Qinline-max-per-routine............678
inline-max-size, Qinline-max-size...............................679
inline-max-total-size, Qinline-max-total-size...............681
inline-min-size, Qinline-min-size...............................683
Qinstall...................................................................684
minstruction, Qinstruction.........................................685
finstrument-functions, Qinstrument-functions...............687
ip, Qip.....................................................................689
ip-no-inlining, Qip-no-inlining....................................690
ip-no-pinlining, Qip-no-pinlining................................691
IPF-flt-eval-method0, QIPF-flt-eval-method0................692
IPF-fltacc, QIPF-fltacc.............................................694
IPF-fma, QIPF-fma....................................................695
IPF-fp-relaxed, QIPF-fp-relaxed................................695
ipo, Qipo..................................................................695
ipo-c, Qipo-c..........................................................697
ipo-jobs, Qipo-jobs...................................................698
ipo-S, Qipo-S...........................................................699
ipo-separate, Qipo-separate.......................................700
ipp, Qipp..................................................................701
prof-data-order, Qprof-data-order...............................779
prof-dir, Qprof-dir.....................................................780
prof-file, Qprof-file....................................................781
prof-func-order, Qprof-func-order...............................783
prof-gen, Qprof-gen..................................................785
prof-hotness-threshold, Qprof-hotness-threshold.............786
prof-src-dir, Qprof-src-dir..........................................788
prof-src-root, Qprof-src-root......................................789
prof-src-root-cwd, Qprof-src-root-cwd..........................791
prof-use, Qprof-use..................................................793
rcd, Qrcd................................................................795
rct, Qrct................................................................796
restrict, Qrestrict.....................................................797
Qsafeseh....................................................................798
save-temps, Qsave-temps............................................799
scalar-rep, Qscalar-rep.............................................801
mserialize-volatile, Qserialize-volatile.......................802
Qsfalign.....................................................................803
std, Qstd.....................................................................804
sox, Qsox....................................................................807
tbb, Qtbb.....................................................................808
tcheck, Qtcheck..........................................................809
tcollect, Qtcollect.......................................................810
tcollect-filter, Qtcollect-filter......................................812
ftemplate-depth, Qtemplate-depth...............................814
ftrapuv, Qftrapuv.......................................................815
unroll-aggressive, Qunroll-aggressive...........................816
unroll, Qunroll..........................................................817
use-asm, Quse-asm......................................................818
use-intel-optimized-headers,
    Quse-intel-optimized-headers..................................819
Quse-msasm-symbols..................................................820
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V (Linux* and Mac OS* X)</td>
<td>821</td>
</tr>
<tr>
<td>Qvc</td>
<td>822</td>
</tr>
<tr>
<td>vec, Qvec</td>
<td>823</td>
</tr>
<tr>
<td>vec-guard-write, Qvec-guard-write</td>
<td>825</td>
</tr>
<tr>
<td>vec-report, Qvec-report</td>
<td>826</td>
</tr>
<tr>
<td>vec-threshold, Qvec-threshold</td>
<td>827</td>
</tr>
<tr>
<td>wd, Qwd</td>
<td>829</td>
</tr>
<tr>
<td>we, Qwe</td>
<td>830</td>
</tr>
<tr>
<td>wn, Qwn</td>
<td>831</td>
</tr>
<tr>
<td>wo, Qwo</td>
<td>832</td>
</tr>
<tr>
<td>wr, Qwr</td>
<td>833</td>
</tr>
<tr>
<td>ww, Qww</td>
<td>834</td>
</tr>
<tr>
<td>x, Qx</td>
<td>835</td>
</tr>
<tr>
<td>rcd, Qrcd</td>
<td>839</td>
</tr>
<tr>
<td>rct, Qrct</td>
<td>840</td>
</tr>
<tr>
<td>reserve-kernel-regs</td>
<td>841</td>
</tr>
<tr>
<td>restrict, Qrestrict</td>
<td>842</td>
</tr>
<tr>
<td>RTC</td>
<td>843</td>
</tr>
<tr>
<td>S</td>
<td>845</td>
</tr>
<tr>
<td>Ssave-temps, Qsave-temps</td>
<td>846</td>
</tr>
<tr>
<td>scalar-rep, Qscalar-rep</td>
<td>848</td>
</tr>
<tr>
<td>shared</td>
<td>849</td>
</tr>
<tr>
<td>shared-intel</td>
<td>850</td>
</tr>
<tr>
<td>shared-libgcc</td>
<td>851</td>
</tr>
<tr>
<td>showIncludes</td>
<td>852</td>
</tr>
<tr>
<td>sox, Qsox</td>
<td>853</td>
</tr>
<tr>
<td>static</td>
<td>854</td>
</tr>
<tr>
<td>staticlib</td>
<td>855</td>
</tr>
<tr>
<td>static-intel</td>
<td>857</td>
</tr>
<tr>
<td>static-libgcc</td>
<td>858</td>
</tr>
</tbody>
</table>
std, Qstd.................................................................859
strict-ansi................................................................861

T
T............................................................................862
tbb, Qtbb................................................................863
Tc.............................................................................864
TC..........................................................................865
tcheck, Qtcheck........................................................866
tcollect, Qtcollect......................................................867
tcollect-filter, Qtcollect-filter.....................................868
Tp...........................................................................870
Kc++, TP................................................................871
tprofile, Qtprofile......................................................872
traceback................................................................873

U
u (Linux*)...............................................................875
u (Windows*)..........................................................876
U............................................................................877
unroll, Qunroll........................................................878
unroll-aggressive, Qunroll-aggressive......................879
use-asm, Quse-asm..................................................880
use-intel-optimized-headers,
  Quse-intel-optimized-headers................................881
use-msasm.............................................................882

V
v............................................................................883
V (Linux* and Mac OS* X)..........................................884
V (Windows*)..........................................................885
vd.............................................................................886
vec, Qvec...............................................................887
vec-guard-write, Qvec-guard-write............................888
vec-report, Qvec-report............................................889
Chapter 15: Quick Reference Guide and Cross Reference

Related Options........................................................................1013
Linking Tools and Options......................................................1013
Portability Options...................................................................1016
Unsupported Microsoft Visual Studio* Compiler
Options..................................................................................1028

Part III: Optimizing Applications

Chapter 16: Overview: Optimizing Applications

Optimizing with the Intel® Compiler........................................1033
Optimizing for Performance....................................................1035
Overview of Parallelism Method..............................................1036
Quick Reference Lists............................................................1038
Other Resources......................................................................1038

Chapter 17: Evaluating Performance

Performance Analysis............................................................1041
Using a Performance Enhancement Methodology.................1041
Intel® Performance Analysis Tools and Libraries...............1044
Performance Enhancement Strategies............................1046
Using Compiler Reports...........................................1055
  Compiler Reports Overview..................................1055
  Compiler Reports Quick Reference..........................1055
  Generating Reports........................................1057
  Interprocedural Optimizations (IPO) Report..............1060
  Profile-guided Optimization (PGO) Report...............1066
  High-level Optimization (HLO) Report....................1069
  High Performance Optimizer (HPO) Report...............1085
  Parallelism Report........................................1086
  Software Pipelining (SWP) Report (Linux* and Windows*)..1088
  Vectorization Report......................................1093
  OpenMP* Report........................................1099

Chapter 18: Using Compiler Optimizations
  Automatic Optimizations Overview.........................1101
  Enabling Automatic Optimizations.........................1102
  Targeting IA-32 and Intel(R) 64 Architecture Processors
    Automatically............................................1106
  Targeting Multiple IA-32 and Intel(R) 64 Architecture Processors
    for Run-time Performance..............................1110
  Targeting IA-32 and Intel 64 Architecture Processors
    Manually................................................1113
  Targeting IA-64 Architecture Processors Automatically....1117
  Restricting Optimizations................................1118

Chapter 19: Using Parallelism: OpenMP* Support
  OpenMP* Support Overview..................................1121
  OpenMP* Options Quick Reference..........................1122
OpenMP* Source Compatibility and Interoperability with Other
Compilers.................................................................1125
Using OpenMP*.........................................................1127
Parallel Processing Model..........................................1130
Worksharing Using OpenMP*.......................................1135
Verifying OpenMP* Using Parallel Lint..........................1150
OpenMP* Directives....................................................1161
  THREADPRIVATE threadprivate Directive....................1161
OpenMP* Advanced Issues..........................................1164
OpenMP* Examples...................................................1168
Libraries, Directives, Clauses, and Environmental Variables..1168
  OpenMP* Environment Variables...............................1168
  OpenMP* Directives and Clauses Summary....................1177
  OpenMP* Library Support........................................1180
Intel(R) Workqueuing Model" product="ClassicC..................1229
  Intel® Workqueuing Model Overview..........................1229
  Workqueuing Constructs........................................1230
  Workqueuing Example Function...............................1236

Chapter 20: Using Parallelism: Automatic Parallelization
  Auto-parallelization Overview..................................1237
  Auto-Parallelization Options Quick Reference.................1242
  Auto-parallelization: Enabling, Options, Directives, and
  Environment Variables............................................1243
  Programming with Auto-parallelization........................1244
  Programming for Multithread Platform Consistency..........1245

Chapter 21: Using Parallelism: Automatic Vectorization
  Automatic Vectorization Overview.............................1251
  Automatic Vectorization Options Quick Reference............1251
  Programming Guidelines for Vectorization.....................1253
  Vectorization and Loops.........................................1254
  Loop Constructs................................................1258
Chapter 22: Using Interprocedural Optimization (IPO)

Interprocedural Optimization (IPO) Overview

Interprocedural Optimization (IPO) Quick Reference

Using IPO

IPO-Related Performance Issues

IPO for Large Programs

Understanding Code Layout and Multi-Object IPO

Creating a Library from IPO Objects

Requesting Compiler Reports with the xi* Tools

Inline Expansion of Functions

Inline Function Expansion

Compiler Directed Inline Expansion of User Functions

Developer Directed Inline Expansion of User Functions

Chapter 23: Using Profile-Guided Optimization (PGO)

Profile-Guided Optimizations Overview

Profile-Guided Optimization (PGO) Quick Reference

Profile an Application

PGO Tools

PGO Tools Overview

code coverage Tool

test prioritization Tool

profmerge and proforder Tools

Using Function Order Lists, Function Grouping, Function Ordering, and Data Ordering Optimizations

Comparison of Function Order Lists and IPO Code Layout

PGO API Support

API Support Overview

PGO Environment Variables
Contents

Dumping Profile Information..........................1352
Interval Profile Dumping...............................1353
Resetting the Dynamic Profile Counters...............1355
Dumping and Resetting Profile Information............1355

Chapter 24: Using High-Level Optimization (HLO)
High-Level Optimizations (HLO) Overview..............1357
Loop Unrolling..............................................1359
Loop Independence........................................1361
Prefetching with Options................................1365

Chapter 25: Optimization Support Features
Prefetching Support.........................................1367
About Register Allocation..................................1371

Chapter 26: Programming Guidelines
Understanding Run-time Performance....................1375
Understanding Data Alignment..........................1379
Timing Your Application..................................1380
Applying Optimization Strategies.......................1381
Optimizing the Compilation Process.....................1392
Symbol Visibility Attribute Options (Linux* and Mac OS* X).........................................................1392

Part IV: Floating-point Operations

Chapter 27: Overview: Floating-point Operations

Chapter 28: Understanding Floating-point Operations
Using the -fp-model (/fp) Option..........................1399
Setting the FTZ and DAZ Flags...........................1403

Chapter 29: Tuning Performance
Overview: Tuning Performance...........................1407
Handling Floating-point Array Operations in a Loop Body......1407
Reducing the Impact of Denormal Exceptions......................1408
Avoiding Mixed Data Type Arithmetic Expressions..............1409
Using Efficient Data Types..............................................1412
Checking the Floating-point Stack State..............................1412

Chapter 30: Understanding IEEE Floating-point Operations
Overview: Understanding IEEE Floating-point Standard........1415
Floating-point Formats..................................................1415
Special Values.............................................................1415

Part V: Intrinsics Reference

Chapter 31: Overview: Intrinsics Reference
Details about Intrinsics.................................................1420
Naming and Usage Syntax..............................................1423
Links and Bibliography................................................1424

Chapter 32: Intrinsics for All Intel Architectures
Overview........................................................................1427
Integer Arithmetic Intrinsics...........................................1427
Floating-point Intrinsics..................................................1428
String and Block Copy Intrinsics......................................1431
Synchronization Primitives..............................................1432
Miscellaneous Intrinsics................................................1434

Chapter 33: Intrinsics for IA-64 Architecture
Overview........................................................................1437
Native Intrinsics ..........................................................1437
Lock and Atomic Operation Related Intrinsics......................1440
Load and Store Intrinsics................................................1444
Operating System Related Intrinsics................................1444
Chapter 34: Data Alignment, Memory Allocation, Intrinsics, and Inline Assembly
Overview.......................................................................1475
Alignment Support..........................................................1475
Allocating and Freeing Aligned Memory Blocks....................1477
Inline Assembly................................................................1477

Chapter 35: MMX(TM) Technology Intrinsics
Overview........................................................................1485
Details about MMX(TM) Technology Intrinsics......................1485
The EMMS Instruction: Why You Need It ...........................1486
EMMS Usage Guidelines...................................................1487
General Support Intrinsics................................................1488
Packed Arithmetic Intrinsics..............................................1491
Shift Intrinsics................................................................1493
Logical Intrinsics.............................................................1496
Compare Intrinsics..........................................................1496
Set Intrinsics..................................................................1498
Intrinsics for IA-64 Architecture........................................1500

Chapter 36: Intrinsics for Intel(R) Streaming SIMD Extensions
Overview........................................................................1503
Details about Intel(R) Streaming SIMD Extension Intrinsics...1503
Chapter 37: Intrinsics for Intel® Streaming SIMD Extensions 2

Overview.................................................................1549
Floating-point Intrinsics..............................................1550
  Arithmetic Intrinsics............................................1550
  Logical Intrinsics................................................1554
  Compare Intrinsics.............................................1555
  Conversion Intrinsics..........................................1566
  Load Intrinsics....................................................1570
  Set Intrinsics....................................................1573
  Store Intrinsics................................................1575
  Integer Intrinsics...............................................1577

Macro Function for Shuffle Operations.......................1543
Macro Functions to Read and Write Control Registers...1544
Macro Function for Matrix Transposition.....................1547

Chapter 37: Intrinsics for Intel(R) Streaming SIMD Extensions 2

Overview.....................................................................1549
Floating-point Intrinsics..........................................1550
  Arithmetic Intrinsics.........................................1550
  Logical Intrinsics..............................................1554
  Compare Intrinsics............................................1555
  Conversion Intrinsics.........................................1566
  Load Intrinsics..................................................1570
  Set Intrinsics....................................................1573
  Store Intrinsics................................................1575
  Integer Intrinsics...............................................1577

Macro Function for Shuffle Operations.......................1543
Macro Functions to Read and Write Control Registers...1544
Macro Function for Matrix Transposition.....................1547
Arithmetic Intrinsics................................................1577
Logical Intrinsics.....................................................1586
Shift Intrinsics........................................................1587
Compare Intrinsics..................................................1592
Conversion Intrinsics..............................................1595
Move Intrinsics.......................................................1597
Load Intrinsics........................................................1598
Set Intrinsics..........................................................1599
Store Intrinsics.......................................................1603
Miscellaneous Functions and Intrinsics.....................1605
   Cacheability Support Intrinsics ..............................1605
   Miscellaneous Intrinsics ......................................1608
   Casting Support Intrinsics ....................................1614
   Pause Intrinsic...................................................1614
   Macro Function for Shuffle...................................1616

Chapter 38: Intrinsics for Intel(R) Streaming SIMD
   Extensions 3
   Overview..............................................................1617
   Integer Vector Intrinsics ......................................1617
   Single-precision Floating-point Vector Intrinsics .........1618
   Double-precision Floating-point Vector Intrinsics ........1619
   Miscellaneous Intrinsics ......................................1621
   Macro Functions ................................................1621

Chapter 39: Intrinsics for Intel(R) Supplemental
   Streaming SIMD Extensions 3
   Overview..............................................................1623
   Addition Intrinsics ...............................................1623
   Subtraction Intrinsics ..........................................1625
   Multiplication Intrinsics ......................................1627
   Absolute Value Intrinsics ......................................1628
   Shuffle Intrinsics................................................1629
Concatenate Intrinsics ..........................................................1630
Negation Intrinsics ..............................................................1631

Chapter 40: Intrinsics for Intel(R) Streaming SIMD Extensions 4
Overview.............................................................................1639
Vectorizing Compiler and Media Accelerators.......................1639
   Overview: Vectorizing Compiler and Media
   Packed Blending Intrinsics .............................................1639
   Floating Point Dot Product Intrinsics .........................1641
   Packed Format Conversion Intrinsics .........................1641
   Packed Integer Min/Max Intrinsics .........................1642
   Floating Point Rounding Intrinsics ..............................1644
   DWORD Multiply Intrinsics ............................................1645
   Register Insertion/Extraction Intrinsics ......................1645
   Test Intrinsics ..........................................................1646
   Packed DWORD to Unsigned WORD Intrinsic .................1648
   Packed Compare for Equal Intrinsic .........................1648
   Cacheability Support Intrinsic ....................................1648
   Efficient Accelerated String and Text Processing ..........1648
   Overview......................................................................1648
   Packed Compare Intrinsics .........................................1649
   Application Targeted Accelerators Intrinsics ...............1651

Chapter 41: Intrinsic Performance Across Intel Architectures
Overview.............................................................................1655
Intrinsics that Increase Performance Across All IA...............1655
Performance of MMX(TM) Technology Intrinsics ..................1659
Performance of Intel(R) Streaming SIMD Extensions
   Intrinsics.......................................................................1663
Chapter 42: Intrinsics for Advanced Encryption Standard Implementation
Overview.................................................................................1671
Intrinsics for Carry-less Multiplication Instruction and Advanced Encryption Standard Instructions...............1672

Chapter 43: Intrinsics for Converting Half Floats
Overview.................................................................................1675
Intrinsics for Converting Half Floats...........................................1676

Part VI: Compiler Reference

Chapter 44: Intel C++ Compiler Pragmas
Overview: Intel® C++ Compiler Pragmas............................1681
Intel-Specific Pragmas.............................................................1682
Intel-specific Pragma Reference..............................................1683
alloc_section.........................................................................1683
distribute_point.................................................................1684
intel_omp_task.................................................................1688
intel_omp_taskq...............................................................1689
ivdep.....................................................................................1691
loop_count............................................................................1693
memref_control.................................................................1695
novector..............................................................................1700
optimize..............................................................................1701
optimization_level.............................................................1703
parallel/noparallel.............................................................1705
prefetch/noprefetch.............................................................1706
swp/noswp.............................................................................1710
unroll/nounroll.................................................................1711
unroll_and_jam/nounroll_and_jam.................................1713
unused...............................................................................1716
Chapter 45: Intel Math Library

Overview: Intel® Math Library...........................................1723
Using the Intel Math Library..........................................1723
Math Functions..................................................................1727
Function List...............................................................1727
Trigonometric Functions..............................................1733
Hyperbolic Functions...............................................1738
Exponential Functions...............................................1740
Special Functions....................................................1745
Nearest Integer Functions........................................1749
Remainder Functions..................................................1752
Miscellaneous Functions...........................................1753
Complex Functions....................................................1759
C99 Macros................................................................1765

Chapter 46: Intel C++ Class Libraries

Introduction to the Class Libraries.................................1767
Overview: Intel C++ Class Libraries..............................1767
Hardware and Software Requirements..............................1767
About the Classes..........................................................1767
Details About the Libraries...........................................1768
C++ Classes and SIMD Operations...............................1769
Capabilities of C++ SIMD Classes...............................1773
Integer Vector Classes..................................................1775
Overview: Integer Vector Classes.................................1775
Terms, Conventions, and Syntax Defined.......................1776
Rules for Operators........................................................1778
Assignment Operator.....................................................1782
Logical Operators.........................................................1782
Addition and Subtraction Operators..........................1784
Multiplication Operators........................................1787
Shift Operators..................................................1789
Comparison Operators..........................................1791
Conditional Select Operators...............................1793
Debug Operations...............................................1796
Unpack Operators..............................................1799
Pack Operators................................................1804
Clear MMX™ State Operator..................................1804
Integer Functions for Streaming SIMD Extensions.....1805
Conversions between Fvec and Ivec.......................1806
Floating-point Vector Classes................................1807
Overview: Floating-point Vector Classes...............1807
Fvec Notation Conventions..................................1808
Data Alignment...............................................1809
Conversions....................................................1809
 Constructors and Initialization............................1810
Arithmetic Operators.........................................1812
Minimum and Maximum Operators..........................1818
Logical Operators.............................................1819
Compare Operators...........................................1821
Conditional Select Operators for Fvec Classes........1826
Cacheability Support Operators............................1831
Debug Operations.............................................1831
Load and Store Operators...................................1833
Unpack Operators.............................................1833
Move Mask Operators........................................1834
Classes Quick Reference....................................1834
Programming Example........................................1844
C++ Librarcref_cls/common/y Extensions..........................1846
Introduction................................................1846
Intel's valarray implementation..........................1846
Chapter 47: Intel’s C/C++ Language Extensions

Introduction........................................................................................................1851
Intel’s C++ lambda extensions........................................................................1851
  Introduction....................................................................................................1851
Details on Using Lambda Expressions in C++.......................................1851
Understanding Lambda-Capture..............................................................1853
Lambda Function Object..............................................................................1855
INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL(R) PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.


* Other names and brands may be claimed as the property of others.

Copyright (C) 1996-2009, Intel Corporation. All rights reserved.

Portions Copyright (C) 2001, Hewlett-Packard Development Company, L.P.
Getting Help and Support

The Intel® C++ Compiler lets you build and optimize C/C++ applications for the Linux* OS (operating system). You can use the compiler on the command line or in the Eclipse* integrated development environment.

For more information about the compiler features and other components, see your Release Notes.

This documentation assumes that you are familiar with the C++ programming language and with your processor's architecture. You should also be familiar with the host computer’s operating system.

Product Website and Support

For general information on support for Intel software products, visit the Intel web site http://www.intel.com/software/products/

At this site, you will find comprehensive product information, including:

- Links to each product, where you will find technical information such as white papers and articles
- Links to user forums
- Links to news and events

To find technical support information, to register your product, or to contact Intel, please visit: http://www.intel.com/software/products/support/

For additional information, see the Technical Support section of your Release Notes.

System Requirements

For detailed information on system requirements, see the Release Notes.
Introduction

Introducing the Intel® C++ Compiler

The Intel® C++ Compiler can generate code for IA-32, Intel® 64, or IA-64 architecture applications on any Intel®-based Linux* system. IA-32 architecture applications (32-bit) can run on all Intel®-based Linux systems. Intel® 64 architecture applications and IA-64 architecture applications can run only on Intel® 64 architecture-based or IA-64 architecture-based Linux systems. You can use the compiler on the command line or in the Eclipse* integrated development environment.

You can find further information in the following documents:

- Building Applications
- Compiler Options
- Optimizing Applications
- Floating-point Operations
- Compiler Reference
- Intrinsics Reference

Notational Conventions

Information in this documentation applies to all supported operating systems and architectures unless otherwise specified.

This documentation uses the following conventions:

Notational Conventions

this type Indicates command-line or option arguments.

This type Indicates a code example.

This type Indicates what you type as input.
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>This type</strong></td>
<td>Indicates menu names, menu items, button names, dialog window names, and other user-interface items.</td>
</tr>
<tr>
<td><strong>File&gt;Open</strong></td>
<td>Menu names and menu items joined by a greater than (&gt;) sign indicate a sequence of actions. For example, &quot;Click File&gt;Open&quot; indicates that in the File menu, click Open to perform this action.</td>
</tr>
<tr>
<td>{value</td>
<td>value}</td>
</tr>
<tr>
<td>[item]</td>
<td>Indicates items that are optional.</td>
</tr>
<tr>
<td>item[, item]...</td>
<td>Indicates that the item preceding the ellipsis (three dots) can be repeated.</td>
</tr>
<tr>
<td>Windows* OS</td>
<td>These terms refer to all supported Microsoft* Windows* operating systems.</td>
</tr>
<tr>
<td>Windows operating system</td>
<td></td>
</tr>
<tr>
<td>Linux* OS</td>
<td>These terms refer to all supported Linux* operating systems.</td>
</tr>
<tr>
<td>Linux operating system</td>
<td></td>
</tr>
<tr>
<td>Mac OS* X</td>
<td>These terms refer to Intel®-based systems running the Mac OS* X operating system.</td>
</tr>
<tr>
<td>Mac OS X operating system</td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows XP*</td>
<td>An asterisk at the end of a word or name indicates it is a third-party product trademark.</td>
</tr>
<tr>
<td>compiler option</td>
<td>This term refers to Windows* OS options, Linux* OS options, or MAC OS* X options that can be used on the compiler command line.</td>
</tr>
<tr>
<td>Option Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>/option or -option</td>
<td>A slash before an option name indicates the option is available on Windows OS. A dash before an option name indicates the option is available on Linux OS* and Mac OS* X systems. For example: Windows option: /fast Linux and Mac OS X option: -fast Note: If an option is available on Windows* OS, Linux* OS, and Mac OS* X systems, no slash or dash appears in the general description of the option. The slash and dash will only appear where the option syntax is described.</td>
</tr>
<tr>
<td>/option:argument or -option argument</td>
<td>Indicates that an option requires a argument (parameter). For example, you must specify an argument for the following options: Windows OS option: /Qdiag-error-limit:n Linux OS and Mac OS X option: -diag-error-limit n</td>
</tr>
<tr>
<td>/option:keyword or -option keyword</td>
<td>Indicates that an option requires one of the keyword values.</td>
</tr>
<tr>
<td>/option[:keyword ] or -option [keyword ]</td>
<td>Indicates that the option can be used alone or with an optional keyword.</td>
</tr>
<tr>
<td>option[n] or option[:n] or option[n]</td>
<td>Indicates that the option can be used alone or with an optional value; for example, in /Qfnalign[:n] and -falign-functions[:n], the n can be omitted or a valid value can be specified for n.</td>
</tr>
<tr>
<td>option[-]</td>
<td>Indicates that a trailing hyphen disables the option; for example, /Qglobal_hoist-disables the Windows OS option /Qglobal_hoist.</td>
</tr>
</tbody>
</table>
Indicates that "no" or "no-" preceding an option disables the option. For example:
In the Windows OS option /[no]traceback, 
/traceback enables the option, while /no-traceback disables it.
In the Linux OS and Mac OS X option -[no-]global_hoist, -global_hoist enables the option, while -no-global_hoist disables it.
In some options, the "no" appears later in the option name; for example, -fno-alias disables the -falias option.

Related Information

Associated Intel Documents

You are strongly encouraged to read the following books for in-depth understanding of threading. Each book discusses general concepts of parallel programming by explaining a particular programming technology:

- Intel® Threading Building Blocks
  See http://oreilly.com/catalog/9780596514808/

- OpenMP* technology
  See http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11387

The following Intel documents provide additional information about the Intel® C++ Compiler, Intel® architecture, Intel® processors, or tools:

- Using the Intel® License Manager for FLEXlm*
- Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture, Intel Corporation
• Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 2B: Instruction Set Reference, N-Z, Intel Corporation
• Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 3B: System Programming Guide, Intel Corporation
• Intel® 64 and IA-32 Architectures Optimization Reference Manual
• Intel® Itanium® Architecture Software Developer’s Manual - Volume 1: Application Architecture, Revision 2.2
• Intel® Itanium® Architecture Software Developer’s Manual - Volume 2: System Architecture, Revision 2.2
• Intel® Itanium® Architecture Software Developer’s Manual - Volume 3: Instruction Set Reference, Revision 2.2
• Intel® Processor Identification with the CPUID Instruction, Intel Corporation, doc. number 241618
• IA-64 Architecture Assembler User’s Guide
• IA-64 Architecture Assembly Language Reference Guide

Most Intel documents can be found at the Intel web site http://www.intel.com/software/products/

Optimization and Vectorization Terminology and Technology

The following documents provide details on basic optimization and vectorization terminology and technology:

• Intel® Architecture Optimization Reference Manual

• High Performance Compilers for Parallel Computers, Michael J. Wolfe. Addison-Wesley, Redwood City. 1996.


• An Auto-vectorizing Compiler for the Intel® Architecture, Aart Bik, Paul Grey, Milind Girkar, and Xinmin Tian. Submitted for publication

• Efficient Exploitation of Parallelism on Pentium® III and Pentium® 4 Processor-Based Systems, Aart Bik, Milind Girkar, Paul Grey, and Xinmin Tian.


**Additional Training**

For additional technical product information including white papers about Intel compilers, open the page associated with your product at http://www.intel.com/software/products/
Part I

Building Applications

Topics:
- Overview: Building Applications
- Building Applications with Eclipse*
- Building Applications from the Command Line
- Using Preprocessor Options
- Modifying the Compilation Environment
- Debugging
- Creating and Using Libraries
- gcc* Compatibility
- Language Conformance
- Porting Applications
- Error Handling
- Reference
Overview: Building Applications

This section describes how to build your applications with the Intel® C++ Compiler. It includes information on using the compiler on the command line and how to use the compiler with supported integrated development environments. Also in this section are helpful topics on linking, debugging, libraries, compatibility, and language conformance.

Introduction to the Compiler

You can invoke the Intel® C++ Compiler from a system command prompt with the icc or icpc command after setting the environment variables.

Getting Help

On the command line, you can execute icpc -help for a summary of command-line options.

Other Resources

For new features and known issues, see the Release Notes.

For general product information or information on support for Intel software products, visit the Intel web site: http://www.intel.com/software/products/. At this site, you will find comprehensive product information, including:

- Links to each product, where you will find technical information such as white papers and articles
- Links to user forums
- Links to news and events

To find technical support information, to register your product, or to contact Intel, please visit: http://www.intel.com/software/products/support/. For additional information, see the Technical Support section of your Release Notes.
Compilation Phases

The Intel® C++ Compiler processes C and C++ language source files. By default, the compiler performs the compile and link phases of compilation and produces an executable file. The compiler also determines which compilation phases to perform based on the file name extension and the compilation options specified. The compiler passes object files and any unrecognized file names to the linker. The linker then determines whether the file is an object file or a library file.

Default Behavior of the Compiler

If you do not specify any options when you invoke the Intel® C++ Compiler, the compiler performs the following:

- produces an executable file
- invokes options specified in a configuration file first
- invokes options specified in the CL environment variable
- searches for header files in known locations
- sets 16 bytes as the strictest alignment constraint for structures
- displays error and warning messages
- uses ANSI with extensions
- performs standard optimizations
- on operating systems that support characters in Unicode* (multi-byte) format, the compiler will process file names containing these characters

Default Output Files

A default invocation of the compiler requires only a C or C++ file name argument, such as:

```
icpc x.cpp
```  

You can compile more than one input file:

```
icpc x.cpp y.cpp z.cpp
```  

This command does the following:

- Compiles and links three input source files.
- Produces one executable file, `a.out`, in the current directory.
Using Compiler Options

A compiler option is a case-sensitive, command-line expression used to change the compiler’s default operation. Compiler options are not required to compile your program, but they are extremely useful in helping you control different aspects of your application, such as:

- code generation
- optimization
- output file (type, name, location)
- linking properties
- size of the executable
- speed of the executable

See Option Categories below for a broader view of the option capabilities included with the Intel® C++ Compiler.

Command-line Syntax

When you specify compiler options on the command-line, the following syntax applies:

```
icc [options] [@response_file] file1 [file2...]
```

where `options` represents zero or more compiler options

where `file` is any of the following:

- C or C++ source file (.C .c .cc .cpp .cxx .c++ .i .ii)
- assembly file (.s .S)
- object file (.o)
- static library (.a)

If you are compiling just C language sources, invoke the compiler with `icc`. You should invoke the compiler with `icpc` if you are compiling just C++ language sources or a combination of C and C++.

The optional `response_file` is a text file that lists compiler options you want to include during compilation. See Using Response Files.
The compiler reads command-line options from left to right. If your compilation includes competing options, then the compiler uses the one furthest to the right. In other words, "the last one wins." In this example:

```
icc -xP main.c file1.c -xW file2.c
```

the compiler sees `-xP` and `-xW` as two forms of the same option where only one form can be used. Since `-xW` is last (furthest to the right), it wins.

All options specified on the command line are used to compile each file. The compiler will NOT compile individual files with specific options as this example may suggest:

```
icc -O3 main.c file1.c -mp1 file2.c
```

It may seem that `main.c` and `file1.c` are compiled with `-O3`, and `file2.c` is compiled with the `-mp1` option. This is not the case. All files are compiled with both options.

A rare exception to this general rule is the `-x` type option:

```
icc -x c file1 -x c++ file2 -x assembler file3
```

where the `type` argument identifies each file type for the compiler.

**Default Operation**

The compiler invokes many options by default. For example, the `-O2` option is on by default for systems based on IA-32 architecture. In this simple example, the compiler includes `-O2` (and the other default options) in the compilation:

```
icc main.c
```

See the Compiler Options reference for the default status of each compiler option.

Each time you invoke the compiler, options listed in the corresponding configuration file (icc.cfg or icp.cfg) override any competing default options. For example, if your icc.cfg file includes the `-O3` option, the compiler will use `-O3` rather than the default `-O2` option. Use the configuration file to list options you'd like the compiler to use for every compilation. See Using Configuration Files.

Finally, options used on the command-line override any competing options specified elsewhere (default options, options in the configuration file). If you specify `-O1` on the command line, this option setting would "win" over competing option defaults and competing options in configuration files, in addition to competing options in the CL environment variable.
Certain \texttt{#pragma} statements in your source code can override competing options specified on the command line. For example, if a function in your code is preceded by \texttt{#pragma optimize("", off)}, then optimization for that function is turned off, even though \texttt{-O2} optimization is on by default, \texttt{-O3} is listed in the configuration file, and \texttt{-O1} is specified on the command-line for the rest of the program.

**Using Options with Arguments**

Compiler options can be as simple as a single letter, such as \texttt{-E/E}. However, many options accept or require arguments. The \texttt{-O} option, for example, accepts a single-value argument that the compiler uses to determine the degree of optimization. Other options require at least one argument and can accept multiple arguments. For most options that accept arguments, the compiler will warn you if your option and argument are not recognized. If you specify \texttt{-O9}, for example, the compiler will issue a warning, ignore the unrecognized \texttt{-O9} option, and proceed with compilation.

While the \texttt{-O} option does not require an argument, there are other options that must include an argument. The \texttt{-I} option requires an argument that identifies the directory to add to the include file search path. If you use this option without an argument, the compiler will not finish compilation.

See the Compiler Options reference for a complete description of options and their supported arguments.

**Other Forms of Options**

You can toggle some options on or off by using the negation convention. For example, the \texttt{-complex-limited-range} option, and many others, include a negation form, \texttt{-no-complex-limited-range}, to change the state of the option. Since this option is disabled by default, using \texttt{-complex-limited-range} on the command line would toggle it to the "ON" state.

**Option Categories**

When you invoke the Intel C++ Compiler and specify a compiler option, you have a wide range of choices to influence the compiler’s default operation. Intel compiler options typically correspond to one or more of the following categories:

- Advanced Optimization
- Code Generation
- Compatibility
• Component Control
• Data
• Deprecated
• Diagnostics
• Floating Point
• Help
• Inlining
• Interprocedural Optimizations (IPO)
• Language
• Linking/Linker
• Miscellaneous
• OpenMP and Parallel Processing
• Optimization
• Output
• Profile Guided Optimization (PGO)
• Preprocessor

To see which options are included in each category, invoke the compiler from the command line with the \texttt{-help} category option. For example:

\texttt{icc -help codegen}

will print to \texttt{stdout} the names and syntax of the options in the Code Generation category.

**Using Compiler Options in the Integrated Development Environment**

If you use the Intel compiler with the Eclipse integrated development environment (IDE) to build your applications, you can specify compiler options for the entire project or individual source files. The compiler integration with the IDE also lets you specify options on the command-line if you need an option that’s not included on the IDE Property Pages.

See Building Applications with Eclipse
Saving Compiler Information in Your Executable

If you want to save information about the compiler in your executable, use the -sox (Linux*) or /Qsox (Windows*) option. When you use this option, the following information is saved:

- compiler version number
- compiler options that were used to produce the executable

**On Linux OS:**
To view the information stored in the object file, use the following command:

```
objdump -sj comment a.out
strings -a a.out |grep comment:
```

**On Windows OS:**
To view the linker directives stored in string format in the object file, use the following command:

```
link /dump /drectives filename.obj
```

In the output, the `?-comment` linker directive displays the compiler version information.

To search your executable for compiler information, use the following command:

```
findstr "Compiler" filename.exe
```

This searches for any strings that have the substring "Compiler" in them.
Building Applications with Eclipse*

Overview: Eclipse Integration

Eclipse* is an open source software development project dedicated to providing a robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools. It is an extensible, open-source integrated development environment (IDE).

The CDT project is dedicated to providing a fully functional C/C++ IDE for the Eclipse platform. CDT is layered on Eclipse and provides a C/C++ development environment perspective.

**NOTE.** Eclipse and CDT are not bundled with the Intel compiler. They must be obtained separately.

The Intel C++ Compiler for Linux* OS provides an integration (also known as an extension) to the Eclipse/CDT IDE that lets you develop, build, and run your Intel C/C++ projects in a visual, interactive environment.

Similarly, the Intel® Debugger for Linux* OS includes a debugger integration to Eclipse and CDT. This integration is also included with the compiler installation.

To use the compiler and the debugger integrations, add them to Eclipse using the **Add an Extension Location** feature. For more information, see **Starting Eclipse**.

This section includes the following topics:

- Starting Eclipse
- Creating a New Project
- Setting Properties
- Updating an Existing Intel(R)C++ Project
- Updating a GCC* Project to Use the Intel(R) Compiler
- Project Types and Makefiles

See Also
- Building Applications with Eclipse*

http://www.eclipse.org/ for further information about Eclipse
http://www.eclipse.org/cdt/ for further information about CDT
Multi-version Compiler Support

For Eclipse Executable, Shared Library, Static Library, and Makefile projects, you can select different versions of the Intel compiler to compile your Eclipse Intel project. Eclipse configurations (and the toolchain that the configuration is based on) are used to provide this support. Refer to the Release Notes for a list of the currently supported compiler versions by platform.

Select the version of the Intel compiler to build your project with. You do this by selecting the configuration associated with the desired version of the compiler prior to building your project. You can create the desired configurations for the versions of the compiler that you would like to build either when you first create an Intel project, or, later, through the Manage Configurations interface within the IDE, accessible via the project’s properties.

To create configurations using the Manage Configurations interface:

1. Right click the project, select Properties > C/C++ Build > Settings
2. On the Configuration line, select the Manage configurations button and then New.

Within configurations, you can set distinct project properties, like compiler options, to be used with different versions of the Intel compiler and freely select and modify which version of the compiler with which to build by changing the active configuration. The active configuration is the configuration that is in effect when your application is built within the Eclipse IDE.

If you have multiple instances of the same major version of the compiler installed on your system (and, for an Eclipse Executable, Shared Library, or Static Library Project, a configuration with that major version is active) with different minor versions, the Eclipse IDE will, by default, use the compiler with its environment established, via execution of <install-dir>/bin/iccvars.*sh. If no compiler environment is established for such a project, then the most current compiler, that is, the one with the highest minor version number, will be used. For an Eclipse/CDT Makefile Project, the compiler environment must be established to enable successful invocation of the compiler, by default. Note also, that for any project, you can set the compiler environment by specifying it within Eclipse. This compiler specification overrides any specification established outside of Eclipse.

Starting Eclipse

If Eclipse and the CDT are installed on your system, follow these steps to use the Intel® C++ Compiler with Eclipse:

1. Initialize the compiler environment by setting environment variables. You can do this by executing iccvars.sh (or iccvars.csh) with the 'source' command. For example, for a root installation to the default directory:

source /opt/intel/Compiler/version_number/package_id/bin/iccvars.sh <arg>
The script takes an argument specifying architecture:
- ia32: Compiler and libraries for IA-32 architectures only
- intel64: Compiler and libraries for Intel® 64 architectures only
- ia64: Compiler and libraries for IA-64 architectures only

Be sure the `LANG` environment variable is set correctly:
```bash
export LANG=en_US
```

Start Eclipse and indicate the JRE, for example:
```bash
<eclipse-install-dir>/eclipse/eclipse -vm
<jre-install-dir>/jrockit-R26.4.0-jre1.5.0_06/bin/java -vmargs -Xmx256m
```

To add the Intel C++ Compiler product extension to your Eclipse configuration, follow these steps from within Eclipse.

1. Open the Product Configuration page by selecting Help > Software Updates > Manage Configuration
2. Under Available Tasks, select Add An Extension Location. A directory browser will open.
3. Browse to the appropriate Eclipse directory; Intel integrations are provided for CDT version 5.0 so the directory path is:
   ```bash
   /opt/intel/Compiler/version_number/package_id/eclipse_support/cdt5.0/eclipse
   ```
4. When asked to restart Eclipse, select Yes. When Eclipse restarts, you will be able to create and work with CDT projects that use the Intel C++ compiler.

If you also installed the Intel Debugger (idb) product extension along with the idb Eclipse product extension and would like to use idb within Eclipse, you should add the idb product extension site to your Eclipse configuration in the same way. For example, if you installed idb as root to the default directory, the idb Eclipse product extension site will be located at:
```bash
/opt/intel/Compiler/version_number/package_id/idb/eclipse_support/cdt5.0/eclipse
```

Creating a Simple Project

Creating a New Project

To create a simple project, start Eclipse* and follow these steps:

1. From the Eclipse File menu, select New > C Project. The C Project wizard opens.
2. For Project name, type `hello_world`. Check the Use default box or specify a directory for your Executable C Project.
3. In the Project Types list, expand the Executable project type list using the arrow character and select Hello World ANSI C Project. In the Toolchain list, select Intel(R) Toolchain for Executable on platform-id where platform-id is either IA-32 or Intel® 64.
4. The Basic Settings page allows you to specify template information, such as Author and Copyright notice, which will appear as a comment at the top of the generated source file. You can also specify the Hello world greeting string to be displayed by your hello_world program as well as a Source directory relative to the project where your generated source file will be created. When you are done filling in the desired fields, click Next.

5. The Select Configurations page opens, allowing you to specify the platforms and configurations for deployment. By default, a Debug and Release configuration will be created for the selected toolchain. Click Finish to complete creation of your new hello_world project.

6. If you are not currently in the C/C++ Development Perspective, you will see the Open Associated Perspective dialog box. In the C/C++ Perspective, click Yes to proceed. In the Project Explorer view, you should now see an entry for your hello_world project.

The next step is Adding a C Source File.

Adding a C Source File

After Creating a New Project, you can add additional source files, then build and run your completed project.

If, however, you chose to create a project type of Empty project for the hello_world project, you would follow these steps to add a hello.c source file to the hello_world project.

1. Select the hello_world project in the Project Explorer view.

2. From the Eclipse* File menu, select New > Source File. Enter hello.c in the Source File text box of the New Source File dialog. Click Finish to add the file to the hello_world project.

3. In the Editor view, add your code for hello.c.

When your code is complete, save your file using File > Save, then proceed to Building a Project.
## Building a Project

To build your project, make sure your `hello_world` project is selected in the **Project Explorer** view, then select **Build all** from the Eclipse® **Project** menu. See the **Build** results in the **Console** view.

```
**** Build of configuration Release for project hello_world ****
```

```
make -k all
```

Building file: ../main.c

Invoking: Compiler

```
icc -MMD -MP -MF"main.d" -MT"main.d" -c -o "main.o" "../main.c"
```

Finished building: ../main.c

Building target: hello_world

Invoking: Linker

```
icc -o"hello_world" ./main.o
```

Finished building target: hello_world

Build complete for project hello_world

## Running a Project

After **Building a Project**, you can run your project by following these steps:

1. Select the **hello_world** project in the **Project Explorer** view.
2. Select **Run > Run As > Local C/C++ Application**.
3. On the **Launch Debug Configuration Selection** dialog, select either the **Intel(R) Debugger** (if installed) or the GDB Debugger*, then click **OK**.
4. After the executable runs, the output of `hello.c` appears in the **Console** view.
Intel Error Parser

The Intel® C/C++ Error Parser (selected by default) lets you track compile-time errors in Eclipse*/CDT*. To confirm that the Intel® C/C++ Error Parser is active:

1. Select the hello_world project in the Project Explorer view.
2. Select Project > Properties.
3. In the Properties dialog boxx, select C/C++ Build>Settings.
4. Click the Error Parsers tab. The Intel(R) C/C++ Error Parser should already be selected. Do not check the CDT Visual C Error Parser.
5. Click OK to update your choices, if you have changed any settings, and close the dialog.

Using the Intel C/C++ Error Parser

If you introduce an error into your hello.c program, such as:

```
#include <xstdio.h>
```

then compile hello.c, the error is reported in the Problems view and a ❌ marker appears in the source file at the line where the error was detected. This processing occurs automatically because the Intel C/C++ Error Parser detects and manages diagnostics generated by the Intel® C++ Compiler. You can double-click on each error in the Problems view to visit the source line in the Editor view.
Correct the error, then rebuild your project.

**Setting Properties**

The Intel® C++ Compiler integration with Eclipse*/CDT* lets you specify compiler, linker, and archiver options at the project and source file level.

**Setting Options for Your Project**

Follow these steps to set options for your project:

1. Select your project in the **Project Explorer** view.
2. From the Eclipse toolbar, select **Project > Properties > C/C++ Build > Settings**.
3. Under **Tool Settings**, click an option category for **C Compiler** or **Linker**.
4. Set the option(s) you want to add to your project compilations, then open other categories if necessary. You can specify option settings independently on each configuration by changing the configuration selection in the Configuration drop-down box. You can set the same option across multiple configurations with one operation. Use the Configuration drop-down box to select this mode.

5. Click OK to complete your selections.

To reset properties to their default setting, click Restore Defaults. The Restore Defaults button appears on each property page, but the Restore Defaults action applies to ALL property pages.

Setting Options for Source Files

In addition to setting compiler options for your entire project, you can also specify options for any source file in your project.

1. Select a source file in your project.
2. Right-click on the source file, and select Properties from the context menu.
3. Select C/C++ Build>Settings to display the Tool Settings for changing compiler options.
4. Change options as described above.
5. In the C/C++ Build dialog box, you may choose Exclude from build if you do not want the source file included in the build.

Specifying Properties

Some properties use check boxes, drop-down boxes, or other input mechanisms to specify a compiler option.
Several options let you specify arguments. Click **New** to add an argument to the list. Enter a valid argument for the option, then click **OK**.

If you want to specify an option that is not available from the **Properties** dialog, use the **Command Line** category. Enter the command line options in the **Additional Options** text box just as you would enter them on the command line.

**Additional Options**

- `-E -march=pentium4`
Updating a Project

Updating an Existing Intel® C++ Compiler Project

You can update an existing Intel compiler project to use the current version of the compiler. When you update the compiler, Tools options and other settings are preserved.

To update an older project to use the current compiler, do the following:

1. Open the project.
2. Select Project>Update to the Latest Intel(R) C++ Compiler.
3. The Project Update dialog box appears:
Use the dialog box as follows:

- Select/Deselect the configurations to update using the checkbox at the beginning of each row. Use **Select All/Deselect All** to select/deselect all configurations.
By default, the compiler saves the original configuration after the update. To delete this saved configuration, unselect the Keep checkbox.

Enter the name of the new configuration in the New Intel Configuration field. The Overwrite field indicates whether a configuration with that same name already exists and whether the existing configuration will be overwritten by the update. If the named configuration does not exist, the field displays ❌. If the named configuration exists, the field displays a checkmark ✅. Unchecking this box will result in a message warning you of any conflict.

You can get brief tips by placing your cursor on the column headers.

You can click the help button to get the help text.

4. Click Finish. A confirmation box similar to the following indicates success:

![Project Update Summary]

The .cproject file is a hidden file used to keep project information. When you update a project, the original .cproject file is backed up and a log file is created in your Eclipse project directory.

**Updating a GCC* Project to Use the Intel® C++ Compiler**

You can update a GCC* Project to use the Intel® C++ Compiler while preserving GCC options and other settings in the original project. Once you successfully update the project, you can use menu items to easily switch back and forth between the Intel compiler and GCC.

To update an existing GCC project to use Intel C++ Compiler, start Eclipse and follow these steps:
1. Open the desired GCC project.
2. Select Project>Use Intel(R) C++ Compiler.

3. The Project Update dialog box appears:
Use the dialog box as follows:

- Select/Deselect the configurations to update using the checkbox at the beginning of each row. Use Select All/Deselect All to select/deselect all configurations.

<table>
<thead>
<tr>
<th>Use</th>
<th>GCC Configuration</th>
<th>Keep</th>
<th>New Intel Configuration</th>
<th>Overwrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>Debug (active)</td>
<td>✔</td>
<td>Debug_i</td>
<td></td>
</tr>
<tr>
<td>✔</td>
<td>Release</td>
<td>✔</td>
<td>Release_i</td>
<td></td>
</tr>
</tbody>
</table>

Use the dialog box as follows:

- Select/Deselect the configurations to update using the checkbox at the beginning of each row. Use Select All/Deselect All to select/deselect all configurations.
• By default, the compiler saves the original configuration after the update. To delete this saved configuration, unselect the Keep checkbox.

• Enter the name of the new configuration in the New Intel Configuration field. The Overwrite field indicates whether a configuration with that same name already exists and whether the existing configuration will be overwritten by the update. If the named configuration does not exist, the field displays ☐. If the named configuration exists, the field displays a checkmark ☑. Unchecking this box will result in a message warning you of any conflict.

• You can get brief tips by placing your cursor on the column headers.

• You can click the help button to get the help text.

• If you have multiple versions of Intel Compiler installed, click Next to select the compiler for use with the new configuration. By default, the most recent Intel compiler will be used.

When a project is successfully updated, the menu item Use GCC appears in the Project menu. Use this menu item, along with the Use Intel(R) C++ Compiler menu item, to switch back and forth between the compilers.

Switching Between Compilers

When you select either the Use GCC or Use Intel(R) C++ Compiler menu item, a dialog box similar to the following displays, asking you to select the active configuration. Choose the configuration and click OK.
Make Files

Project Types and Makefiles

When you create a new Intel C project in Eclipse*/CDT*, you can select **Executable**, **Shared Library**, or **Static Library** projects or **Makefile** projects.
Select Makefile Project if your project already includes a makefile. Use Executable, Shared Library, or Static Library Project to build a makefile using Intel compiler-specific options assigned from property pages.

Exporting Makefiles

If you created an Executable, Shared Library, or Static Library Project, you can use Eclipse to build a makefile that includes Intel compiler options. See Setting Properties. When your project is complete, you can export your makefile and project source files to another directory, then build your project from the command line using make.
Exporting makefiles

To export your makefile:

1. Select your project in the Eclipse Project Explorer view.
2. From the Eclipse File menu, select Export to launch the Export Wizard.
3. On the Select dialog of the Export Wizard, select File system, then click Next.

4. On the File system dialog, check both the helloworld and Release directories in the left-hand pane. Be sure all the project sources in the right-hand pane are also checked.

NOTE. You may deselect some files in the right-hand pane, such as the hello.o object file and helloworld executable. However, you must also select Create directory structure for files in the Options section to successfully create the export directory. This also applies to project files in the helloworld directory.
5. Use the **Browse** button to target the export to an existing directory. Eclipse can also create a new directory for full paths entered in the **To directory** text box. If, for example, you specified `/code/makefile` as the export directory, Eclipse creates two new sub-directories:

- `/code/makefile/helloworld`
- `/code/makefile/helloworld/Release`
6. Click **Finish** to complete the export.

**Running make**

In a terminal window, change to the `/cpp/hello_world/Release` directory, then run `make` by typing:
make clean all
You should see the following output:
rm -rf ./hello.o ./hello.d hello_world

Building file: ../hello.c
Invoking: C Compiler
icc -c -o hello.o ../hello.c
Finished building: ../hello.c

Building target: hello_world
Invoking: Linker
icc -o hello_world ./hello.o
Finished building target: hello_world
This generates the hello_world executable in the same directory.

Using Intel(R) Performance Libraries
The Intel® C++ Compiler now comes bundled with several Intel® Performance Libraries. You can access these libraries in Eclipse, using the following property pages, located in the Performance Library Build Components category:

• Use Intel(R) Threading Building Blocks Library
• Use Intel(R) Integrated Performance Primitives Libraries
• Use Intel(R) Math Kernel Library

The Use Intel Threading Building Blocks property page lets you link to the libraries and bring in the associated headers.

The Use Intel Integrated Performance Primitives property page provides the following choices in a drop-down menu:

• None, to disable use of Intel Integrated Performance Primitives
• Use main libraries set, to use all libraries except Crypto libraries.
• Use main libraries and cryptography library, to use Cryptography libraries in addition to the main libraries.
• **Use non-pic version of libraries**, to use the non-pic version of the main libraries

• **Use cryptography library and non-pic version of libraries**, to use Cryptography libraries in addition to the non-pic version of the main libraries

---

**NOTE.** The Cryptography libraries are subject to export laws.

---

The **Use Intel Math Kernel Library** property page provides the following choices in a drop-down menu:

- **None**, to disable use of Intel Math Kernel Library
- **Use threaded Intel(R) MKL library**, to link using threaded version of the library.
- **Use non-threaded Intel(R) MKL library**, to link using non-threaded version of the library.
- **Use Intel(R) MKL Cluster and sequential Intel(R) MKL libraries**, to link using Intel(R) MKL Cluster libraries and the sequential Intel(R) MKL libraries

For more information, see the Intel Threading Building Blocks, Intel Integrated Performance Primitives, and Intel Math Kernel Library documentation.
Invoking the Compiler from the Command Line

There are two necessary steps to invoke the Intel® C++ Compiler from the command line:

1. set the environment
2. invoke the compiler

Set the Environment Variables

Before you can operate the compiler, you must set the environment variables to specify locations for the various components. The Intel C++ Compiler installation includes shell scripts that you can "source" to set environment variables. With the default compiler installation, these scripts are:

<install-dir>/bin/iccvars.sh <arg>

or

<install-dir>/bin/iccvars.csh <arg>

The scripts take an argument specifying architecture:

• ia32: Compiler and libraries for IA-32 architectures only
• intel64: Compiler and libraries for Intel® 64 architectures only
• ia64: Compiler and libraries for IA-64 architectures only

To source an environment script, enter one of the following on the command line:

source <install-dir>/bin/iccvars.sh <arg>

or

source <install-dir>/bin/iccvars.csh <arg>

If you want the script to run automatically, add the same command to the end of your startup file.

Sample .bash_profile entry for iccvars.sh:

# set environment vars for Intel C++ compiler
source <install-dir>/bin/iccvars.sh ia32
With some Linux* distributions, if you source iccvars.sh from your .bash_profile, the location of LIBRARY_PATH may not be set as you would expect. It may be necessary to source iccvars.sh after starting your terminal session. This affects the Intel C++ compiler (icpc) only.

**Invoking the Compiler with icc or icpc**

You can invoke the Intel C++ Compiler on the command line with either icc or icpc.

- When you invoke the compiler with icc, the compiler builds C source files using C libraries and C include files. If you use icc with a C++ source file, it is compiled as a C++ file. Use icc to link C object files.
- When you invoke the compiler with icpc the compiler builds C++ source files using C++ libraries and C++ include files. If you use icpc with a C source file, it is compiled as a C++ file. Use icpc to link C++ object files.

**Command-line Syntax**

When you invoke the Intel C++ Compiler with icc or icpc, use the following syntax:

```
{icc|icpc} [options] file1 [file2 . . .]
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Indicates one or more command-line options. The compiler recognizes one or more letters preceded by a hyphen (-). This includes linker options.</td>
</tr>
<tr>
<td>file1, file2 . . .</td>
<td>Indicates one or more files to be processed by the compiler. You can specify more than one file. Use a space as a delimiter for multiple files.</td>
</tr>
</tbody>
</table>

**Invoking the Compiler from the Command Line with make**

To run make from the command line using Intel® C++ Compiler, make sure that /usr/bin is in your path.

To use the Intel compiler, your makefile must include the setting CC=icpc. Use the same setting on the command line to instruct the makefile to use the Intel compiler. If your makefile is written for gcc, the GNU* C compiler, you will need to change those command line options not recognized by the Intel compiler. Then you can compile:

```
make -f my_makefile
```
See Also
• Building Applications from the Command Line
• Modifying Your makefile

Passing Options to the Linker

This topic describes the options that let you control and customize the linking with tools and libraries and define the output of the ld linker. See the ld man page for more information on the linker.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Ldirectory</td>
<td>Instruct the linker to search directory for libraries.</td>
</tr>
<tr>
<td>-Q&lt;tool&gt;,&lt;list&gt;</td>
<td>Passes an argument list to another program in the compilation sequence, such as the assembler or linker.</td>
</tr>
<tr>
<td>-shared</td>
<td>Instructs the compiler to build a Dynamic Shared Object (DSO) instead of an executable.</td>
</tr>
<tr>
<td>-shared-libgcc</td>
<td>-shared-libgcc has the opposite effect of -static-libgcc. When it is used, the GNU standard libraries are linked in dynamically, allowing the user to override the static linking behavior when the -static option is used. Note: By default, all C++ standard and support libraries are linked dynamically.</td>
</tr>
<tr>
<td>-shared-intel</td>
<td>Specifies that all Intel-provided libraries should be linked dynamically.</td>
</tr>
<tr>
<td>-static</td>
<td>Causes the executable to link all libraries statically, as opposed to dynamically. When -static is not used:</td>
</tr>
<tr>
<td></td>
<td>• /lib/ld-linux.so.2 is linked in</td>
</tr>
<tr>
<td></td>
<td>• all other libs are linked dynamically</td>
</tr>
<tr>
<td></td>
<td>When -static is used:</td>
</tr>
<tr>
<td></td>
<td>• /lib/ld-linux.so.2 is not linked in</td>
</tr>
<tr>
<td></td>
<td>• all other libs are linked statically</td>
</tr>
</tbody>
</table>
This option causes the GNU standard libraries to be linked in statically.

- Bstatic

This option is placed in the linker command line corresponding to its location on the user command line. This option is used to control the linking behavior of any library being passed in via the command line.

- Bdynamic

This option is placed in the linker command line corresponding to its location on the user command line. This option is used to control the linking behavior of any library being passed in via the command line.

- static-intel

This option causes Intel-provided libraries to be linked in statically. It is the opposite of -shared-intel.

- Wl, optlist

This option passes a comma-separated list (optlist) of linker options to the linker.

- Xlinker val

This option passes a value (val), such as a linker option, an object, or a library, directly to the linker.

### Compiler Input Files

The Intel® C++ Compiler recognizes input files with the extensions listed in the following table:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Interpretation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.c</td>
<td>C source file</td>
<td>Passed to compiler</td>
</tr>
<tr>
<td>file.C</td>
<td>C++ source file</td>
<td>Passed to compiler</td>
</tr>
<tr>
<td>file.CC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cpp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.a</td>
<td>Library file</td>
<td>Passed to linker</td>
</tr>
<tr>
<td>file.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.i</td>
<td>Preprocessed file</td>
<td>Passed to stdout</td>
</tr>
</tbody>
</table>
### Output Files

The Intel® C++ Compiler produces output files with the extensions listed in the following table:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.i</td>
<td>Preprocessed file -- produced with the -P option.</td>
<td>Passed to linker</td>
</tr>
<tr>
<td>file.o</td>
<td>Object file -- produced with the -c option.</td>
<td>Passed to linker</td>
</tr>
<tr>
<td>file.s</td>
<td>Assembly language file -- produced with the -S option.</td>
<td>Passed to assembler</td>
</tr>
<tr>
<td>a.out</td>
<td>Executable file -- produced by default compilation.</td>
<td>Passed to assembler</td>
</tr>
</tbody>
</table>

### See Also

- Building Applications from the Command Line
- Using Options for Preprocessing
- Specifying Object Files
- Specifying Assembly Files
- Specifying Executable Files

### Specifying Compilation Output

#### Specifying Executable Files

You can use the -o option to specify the name of the executable file. In the following example, the compiler produces an executable file named `startup`.

```
icpc -o startup prog1.cpp
```

### See Also

- Specifying Compilation Output
Specifying Object Files
You can use the \texttt{-c} and \texttt{-o} options to specify an alternate name for an object file. In this example, the compiler generates an object file name \texttt{myobj.o}:
\begin{verbatim}
icpc -c -omyobj.o x.cpp
\end{verbatim}

See Also
- Specifying Compilation Output
- \texttt{-c}
- \texttt{-o}

Specifying Assembly Files
You can use the \texttt{-S} and \texttt{-o} options to specify an alternate name for an assembly file. In this example, the compiler generates an assembly file named \texttt{myasm.s}:
\begin{verbatim}
icpc -S -omyasm.s x.cpp
\end{verbatim}

See Also
- Specifying Compilation Output
- \texttt{-S}

Specifying Alternate Tools and Paths
Use the \texttt{-Qlocation} option to specify an alternate path for a tool. This option accepts two arguments using the following syntax:
\begin{verbatim}
-Qlocation,tool,path
\end{verbatim}
where \texttt{tool} designates which compilation tool is associated with the alternate \texttt{path}.

\begin{tabular}{|l|l|}
\hline
\textbf{tool} & \textbf{Description} \\
\hline
cpp & Specifies the compiler front-end preprocessor. \\
c & Specifies the C++ compiler. \\
asm & Specifies the assembler. \\
link & Specifies the linker. \\
\hline
\end{tabular}
Use the `-Qoption` option to pass an option specified by `optlist` to a `tool`, where `optlist` is a comma-separated list of options. The syntax for this command is:

```
-Qoption,tool,optlist
```

where `tool` designates which compilation tool receives the `optlist`.

<table>
<thead>
<tr>
<th><code>tool</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpp</td>
<td>Specifies the compiler front-end preprocessor.</td>
</tr>
<tr>
<td>c</td>
<td>Specifies the C++ compiler.</td>
</tr>
<tr>
<td>asm</td>
<td>Specifies the assembler.</td>
</tr>
<tr>
<td>link</td>
<td>Specifies the linker.</td>
</tr>
</tbody>
</table>

`optlist` indicates one or more valid argument strings for the designated program. If the argument is a command-line option, you must include the hyphen. If the argument contains a space or tab character, you must enclose the entire argument in quotation characters ("""). You must separate multiple arguments with commas.

**Using Precompiled Header Files**

The Intel® C++ Compiler supports precompiled header (PCH) files to significantly reduce compile times using the following options:

- `-pch`
- `-pch-dir dirname`
- `-pch-create filename`
- `-pch-use filename`

**CAUTION.** Depending on how you organize the header files listed in your sources, these options may increase compile times. See Organizing Source Files to learn how to optimize compile times using the PCH options.

**Using `-pch`**

The `-pch` option directs the compiler to use appropriate PCH files. If none are available, they are created as `sourcefile.pchi`. This option supports multiple source files, such as the ones shown in Example 1:
Example 1 command line:
`icpc -pch source1.cpp source2.cpp`

Example 1 output when .pchi files do not exist:
"source1.cpp": creating precompiled header file "source1.pchi"
"source2.cpp": creating precompiled header file "source2.pchi"

Example 1 output when .pchi files do exist:
"source1.cpp": using precompiled header file "source1.pchi"
"source2.cpp": using precompiled header file "source2.pchi"

**NOTE.** The `-pch` option will use PCH files created from other sources if the headers files are the same. For example, if you compile `source1.cpp` using `-pch`, then `source1.pchi` is created. If you then compile `source2.cpp` using `-pch`, the compiler will use `source1.pchi` if it detects the same headers.

**Using -pch-create**

Use the `-pch-create filename` option if you want the compiler to create a PCH file called `filename`. Note the following regarding this option:

- The `filename` parameter must be specified.
- The `filename` parameter can be a full path name.
- The full path to `filename` must exist.
- The `.pchi` extension is not automatically appended to `filename`.
- This option cannot be used in the same compilation as `-pch-use filename`.
- The `-pch-create filename` option is supported for single source file compilations only.

Example 2 command line:
`icpc -pch-create /pch/source32.pchi source.cpp`

Example 2 output:
"source.cpp": creating precompiled header file "/pch/source32.pchi"
**Using -pch-use filename**

This option directs the compiler to use the PCH file specified by `filename`. It cannot be used in the same compilation as `-pch-create filename`. The `-pch-use filename` option supports full path names and supports multiple source files when all source files use the same `.pchi` file.

Example 3 command line:

```sh
icpc -pch-use /pch/source32.pchi source.cpp
```

Example 3 output:

"source.cpp": using precompiled header file /pch/source32.pchi

**Using -pch-dir dirname**

Use the `-pch-dir dirname` option to specify the path (`dirname`) to the PCH file. You can use this option with `-pch`, `-pch-create filename`, and `-pch-use filename`.

Example 4 command line:

```sh
icpc -pch -pch-dir /pch/source32.cpp
```

Example 4 output:

"source32.cpp": creating precompiled header file /pch/source32.pchi

**Organizing Source Files**

If many of your source files include a common set of header files, place the common headers first, followed by the `#pragma hdrstop` directive. This pragma instructs the compiler to stop generating PCH files. For example, if `source1.cpp`, `source2.cpp`, and `source3.cpp` all include `common.h`, then place `#pragma hdrstop after common.h to optimize compile times.`

```c
#include "common.h"

#pragma hdrstop

#include "noncommon.h"
```

When you compile using the `-pch` option:

```sh
icpc -pch source1.cpp source2.cpp source3.cpp
```

the compiler will generate one PCH file for all three source files:

"source1.cpp": creating precompiled header file "source1.pchi"
If you don't use \texttt{\#pragma hdrstop}, a different PCH file is created for each source file if different headers follow \texttt{common.h}, and the subsequent compile times will be longer. \texttt{\#pragma hdrstop} has no effect on compilations that do not use these PCH options.

\textbf{See Also}
- Building Applications from the Command Line
- \texttt{-pch}
- \texttt{-pch-dir}
- \texttt{-pch-create}
- \texttt{-pch-use}

\textbf{Compiler Option Mapping Tool}

The Intel compiler’s Option Mapping Tool provides an easy method to derive equivalent options between Windows* and Linux*. If you are a Windows-based application developer who is developing an application for Linux OS, you may want to know, for example, the Linux OS equivalent for the \texttt{/Oy-} option. Likewise, the Option Mapping Tool provides Windows OS equivalents for Intel compiler options supported on Linux OS.

The Option Mapping Tool is not supported on Mac OS* X.

\textbf{Using the Compiler Option Mapping Tool}

You can start the Option Mapping Tool from the command line by:
- invoking the compiler and using the \texttt{-map-opts} option
- or, executing the tool directly

\textbf{NOTE.} Compiler options are mapped to their equivalent on the architecture you are using. It will not, for example, map an option that is specific to the IA-64 architecture to a like option available on the IA-32 architecture or Intel\textsuperscript{®} 64 architecture.

\textbf{Calling the Option Mapping Tool with the Compiler}

If you use the compiler to execute the Option Mapping Tool, the following syntax applies:
\texttt{<compiler command> <map-opts option> <compiler option(s)>}


**Example:** Finding the Windows OS equivalent for `-fp`

icpc -map-opts -fp

Intel(R) Compiler option mapping tool

Mapping Linux options to Windows OS for C++

'-map-opts' Linux option maps to

  --> '-Qmap-opts' option on Windows
  --> '-Qmap_opts' option on Windows

'-fp' Linux option maps to

  --> '-Oy-' option on Windows

Output from the Option Mapping Tool also includes:

- option mapping information (not shown here) for options included in the compiler configuration file
- alternate forms of the option that are supported but may not be documented

When you call the Option Mapping Tool with the compiler, your source file is not compiled.

**Calling the Option Mapping Tool Directly**

Use the following syntax to execute the Option Mapping Tool directly from a command line environment where the full path to the map_opts executable is known (compiler bin directory):

```
map_opts [-nologo] -t<target OS> -l<language> -opts <compiler option(s)>
```

where values for:

- `<target OS> = {l|linux|w|windows}`
- `<language> = {f|fortran|c}`
Example: Finding the Windows OS equivalent for -fp
map_opts -tw -lc -opts -fp
Intel(R) Compiler option mapping tool

mapping Linux options to Windows for C++

'-fp' Linux option maps to
  --> '-Oy-' option on Windows

Open Source Tools
This version of the Intel® C++ Compiler includes improved support for the following open source tools:

- GNU Libtool – a script that allows package developers to provide generic shared library support.
- Valgrind – a flexible system for debugging and profiling executables running on x86 processors.
- GNU Automake – a tool for automatically generating Makefile.ins from files called Makefile.am.

See Also
- Building Applications from the Command Line
  GNU Automake documentation – http://sources.redhat.com/automake/automake.html
Using Preprocessor Options

About Preprocessor Options

This section explains how preprocessor options are implemented in the Intel® C++ Compiler to perform preliminary operations on C and C++ source files. The preprocessor options are summarized in the following table:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-E</td>
<td>preprocess to stdout. #line directives included.</td>
</tr>
<tr>
<td>-P</td>
<td>preprocess to a file. #line directives omitted.</td>
</tr>
<tr>
<td>-EP</td>
<td>preprocess to stdout omitting #line directives.</td>
</tr>
<tr>
<td>-C</td>
<td>retain comments in intermediate file (use with -E or -P).</td>
</tr>
<tr>
<td>-D</td>
<td>define a macro.</td>
</tr>
<tr>
<td>-U</td>
<td>undefine a macro.</td>
</tr>
<tr>
<td>-I</td>
<td>add directory to include file search path.</td>
</tr>
<tr>
<td>-X</td>
<td>remove standard directories from include file search path.</td>
</tr>
<tr>
<td>-H</td>
<td>print include file order.</td>
</tr>
<tr>
<td>-M</td>
<td>generate makefile dependency information.</td>
</tr>
</tbody>
</table>

See Also
- Using Preprocessor Options
- -E
- -P
- -EP
- -C
- -D
Using Options for Preprocessing

Use these options to preprocess your source files without compiling them. When using these options, only the preprocessing phase of compilation is activated.

**Using -E**

Use this option to preprocess to stdout. For example, to preprocess two source files and write them to stdout, enter the following command:

```
icpc -E prog1.cpp prog2.cpp
```

**Using -P**

Use this option to preprocess to a .i file omitting #line directives. For example, the following command creates two files named `prog1.i` and `prog2.i`, which you can use as input to another compilation:

```
icpc -P prog1.cpp prog2.cpp
```

---

**CAUTION.** Existing files with the same name and extension are overwritten when you use this option.

**Using -EP**

Use this option to preprocess to stdout omitting #line directives.

```
icpc -EP prog1.cpp prog2.cpp
```

**Using -C**

Use this option to retain comments. In this example:

```
icpc -C -P prog1.cpp prog2.cpp
```
the compiler preserves comments in the prog1.i preprocessed file.

**Option Summary**

The following table summarizes the preprocessing options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Output Includes #line Directives</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>-E</td>
<td>Yes</td>
<td>stdout</td>
</tr>
<tr>
<td>-P</td>
<td>No</td>
<td>.i file</td>
</tr>
<tr>
<td>-EP</td>
<td>No</td>
<td>stdout</td>
</tr>
<tr>
<td>-P -EP</td>
<td>No</td>
<td>.i file</td>
</tr>
</tbody>
</table>

**See Also**
- Using Preprocessor Options
- -E
- -P
- -EP
- -C

**Using Options to Define Macros**

You can use compiler options to define or undefine predefined macros.

**Using -D**

Use this option to define a macro. For example, to define a macro called SIZE with the value 100 use the following command:

```
icpc -DSIZE=100 prog1.cpp
```

If you define a macro, but do not assign a value, the compiler defaults to 1 for the value of the macro.
Using -U

Use this option to undefine a macro. For example, this command:

```
icpc -Uia32 prog1.cpp
```

undefines the `ia32` predefined macro. If you attempt to undefine an ANSI C macro, the compiler
will emit an error:

```
invalid macro undefined: <name of macro>
```

See Also

- Using Preprocessor Options
- ANSI Standard Predefined Macros
- Additional Predefined Macros
Modifying the Compilation Environment

About Modifying the Compilation Environment

To run the Intel® C++ Compiler, you need to start with the proper environment. The compiler includes scripts which set the environment for all necessary components. You can modify the compilation environment by specifying different settings for:

- Environment Variables
- Configuration Files
- Include Files

See Also

- Modifying the Compilation Environment
- Response Files

Setting Environment Variables

You can customize your system environment by specifying paths where the compiler searches for special files such as libraries, include files, and configuration files. The Intel® C++ Compiler supports the environment variables listed in the following table:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GXX_INCLUDE</td>
<td>Specifies the location of the gcc headers. Set this variable only when the compiler cannot locate the gcc headers when using the -gcc-name option.</td>
</tr>
<tr>
<td>GXX_ROOT</td>
<td>Specifies the location of the gcc binaries. Set this variable only when the compiler cannot locate the gcc binaries when using the -gcc-name option.</td>
</tr>
<tr>
<td>IA32ROOT (IA-32 architecture and Intel® 64 architecture)</td>
<td>Points to the directories containing the include and library files for a non-standard installation structure.</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IA64ROOT (IA-64 architecture based systems)</td>
<td>Points to the directories containing the include and library files for a non-standard installation structure.</td>
</tr>
<tr>
<td>ICCCFG</td>
<td>Specifies the configuration file for customizing compilations when invoking the compiler using <code>icc</code>.</td>
</tr>
<tr>
<td>ICPCCFG</td>
<td>Specifies the configuration file for customizing compilations when invoking the compiler using <code>icpc</code>.</td>
</tr>
<tr>
<td>INTEL_LICENSE_FILE</td>
<td>Specifies the location for the Intel license file.</td>
</tr>
<tr>
<td>KMP_ALL_THREADS</td>
<td>Limits the number of simultaneously executing threads in an OpenMP* program. If this limit is reached and another native operating system thread encounters OpenMP* API calls or constructs, then the program may abort with an error message. If this limit is reached at the time an OpenMP parallel region begins, a one-time warning message may be generated indicating that the number of threads in the team was reduced, but the program will continue execution. This environment variable is only used for programs compiled with <code>-openmp</code> or <code>-openmp-profile</code>.</td>
</tr>
<tr>
<td>KMP_LIBRARY</td>
<td>Selects the OpenMP run-time library execution mode. The values for this variable are <code>serial</code>, <code>turnaround</code>, or <code>throughput</code> (default).</td>
</tr>
<tr>
<td>KMP_STACKSIZE</td>
<td>Sets the number of bytes to allocate for each OpenMP* thread to use as its private stack. Use the optional suffix b, k, m, g, or t, to specify bytes, kilobytes, megabytes, gigabytes, or terabytes. Note that this variable does not have any effect on native operating system threads created by the user program or the thread executing the sequential part of an OpenMP* program. Default: IA-32 architecture: 2m, Intel® 64 architecture: 4m, IA-64 architecture: 4m.</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KMP_VERSION</td>
<td>Enables (1) or disables (0) the printing of OpenMP run-time library version information during program execution. Default: disabled.</td>
</tr>
<tr>
<td>LD_LIBRARY_PATH</td>
<td>Specifies the location for shared objects.</td>
</tr>
<tr>
<td>OMP_DYNAMIC</td>
<td>Enables (1) or disables (0) the dynamic adjustment of the number of threads. Default is 0 (disabled).</td>
</tr>
<tr>
<td>OMP_NESTED</td>
<td>Enables (1) or disables (0) nested parallelism. Default is 0 (nested parallelism disabled).</td>
</tr>
<tr>
<td>OMP_NUM_THREADS</td>
<td>Sets the maximum number of threads to use for OpenMP* parallel regions if no other value is specified in the program itself. Default is the number of processors currently visible to the operating system on which the program is executed.</td>
</tr>
<tr>
<td>PATH</td>
<td>Specifies the directories the system searches for binary executable files.</td>
</tr>
<tr>
<td>PROF_DIR</td>
<td>Specifies the directory where profiling files (files with extensions .dyn, .dpi, .spi and so forth) are stored. The default is to store the .dyn files in the source directory of the file containing the first executed instrumented routine in the binary compiled with -prof-gen . Name for the .dpi file. The default is pgopti.dpi.</td>
</tr>
<tr>
<td>PROF_DPI</td>
<td></td>
</tr>
<tr>
<td>TMP</td>
<td>Specifies the location for temporary files. If none of these are specified, the compiler stores temporary files in /tmp.</td>
</tr>
<tr>
<td>TMPDIR</td>
<td></td>
</tr>
<tr>
<td>TEMP</td>
<td></td>
</tr>
</tbody>
</table>

**GNU* Environment Variables**

The Intel C++ Compiler also supports the GNU environment variables listed in the following table:
### Environment Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPATH</td>
<td>Path to include directory for C/C++ compilations.</td>
</tr>
<tr>
<td>C_INCLUDE_PATH</td>
<td>Path include directory for C compilations.</td>
</tr>
<tr>
<td>CPLUS_INCLUDE_PATH</td>
<td>Path include directory for C++ compilations.</td>
</tr>
<tr>
<td>DEPENDENCIES_OUTPUT</td>
<td>If this variable is set, its value specifies how to output dependencies for make based on the non-system header files processed by the compiler. System header files are ignored in the dependency output.</td>
</tr>
<tr>
<td>GCC_EXEC_PREFIX</td>
<td>This variable specifies alternative names for the linker (ld) and assembler (as).</td>
</tr>
<tr>
<td>LIBRARY_PATH</td>
<td>The value of LIBRARY_PATH is a colon-separated list of directories, much like PATH.</td>
</tr>
<tr>
<td>SUNPRO_DEPENDENCIES</td>
<td>This variable is the same as DEPENDENCIES_OUTPUT, except that system header files are not ignored.</td>
</tr>
</tbody>
</table>

### Using Configuration Files

You can decrease the time you spend entering command-line options by using the configuration file to automate command-line entries. Add any valid command-line option to the configuration file. The compiler processes options in the configuration file in the order they appear followed by the command-line options that you specify when you invoke the compiler. Options in the configuration file are executed every time you run the compiler. If you have varying option requirements for different projects, use response files.
How to Use Configuration Files

The following example illustrates a basic configuration file. The text following the "#" character is recognized as a comment. The configuration file, icc.cfg and icpc.cfg, is located in the same directory as the compiler's executable file. You should modify the configuration file environment variable if you need to specify a different location for the configuration file.

```bash
# Sample configuration file.
-I/my_headers
```

In this example, the compiler reads the configuration file and invokes the 
-I option every time you run the compiler, along with any options you specify on the command line.

See Also

- Modifying the Compilation Environment
- Environment Variables
- Using Response Files

Specifying Include Files

The Intel® C++ Compiler searches the default system areas for include files and whatever is specified by the 
-I compiler option. The compiler searches directories for include files in the following order:

1. Directories specified by the 
-I option
2. Directories specified in the environment variables
3. Default include directory

Use the 
-X option to remove default directories from the include file search path.

For example, to direct the compiler to search the path /alt/include instead of the default path, do the following:

```bash
icpc -X -I/alt/include prog1.cpp
```

See Also

- Modifying the Compilation Environment
- 
-I compiler option
- 
-X compiler option
- Environment Variables
Using Response Files

Use response files to specify options used during particular compilations. Response files are invoked as an option on the command line. Options in a response file are inserted in the command line at the point where the response file is invoked.

Sample Response Files

# response file: response1.txt
# compile with these options
-w0
# end of response1 file
# response file: response2.txt
# compile with these options
-00
# end of response2 file

Use response files to decrease the time spent entering command-line options and to ensure consistency by automating command-line entries. Use individual response files to maintain options for specific projects.

Any number of options or file names can be placed on a line in a response file. Several response files can be referenced in the same command line.

The following example shows how to specify a response file on the command line:

icpc @response1.txt prog1.cpp @response2.txt prog2.cpp

NOTE. An "@" symbol must precede the name of the response file on the command line.

See Also

- Modifying the Compilation Environment
- Using Configuration Files
Debugging

Using the Debugger

See the Intel® Debugger (IDB) documentation for complete information on using the Intel Debugger. You can also use the GNU Debugger (gdb) to debug programs compiled with the Intel® C++ Compiler.

Preparing for Debugging

Use this option to direct the compiler to generate code to support symbolic debugging. For example:

```
icpc -g prog1.cpp
```

The compiler does not support the generation of debugging information in assembly files. When you specify this option, the resulting object file will contain debugging information, but the assembly file will not.

This option changes the default optimization from `-O2` to `-O0`.

See Also

- Debugging
- `-g`

Symbolic Debugging and Optimizations

When you use debugging options to generate code with debug symbols, the compiler disables `O`n optimizations. If you specify an `O`n option with debugging options, some of the generated debug information may be inaccurate as a side-effect of optimization.

The table below summarizes the effects of using the debugging options with the optimization options.

<table>
<thead>
<tr>
<th>Option(s)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-g</code></td>
<td>Debugging information produced, <code>-O0</code> and <code>-fp</code> enabled.</td>
</tr>
<tr>
<td><code>-g -O1</code></td>
<td>Debugging information produced, <code>-O1</code> optimizations enabled.</td>
</tr>
<tr>
<td><code>-g -O2</code></td>
<td>Debugging information produced, <code>-O2</code> optimizations enabled.</td>
</tr>
</tbody>
</table>
Result

Option(s)  Result
-\( g \) -O3 -fp  Debugging information produced, -O3 optimizations and -fp are enabled.

See Also

- Debugging
- -g
- -fp
- -O1, -O2, -O3

Using Options for Debug Information

The Intel® C++ Compiler provides basic debugging information and new features for enhanced debugging of code. The basic debugging options are listed in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-debug all</td>
<td>These options are equivalent to -g. They turn on production of basic debug information. They are off by default.</td>
</tr>
<tr>
<td>-debug full</td>
<td>These options are equivalent to -g. They turn on production of basic debug information. They are off by default.</td>
</tr>
<tr>
<td>-debug none</td>
<td>This option turns off production of debug information. This option is on by default.</td>
</tr>
</tbody>
</table>

The Intel C++ Compiler improves debuggability of optimized code through enhanced support for:

- tracebacks
- variable locations
- breakpoints and stepping

The options described in the following table control generation of enhanced debug information.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-debug expr-source-pos</td>
<td>This option enables expanded line number information.</td>
</tr>
<tr>
<td>-debug inline-debug-info</td>
<td>This option produces enhanced debug information for inlined code. It provides more information to debuggers for function call traceback.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-debug emit_column</code></td>
<td>This option generates column number information for debugging.</td>
</tr>
<tr>
<td><code>-debug semantic-stepping</code></td>
<td>This option generates information useful for breakpoints and stepping. It tells the debugger to stop only at machine instructions that achieve the final effect of a source statement.</td>
</tr>
<tr>
<td><code>-debug variable-locations</code></td>
<td>This option produces additional debug information for scalar local variables using a feature of the DWARF object module format known as &quot;location lists.&quot; The runtime locations of local scalar variables are specified more accurately using this feature, i.e. whether at a given position in the code, a variable value is found in memory or a machine register.</td>
</tr>
<tr>
<td><code>-debug extended</code></td>
<td>This option turns on the following <code>-debug options:</code></td>
</tr>
<tr>
<td></td>
<td>• <code>-debug semantic-stepping</code></td>
</tr>
<tr>
<td></td>
<td>• <code>-debug variable-locations</code></td>
</tr>
<tr>
<td></td>
<td>It also specifies that column numbers should appear in the line information.</td>
</tr>
<tr>
<td><code>-debug parallel</code> (Linux OS IA-32 and Intel® 64 architectures only)</td>
<td>This option determines whether the compiler generates parallel debug code instrumentations useful for thread data sharing and reentrant call detection. Must be used in conjunction with the <code>-g</code> option</td>
</tr>
</tbody>
</table>

**NOTE.** When the compiler needs to choose between optimization and quality of debug information, optimization is given priority.
Creating and Using Libraries

Overview: Using Libraries

The Intel® C++ Compiler uses the GNU* C Library and the Standard C++ Library. These libraries are documented at the following URLs:


Default Libraries

The following libraries are supplied with the Intel® C++ Compiler:

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libguide.a</td>
<td>For OpenMP* implementation</td>
</tr>
<tr>
<td>libguide.so</td>
<td></td>
</tr>
<tr>
<td>libguide_stats.a</td>
<td>OpenMP static library for the parallelizer tool with performance statistics</td>
</tr>
<tr>
<td>libguide_stats.so</td>
<td>and profile information</td>
</tr>
<tr>
<td>libompstub.a</td>
<td>Library that resolves references to OpenMP subroutines when OpenMP is</td>
</tr>
<tr>
<td></td>
<td>not in use</td>
</tr>
<tr>
<td>libsvml.a</td>
<td>Short vector math library</td>
</tr>
<tr>
<td>libsvml.so</td>
<td></td>
</tr>
<tr>
<td>libirc.a</td>
<td>Intel support library for PGO and CPU dispatch</td>
</tr>
<tr>
<td>libirc_s.a</td>
<td></td>
</tr>
<tr>
<td>libintlc.so.5</td>
<td></td>
</tr>
<tr>
<td>libimf.a</td>
<td>Intel math library</td>
</tr>
<tr>
<td>libimf.so</td>
<td></td>
</tr>
<tr>
<td>libcxxguard.a</td>
<td>Used for interoperability support with the -cxxxlib option.</td>
</tr>
<tr>
<td>libcxxguard.so</td>
<td>See gcc Interoperability.</td>
</tr>
<tr>
<td>libcxxguard.so.5</td>
<td></td>
</tr>
</tbody>
</table>
When you invoke the `-cxxlib` option, libcprts is replaced with libstdc++ from the gcc* distribution (3.2 or newer)

**CAUTION.** The Linux* OS system libraries and the compiler libraries are not built with the `-align` option. Therefore, if you compile with the `-align` option and make a call to a compiler distributed or system library, and have long long, double, or long double types in your interface, you will get the wrong answer due to the difference in alignment. Any code built with `-align` cannot make calls to libraries that use these types in their interfaces unless they are built with `-align` (in which case they will not work without `-align`).

**Math Libraries**

The Intel math library, libimf.a, contains optimized versions of math functions found in the standard C run-time library. The functions in libimf.a are optimized for program execution speed on Intel processors. The Intel math library is linked by default.

**See Also**

- Creating and Using Libraries
- Managing Libraries and Intel Math Library

**Managing Libraries**

During compilation, the compiler reads the LIBRARY_PATH environment variable for static libraries it needs to link when building the executable. At runtime, the executable will link against dynamic libraries referenced in the LD_LIBRARY_PATH environment variable.

**Modifying LIBRARY_PATH**

If you want to add a directory, `/libs` for example, to the LIBRARY_PATH, you can do either of the following:

- command line: prompt> export LIBRARY_PATH=/libs:$LIBRARY_PATH
- startup file: export LIBRARY_PATH=/libs:$LIBRARY_PATH

To compile `file.cpp` and link it with the library `mylib.a`, enter the following command:

```
icpc file.cpp mylib.a```

The compiler passes file names to the linker in the following order:
Creating Libraries

Libraries are simply an indexed collection of object files that are included as needed in a linked program. Combining object files into a library makes it easy to distribute your code without disclosing the source. It also reduces the number of command-line entries needed to compile your project.

Static Libraries

Executables generated using static libraries are no different than executables generated from individual source or object files. Static libraries are not required at runtime, so you do not need to include them when you distribute your executable. At compile time, linking to a static library is generally faster than linking to individual source files.

To build a static library on Linux OS:

1. use the `-c` option to generate object files from the source files:
   ```
icpc -c my_source1.cpp my_source2.cpp my_source3.cpp
   ```

2. use the GNU tool `ar` to create the library file from the object files:
   ```
ar rc my_lib.a my_source1.o my_source2.o my_source3.o
   ```

3. compile and link your project with your new library:
   ```
icpc main.cpp my_lib.a
   ```

If your library file and source files are in different directories, use the `-Ldir` option to indicate where your library is located:
```
icpc -L/cpp/libs main.cpp my_lib.a
```

To build a static library on Mac OS X:

1. use the following command line to generate object files and create the library file:
   ```
icpc -fpic -o mylib.a -staticlib my_source1.cpp my_source2.cpp
   ```

2. compile and link your project with your new library:
   ```
icpc main.cpp my_lib.a
   ```
If your library file and source files are in different directories, use the `-L` option to indicate where your library is located:

```
icpc -L/cpp/libs main.cpp my_lib.a
```

If you are using Interprocedural Optimization, see Creating a Library from IPO Objects using `xiar`.

**Shared Libraries**

Shared libraries, also referred to as dynamic libraries or Dynamic Shared Objects (DSO), are linked differently than static libraries. At compile time, the linker insures that all the necessary symbols are either linked into the executable, or can be linked at runtime from the shared library. Executables compiled from shared libraries are smaller, but the shared libraries must be included with the executable to function correctly. When multiple programs use the same shared library, only one copy of the library is required in memory.

To build a shared library on Linux* OS:

1. use the `-fPIC` and `-c` options to generate object files from the source files:
   ```
icpc -fPIC -c my_source1.cpp my_source2.cpp my_source3.cpp
   ```
2. use the `-shared` option to create the library file from the object files:
   ```
icpc -shared -o my_lib.so my_source1.o my_source2.o my_source3.o
   ```
3. compile and link your project with your new library:
   ```
icpc main.cpp my_lib.so
   ```

To build a shared library on Mac OS* X:

1. use the following command line to generate object files and create the library file:
   ```
icpc -fPIC -o my_lib.so -dynamiclib my_source1.cpp my_source2.cpp my_source3.cpp
   ```
2. compile and link your project with your new library:
   ```
icpc main.cpp my_lib.dylib
   ```

**See Also**

- Creating and Using Libraries
- Using Intel Shared Libraries
- Compiling for Non-shared Libraries
Using Intel Shared Libraries

By default, the Intel® C++ Compiler links Intel-provided C++ libraries dynamically. The GNU*, Linux* OS, and Mac OS* X system libraries are also linked dynamically.

Options for Shared Libraries (Linux* OS)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-shared-intel</td>
<td>Use the -shared-intel option to link Intel-provided C++ libraries dynamically (default). This has the advantage of reducing the size of the application binary, but it also requires the libraries to be on the systems where the application runs.</td>
</tr>
<tr>
<td>-shared</td>
<td>The -shared option instructs the compiler to build a Dynamic Shared Object (DSO) instead of an executable. For more details, refer to the ld man page documentation.</td>
</tr>
<tr>
<td>-fpic</td>
<td>Use the -fpic option when building shared libraries. It is required for the compilation of each object file included in the shared library.</td>
</tr>
</tbody>
</table>

Options for Shared Libraries (Mac OS* X)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dynamiclib</td>
<td>Use the -dynamiclib option to invoke the libtool command to generate dynamic libraries.</td>
</tr>
<tr>
<td>-fpic</td>
<td>Use the -fpic option when building shared libraries. It is required for the compilation of each object file included in the shared library.</td>
</tr>
</tbody>
</table>

Compiling for Non-shared Libraries

Overview: Compiling for Non-shared Libraries

This section includes information on:
Global Symbols and Visibility Attributes

A global symbol is one that is visible outside the compilation unit (single source file and its include files) in which it is declared. In C/C++, this means anything declared at file level without the `static` keyword. For example:

```c
int x = 5;       // global data definition
extern int y;   // global data reference
int five()       // global function definition
{ return 5; }
extern int four(); // global function reference
```

A complete program consists of a main program file and possibly one or more shareable object (.so) files that contain the definitions for data or functions referenced by the main program. Similarly, shareable objects might reference data or functions defined in other shareable objects. Shareable objects are so called because if more than one simultaneously executing process has the shareable object mapped into its virtual memory, there is only one copy of the read-only portion of the object resident in physical memory. The main program file and any shareable objects that it references are collectively called the components of the program.

Each global symbol definition or reference in a compilation unit has a visibility attribute that controls how (or if) it may be referenced from outside the component in which it is defined. There are five possible values for visibility:

- **EXTERNAL** – The compiler must treat the symbol as though it is defined in another component. For a definition, this means that the compiler must assume that the symbol will be overridden (preempted) by a definition of the same name in another component. See Symbol Preemption. If a function symbol has external visibility, the compiler knows that it must be called indirectly and can inline the indirect call stub.
- **DEFAULT** – Other components can reference the symbol. Furthermore, the symbol definition may be overridden (preempted) by a definition of the same name in another component.
- **PROTECTED** – Other components can reference the symbol, but it cannot be preempted by a definition of the same name in another component.
• **HIDDEN** – Other components cannot directly reference the symbol. However, its address might be passed to other components indirectly (for example, as an argument to a call to a function in another component, or by having its address stored in a data item reference by a function in another component).

• **INTERNAL** – The symbol cannot be referenced outside its defining component, either directly or indirectly.

Static local symbols (in C/C++, declared at file scope or elsewhere with the keyword static) usually have HIDDEN visibility—they cannot be referenced directly by other components (or, for that matter, other compilation units within the same component), but they might be referenced indirectly.

**NOTE.** Visibility applies to references as well as definitions. A symbol reference's visibility attribute is an assertion that the corresponding definition will have that visibility.

---

**Symbol Preemption**

Sometimes you may need to use some of the functions or data items from a shareable object, but may wish to replace others with your own definitions. For example, you may want to use the standard C runtime library shareable object, `libc.so`, but to use your own definitions of the heap management routines `malloc()` and `free()`. In this case it is important that calls to `malloc()` and `free()` within `libc.so` call your definition of the routines and not the definitions present in `libc.so`. Your definition should override, or preempt, the definition within the shareable object.

This feature of shareable objects is called symbol preemption. When the runtime loader loads a component, all symbols within the component that have default visibility are subject to preemption by symbols of the same name in components that are already loaded. Since the main program image is always loaded first, none of the symbols it defines will be preempted.

The possibility of symbol preemption inhibits many valuable compiler optimizations because symbols with default visibility are not bound to a memory address until runtime. For example, calls to a routine with default visibility cannot be inlined because the routine might be preempted if the compilation unit is linked into a shareable object. A preemptable data symbol cannot be accessed using GP-relative addressing because the name may be bound to a symbol in a different component; the GP-relative address is not known at compile time.

Symbol preemption is a very rarely used feature that has drastic negative consequences for compiler optimization. For this reason, by default the compiler treats all global symbol definitions as non-preemptable (i.e., protected visibility). Global references to symbols defined in other compilation units are assumed by default to be preemptable (i.e., default visibility). In those rare cases when you need all global definitions, as well as references, to be preemptable, specify the `-fpic` option to override this default.
Specifying Symbol Visibility Explicitly

You can explicitly set the visibility of an individual symbol using the `visibility` attribute on a data or function declaration. For example:

```c
int i __attribute__((visibility("default")));
void __attribute__((visibility("hidden"))) x () {...}
extern void y() __attribute__((visibility("protected"));
```

The `visibility` declaration attribute accepts one of the five keywords:

- `external`
- `default`
- `protected`
- `hidden`
- `internal`

The value of the `visibility` declaration attribute overrides the default set by the `-fvisibility`, `-fpic`, or `-fno-common` attributes.

If you have a number of symbols for which you wish to specify the same `visibility` attribute, you can set the visibility using one of the five command line options:

- `-fvisibility-external=file`
- `-fvisibility-default=file`
- `-fvisibility-protected=file`
- `-fvisibility-hidden=file`
- `-fvisibility-internal=file`

where `file` is the pathname of a file containing a list of the symbol names whose visibility you wish to set. The symbol names in the file are separated by white space (blanks, TAB characters, or newlines). For example, the command line option:

```bash
-fvisibility-protected=prot.txt
```

where file `prot.txt` contains:

```
a
  b c d
 e
```
sets protected visibility for symbols a, b, c, d, and e. This has the same effect as
\_attribute\_ ((visibility="protected"))
on the declaration for each of the symbols. Note that these two ways to explicitly set visibility
are mutually exclusive – you may use \_attribute\((visibility())\) on the declaration, or
specify the symbol name in a file, but not both.
You can set the default visibility for symbols using one of the command line options:
- \_fvisibility=external
- \_fvisibility=default
- \_fvisibility=protected
- \_fvisibility=hidden
- \_fvisibility=internal
This option sets the visibility for symbols not specified in a visibility list file and that do not have
\_attribute\((visibility())\) in their declaration. For example, the command line options:
\_fvisibility=protected -fvisibility-default=prot.txt
where file prot.txt is as previously described, will cause all global symbols except a, b, c, d,
and e to have protected visibility. Those five symbols, however, will have default visibility and
thus be preemptable.

Other Visibility-related Command-line Options

\_fminshared
The \_fminshared option specifies that the compilation unit will be part of a main program
component and will not be linked as part of a shareable object. Since symbols defined in the
main program cannot be preempted, this allows the compiler to treat symbols declared with
default visibility as though they have protected visibility (i.e., \_fminshared implies \_fvisibility=protected). Also, the compiler need not generate position-independent code for the main
program. It can use absolute addressing, which may reduce the size of the global offset table
(GOT) and may reduce memory traffic.

\_fpic
The \_fpic option specifies full symbol preemption. Global symbol definitions as well as global
symbol references get default (i.e., preemptable) visibility unless explicitly specified otherwise.
-fno-common

Normally a C/C++ file-scope declaration with no initializer and without the extern or static keyword

    int i;

is represented as a common symbol. Such a symbol is treated as an external reference, except that if no other compilation unit has a global definition for the name, the linker allocates memory for it. The -fno-common option causes the compiler to treat what otherwise would be common symbols as global definitions and to allocate memory for the symbol at compile time. This may permit the compiler to use the more efficient GP-relative addressing mode when accessing the symbol.

See Also
• Compiling for Non-shared Libraries
• -fminshared compiler
• -fpic
• -fno-common
**gcc Compatibility**

C language object files created with the Intel® C++ Compiler are binary compatible with the GNU gcc* compiler and glibc*, the GNU C language library. You can use the Intel compiler or the gcc compiler to pass object files to the linker. However, to correctly pass the Intel libraries to the linker, use the Intel compiler.

The Intel C++ Compiler supports many of the language extensions provided by the GNU compilers.

**gcc Extensions to the C Language**

GNU C includes several, non-standard features not found in ISO standard C. This version of the Intel C++ Compiler supports most of these extensions listed in the following table. See http://www.gnu.org for more information.

<table>
<thead>
<tr>
<th>gcc Language Extension</th>
<th>Intel Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statements and Declarations in Expressions</td>
<td>Yes</td>
</tr>
<tr>
<td>Locally Declared Labels</td>
<td>Yes</td>
</tr>
<tr>
<td>Labels as Values</td>
<td>Yes</td>
</tr>
<tr>
<td>Nested Functions</td>
<td>No</td>
</tr>
<tr>
<td>Constructing Function Calls</td>
<td>No</td>
</tr>
<tr>
<td>Naming an Expression's Type</td>
<td>Yes</td>
</tr>
<tr>
<td>Referring to a Type with typeof</td>
<td>Yes</td>
</tr>
<tr>
<td>Generalized Lvalues</td>
<td>Yes</td>
</tr>
<tr>
<td>Conditionals with Omitted Operands</td>
<td>Yes</td>
</tr>
<tr>
<td>Double-Word Integers</td>
<td>Yes</td>
</tr>
<tr>
<td>gcc Language Extension</td>
<td>Intel Support</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Complex Numbers</td>
<td>Yes</td>
</tr>
<tr>
<td>Hex Floats</td>
<td>Yes</td>
</tr>
<tr>
<td>Arrays of Length Zero</td>
<td>Yes</td>
</tr>
<tr>
<td>Arrays of Variable Length</td>
<td>Yes</td>
</tr>
<tr>
<td>Macros with a Variable Number of Arguments.</td>
<td>Yes</td>
</tr>
<tr>
<td>Slightly Looser Rules for Escaped Newlines</td>
<td>No</td>
</tr>
<tr>
<td>String Literals with Embedded Newlines</td>
<td>Yes</td>
</tr>
<tr>
<td>Non-Lvalue Arrays May Have Subscripts</td>
<td>Yes</td>
</tr>
<tr>
<td>Arithmetic on void-Pointers</td>
<td>Yes</td>
</tr>
<tr>
<td>Arithmetic on Function-Pointers</td>
<td>Yes</td>
</tr>
<tr>
<td>Non-Constant Initializers</td>
<td>Yes</td>
</tr>
<tr>
<td>Compound Literals</td>
<td>Yes</td>
</tr>
<tr>
<td>Designated Initializers</td>
<td>Yes</td>
</tr>
<tr>
<td>Cast to a Union Type</td>
<td>Yes</td>
</tr>
<tr>
<td>Case Ranges</td>
<td>Yes</td>
</tr>
<tr>
<td>Mixed Declarations and Code</td>
<td>Yes</td>
</tr>
<tr>
<td>Declaring Attributes of Functions</td>
<td>Yes</td>
</tr>
<tr>
<td>Attribute Syntax</td>
<td>Yes</td>
</tr>
<tr>
<td>Prototypes and Old-Style Function Definitions</td>
<td>No</td>
</tr>
<tr>
<td>C++ Style Comments</td>
<td>Yes</td>
</tr>
<tr>
<td>gcc Language Extension</td>
<td>Intel Support</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Dollar Signs in Identifier Names</td>
<td>Yes</td>
</tr>
<tr>
<td>ESC Character in Constants</td>
<td>Yes</td>
</tr>
<tr>
<td>Specifying Attributes of Variables</td>
<td>Yes</td>
</tr>
<tr>
<td>Specifying Attributes of Types</td>
<td>Yes</td>
</tr>
<tr>
<td>Inquiring on Alignment of Types or Variables</td>
<td>Yes</td>
</tr>
<tr>
<td>Inline Function is As Fast As a Macro</td>
<td>Yes</td>
</tr>
<tr>
<td>Assembler Instructions with C Expression Operands</td>
<td>Yes</td>
</tr>
<tr>
<td>Controlling Names Used in Assembler Code</td>
<td>Yes</td>
</tr>
<tr>
<td>Variables in Specified Registers</td>
<td>Yes</td>
</tr>
<tr>
<td>Alternate Keywords</td>
<td>Yes</td>
</tr>
<tr>
<td>Incomplete enum Types</td>
<td>Yes</td>
</tr>
<tr>
<td>Function Names as Strings</td>
<td>Yes</td>
</tr>
<tr>
<td>Getting the Return or Frame Address of a Function</td>
<td>Yes</td>
</tr>
<tr>
<td>Using Vector Instructions Through Built-in Functions</td>
<td>No</td>
</tr>
<tr>
<td>Other built-in functions provided by GCC</td>
<td>Yes</td>
</tr>
<tr>
<td>Built-in Functions Specific to Particular Target Machines</td>
<td>No</td>
</tr>
<tr>
<td>Pragmas Accepted by GCC</td>
<td>Yes</td>
</tr>
<tr>
<td>Unnamed struct/union fields within structs/unions</td>
<td>Yes</td>
</tr>
<tr>
<td>Decimal floating types</td>
<td>Yes</td>
</tr>
</tbody>
</table>
GNU C++ includes several, non-standard features not found in ISO standard C++. This version of the Intel C++ Compiler supports many of these extensions listed in the following table. See http://www.gnu.org for more information.

<table>
<thead>
<tr>
<th>g++ Language Extension</th>
<th>Intel Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum and Maximum operators in C++</td>
<td>Yes</td>
</tr>
<tr>
<td>When is a Volatile Object Accessed?</td>
<td>No</td>
</tr>
<tr>
<td>Restricting Pointer Aliasing</td>
<td>Yes</td>
</tr>
<tr>
<td>Vague Linkage</td>
<td>Yes</td>
</tr>
<tr>
<td>Declarations and Definitions in One Header</td>
<td>No</td>
</tr>
<tr>
<td>Where's the Template?</td>
<td>extern template supported</td>
</tr>
<tr>
<td>Extracting the function pointer from a bound pointer to member function</td>
<td>Yes</td>
</tr>
<tr>
<td>C++-Specific Variable, Function, and Type Attributes</td>
<td>Yes</td>
</tr>
<tr>
<td>Java Exceptions</td>
<td>No</td>
</tr>
<tr>
<td>Deprecated Features</td>
<td>No</td>
</tr>
<tr>
<td>Backwards Compatibility</td>
<td>No</td>
</tr>
</tbody>
</table>
NOTE. Statement expressions are supported, except the following are prohibited inside them:

- dynamically-initialized local static variables
- local non-POD class definitions
- try/catch

Also, branching out of a statement expression is not allowed, and statement expressions may not be used in default argument expressions. Variable-length arrays are no longer allowed in statement expressions.

NOTE. The Intel C++ Compiler supports gcc-style inline ASM if the assembler code uses AT&T* System V/386 syntax.

**gcc** Interoperability

**gcc Interoperability**

C++ compilers are interoperable if they can link object files and libraries generated by one compiler with object files and libraries generated by the second compiler, and the resulting executable runs successfully. The Intel® C++ Compiler is highly compatible with the gcc and g++ compilers. This section describes features of the Intel C++ Compiler that provide interoperability with gcc and g++.

**See Also**

- gcc* Interoperability
- gcc Compatibility
- Compiler Options for Interoperability
- Predefined Macros for Interoperability

**Compiler Options for Interoperability**

The Intel® C++ Compiler options that affect gcc* interoperability include:

- `-gcc-name=dir`
- `-gcc-version=nnn`
- `-gxx-name=dir`
• -cxxlib
• -fabi-version=n
• -no-gcc

-gcc-name option
The -gcc-name=dir option, used with -cxxlib, lets you specify the full-path location of gcc if the compiler cannot locate the gcc C++ libraries. Use this option when referencing a non-standard gcc installation.

-gcc-version option
The -gcc-version=nnn option provides compatible behavior with gcc, where nnn indicates the gcc version. The -gcc-version option is ON by default, and the value of nnn depends on the version of gcc installed on your system. This option selects the version of gcc with which you achieve ABI interoperability.

<table>
<thead>
<tr>
<th>Installed Version of gcc</th>
<th>Default Value of -gcc-version</th>
</tr>
</thead>
<tbody>
<tr>
<td>older than version 3.2</td>
<td>not set</td>
</tr>
<tr>
<td>3.2</td>
<td>320</td>
</tr>
<tr>
<td>3.3</td>
<td>330</td>
</tr>
<tr>
<td>3.4</td>
<td>340</td>
</tr>
<tr>
<td>4.0</td>
<td>400</td>
</tr>
<tr>
<td>4.1</td>
<td>410</td>
</tr>
<tr>
<td>4.2</td>
<td>420</td>
</tr>
<tr>
<td>4.3</td>
<td>430</td>
</tr>
</tbody>
</table>
-gxx-name option
The -gxx-name=dir option specifies that the g++ compiler should be used to set up the environment for C++ compilations.

-cxxlib option
The -cxxlib[=dir] option (ON by default) builds your applications using the C++ libraries and header files included with the gcc compiler. They include:

- libstdc++ standard C++ header files
- libstdc++ standard C++ library
- libgcc C++ language support

Use the optional argument, =dir, to specify the top-level location for the gcc binaries and libraries.

**NOTE.** The Intel C++ Compiler is compatible with gcc 3.2, 3.3, 3.4, 4.0, 4.1 and 4.2.

When you compile and link your application, the resulting C++ object files and libraries can interoperate with C++ object files and libraries generated by gcc 3.2 or higher. This means that third-party C++ libraries built with gcc 3.2 will work with C++ code generated by the Intel Compiler.

**NOTE.** gcc 3.2, 3.3, and 3.4 are not interoperable. gcc 4.0, 4.1, and 4.2 are interoperable. By default, the Intel compiler will generate code that is interoperable with the version of gcc it finds on your system.

By default, the Intel C++ Compiler uses headers and libraries included with the product, when the system includes a version of gcc less than 3.2.

If you build one shared library against the Intel C++ libraries, build a second shared library against the gnu C++ libraries, and use both libraries in a single application, you will have two C++ run-time libraries in use. Since the application might use symbols from both libraries, the following problems may occur:

- partially initialized libraries
- lost I/O operations from data put in unaccessed buffers
• other unpredictable results, such as jumbled output

The Intel C++ Compiler does not support more than one run-time library in one application.

CAUTION. If you successfully compile your application using more than one run-time library, the resulting program will likely be very unstable, especially when new code is linked against the shared libraries.

-fabi-version

The `-fabi-version=n` option directs the compiler to select a specific ABI implementation. By default, the Intel compiler uses the ABI implementation that corresponds to the installed version of gcc. Both gcc 3.2 and 3.3 are not fully ABI-compliant.

<table>
<thead>
<tr>
<th>Value of n</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=0</td>
<td>Select most recent ABI implementation</td>
</tr>
<tr>
<td>n=1</td>
<td>Select g++ 3.2 compatible ABI implementation</td>
</tr>
<tr>
<td>n=2</td>
<td>Select most conformant ABI implementation</td>
</tr>
</tbody>
</table>

See Also
• gcc* Interoperability
• Specifying Alternate Tools and Paths
• -gcc-name
• -gcc-version
• -cxxlib
• -fabi-version
• -no-gcc

For more information on ABI conformance, see http://www.codesourcery.com/

Predefined Macros for Interoperability

The Intel® C++ Compiler and gcc*/g++* support the following predefined macros:
• __GNUC__
• __GNUG__
• __GNUC_MINOR__
• __GNUC_PATCHLEVEL__

You can specify the -no-gcc option to undefine these macros. If you need gcc interoperability (-cxxlib), do not use the -no-gcc compiler option.

⚠️ **CAUTION.** Not defining these macros results in different paths through system header files. These alternate paths may be poorly tested or otherwise incompatible.

### See Also
- gcc* Interoperability
- Predefined Macros
- GNU Environment Variables

### gcc Built-in Functions

This version of the Intel® C++ compiler supports the following gcc* built-in functions:

```c
__builtin_abs
__builtin_labs
__builtin_cos
__builtin_cosf
__builtin_expect
__builtin_fabs
__builtin_fabsf
__builtin_memcmp
__builtin_memcpy
__builtin_sin
__builtin_sinf
__builtin_sqrt
__builtin_sqrtf
__builtin_strcmp
__builtin_strlen
__builtin_strncmp
__builtin_abort
__builtin_prefetch
__builtin_constant_p
__builtin_printf
__builtin_fprintf
__builtin_fscanf
```
The Intel® C++ Compiler supports the storage class keyword __thread, which can be used in variable definitions and declarations. Variables defined and declared this way are automatically allocated locally to each thread:

__thread int i;
__thread struct state s;
extern __thread char *p;

See Also
• gcc* Interoperability
• fbuiltin, Oi

Thread-local Storage
NOTE. The __thread keyword is only recognized when the GNU compatibility version is 3.3 or higher. You may need to specify the -gcc-version=330 compiler option to enable thread-local storage.
Conformance to the C Standard

The Intel® C++ Compiler provides conformance to the ANSI/ISO standard for C language compilation (ISO/IEC 9899:1990). This standard requires that conforming C compilers accept minimum translation limits. This compiler exceeds all of the ANSI/ISO requirements for minimum translation limits.

C99 Support

The following C99 features are supported in this version of the Intel C++ Compiler:

- restricted pointers (restrict keyword).
- variable-length Arrays
- flexible array members
- complex number support (_Complex keyword)
- hexadecimal floating-point constants
- compound literals
- designated initializers
- mixed declarations and code
- macros with a variable number of arguments
- inline functions (inline keyword)
- boolean type (_Bool keyword)

Conformance to the C++ Standard


Exported Templates

The Intel® C++ Compiler supports exported templates using the following options:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-export</code></td>
<td>Enable recognition of exported templates. Supported in C++ mode only.</td>
</tr>
<tr>
<td><code>-export-dir dir</code></td>
<td>Specifies a directory name to be placed on the exported template search path. The directories are used to find the definitions of exported templates and are searched in the order in which they are specified on the command-line. The current directory is always the first entry on the search path.</td>
</tr>
</tbody>
</table>

**NOTE.** Exported templates are not supported with gcc 3.3 or earlier. You need gcc 3.4 or newer to use this feature.
Exported templates are templates declared with the `export` keyword. Exporting a class template is equivalent to exporting each of its static data members and each of its non-inline member functions. An exported template is unique because its definition does not need to be present in a translation unit that uses that template. For example, the following C++ program consists of two separate translation units:

```cpp
// file1.cpp
#include <stdio.h>
static void trace() { printf("File 1\n"); }  
export template<class T> T const & min(T const &a, T const &b); 
int main() {
    trace();  
    return min(2, 3);  
}
```

```cpp
// file2.cpp
#include <stdio.h>
static void trace() { printf("File 2\n"); }  
export template<class T> T const & min(T const &a, T const &b) {
    trace();  
    return a<b? a: b;  
}
```

Note that these two files are separate translation units: one is not included in the other. That allows the two functions `trace()` to coexist (with internal linkage).

**Usage**

```bash
icpc -export -export-dir /usr2/export/ -c file1.cpp icpc -export -export-dir /usr2/export/ -c file2.cpp icpc -export -export-dir /usr2/export/ file1.o file2.o
```

**See Also**

- Language Conformance
- `export`
Template Instantiation

The Intel® C++ Compiler supports `extern template`, which lets you specify that a template in a specific translation unit will not be instantiated because it will be instantiated in a different translation unit or different library. The compiler now includes additional support for:

- **inline template** – instantiates the compiler support data for the class (i.e. the vtable) for a class without instantiating its members.
- **static template** – instantiates the static data members of the template, but not the virtual tables or member functions.

You can now use the following options to gain more control over the point of template instantiation:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-fno-implicit-templates</code></td>
<td>Never emit code for non-inline templates which are instantiated implicitly (i.e. by use). Only emit code for explicit instantiations.</td>
</tr>
<tr>
<td><code>-fno-implicit-inline-templates</code></td>
<td>Do not emit code for implicit instantiations of inline templates either. The default is to handle inlines differently so that compilations, with and without optimization, will need the same set of explicit instantiations.</td>
</tr>
</tbody>
</table>

**See Also**

- Language Conformance
- `-fno-implicit-templates`
- `-fno-implicit-inline-templates`
Overview: Porting Applications

This section describes a basic approach to porting applications from GCC's C/C++ compilers to the Intel® C/C++ compilers. These compilers correspond to each other in the following ways:

<table>
<thead>
<tr>
<th>Language</th>
<th>Intel Compiler</th>
<th>GCC* Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>icc</td>
<td>gcc</td>
</tr>
<tr>
<td>C++</td>
<td>icpc</td>
<td>g++</td>
</tr>
</tbody>
</table>

It also contains information on how to use the -diag enable port-win option to issue warnings about general syntactical problems when porting from GNU gcc* to Microsoft C++.

NOTE. To simplify this discussion on porting applications, the term "gcc", unless otherwise indicated, refers to both gcc and g++ compilers from the GNU Compiler Collection*.

Advantages to Using the Intel Compiler

In many cases, porting applications from gcc to the Intel compiler can be as easy as modifying your makefile to invoke the Intel compiler (icc) instead of gcc. Using the Intel compiler typically improves the performance of your application, especially for those that run on Intel processors. In many cases, your application's performance may also show improvement when running on non-Intel processors. When you compile your application with the Intel compiler, you have access to:

- compiler options that optimize your code for the latest Intel processors on IA-32 architecture, Intel® 64 architecture, and IA-64 architecture.
- advanced profiling tools (PGO) similar to gprof.
- high-level optimizations (HLO).
- interprocedural optimization (IPO).
- Intel intrinsic functions that the compiler uses to inline instructions, including SSE, SSE2, SSE3, SSSE3, and SSE4.
- the highly-optimized Intel Math Library for improved accuracy.
Since the Intel compiler is compatible and interoperable with gcc, porting your gcc application to the Intel compiler includes the benefits of binary compatibility. As a result, you should not have to re-build libraries from your gcc applications. The Intel compiler also supports many of the same compiler options, macros, and environment variables you already use in your gcc work.

**Porting Strategy**

For many gcc applications, porting to the Intel compiler requires little more than modifying your makefile to account for differences that may exist between compiling with gcc and compiling with icc.

One challenge in porting applications from one compiler to another is making sure there is support for the compiler options you use to build your application. The Compiler Options reference lists compiler options that are supported by both the Intel® C++ Compiler and gcc.

**Next Steps**

- Modifying Your Makefile
- Other Considerations

**Modifying Your makefile**

If you use makefiles to build your gcc application, you need to change the value for the `CC` compiler variable to use the Intel compiler. You may also want to review the options specified by `CFLAGS`. A simple example follows:
gcc makefile

# Use gcc compiler
CC = gcc

# Compile-time flags
CFLAGS = -O2 -std=c99

all: area_app
area_app: area_main.o area_functions.o
  $(CC) area_main.o area_functions.o -o area

area_main.o: area_main.c
  $(CC) -c $(CFLAGS) area_main.c

area_functions.o: area_functions.c
  $(CC) -c -fno-asm $(CFLAGS) area_functions.c

clean:
  rm -rf *o area

Modified makefile for Intel Compiler

# Use Intel C compiler
CC = icc

# Compile-time flags
CFLAGS = -std=c99

all: area-app
area-app: area_main.o area_functions.o
  $(CC) area_main.o area_functions.o -o area

area_main.o: area_main.c
  $(CC) -c $(CFLAGS) area_main.c

area_functions.o: area_functions.c
  gcc -c -O2 -fno-asm $(CFLAGS) area_functions.c

clean:
  rm -rf *o area
If your gcc code includes features that are not supported with the Intel compiler, such as compiler options, language extensions, macros, pragmas, etc., you can compile those sources separately with gcc if necessary.

In the above makefile, `area_functions.c` is an example of a source file that includes features unique to gcc. Since the Intel compiler uses the `-O2` compiler option by default and gcc's default is `-O0`, we instruct gcc to compile with `-O2`. We also include the `-fno-asm` switch from the original makefile since this switch is not supported with the Intel compiler. With the modified makefile, the output of `make` is:

```
icc -c -std=c99 area_main.c
gcc -c -O2 -fno-asm -std=c99 area_functions.c
icc area_main.o area_functions.o -o area
```

See Also

- Porting Applications
- Equivalent Macros
- Equivalent Environment Variables
- Compiler Options for Interoperability
- Support for gcc and g++ Language Extensions
- Predefined Macros
- gcc Builtin Functions

Equivalent Macros

Macro support is an important aspect in porting applications from gcc to the Intel compiler. The following table lists the most common macros used in both compilers.

```
__CHAR_BIT__
__DATE__
__DBL_DENORM_MIN__
__DBL_DIG__
__DBL_EPSILON__
__DBL_HAS_INFINITY__
```

---

140
### Equivalent Environment Variables

The Intel® C++ Compiler supports many of the same environment variables used with gcc as well as other compiler-specific variables. See Setting Environment Variables for a list of supported variables.

### Other Considerations

There are some notable differences between the Intel® C++ Compiler and gcc. Consider the following as you begin compiling your source code with the Intel C++ Compiler.

#### Setting the Environment

The Intel compilers rely on environment variables for the location of compiler binaries, libraries, man pages, and license files. In some cases these are different from the environment variables that gcc uses. Another difference is that these variables are not set by default after installing the Intel compiler. The following environment variables need to be set prior to running the Intel compiler:

- **PATH** – add the location of the compiler binaries to `PATH`
- **LD_LIBRARY_PATH** (Linux OS) – sets the location of the compiler libraries as well as the resulting binary generated by the compiler
- **DYLD_LIBRARY_PATH** (Mac OS X) - sets the location of the compiler libraries as well as the resulting binary generated by the compiler
- **MANPATH** – add the location of the compiler man pages (`icc` and `icpc`) to `MANPATH`
• **INTEL_LICENSE_FILE** – sets the location of the Intel compiler license file

You can `source` the `iccvars.sh` shell script (included with the compiler) to set these environment variables for you. See Invoking the Compiler from the Command Line for details on using `iccvars.sh`.

**NOTE.** Setting these environment variables with `iccvars.sh` does not impose a conflict with gcc. You should be able to use both compilers in the same shell.

### Using Optimization

The Intel C++ Compiler is an optimizing compiler that begins with the assumption that you want improved performance from your application when it is executed on Intel architecture. Consequently, certain optimizations, such as `-O2`, are part of the default invocation of the Intel compiler. By default, gcc turns off optimization - the equivalent of compiling with `-O` or `-O0`. Other forms of the `-O<n>` option compare as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Intel</th>
<th>gcc</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-00</code></td>
<td>Turns off optimization.</td>
<td>Default. Turns off optimization. Same as <code>-O</code>.</td>
</tr>
<tr>
<td><code>-01</code></td>
<td>Decreases code size with some increase in speed.</td>
<td>Decreases code size with some increase in speed.</td>
</tr>
<tr>
<td><code>-02</code></td>
<td>Default. Favors speed optimization with some increase in code size. Same as <code>-O</code>. Intrinsics, loop unrolling, and inlining are performed.</td>
<td>Optimizes for speed as long as there is not an increase in code size. Loop unrolling and function inlining, for example, are not performed.</td>
</tr>
<tr>
<td><code>-03</code></td>
<td>Enables <code>-O2</code> optimizations plus more aggressive optimizations, such as prefetching, scalar replacement, and loop and memory access transformations.</td>
<td>Optimizes for speed while generating larger code size. Includes <code>-O2</code> optimizations plus loop unrolling and inlining. Similar to <code>-O2 -ip</code> on Intel compiler.</td>
</tr>
</tbody>
</table>
Targeting Intel Processors

While many of the same options that target specific processors are supported with both compilers, Intel includes options that utilize processor-specific instruction scheduling to target the latest Intel processors. If you compile your gcc application with the \texttt{-march} or \texttt{-mtune} option, consider using Intel's \texttt{-x} or \texttt{-ax} options for applications that run on IA-32 or Intel® 64 architecture.

Modifying Your Configuration

The Intel compiler lets you maintain configuration and response files that are part of compilation. Options stored in the configuration file apply to every compilation, while options stored in response files apply only where they are added on the command line. If you have several options in your makefile that apply to every build, you may find it easier to move these options to the configuration file (.../bin/icc.cfg and .../bin/icpc.cfg).

In a multi-user, networked environment, options listed in the icc.cfg and icpc.cfg files are generally intended for everyone who uses the compiler. If you need a separate configuration, you can use the ICCCFFG or ICPCCFFG environment variable to specify the name and location of your own .cfg file, such as /my_code/my_config.cfg. Anytime you instruct the compiler to use a different configuration file, the system configuration files (icc.cfg and icpc.cfg) are ignored.
Using the Intel Math Library

With the Intel C++ Compiler, the Intel Math Library, libimf, is linked by default when calling math functions that require the library. Some functions, such as sin, do not require a call to the library, since the compiler already knows how to compute the sin function. The Intel Math Library includes many functions not found in the standard libm. A simple example using the sind function follows:

```c
#include <stdio.h>
#include <mathimf.h>
double sind(double);
double x = 90.0;
int main(void){
    printf("The sine of %f degrees is %f\n", x, sind(x) );
    return(0);
}
```

**NOTE.** You cannot make calls the Intel Math Library with gcc.

See Also
- Porting Applications
- -O
- Using Configuration Files
- Using Response Files

Porting from GNU gcc* to Microsoft Visual C++*

You can use the -diag-enable port-win option to alert you to general syntactical problems when porting an application from GNU gcc* to Microsoft Visual C++*. The following examples illustrate use of this option.
Example 1:
$ cat -n attribute.c
  1 int main()
  2 {
  3   int i __attribute__((unused));
  4   return 0;
  5 }
$ icc -c -diag-enable port-win attribute.c
attribute.c(3): warning #2133: attributes are a GNU extension
    int i __attribute__((unused));

Example 2:
$ cat -n state_expr.c
  1 int main()
  2 {
  3   int i = ({ int j; j = 1; j--; j;});
  4   return i;
  5 }
$ icc -c -diag-enable port-win state_expr.c
state_expr.c(3): warning #2132: statement expressions are a GNU extension
    int i = ({ int j; j = 1; j--; j;});
Remarks, Warnings, and Errors

This topic describes compiler remarks, warnings, and errors. The Intel® C++ Compiler displays these messages, along with the erroneous source line, on the standard output.

Remarks

Remark messages report common but sometimes unconventional use of C or C++. The compiler does not print or display remarks unless you specify the -w2 option. Remarks do not stop translation or linking. Remarks do not interfere with any output files. The following are some representative remark messages:

- function declared implicitly
- type qualifiers are meaningless in this declaration
- controlling expression is constant

Warnings

Warning messages report legal but questionable use of the C or C++. The compiler displays warnings by default. You can suppress all warning messages with the -w compiler option. Warnings do not stop translation or linking. Warnings do not interfere with any output files. The following are some representative warning messages:

- declaration does not declare anything
- pointless comparison of unsigned integer with zero
- possible use of = where == was intended

Additional Warnings

This version of Intel C++ Compiler includes support for the following options:
<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-W[no-]missing-prototypes</td>
<td>Warn for missing prototypes</td>
</tr>
<tr>
<td>-W[no-]pointer-arith</td>
<td>Warn for questionable pointer arithmetic</td>
</tr>
<tr>
<td>-W[no-]uninitialized</td>
<td>Warn if a variable is used before being initialized</td>
</tr>
<tr>
<td>-W[no-]deprecated</td>
<td>Display warnings related to deprecated features</td>
</tr>
<tr>
<td>-W[no-]abi</td>
<td>Warn if generated code is not C++ ABI compliant</td>
</tr>
<tr>
<td>-W[no-]unused-function</td>
<td>Warn if declared function is not used</td>
</tr>
<tr>
<td>-W[no-]unknown-pragma-directiv</td>
<td>Warn if an unknown #pragma directive is used</td>
</tr>
<tr>
<td>-W[no-]main</td>
<td>Warn if return type of main is not expected</td>
</tr>
<tr>
<td>-W[no-]comment[s]</td>
<td>Warn when /* appears in the middle of a /* */ comment</td>
</tr>
<tr>
<td>-W[no-]return-type</td>
<td>Warn when a function uses the default int return type</td>
</tr>
<tr>
<td></td>
<td>Warn when a return statement is used in a void function</td>
</tr>
</tbody>
</table>

**Errors**

These messages report syntactic or semantic misuse of C or C++. The compiler always displays error messages. Errors suppress object code for the module containing the error and prevent linking, but they allow parsing to continue to detect other possible errors. Some representative error messages are:

- missing closing quote
- expression must have arithmetic type
- expected a ";;"
Option Summary

Use the following compiler options to control remarks, warnings, and errors:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w</td>
<td>Suppress all warnings</td>
</tr>
<tr>
<td>-w0</td>
<td>Display only errors</td>
</tr>
<tr>
<td>-w1</td>
<td>Display only errors and warnings</td>
</tr>
<tr>
<td>-w2</td>
<td>Display errors, warnings, and remarks</td>
</tr>
<tr>
<td>-Wbrief</td>
<td>Display brief one-line diagnostics</td>
</tr>
<tr>
<td>-Wcheck</td>
<td>Enable more strict diagnostics</td>
</tr>
<tr>
<td>-Werror-all</td>
<td>Change all warnings and remarks to errors</td>
</tr>
<tr>
<td>-Werror</td>
<td>Change all warnings to errors</td>
</tr>
</tbody>
</table>

You can also control the display of diagnostic information with variations of the -diag compiler option. This compiler option accepts numerous arguments and values, allowing you wide control over displayed diagnostic messages and reports.

Some of the most common variations include the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-diag-enable list</td>
<td>Enables a diagnostic message or a group of messages</td>
</tr>
<tr>
<td>-diag-disable list</td>
<td>Disables a diagnostic message or a group of messages</td>
</tr>
<tr>
<td>-diag-warning list</td>
<td>Tells the compiler to change diagnostics to warnings</td>
</tr>
<tr>
<td>-diag-error list</td>
<td>Tells the compiler to change diagnostics to errors</td>
</tr>
<tr>
<td>-diag-remark list</td>
<td>Tells the compiler to change diagnostics to remarks (comments)</td>
</tr>
</tbody>
</table>
The list items can be specific diagnostic IDs, one of the keywords warn, remark, or error, or a keyword specifying a certain group (par, vec, sc, driver, thread, par,, port-win, sc,openmp). For more information, see -diag.

Other diagnostic-related options include the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-diag-dump</td>
<td>Tells the compiler to print all enabled diagnostic messages and stop compilation</td>
</tr>
<tr>
<td>-diag-file[=file]</td>
<td>Causes the results of diagnostic analysis to be output to a file</td>
</tr>
<tr>
<td>-diag-file-append[=file]</td>
<td>Causes the results of diagnostic analysis to be appended to a file</td>
</tr>
<tr>
<td>-diag-error-limit n</td>
<td>Specifies the maximum number of errors allowed before compilation stops</td>
</tr>
</tbody>
</table>

Using Source Code Verification

Source Checker Overview

The Intel® C++ Compiler Professional product provides the following source code analysis features:

- source checker analysis
- parallel lint

Source Checker

The source checker is a compiler feature that provides advanced diagnostics based on detailed analysis of source code. It performs static global analysis to find errors in software that go undetected by the compiler itself. A general source code analysis tool that provides an additional diagnostic capability to help you debug your programs. You can use source code analysis options to detect potential errors in your compiled code including:

- incorrect usage of OpenMP* directives
- inconsistent object declarations in different program units
- boundary violations
- uninitialized memory
- memory corruptions
- memory leaks
- incorrect usage of pointers and allocatable arrays
- dead code and redundant executions
- typographical errors or uninitialized variables
- dangerous usage of unchecked input

The source checker can be used to analyze and find issues with source files; these source files need not form a whole program (for instance, you can check a library source). In such cases, due to the lack of full information on usage and modification of global objects, calls to routines, and so forth, analysis will be less exact.

Your code must successfully compile, with no errors, for source code analysis options to take effect.

The intended output from the source checker are useful diagnostics; no executable is generated. Object files and library files generated during source checker analysis cannot be used to generate an executable or a dynamic or static library.

Source checker analysis performs a general overview check of a program for all possible values simultaneously. This is in contrast to run-time checking tools that execute a program with a fixed set of values for input variables; such checking tools cannot easily check all edge effects. By not using a fixed set of input values, the source checker analysis can check for all possible corner cases.

**Limitations of Source Checker Analysis**

Since the source checker does not perform full interpretation of analyzed programs, it can generate so called false-positive messages. This is a fundamental difference between the compiler and source checker generated errors; in the case of the source checker, you decide whether the generated error is legitimate and needs to be fixed.

**Example 1: Incorrect message about division by 0**
In this example, possible values for parameter $x$ are $\{6,3\}$, for parameter $y$ are $\{3,0\}$. If $x$ and $y$ both have the value 3, the expression $x-y$ is equal to 0. The source checker cannot identify that the value 3 for $x$ and $y$ cannot coexist.

```c
#include <stdio.h>

double foo(int x, int y) {
    return 1 / (x - y);
}

int main() {
    printf("%f\n", foo(6, 3));
    printf("%f\n", foo(3, 0));
    return 0;
}
```

The source checker issues the following message:

```
f1.c(4): error #12062: possible division by 0
```

**Example 2: Incorrect message about uninitialization**
This example illustrates how a false positive can appear from conditional statements.

```c
#include <stdio.h>

int main(int n) {
    int j;
    if (n != 0) {
        j = n;
    }
    if (n != 0) {
        printf("%d\n", j);
    }
    return 0;
}
```

The source checker issues the following message:

```
f1.c(9): error #12144: "j" is possibly uninitialized
```

**Parallel Lint**

Writing and debugging parallel programs requires specific knowledge and tools. Parallel lint can help in both the development of parallel applications and the parallelizing of existing serial applications. Based on source checker algorithms, parallel lint is a source code analysis capability that performs parallelization analysis of OpenMP* programs. The OpenMP 3.0 standard is supported. OpenMP provides a variety of ways for expressing parallelization. Parallel lint can diagnose problems with OpenMP directives and clauses, including:

- nested parallel regions including dynamic extent of parallel regions
- private/shared/reduction variables in parallel regions
- threadprivate variables
- expressions used in OpenMP clauses

In certain cases, an OpenMP program can meet all requirements of the specification but still have serious semantic issues. Parallel lint can help diagnose:

- some types of deadlocks
- data races or potential data dependency
- side effects without proper synchronization

Parallel lint also performs interprocedural analysis and can find issues with parallel directives located in different procedures or files.

**Using the Source Code Analysis Options**

Source code analysis options include the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-diag-enable sc{[1</td>
<td>2</td>
</tr>
<tr>
<td>/Qdiag-enable:sc{[1</td>
<td>2</td>
</tr>
<tr>
<td>-diag-disable sc</td>
<td>Disables source checker analysis</td>
</tr>
<tr>
<td>/Qdiag-disable:sc</td>
<td></td>
</tr>
<tr>
<td>-diag-enable sc-parallel{[1</td>
<td>2</td>
</tr>
<tr>
<td>/Qdiag-enable:sc-parallel{[1</td>
<td>2</td>
</tr>
<tr>
<td>-diag-disable sc-parallel</td>
<td>Disables parallel lint analysis</td>
</tr>
<tr>
<td>/Qdiag-disable:sc-parallel</td>
<td></td>
</tr>
<tr>
<td>-diag-enable sc-include</td>
<td>Analyzes include files as well as source files.</td>
</tr>
<tr>
<td>/Qdiag-enable:sc-include</td>
<td></td>
</tr>
<tr>
<td>-diag-disable warn</td>
<td>Suppresses all warnings, cautions and comments (issues errors only), including those specific to source code analysis</td>
</tr>
<tr>
<td>/Qdiag-disable:warn</td>
<td></td>
</tr>
</tbody>
</table>
Parallel lint diagnostics are a subset of source checker diagnostics. If both the source checker and parallel lint options are specified, only source checker options are used.

When using parallel lint, be sure to specify the OpenMP compiler option. Add the -openmp (Linux OS and Mac OS X) or /Qopenmp (Windows OS) option to the command line.

Specify the -diag-enable sc-included (Linux OS and Mac OS X) or /Qdiag-enable sc-included (Windows OS) option to analyze both source files and include files.

Using Source Code Analysis Options with Other Compiler Options

If the -c (Linux OS and Mac OS X) or /c (Windows OS) compiler option is used on the command line along with a command line option to enable source code analysis, an object file is created; source code analysis diagnostics are not produced. This object file may be used in
a further invocation of source code analysis. To receive complete source code diagnostics, specify source code analysis options for both compilation and linking phases. This feature is useful when a program consists of files written in different languages (C/C++ and Fortran).

```bash
icc -c -diag-enable sc2 file1.c
ifort -c -diag-enable sc2 file2.f90
icc -diag-enable sc2 file1.obj file2.obj
```

To analyze OpenMP directives, add the `-openmp` (Linux OS and Mac OS X) or `/Qopenmp` (Windows OS) option to the command line.

**Using Source Code Analysis within the IDE**

When source code analysis support is enabled within the IDE, the customary final build target (e.g. an executable image) is not created. Therefore, you should create a separate "Source Code Analysis" configuration.

**In the Eclipse* IDE, do the following:**

1. Open the property pages for the project and select C/C++ Build.
2. Click the Manage... button.
3. In the Manage dialog box, click the New... button to open the Create configuration dialog box.
4. Supply a name for the new configuration in the Name box; for example, Source Code Analysis.
5. Supply a Description for the configuration if you want (optional).
6. You can choose to Copy settings from a Default configuration or an Existing configuration by clicking the appropriate radio button and then selecting a configuration from the corresponding drop down menu.
7. Click OK to close the Create configuration dialog box.
8. Click OK to close the Manage dialog box (with your new configuration name selected).

The property pages will now display the settings for your new configuration; this becomes the active build configuration. Navigate to the Intel compiler's Compilation Diagnostics properties. Use the Level of Source Code Parallelization Analysis, Level of Source Code Analysis, and Analyze Include Files properties to control source code analysis.
**Source Checker Capabilities**

Source checker analysis capabilities include the following:

- **Interprocedural Analysis** for detecting inconsistent objects declarations in different program units and for propagation of data (for example, pointer aliases and argument constant values) using procedure calls.
- **Local Program Analysis** for analyzing each program unit separately and checking for various kinds of problems that will errors or warnings.
- **C/C++ specific Analysis** for analyzing C/C++ source code and checking for C/C++ specific error and warning conditions. Source code analysis also detects improper code style and flaws in object-oriented design solutions.
- **Fortran-specific Analysis** for analyzing Fortran source code and checking for Fortran-specific error and warning conditions.
- **OpenMP Analysis** for checking for OpenMP API restrictions.

**Interprocedural Analysis**

Source code analysis detects inconsistent object declarations in different program units, for example:

- Different external objects with the same name.
- Inconsistent declarations of a COMMON block (Fortran-specific).
- Mismatched number of arguments.
- Mismatched type, rank, shape, or/and size of an argument.
- Inconsistent declaration of a procedure.

The following examples illustrate interprocedural analysis.

**Example 1: Wrong number of arguments**

File f1.c contains this function declaration:

```c
1    void Do_hello() {
2       printf("helo everybody\n");
3     }
```

```c
```
File f2.c contains a call to the routine:
1 extern void Do_hello(int);
2
3 void foo() {
4   Do_hello(1);
5 }

The source checker issues the following message:
f2.c(4): error #12020: number of actual arguments (1) in call of "Do_hello"
doesn't match the number of formal arguments (0); "Do_hello" is defined at
(file:f1.c line:1)

The other goal of interprocedural analysis is to propagate information about program objects
across the program through procedure calls. The following information is propagated:

- Ranges of constant values
- Pointer attributes
- Information about usage and modification of objects

Example 2: Out of Boundaries
This example demonstrates the propagation of pointer A to Arr and the constant value 5 to x:
1 void foo( int* Arr, int x) {
2   for(; 0<=x; x--) {
3     Arr[x] = x;
4   }
5 }
6
7 void bar() {
8   int A[5];
9   foo(A,5);
10 }

The source checker issues the following message:
Local Program Analysis

The source checker uses local analysis of each program unit to check for various kinds of errors, warnings, and/or debatable points in a program. Examples of these errors are:

- Incorrect use or modification of an object
- Problems with memory (for example, leaks, corruptions, uninitialized memory)
- Incorrect use with pointers
- Boundaries violations
- Wrong value of an argument in an intrinsic call
- Dead code and redundant executions

The following examples illustrate local program analysis.

Example 1: Object is smaller than required size

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
    char string[10];

    strcpy(string, "Hello world from");
    printf("\n",string);

    return 0;
}
```

The following message is issued:

```c
f1.c(7): error #12224: Buffer overflow: size of object "string" (10 bytes) is less than required size (17 bytes)
```

Example 2: Memory Leak
File `f1.c` contains the following:

```c
1 #include <stdio.h>
2 #include <malloc.h>
3
4 int main(void) {
5  float **ptr;
6
7  ptr = (float **)malloc(8);
8  if (ptr == NULL) exit(1);
9  *ptr = (float*)malloc(sizeof(float));
10  if (*ptr == NULL) exit(1);
11  **ptr = 3.14;
12  printf("%f\n", **ptr);
13  free(ptr);
14  return 0;
15 }
```

The source checker issues the following message:

`f1.c(14): error #12121: memory leak: dynamic memory allocated at (file:f1.c line:9) is not freed at this point`

**C/C++ Specific Analysis**

Source code analysis examines C/C++ source code and checks for C++ specific errors. It also points out places of improper code style and flaws in object-oriented design solutions.

The source checker detects issues with the following:

- Memory management (leaks, mixing C and C++ memory management routines, smart pointer usage)
- C++ exception handling (uncaught exception, exception from destructor/operator delete)
- Misuse of operator new/operator delete
- Misuse of virtual functions
Example 1: Call of virtual function from constructor

```c++
#include "stdio.h"

class A {
public:
  A() { destroy(); }
  void destroy() { clear0();}
  virtual void clear()=0;
  void clear0() { clear(); ;}
};

class B : public A {
public:
  B(){ }
  virtual void clear(){ printf("overloaded clear"); }
  virtual ~B() { }
};

int main() {
  B b;
  return 0;
}
```

The source checker issues the following message:

f1.cpp(8): warning #12327: pure virtual function "clear" is called from constructor (file:f1.cpp line:5)

Fortran-Specific Analysis

The source checker is able to detect issues with the following:

- Mismatched type, rank, shape, or/and size of an argument
Incorrect usage of ALLOCATABLE arrays
Inconsistency in COMMON blocks

The following example illustrates Fortran-specific analysis.

**Example 1: Undefined function result**

File f1.f contains the following lines:

```fortran
1 subroutine foo(m)
2 integer, dimension(2,3) :: m
3 do i=1,3
4 print *,m(:,i)
5 end do
6 end
7 integer, dimension(3,2) :: n
8 do i=1,2
9 n(:,i) = i
10 end do
11 call foo(n)
12 ! shapes of argument #1 and dummy argument are different.
13 do i=1,2
14 print *,n(:,i)
15 end do
16 end
```

Source code analysis issues the following message:

`f1.f(11): error #12028: shape of actual argument 1 in call of "FOO" doesn't match the shape of formal argument "M"; "FOO" is defined`

**OpenMP* Analysis**

The compiler detects some restrictions noted in the OpenMP* API. With OpenMP analysis, additional checks for misuse of the OpenMP API are performed.

**Example 1: Incorrect usage of OpenMP directives**
File f1.c contains the following lines:
1 #include <stdio.h>
2 #include <omp.h>
3
4 void fff(int ii) {
5 #pragma omp barrier
6 printf("Val = %d \n", ii);
7 }
8
9 int main(void) {
10 int i=3;
11 omp_set_num_threads(3);
12 #pragma omp parallel
13 #pragma omp master
14 fff(i);
15 return 0;
16 }

Source code analysis issues the following message:
f1.c(5): error #12200: BARRIER directive is not allowed in the dynamic extent of MASTER directive (file:f1.c line:13)

Example 2: Incorrect data dependency
To enable data dependency analysis for a parallel program, enable diagnostics at level 3.

```c
int main(void) {
    int i, sum = 0;
    int a[1000];
    
    #pragma omp parallel for reduction(+:sum)
    for (i = 1; i < 999; i++) {
        a[i] = i;
        sum = sum + a[i + 1];
    }
}
```

Source code analysis issues the following message:

```
f1.c(8): warning #12247: anti data dependence from (file:f1.c line:8) to (file:f1.c line:7), due to "a" may lead to incorrect program execution in parallel mode
```

**Example 3: Incorrect synchronization**
File f1.c contains the following lines:

1 #include <stdio.h>
2 #include <omp.h>
3
4 int a[1000];
5 int sum = 0;
6
7 void moo() {
8 int i;
9
10 #pragma omp task
11 for (i=0; i<1000; i++) {
12 a[i] = i;
13 sum = sum + a[i];
14 }
15 }
16
17 void foo() {
18 printf("\%d\n", sum);
19 }
20
21 int main(void) {
22 int i;
23

24 #pragma omp parallel shared(sum)

25 #pragma omp single

26 {

27 moo();

28 foo();

29 }

30 return 0;

31 }

Source code analysis issues the following message:
f1.c(18): error #12365: variable "sum" is defined at (file:f1.c line:13) in
TASK region (file:f1.c line:10) and is used before synchronization
ANSI Standard Predefined Macros

The ANSI/ISO standard for the C language requires that certain predefined macros be supplied with conforming compilers. The following table lists the macros that the Intel® C++ Compiler supplies in accordance with this standard:

The compiler includes predefined macros in addition to those required by the standard. The default predefined macros differ among Windows*, Linux*, and Mac OS* X operating systems due to the default /Za compiler option on Windows. Differences also exist on Linux OS and Mac OS X as a result of the -std compiler option.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DATE</strong></td>
<td>The date of compilation as a string literal in the form Mmm dd yyyy.</td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>A string literal representing the name of the file being compiled.</td>
</tr>
<tr>
<td><strong>LINE</strong></td>
<td>The current line number as a decimal constant.</td>
</tr>
<tr>
<td><strong>STDC</strong></td>
<td>The name <strong>STDC</strong> is defined when compiling a C translation unit.</td>
</tr>
<tr>
<td><strong>STDC_HOSTED</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>The time of compilation as a string literal in the form hh:mm:ss.</td>
</tr>
</tbody>
</table>

See Also

- Reference
- Additional Predefined Macros

Additional Predefined Macros

The Intel® C++ Compiler supports the predefined macros listed in the table below. The compiler also includes predefined macros specified by the ISO/ANSI standard.

The following designations apply:
### Label Meaning

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td>Included on systems based on IA-32 architecture.</td>
</tr>
<tr>
<td>i64em</td>
<td>Included on systems based on Intel® 64 architecture.</td>
</tr>
<tr>
<td>i64</td>
<td>Included on systems based on IA-64 architecture.</td>
</tr>
</tbody>
</table>

### Macro Name | Value | i32 | i64em | i64 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARRAY_OPERATORS</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>BASE_FILE</strong></td>
<td>Name of source file</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><em>BOOL</em></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>cplusplus</strong></td>
<td>1 (with C++ compiler)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>DEPRECATED</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>ECC</strong></td>
<td>Intel Compiler Version</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EDG</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>EDG_VERSION</strong></td>
<td>EDG version</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>ELF</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>extension</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>EXCEPTIONS</strong></td>
<td>Defined as 1 when -fno-except- tions is not used.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>GNUC</strong></td>
<td>The major version number of gcc installed on the system.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>GNUG</strong></td>
<td>The major version number of g++ installed on the system.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>gnuinux</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>GNUC_MINOR</strong></td>
<td>The minor version number of gcc or g++ installed on the system.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Macro Name</td>
<td>Value</td>
<td>i32</td>
<td>i64em</td>
<td>i64</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------</td>
<td>-----</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td><strong>GNUC_PATCHLEVEL</strong></td>
<td>The patch level version number of gcc or g++ installed on the system.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__GXX_ABI_VERSION</td>
<td>102</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__HONOR_STD</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>__i386</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>i386</strong></td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i386</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__IA64</td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>IA64</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>__ICC</td>
<td>Intel compiler version</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_INTEGRAL_MAX_BITS</td>
<td>64</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>__INTEL_COMPILER</td>
<td>Intel compiler version</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__INTEL_COMPILER_BUILD_DATE</td>
<td>YYYYMMDD</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>INTEL_RTTI</strong></td>
<td>Defined as 1 when <code>-fno-rtti</code> is not specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>INTEL_STRICT_ANSI</strong></td>
<td>Defined as 1 when <code>-strict-ansi</code> is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>itanium</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>__linux</td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>linux</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>linux</td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>LONG_DOUBLE_SIZE</strong></td>
<td>80</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
## Blacksight Data

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
<th>i32</th>
<th>i64em</th>
<th>i64</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LONG_MAX</strong></td>
<td>9223372036854775807L</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__lp64</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>LP64</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><em>LP64</em></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><em>MT</em></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>MMX</strong></td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NO_INLINE</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__NO_MATH_INLINES</td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__NO_STRING_INLINES</td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_OPENMP</td>
<td>Defined as 200805 when -openmp is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>OPTIMIZE</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><em>pentium4</em></td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><em>pentium4</em>_</td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>PIC</strong></td>
<td>Defined as 1 when -fPIC is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>pic</strong></td>
<td>Defined as 1 when -fPIC is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_PGO_INSTRUMENT</td>
<td>Defined as 1 when -prof-gen[x] is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_PLACEMENT_DELETE</td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>PTRDIFF_TYPE</strong></td>
<td>int on IA-32 architecture long on Intel 64 architecture long on IA-64 architecture</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Macro Name</td>
<td>Value</td>
<td>i32</td>
<td>i64em</td>
<td>i64</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------</td>
<td>-----</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td><strong>REGISTER_PREFIX</strong></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>SIGNED_CHARS</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>SIZE_TYPE</strong></td>
<td>unsigned on IA-32 architecture</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>unsigned long on Intel® 64 architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned long on IA-64 architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SSE</strong></td>
<td>Defined as 1 for processors that support SSE instructions.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>SSE2</strong></td>
<td>Defined as 1 for processors that support SSE2 instructions.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>SSE3</strong></td>
<td>Defined as 1 for processors that support SSE3 instructions.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>SSSE3</strong></td>
<td>Defined as 1 for processors that support SSSE3 instructions.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>unix</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>unix</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>unix</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>USER_LABEL_PREFIX</strong></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>VERSION</strong></td>
<td>Intel version string</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>WCHAR_T</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>WCHAR_TYPE</strong></td>
<td>long int on IA-32 architecture</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>int on Intel® 64 architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>int on IA-64 architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WINT_TYPE</strong></td>
<td>unsigned int</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__x86_64</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
## See Also

- Reference
- `-D`
- `-U`
- ANSI Standard Predefined Macros

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
<th>i32</th>
<th>i64em</th>
<th>i64</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x86_64</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Part II

Compiler Options

Topics:

- Overview: Compiler Options
- Alphabetical Compiler Options
- Quick Reference Guide and Cross Reference
Overview: Compiler Options

This document provides details on all current Linux* OS, Mac OS* X, and Windows* OS compiler options.

It provides the following information:

- **New options**
  This topic lists new compiler options in this release.

- **Deprecated**
  This topic lists deprecated and removed compiler options for this release. Some deprecated options show suggested replacement options.

- **Alphabetical Compiler Options**
  This topic is the main source in the documentation set for general information on all compiler options. Options are described in alphabetical order. The Overview describes what information appears in each compiler option description.

- **Quick Reference Guide and Cross Reference**
  This topic contains a table summarizing compiler options. The table shows the option name, a short description of the option, the default setting for the option, and the equivalent option on the alternate operating system.

- **Related Options**
  This topic lists related options that can be used under certain conditions.

In this guide, compiler options are available on all supported operating systems and architectures unless otherwise identified.

For information on compiler options that are equivalent to gcc options, see Portability Options.

For further information on compiler options, see Building Applications and Optimizing Applications.

**Functional Groupings of Compiler Options**

To see functional groupings of compiler options, specify a functional category for option help on the command line. For example, to see a list of options that affect diagnostic messages displayed by the compiler, enter one of the following commands:

```
-<b>help</b> diagnostics  ! Linux and Mac OS X systems
/help diagnostics       ! Windows systems
```
For details on the categories you can specify, see help.

New Options

This topic lists the options that provide new functionality in this release.

Some compiler options are only available on certain systems, as indicated by these labels:

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td>The option is available on systems using IA-32 architecture.</td>
</tr>
<tr>
<td>i64em</td>
<td>The option is available on systems using Intel® 64 architecture.</td>
</tr>
<tr>
<td>i64</td>
<td>The option is available on systems using IA-64 architecture.</td>
</tr>
</tbody>
</table>

If no label appears, the option is available on all supported systems.

If "only" appears in the label, the option is only available on the identified system.

For more details on the options, refer to the Alphabetical Compiler Options section.

For information on conventions used in this table, see Conventions.

New compiler options are listed in tables below:

- The first table lists new options that are available on Windows* systems.
- The second table lists new options that are available on Linux* and Mac OS* X systems. If an option is only available on one of these operating systems, it is labeled.

### Windows* OS Options

<table>
<thead>
<tr>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>Generates code that will run on any Pentium or later processor.</td>
</tr>
<tr>
<td>OFF</td>
<td>Optimizes for Intel® Streaming SIMD Extensions 3 (Intel® SSE3).</td>
</tr>
<tr>
<td>OFF</td>
<td>Optimizes for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3).</td>
</tr>
</tbody>
</table>

---

177
<table>
<thead>
<tr>
<th>Windows* OS Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>/arch:SSE4.1 (i32, i64em)</td>
<td>Optimizes for Intel® Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators.</td>
<td>OFF</td>
</tr>
<tr>
<td>/bigobj</td>
<td>Increases the number of sections that an object file can contain.</td>
<td>OFF</td>
</tr>
<tr>
<td>/debug:[no]ex-pr-source-pos</td>
<td>Generates source position information at the statement level of granularity.</td>
<td>OFF</td>
</tr>
<tr>
<td>/debug:[no]parallel(i32, i64em)</td>
<td>Generates parallel debug code instrumentations needed for the thread data sharing and reentrant call detection of the Intel® Parallel Debugger Extension.</td>
<td>OFF</td>
</tr>
<tr>
<td>/homeparams</td>
<td>Tells the compiler to store parameters passed in registers to the stack.</td>
<td>OFF</td>
</tr>
<tr>
<td>/hotpatch (i32, i64em)</td>
<td>Tells the compiler to prepare a routine for hotpatching</td>
<td>OFF</td>
</tr>
<tr>
<td>/MP</td>
<td>Creates multiple processes that can be used to compile large numbers of source files at the same time.</td>
<td>OFF</td>
</tr>
<tr>
<td>/QaxSSE2 (i32, i64em)</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2.</td>
<td>OFF</td>
</tr>
<tr>
<td>/QaxSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>/QaxSSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family.</td>
<td>OFF</td>
</tr>
</tbody>
</table>
### Compiler Options

<table>
<thead>
<tr>
<th>Windows* OS Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>/QaxSSE4.1 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>/QaxSSE4.2 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qcov-dir dir (i32, i64em)</td>
<td>Specifies a directory for profiling information output files that can be used with the codecov or tselect tool.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qcov-file file (i32, i64em)</td>
<td>Specifies an alternate file name for the profiling summary files that can be used with the codecov or tselect tool.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qdiag-enable:sc-parallel (i32, i64em)</td>
<td>Enables analysis of parallelization in source code (parallel lint diagnostics).</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qdiag-error-limit:n</td>
<td>Specifies the maximum number of errors allowed before compilation stops.</td>
<td>n=30</td>
</tr>
<tr>
<td>/Qdiag-once:id[,id,...]</td>
<td>Tells the compiler to issue one or more diagnostic messages only once.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qdiag-error-limit:n</td>
<td>Specifies the maximum number of errors allowed before compilation stops.</td>
<td>n=30</td>
</tr>
<tr>
<td>/Qdiag-once:id[,id,...]</td>
<td>Tells the compiler to issue one or more diagnostic messages only once.</td>
<td>OFF</td>
</tr>
<tr>
<td>Windows* OS Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>/Qfast-trancendentals</td>
<td>Enables the compiler to replace calls to transcendental functions with faster but less precise implementations.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qfma (i64 only)</td>
<td>Enables the combining of floating-point multiplies and add/subtract operations.</td>
<td>ON</td>
</tr>
<tr>
<td>/Qfp-relaxed (i64 only)</td>
<td>Enables use of faster but slightly less accurate code sequences for math functions.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qfreestanding</td>
<td>Ensures that compilation takes place in a freestanding environment.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qhelp-pragmas (i32, i64em)</td>
<td>Displays all supported pragmas.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qinstruction:[no]movbe (i32, i64em)</td>
<td>Determines whether MOVBE instructions are generated for Intel processors.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qipp</td>
<td>Tells the compiler to link to the some or all of the Intel® IPP libraries.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qmkl</td>
<td>Tells the compiler to link to certain parts of the Intel® Math Kernel Library.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qopenmp-link:library</td>
<td>Controls whether the compiler links to static or dynamic OpenMP run-time libraries.</td>
<td>/Qopenmp-link:dynami</td>
</tr>
<tr>
<td>/Qopenmp-task:model</td>
<td>Lets you choose an OpenMP* tasking model.</td>
<td>/Qopenmp-task:omp</td>
</tr>
<tr>
<td>/Qopenmp-threadprivate:type</td>
<td>Lets you specify an OpenMP* threadprivate implementation.</td>
<td>/Qopenmp-threadpri-vate:legacy</td>
</tr>
<tr>
<td>/Qopt-block-factor:n</td>
<td>Lets you specify a loop blocking factor.</td>
<td>OFF</td>
</tr>
</tbody>
</table>
## Compiler Options

<table>
<thead>
<tr>
<th>Windows* OS Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/Qopt-jump-tables:keyword</code></td>
<td>Enables or disables generation of jump tables for switch statements.</td>
<td><code>/Qopt-jump-tables:default</code></td>
</tr>
<tr>
<td><code>/Qopt-load-pair</code></td>
<td>Enables loadpair optimization.</td>
<td><code>/Qopt-load-pair</code></td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>/Qopt-mod-versioning</code></td>
<td>Enables versioning of modulo operations for certain types of operands.</td>
<td><code>/Qopt-mod-versioning</code></td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>/Qopt-prefetch-initial-values</code></td>
<td>Enables or disables prefetches that are issued before a loop is entered.</td>
<td><code>/Qopt-prefetch-initial-values</code></td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>/Qopt-prefetch-issue-excl-hint</code></td>
<td>Determines whether the compiler issues prefetches for stores with exclusive hint.</td>
<td><code>/Qopt-prefetch-issue-excl-hint</code></td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>/Qopt-prefetch-next-iteration</code></td>
<td>Enables or disables prefetches for a memory access in the next iteration of a loop.</td>
<td><code>/Qopt-prefetch-next-iteration</code></td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>/Qopt-subscript-in-range</code></td>
<td>Determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.</td>
<td><code>/Qopt-subscript-in-range-</code></td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows* OS Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>/Qpar-affinity:[modifier,...]type,[permute][,off-set]</td>
<td>Specifies thread affinity.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qpar-num-threads:n</td>
<td>Specifies the number of threads to use in a parallel region.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qprof-data-order</td>
<td>Enables or disables data ordering if profiling information is enabled.</td>
<td>/Qprof-data-order</td>
</tr>
<tr>
<td>/Qprof-func-order</td>
<td>Enables or disables function ordering if profiling information is enabled.</td>
<td>/Qprof-func-order</td>
</tr>
<tr>
<td>/Qprof-hotness-threshold</td>
<td>Lets you set the hotness threshold for function grouping and function ordering.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qprof-src-dir</td>
<td>Determines whether directory information of the source file under compilation is considered when looking up profile data records.</td>
<td>/Qprof-src-dir</td>
</tr>
<tr>
<td>/Qprof-src-root</td>
<td>Lets you use relative directory paths when looking up profile data and specifies a directory as the base.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qprof-src-root-cwd</td>
<td>Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qtbb</td>
<td>Tells the compiler to link to the Intel® TBB libraries.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qtcollect-filter</td>
<td>Lets you enable or disable the instrumentation of specified functions.</td>
<td>OFF</td>
</tr>
</tbody>
</table>
### Compiler Options

<table>
<thead>
<tr>
<th>Windows* OS Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Quse-intel-</td>
<td>Determines whether the performance headers directory is added to the include path search list.</td>
<td>OFF</td>
</tr>
<tr>
<td>optimized-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>headers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Quse-msasm-</td>
<td>Tells the compiler to use a dollar sign (&quot;$&quot;) when producing symbol names.</td>
<td>OFF</td>
</tr>
<tr>
<td>symbols</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qvec9</td>
<td>Specifies compatibility with Microsoft* Visual Studio 2008.</td>
<td>varies</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qvec</td>
<td>Enables or disables vectorization and transformations enabled for vectorization.</td>
<td>/Qvec</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qvec-thresh-</td>
<td>Sets a threshold for the vectorization of loops.</td>
<td>/Qvec-thresh-</td>
</tr>
<tr>
<td>old</td>
<td></td>
<td>old100</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxHost</td>
<td>Can generate instructions for the highest instruction set available on the compilation host processor.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxAVX</td>
<td>Optimizes for Intel processors that support Intel® Advanced Vector Extensions (Intel® AVX).</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxSSE2</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2.</td>
<td>ON</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxSSE3</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors, and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxSSE3_ATOM</td>
<td>Optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows* OS Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>/QxSSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family.</td>
<td>OFF</td>
</tr>
<tr>
<td>/QxSSE4.1 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>/QxSSE4.2 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Werror-all</td>
<td>Changes all warnings and remarks to errors.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Yd</td>
<td>Tells the compiler to add complete debugging information in all object files created from a precompiled header (.pch) file when option /Zi or /Z7 is specified.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Zx (i64)</td>
<td>Disables certain optimizations, such as software pipelining and global scheduling optimizations, that make it difficult to debug optimized code.</td>
<td>OFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linux* OS and Mac OS* X Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-axSSE2 (i32, i64em; Linux only)</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2.</td>
<td>OFF</td>
</tr>
</tbody>
</table>
### Compiler Options

<table>
<thead>
<tr>
<th>Linux* OS and Mac OS* X Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-axSSE3</code> (i32, i64em)</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors, and it can optimize for processors based on Intel® Core microarchitecture and Intel NetBurst® microarchitecture. On Mac OS* X systems, this option is only available on IA-32 architecture.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-axSSSE3</code> (i32, i64em)</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™ 2 Duo processor family. On Mac OS* X systems, this option is only available on Intel® 64 architecture.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-axSSE4.1</code> (i32, i64em)</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-axSSE4.2</code> (i32, i64em)</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-debug parallel</code> (i32, i64em; Linux* OS only)</td>
<td>Generates parallel debug code instrumentations needed for the thread data sharing and reentrant call detection of the Intel® Parallel Debugger Extension.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-diag-enable sc-parallel</code> (i32, i64em)</td>
<td>Enables analysis of parallelization in source code (parallel lint diagnostics).</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-diag-error-limit n</code></td>
<td>Specifies the maximum number of errors allowed before compilation stops.</td>
<td>n=30</td>
</tr>
<tr>
<td>Linux* OS and Mac OS* X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>-diag-once id[,id,...]</td>
<td>Tells the compiler to issue one or more diagnostic messages only once.</td>
<td>OFF</td>
</tr>
<tr>
<td>-falign-stack (i32 only)</td>
<td>Tells the compiler the stack alignment to use on entry to routines.</td>
<td>-falign-stack=default</td>
</tr>
<tr>
<td>-fasm-blocks (i32, i64em; Mac OS* X only)</td>
<td>Enables the use of blocks and entire functions of assembly code within a C or C++ file.</td>
<td>OFF</td>
</tr>
<tr>
<td>-fast-transcendentals</td>
<td>Enables the compiler to replace calls to transcendental functions with faster but less precise implementation.</td>
<td>OFF</td>
</tr>
<tr>
<td>-ffreestanding</td>
<td>Ensures that compilation takes place in a freestanding environment.</td>
<td>OFF</td>
</tr>
<tr>
<td>-fma (i64 only; Linux* OS only)</td>
<td>Enables the combining of floating-point multiplies and add/subtract operations.</td>
<td>ON</td>
</tr>
<tr>
<td>-fnon-call-exceptions (i32, i64em)</td>
<td>Allows trapping instructions to throw C++ exceptions.</td>
<td>-fno-non-call-exceptions</td>
</tr>
<tr>
<td>-fp-relaxed (i64 only; Linux* OS only)</td>
<td>Enables use of faster but slightly less accurate code sequences for math functions.</td>
<td>OFF</td>
</tr>
<tr>
<td>-fpie (Linux* OS only)</td>
<td>Tells the compiler to generate position-independent code to link into executables.</td>
<td>OFF</td>
</tr>
<tr>
<td>-fstack-protector (i32, i64em)</td>
<td>Determines whether the compiler generates code that detects some buffer overruns. Same as option -fstack-security-check.</td>
<td>-fno-stack-protector</td>
</tr>
<tr>
<td>-help-pragmas</td>
<td>Displays all supported pragmas.</td>
<td>OFF</td>
</tr>
</tbody>
</table>
### Compiler Options

<table>
<thead>
<tr>
<th>Linux* OS and Mac OS* X Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-ipp</td>
<td>Tells the compiler to link to the some or all of the Intel® IPP libraries.</td>
<td>OFF</td>
</tr>
<tr>
<td>-m32, -m64 (i32, i64em)</td>
<td>Tells the compiler to generate code for a specific architecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>-mia32 (i32 only)</td>
<td>Generates code that will run on any Pentium or later processor.</td>
<td>OFF</td>
</tr>
<tr>
<td>-minstruction=[no]movbe (i32, i64em)</td>
<td>Determines whether MOVBE instructions are generated for Intel processors.</td>
<td>OFF</td>
</tr>
<tr>
<td>-mkl</td>
<td>Tells the compiler to link to certain parts of the Intel® Math Kernel Library.</td>
<td>OFF</td>
</tr>
<tr>
<td>-mssse3 (i32, i64em)</td>
<td>Generates code for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3).</td>
<td>Linux systems: OFF Mac OS X systems using Intel® 64 architecture: ON</td>
</tr>
<tr>
<td>-msse4.1 (i32, i64em)</td>
<td>Generates code for Intel® Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators.</td>
<td>OFF</td>
</tr>
<tr>
<td>-multiple-processes</td>
<td>Creates multiple processes that can be used to compile large numbers of source files at the same time.</td>
<td>OFF</td>
</tr>
<tr>
<td>-openmp-link library</td>
<td>Controls whether the compiler links to static or dynamic OpenMP run-time libraries.</td>
<td>-openmp-link dynamic</td>
</tr>
<tr>
<td>-openmp-task model</td>
<td>Lets you choose an OpenMP* tasking model.</td>
<td>-openmp-task omp</td>
</tr>
<tr>
<td>Linux* OS and Mac OS X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>-openmp-threadprivate=type (Linux* OS only)</td>
<td>Lets you specify an OpenMP* threadprivate implementation.</td>
<td>-openmp-threadprivate=legacy</td>
</tr>
<tr>
<td>-opt-block-factor=n</td>
<td>Lets you specify a loop blocking factor.</td>
<td>OFF</td>
</tr>
<tr>
<td>-opt-calloc (Linux* OS only)</td>
<td>Tells the compiler to substitute a call to _intel_fast_calloc() for a call to calloc().</td>
<td>OFF</td>
</tr>
<tr>
<td>-opt-jump-tables=keyword</td>
<td>Enables or disables generation of jump tables for switch statements.</td>
<td>-opt-jump-tables=default</td>
</tr>
<tr>
<td>-opt-loadpair (i64 only; Linux* OS only)</td>
<td>Enables loadpair optimization.</td>
<td>-no-opt-loadpair</td>
</tr>
<tr>
<td>-opt-mod-versioning (i64 only; Linux* OS only)</td>
<td>Enables versioning of modulo operations for certain types of operands.</td>
<td>-no-opt-mod-versioning</td>
</tr>
<tr>
<td>-opt-prefetch-initial-values (i64 only; Linux* OS only)</td>
<td>Enables or disables prefches that are issued before a loop is entered.</td>
<td>-opt-prefetch-initial-values</td>
</tr>
<tr>
<td>-opt-prefetch-issue-excl-hint (i64 only)</td>
<td>Determines whether the compiler issues prefetches for stores with exclusive hint.</td>
<td>-no-opt-prefetch-issue-excl-hint</td>
</tr>
</tbody>
</table>
### Compiler Options

<table>
<thead>
<tr>
<th><em><em>Linux</em> OS and Mac OS</em> X Options**</th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>-opt-prefetch-next-iteration</td>
<td>Enables or disables prefetches for a memory access in the next iteration of a loop.</td>
<td>-opt-prefetch-next-iteration</td>
</tr>
<tr>
<td>(i64 only; Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-opt-subscript-in-range</td>
<td>Determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.</td>
<td>-no-opt-subscript-in-range</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-par-affinity=[modifier,...]type[,permute][,offset]</td>
<td>Specifies thread affinity.</td>
<td>OFF</td>
</tr>
<tr>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-par-num-threads=n</td>
<td>Specifies the number of threads to use in a parallel region.</td>
<td>OFF</td>
</tr>
<tr>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-pie</td>
<td>Produces a position-independent executable on processors that support it.</td>
<td>OFF</td>
</tr>
<tr>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-prof-data-order</td>
<td>Enables or disables data ordering if profiling information is enabled.</td>
<td>-no-prof-data-order</td>
</tr>
<tr>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-prof-func-groups</td>
<td>Enables or disables function grouping if profiling information is enabled.</td>
<td>-no-prof-func-groups</td>
</tr>
<tr>
<td>(i32, i64em; Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-prof-func-order</td>
<td>Enables or disables function ordering if profiling information is enabled.</td>
<td>-no-prof-func-order</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><em>Linux</em> OS and Mac OS</em> X Options**</td>
<td><strong>Description</strong></td>
<td><strong>Default</strong></td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-prof-hotness-threshold</td>
<td>Lets you set the hotness threshold for function grouping and function ordering.</td>
<td>OFF</td>
</tr>
<tr>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-prof-src-root</td>
<td>Lets you use relative directory paths when looking up profile data and specifies a directory as the base.</td>
<td>OFF</td>
</tr>
<tr>
<td>-prof-src-root-cwd</td>
<td>Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.</td>
<td>OFF</td>
</tr>
<tr>
<td>-staticlib</td>
<td>Invokes the libtool command to generate static libraries.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em; Mac OS* X only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-tbb</td>
<td>Tells the compiler to link to the Intel® TBB libraries.</td>
<td>OFF</td>
</tr>
<tr>
<td>-tcollect-filter</td>
<td>Lets you enable or disable the instrumentation of specified functions.</td>
<td>OFF</td>
</tr>
<tr>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-use-intel-optimized-headers</td>
<td>Determines whether the performance headers directory is added to the include path search list.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-vec</td>
<td>Enables or disables vectorization and transformations enabled for vectorization.</td>
<td>-vec</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-vec-thresh-old</td>
<td>Sets a threshold for the vectorization of loops.</td>
<td>-vec-thresh-old100</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-Werror-all</td>
<td>Changes all warnings and remarks to errors.</td>
<td>OFF</td>
</tr>
</tbody>
</table>
### Compiler Options

<table>
<thead>
<tr>
<th>Linux* OS and Mac OS* X Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Wformat-security</td>
<td>Issues a warning when the use of format functions may cause security problems.</td>
<td>-Wno-format-security</td>
</tr>
<tr>
<td>-xHost (i32, i64em)</td>
<td>Can generate instructions for the highest instruction set available on the compilation host processor.</td>
<td>OFF</td>
</tr>
<tr>
<td>-xAVX (i32, i64em)</td>
<td>Optimizes for Intel processors that support Intel® Advanced Vector Extensions (Intel® AVX).</td>
<td>OFF</td>
</tr>
<tr>
<td>-xsSSE2 (i32, i64em; Linux only)</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2.</td>
<td>ON</td>
</tr>
<tr>
<td>-xsSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. On Mac OS* X systems, this option is only available on IA-32 architecture.</td>
<td>Linux systems:OFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mac OS X systems using IA-32 architecture: ON</td>
</tr>
<tr>
<td>-xsSSE3_ATOM (i32, i64em)</td>
<td>Optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology.</td>
<td>OFF</td>
</tr>
<tr>
<td>-xsSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. On Mac OS* X systems, this option is only available on Intel® 64 architecture.</td>
<td>Linux systems:OFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mac OS X systems using Intel® 64 architecture: ON</td>
</tr>
<tr>
<td>-xsSSE4.1 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.</td>
<td>OFF</td>
</tr>
</tbody>
</table>
Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.

**Deprecated and Removed Compiler Options**

This topic lists deprecated and removed compiler options and suggests replacement options, if any are available.

**Deprecated Options**

Occasionally, compiler options are marked as "deprecated." Deprecated options are still supported in the current release, but are planned to be unsupported in future releases.

The following options are deprecated in this release of the compiler:

<table>
<thead>
<tr>
<th>Linux* OS and Mac OS* X Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-xsse4.2</code> (i32, i64em)</td>
<td>None</td>
</tr>
<tr>
<td><code>-fargument-alias-alias-args</code></td>
<td><code>-fargument-alias</code></td>
</tr>
<tr>
<td><code>-axK</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-axN</code></td>
<td>Linux* OS: <code>-axSSE2</code></td>
</tr>
<tr>
<td></td>
<td>Mac OS* X: None</td>
</tr>
<tr>
<td><code>-axP</code></td>
<td>Linux* OS: <code>-axSSE3</code></td>
</tr>
<tr>
<td></td>
<td>Mac OS* X on IA-32 architecture: <code>-axSSE3</code></td>
</tr>
<tr>
<td></td>
<td>Mac OS* X on Intel® 64 architecture: None</td>
</tr>
<tr>
<td><code>-axS</code></td>
<td><code>-axSSE4.1</code></td>
</tr>
<tr>
<td>Linux* OS and Mac OS* X Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><code>-axT</code></td>
<td>Linux* OS: <code>-axSSSE3</code></td>
</tr>
<tr>
<td></td>
<td>Mac OS X on IA-32 architecture: None</td>
</tr>
<tr>
<td></td>
<td>Mac OS X on Intel® 64 architecture: <code>-axSSSE3</code></td>
</tr>
<tr>
<td><code>-axW</code></td>
<td><code>-msse2</code></td>
</tr>
<tr>
<td><code>-c99</code></td>
<td><code>-std=c99</code></td>
</tr>
<tr>
<td><code>-create-pch</code></td>
<td><code>-pch-create</code></td>
</tr>
<tr>
<td><code>-diag&lt;-type&gt; sv[&lt;n&gt;]</code></td>
<td><code>-diag&lt;-type&gt; sc[&lt;n&gt;]</code></td>
</tr>
<tr>
<td><code>-diag-enable sv-include</code></td>
<td><code>-diag-enable sc-include</code></td>
</tr>
<tr>
<td><code>-func-groups</code></td>
<td><code>-prof-func-groups</code></td>
</tr>
<tr>
<td><code>-fwrtable-strings</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-i-dynamic</code></td>
<td><code>-shared-intel</code></td>
</tr>
<tr>
<td><code>-i-static</code></td>
<td><code>-static-intel</code></td>
</tr>
<tr>
<td><code>-inline-debug-info</code></td>
<td><code>-debug</code></td>
</tr>
<tr>
<td><code>-IPF-flt-eval-method0</code></td>
<td><code>-fp-model source</code></td>
</tr>
<tr>
<td><code>-IPF-fltacc</code></td>
<td><code>-fp-model precise</code></td>
</tr>
<tr>
<td><code>-no-IPF-fltacc</code></td>
<td><code>-fp-model fast</code></td>
</tr>
<tr>
<td><code>-IPF-fma</code></td>
<td><code>-fma</code></td>
</tr>
<tr>
<td><code>-IPF-fp-relaxed</code></td>
<td><code>-fp-relaxed</code></td>
</tr>
<tr>
<td><code>-Kcpp</code></td>
<td><code>-x c++</code></td>
</tr>
<tr>
<td><code>-march=pentiumii</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-march=pentiumiii</code></td>
<td><code>-march=pentium3</code></td>
</tr>
<tr>
<td>Linux* OS and Mac OS* X Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td><code>-mcu</code></td>
<td><code>-mtune</code></td>
</tr>
<tr>
<td><code>-mp</code></td>
<td><code>-fp-model</code></td>
</tr>
<tr>
<td><code>-Ob</code></td>
<td><code>-inline-level</code></td>
</tr>
<tr>
<td><code>-openmp-lib legacy</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-openmpP</code></td>
<td><code>-openmp</code></td>
</tr>
<tr>
<td><code>-openmpS</code></td>
<td><code>-openmp-stubs</code></td>
</tr>
<tr>
<td><code>-prefetch</code></td>
<td><code>-opt-prefetch</code></td>
</tr>
<tr>
<td><code>-prof-genx</code></td>
<td><code>-prof-gen=srcpos</code></td>
</tr>
<tr>
<td><code>-use-asm</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-use-pch</code></td>
<td><code>-pch-use</code></td>
</tr>
<tr>
<td><code>-wd</code></td>
<td><code>-diag-disable</code></td>
</tr>
<tr>
<td><code>-we</code></td>
<td><code>-diag-error</code></td>
</tr>
<tr>
<td><code>-wn</code></td>
<td><code>-diag-error-limit</code></td>
</tr>
<tr>
<td><code>-wo</code></td>
<td><code>-diag-once id[,id,...]</code></td>
</tr>
<tr>
<td><code>-wr</code></td>
<td><code>-diag-remark</code></td>
</tr>
<tr>
<td><code>-ww</code></td>
<td><code>-diag-warning</code></td>
</tr>
<tr>
<td><code>-xK</code></td>
<td><code>-mia32</code></td>
</tr>
<tr>
<td><code>-xN</code></td>
<td><code>Linux* OS: -xSSE2&lt;br&gt;Mac OS* X: None</code></td>
</tr>
<tr>
<td><code>-xO</code></td>
<td><code>-msse3</code></td>
</tr>
</tbody>
</table>
## Compiler Options

<table>
<thead>
<tr>
<th>Linux* OS and Mac OS* X Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-xP</code></td>
<td>Linux* OS: <code>-xSSE3</code></td>
</tr>
<tr>
<td></td>
<td>Mac OS* X on IA-32 architecture: <code>-xSSE3</code></td>
</tr>
<tr>
<td></td>
<td>Mac OS* X on Intel® 64 architecture: None</td>
</tr>
<tr>
<td><code>-xS</code></td>
<td><code>-xSSE4.1</code></td>
</tr>
<tr>
<td><code>-xT</code></td>
<td>Linux* OS: <code>-xSSSE3</code></td>
</tr>
<tr>
<td></td>
<td>Mac OS* X on IA-32 architecture: None</td>
</tr>
<tr>
<td></td>
<td>Mac OS* X on Intel® 64 architecture: <code>-xSSSE3</code></td>
</tr>
<tr>
<td><code>-xW</code></td>
<td><code>-msse2</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Windows* OS Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/Fm</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/Fr</code></td>
<td><code>/FR</code></td>
</tr>
<tr>
<td><code>/G5</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/G6 (or /GB)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/G7</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/Ge</code></td>
<td><code>/Gs0</code></td>
</tr>
<tr>
<td><code>/Gf</code></td>
<td><code>/GF</code></td>
</tr>
<tr>
<td><code>/GX</code></td>
<td><code>/EHs</code></td>
</tr>
<tr>
<td><code>/Gy</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/GZ</code></td>
<td><code>/RTC</code></td>
</tr>
<tr>
<td><code>/H</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/ML and/MLd</code></td>
<td>None</td>
</tr>
<tr>
<td>Windows* OS Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>/Op</td>
<td>/fp</td>
</tr>
<tr>
<td>/QA-</td>
<td>None</td>
</tr>
<tr>
<td>/QaxK</td>
<td>None</td>
</tr>
<tr>
<td>/QaxN</td>
<td>/QaxSSE2</td>
</tr>
<tr>
<td>/QaxP</td>
<td>/QaxSSE3</td>
</tr>
<tr>
<td>/QaxS</td>
<td>/QaxSSE4.1</td>
</tr>
<tr>
<td>/QaxT</td>
<td>/QaxSSSE3</td>
</tr>
<tr>
<td>/QaxW</td>
<td>/arch:SSE2</td>
</tr>
<tr>
<td>/Qc99</td>
<td>/Qstd=c99</td>
</tr>
<tr>
<td>/Qdiag-&lt;type&gt; sv[n]</td>
<td>/Qdiag-&lt;type&gt; sc[n]</td>
</tr>
<tr>
<td>/Qdiag-enable:sv-include</td>
<td>/Qdiag-enable:sc-include</td>
</tr>
<tr>
<td>/Qinline-debug-info</td>
<td>None</td>
</tr>
<tr>
<td>/QIPF-flt-eval-method0</td>
<td>/fp:source</td>
</tr>
<tr>
<td>/QIPF-fltacc</td>
<td>/fp:precise</td>
</tr>
<tr>
<td>/QIPF-fltacc-</td>
<td>/fp:fast</td>
</tr>
<tr>
<td>/QIPF-fma</td>
<td>/Qfma</td>
</tr>
<tr>
<td>/QIPF-fp-relaxed</td>
<td>/Qfp-relaxed</td>
</tr>
<tr>
<td>/Qmspp</td>
<td>None</td>
</tr>
<tr>
<td>/Openmp-lib:legacy</td>
<td>None</td>
</tr>
<tr>
<td>/Qprefetch</td>
<td>/Qopt-prefetch</td>
</tr>
</tbody>
</table>
### Compiler Options

<table>
<thead>
<tr>
<th>Windows* OS Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Qprof-genx</td>
<td>/Qprof-gen=srcpos</td>
</tr>
<tr>
<td>/Quse-asm</td>
<td>None</td>
</tr>
<tr>
<td>/Qwd</td>
<td>/Qdiag-disable</td>
</tr>
<tr>
<td>/Qwe</td>
<td>/Qdiag-error</td>
</tr>
<tr>
<td>/Qwn</td>
<td>/Qdiag-error-limit</td>
</tr>
<tr>
<td>/Qwo</td>
<td>/Qdiag-once id[,id,...]</td>
</tr>
<tr>
<td>/Qwr</td>
<td>/Qdiag-remark</td>
</tr>
<tr>
<td>/Qww</td>
<td>/Qdiag-warning</td>
</tr>
<tr>
<td>/QxK</td>
<td>None</td>
</tr>
<tr>
<td>/QxN</td>
<td>/QxSSE2</td>
</tr>
<tr>
<td>/QxO</td>
<td>/arch:SSE3</td>
</tr>
<tr>
<td>/QxP</td>
<td>/QxSSE3</td>
</tr>
<tr>
<td>/QxS</td>
<td>/QxSSE4.1</td>
</tr>
<tr>
<td>/QxT</td>
<td>/QxSSSE3</td>
</tr>
<tr>
<td>/QxW</td>
<td>/arch:SSE2</td>
</tr>
<tr>
<td>/Zd</td>
<td>/debug:minimal</td>
</tr>
<tr>
<td>/Ze</td>
<td>None</td>
</tr>
</tbody>
</table>

Deprecated options are not limited to this list.
**Removed Options**

Some compiler options are no longer supported and have been removed. If you use one of these options, the compiler issues a warning, ignores the option, and then proceeds with compilation.

This version of the compiler no longer supports the following compiler options:

<table>
<thead>
<tr>
<th>Linux* OS and Mac OS* X Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0f_check</td>
<td>None</td>
</tr>
<tr>
<td>-axB</td>
<td>-axSSE2</td>
</tr>
<tr>
<td>-axi</td>
<td>None</td>
</tr>
<tr>
<td>-axM</td>
<td>None</td>
</tr>
<tr>
<td>-cxxlib-gcc[=dir]</td>
<td>-cxxlib[=dir]</td>
</tr>
<tr>
<td>-cxxlib-icc</td>
<td>None</td>
</tr>
<tr>
<td>-F</td>
<td>-P</td>
</tr>
<tr>
<td>-fdiv_check</td>
<td>None</td>
</tr>
<tr>
<td>-fp</td>
<td>-fno-omit-frame-pointer</td>
</tr>
<tr>
<td>-fpstkchk</td>
<td>-fp-stack-check</td>
</tr>
<tr>
<td>-IPF-fp-speculation</td>
<td>-fp-speculation</td>
</tr>
<tr>
<td>-ipo-obj (and -ipo_obj)</td>
<td>None</td>
</tr>
<tr>
<td>-Knopic, -KNOPIC</td>
<td>-fpic</td>
</tr>
<tr>
<td>-Kpic, -KPIC</td>
<td>-fpic</td>
</tr>
<tr>
<td>-mtune=itanium</td>
<td>None</td>
</tr>
<tr>
<td>-no-c99</td>
<td>-std=c89</td>
</tr>
<tr>
<td>Linux* OS and Mac OS* X Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>-no-cpprt</td>
<td>-no-cxxlib</td>
</tr>
<tr>
<td>-nobss-init</td>
<td>-no-bss-init</td>
</tr>
<tr>
<td>-norestrict</td>
<td>-no-restrict</td>
</tr>
<tr>
<td>-opt-report-level</td>
<td>-opt-report</td>
</tr>
<tr>
<td>-prof-format-32</td>
<td>None</td>
</tr>
<tr>
<td>-prof-gen-sampling</td>
<td>None</td>
</tr>
<tr>
<td>-qp</td>
<td>-p</td>
</tr>
<tr>
<td>-shared-libcxa</td>
<td>-shared-libgcc</td>
</tr>
<tr>
<td>-ssp</td>
<td>None</td>
</tr>
<tr>
<td>-static-libcxa</td>
<td>-static-libgcc</td>
</tr>
<tr>
<td>-syntax</td>
<td>-fsyntax-only</td>
</tr>
<tr>
<td>-tpp1</td>
<td>None</td>
</tr>
<tr>
<td>-tpp2</td>
<td>-mtune=itanium2</td>
</tr>
<tr>
<td>-tpp5</td>
<td>None</td>
</tr>
<tr>
<td>-tpp6</td>
<td>None</td>
</tr>
<tr>
<td>-tpp7</td>
<td>-mtune=pentium4</td>
</tr>
<tr>
<td>-use-pch</td>
<td>-pch-use</td>
</tr>
<tr>
<td>-xB</td>
<td>-xSSE2</td>
</tr>
<tr>
<td>-xi</td>
<td>None</td>
</tr>
<tr>
<td>-xM</td>
<td>None</td>
</tr>
<tr>
<td>Windows* OS Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>/QaxB</td>
<td>/QaxSSE2</td>
</tr>
<tr>
<td>/Qaxi</td>
<td>None</td>
</tr>
<tr>
<td>/QaxM</td>
<td>None</td>
</tr>
<tr>
<td>/Qfpstkchk</td>
<td>/Qfp-stack-check</td>
</tr>
<tr>
<td>/QIfdiv</td>
<td>None</td>
</tr>
<tr>
<td>/QIPIF-fp-speculation</td>
<td>/Qfp-speculation</td>
</tr>
<tr>
<td>/Qipo-obj (and /Qipo_obj)</td>
<td>None</td>
</tr>
<tr>
<td>/QI0f</td>
<td>None</td>
</tr>
<tr>
<td>/Qopt-report-level</td>
<td>/Qopt-report</td>
</tr>
<tr>
<td>/Qprof-format-32</td>
<td>None</td>
</tr>
<tr>
<td>/Qprof-gen-sampling</td>
<td>None</td>
</tr>
<tr>
<td>/Qssp</td>
<td>None</td>
</tr>
<tr>
<td>/Qvc6</td>
<td>None</td>
</tr>
<tr>
<td>/Qvc7</td>
<td>None</td>
</tr>
<tr>
<td>/QxB</td>
<td>/QxsSSE2</td>
</tr>
<tr>
<td>/Qxi</td>
<td>None</td>
</tr>
<tr>
<td>/QxM</td>
<td>None</td>
</tr>
</tbody>
</table>

Removed options are not limited to these lists.
Compiler Option Descriptions and General Rules

This section describes all the current Linux* OS, Mac OS* X, and Windows* OS compiler options in alphabetical order.

Option Descriptions

Each option description contains the following information:

- A short description of the option.
- IDE Equivalent
  This shows information related to the integrated development environment (IDE) Property Pages on Windows*, Linux*, and Mac OS* X systems. It shows on which Property Page the option appears, and under what category it’s listed. The Windows IDE is Microsoft* Visual Studio* .NET; the Linux IDE is Eclipse*; the Mac OS X IDE is Xcode*. If the option has no IDE equivalent, it will specify "None".
- Architectures
  This shows the architectures where the option is valid. Possible architectures are:
  - IA-32 architecture
  - Intel® 64 architecture
  - IA-64 architecture
- Syntax
  This shows the syntax on Linux and Mac OS X systems and the syntax on Windows systems. If the option has no syntax on one of these systems, that is, the option is not valid on a particular system, it will specify "None".
- Arguments
  This shows any arguments (parameters) that are related to the option. If the option has no arguments, it will specify "None".
- Default
  This shows the default setting for the option.
- Description
This shows the full description of the option. It may also include further information on any applicable arguments.

- **Alternate Options**
  These are options that are synonyms of the described option. If there are no alternate options, it will specify "None".
  Many options have an older spelling where underscores ("_") instead of hyphens ("-") connect the main option names. The older spelling is a valid alternate option name.

Some option descriptions may also have the following:

- **Example**
  This shows a short example that includes the option

- **See Also**
  This shows where you can get further information on the option or related options.

### General Rules for Compiler Options

You cannot combine options with a single dash (Linux OS and Mac OS X) or slash (Windows OS). For example:

- **On Linux and Mac OS X systems**: This is incorrect: `-wc`; this is correct: `−w −c`
- **On Windows systems**: This is incorrect: `/wc`; this is correct: `/w /c`

All compiler options are case sensitive. Some options have different meanings depending on their case; for example, option "c" prevents linking, but option "C" places comments in preprocessed source output.

Options specified on the command line apply to all files named on the command line.

Options can take arguments in the form of file names, strings, letters, or numbers. If a string includes spaces, the string must be enclosed in quotation marks. For example:

- **On Linux and Mac OS X systems**, `-dynamic-linker mylink (file name) or -Umacro3 (string)`
- **On Windows systems**, `/Famyfile.s (file name) or /V"version 5.0" (string)`

Compiler options can appear in any order.

On Windows systems, all compiler options must precede `/link` options, if any, on the command line.

Unless you specify certain options, the command line will both compile and link the files you specify.
You can abbreviate some option names, entering as many characters as are needed to uniquely identify the option.

Certain options accept one or more keyword arguments following the option name. For example, the arch option accepts several keywords.

To specify multiple keywords, you typically specify the option multiple times. However, there are exceptions; for example, the following are valid: –axNB (Linux OS) or /QaxNB (Windows OS).

Compiler options remain in effect for the whole compilation unless overridden by a compiler #pragma.

To disable an option, specify the negative form of the option.

On Windows systems, you can also disable one or more options by specifying option /Od last on the command line.

**NOTE.** On Windows systems, the /Od option is part of a mutually-exclusive group of options that includes /Od, /O1, /O2, /O3, and /Ox. The last of any of these options specified on the command line will override the previous options from this group.

If there are enabling and disabling versions of an option on the command line, the last one on the command line takes precedence.

### Lists and Functional Groupings of Compiler Options

To see a list of all the compiler options, specify option help on the command line.

To see functional groupings of compiler options, specify a functional category for option help. For example, to see a list of options that affect diagnostic messages displayed by the compiler, enter one of the following commands:

- `-help diagnostics` ! Linux and Mac OS X systems
- `/help diagnostics` ! Windows systems

For details on the categories you can specify, see help.
A, QA

Specifies an identifier for an assertion.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-A name[(value)]

Windows:
/QA name[(value)]

Arguments

<table>
<thead>
<tr>
<th>name</th>
<th>Is the identifier for the assertion.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Is an optional value for the assertion. If a value is specified, it must be within quotes, including the parentheses delimiting it.</td>
</tr>
</tbody>
</table>

Default

OFF Assertions have no identifiers or symbol names.

Description

This option specifies an identifier (symbol name) for an assertion. It is equivalent to an #assert preprocessing directive.

Note that this option is not the positive form of the C++ /QA- option.

Alternate Options
None
Example

To make an assertion for the identifier fruit with the associated values orange and banana use the following command.

On Windows* systems:
icl /QA"fruit(orange,banana)" prog1.cpp

On Linux* and Mac OS* X systems:
icpc -A"fruit(orange,banana)" prog1.cpp

A-, QA-
Disables all predefined macros. This is a deprecated option.

IDE Equivalent

Windows: None
Linux: Preprocessor > Undefine All Preprocessor Definitions
Mac OS X: Preprocessor > Undefine All Preprocessor Definitions

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-A-

Windows:
/QA-

Arguments

None

Default

OFF Predefined macros remain enabled.
**Description**
This option disables all predefined macros. It causes all predefined macros and assertions to become inactive.

Note that this option is not the negative form of the C++ /QA option.

**Alternate Options**
None

**fargument-alias, Qalias-args**
*Determines whether function arguments can alias each other.*

**IDE Equivalent**
Windows: None
Linux: Data > Enable Argument Aliasing
Mac OS X: Data > Enable Argument Aliasing

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
-fargument-alias
-fargument-noalias

**Windows:**
/Qalias-args
/Qalias-args-

**Arguments**
None
Default

-fargument-alias or /Qalias-args     Function arguments can alias each other and can alias global storage.

Description

This option determines whether function arguments can alias each other. If you specify -fargument-noalias or /Qalias-args-, function arguments cannot alias each other, but they can alias global storage.

On Linux and Mac OS X systems, you can also disable aliasing for global storage, by specifying option -fargument-noalias-global.

Alternate Options

Linux and Mac OS X: -[no-]alias-args (this is a deprecated option)

Windows: None

See Also

- alias-const, Qalias-const

Determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const.

IDE Equivalent

Windows: None

Linux: Data > Assume Restrict Semantics for Const

Mac OS X: Data > Assume Restrict Semantics for Const

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
- alias-const
- no-alias-const

Windows:
/Qalias-const
/Qalias-const-

Arguments
None

Default
-no-alias-const
or /Qalias-const-

The compiler uses standard C/C++ rules for the interpretation of const.

Description

This option determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const. It implies an additional attribute for const.

This functionality complies with the input/output buffer rule, which assumes that input and output buffer arguments do not overlap. This option allows the compiler to do some additional optimizations with those parameters.

In C99, you can also get the same result if you additionally declare your pointer parameters with the restrict keyword.

Alternate Options
None
**align**

*Determine whether variables and arrays are naturally aligned.*

---

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-align`
- `-noalign`

**Windows:**

None

**Arguments**

None

**Default**

OFF

Variables and arrays are aligned according to the gcc model, which means they are aligned to 4-byte boundaries.

**Description**

This option determines whether variables and arrays are naturally aligned. Option `-align` forces the following natural alignment:

<table>
<thead>
<tr>
<th>Type</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>8 bytes</td>
</tr>
<tr>
<td>long long</td>
<td>8 bytes</td>
</tr>
<tr>
<td>long double</td>
<td>16 bytes</td>
</tr>
</tbody>
</table>
If you are not interacting with system libraries or other libraries that are compiled without `-align`, this option can improve performance by reducing misaligned accesses.

**CAUTION.** If you are interacting with system libraries or other libraries that are compiled without `-align`, your application may not perform as expected.

**Alternate Options**

None

**ansi**

*Enables language compatibility with the gcc option* `-ansi`.

**IDE Equivalent**

Windows: None

Linux: *Language > ANSI Conformance*

Mac OS X: *Language > C ANSI Conformance*

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

`-ansi`

Windows:

None

**Arguments**

None

**Default**

OFF

GNU C++ is more strongly supported than ANSI C.
**Description**

This option enables language compatibility with the gcc option `-ansi` and provides the same level of ANSI standard conformance as that option.

This option sets option `fmath-errno`.

If you want strict ANSI conformance, use the `-strict-ansi` option.

**Alternate Options**

None

**ansi-alias, Qansi-alias**

*Enable use of ANSI aliasing rules in optimizations.*

**IDE Equivalent**

Windows: None

Linux: **Language > Enable Use of ANSI Aliasing Rules in Optimizations**

Mac OS X: **Language > Enable ANSI Aliasing**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-ansi-alias`
- `-no-ansi-alias`

Windows:

- `/Qansi-alias`
- `/Qansi-alias-`

**Arguments**

None
Default
-no-ansi-alias or /Qan-ali consort use of ANSI aliasing rules in optimizations.

Description
This option tells the compiler to assume that the program adheres to ISO C Standard aliasability rules.

If your program adheres to these rules, then this option allows the compiler to optimize more aggressively. If it doesn't adhere to these rules, then it can cause the compiler to generate incorrect code.

Alternate Options
None

Ap64
Enables 64-bit pointers.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux and Mac OS X:
None

Windows:
/An64

Arguments
None

Default
ON 64-bit pointers are enabled.
Description
This option enables 64-bit pointers.

Alternate Options
None

arch
*Tells the compiler to generate optimized code specialized for the processor that executes your program.*

IDE Equivalent
Windows: **Code Generation > Enable Enhanced Instruction Set**
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64 architectures

Syntax
- **Linux and Mac OS X:** None
- **Windows:**
  ```
  /arch:processor
  ```

Arguments
*processor* is the processor type. Possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA32</td>
<td>Generates code that will run on any Pentium or later processor. Disables any default extended instruction settings, and any previously set extended instruction settings. This value is only available on IA-32 architecture.</td>
</tr>
<tr>
<td>SSE</td>
<td>This is the same as specifying IA32.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>SSE2</td>
<td>Generates code for Intel® Streaming SIMD Extensions 2 (Intel® SSE2).</td>
</tr>
<tr>
<td>SSE3</td>
<td>Generates code for Intel® Streaming SIMD Extensions 3 (Intel® SSE3).</td>
</tr>
<tr>
<td>SSSE3</td>
<td>Generates code for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3).</td>
</tr>
<tr>
<td>SSE4.1</td>
<td>Generates code for Intel® Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators.</td>
</tr>
</tbody>
</table>

**Default**

SSE2

The compiler generates code for Intel® Streaming SIMD Extensions 2 (Intel® SSE2).

**Description**

This option tells the compiler to generate optimized code specialized for the processor that executes your program.

Code generated with the values IA32, SSE, SSE2, or SSE3 should execute on any compatible non-Intel processor with support for the corresponding instruction set.

Options /arch and /Qx are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning.

**Alternate Options**

Linux and Mac OS X: –m

Windows: None

**See Also**

- x, Qx
- ax, Qax
- m
As

Determines the size of virtual address space.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux and Mac OS X:

None

Windows:

/Asi

Arguments

Is the virtual address space. Possible values are 32 or 64.

Default

/As64

The virtual address space is 16 exabytes.

Description

This option determines the size of virtual address space.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/As32</td>
<td>Sets the virtual address space to 4 gigabytes (GB).</td>
</tr>
<tr>
<td>/As64</td>
<td>Sets the virtual address space to 16 exabytes (EB).</td>
</tr>
</tbody>
</table>

Alternate Options

None
auto-ilp32, Qauto-ilp32
Instruction the compiler to analyze the program to determine if there are 64-bit pointers which can be safely shrunk into 32-bit pointers.

IDE Equivalent
None

Architectures
Intel® 64 architecture, IA-64 architecture

Syntax
Linux and Mac OS X:
- auto-ilp32
Windows:
/Qauto-ilp32

Arguments
None

Default
OFF The optimization is not attempted.

Description
This option instructs the compiler to analyze and transform the program so that 64-bit pointers are shrunk to 32-bit pointers, and 64-bit longs (on Linux) are shrunk into 32-bit longs wherever it is legal and safe to do so. In order for this option to be effective the compiler must be able to optimize using the -ipo/-Qipo option and must be able to analyze all library/external calls the program makes.

This option requires that the size of the program executable never exceeds $2^{32}$ bytes and all data values can be represented within 32 bits. If the program can run correctly in a 32-bit system, these requirements are implicitly satisfied. If the program violates these size restrictions, unpredictable behavior might occur.
Alternate Options
None

ax, Qax
*Tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit.*

IDE Equivalent
Windows: Code Generation > Add Processor-Optimized Code Path
Linux: Code Generation > Add Processor-Optimized Code Path
Mac OS X: Code Generation > Add Processor-Optimized Code Path

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-ax processor

Windows:
/Qax processor

Arguments
processor
Indicates the processor for which code is generated. The following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE). Possible values are:

SSE4.2 Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media
Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.

**SSE4.1** Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture. This replaces value S, which is deprecated.

**SSSE3** Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. For Mac OS* X systems, this value is only supported on Intel® 64 architecture. This replaces value T, which is deprecated.

**SSE3** Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. For Mac OS* X systems, this value is only supported on IA-32 architecture. This replaces value P, which is deprecated.

**SSE2** Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2. This value is not available on Mac OS* X systems. This replaces value N, which is deprecated.
Default

OFF

No auto-dispatch code is generated. Processor-specific code is generated and is controlled by the setting of compiler option `-m` (Linux), compiler option `/arch` (Windows), or compiler option `-x` (Mac OS* X).

Description

This option tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit. It also generates a baseline code path. The baseline code is usually slower than the specialized code.

The baseline code path is determined by the architecture specified by the `-x` (Linux and Mac OS X) or `/Qx` (Windows) option. While there are defaults for the `-x` or `/Qx` option that depend on the operating system being used, you can specify an architecture for the baseline code that is higher or lower than the default. The specified architecture becomes the effective minimum architecture for the baseline code path.

If you specify both the `-ax` and `-x` options (Linux and Mac OS X) or the `/Qax` and `/Qx` options (Windows), the baseline code will only execute on processors compatible with the processor type specified by the `-x` or `/Qx` option.

This option tells the compiler to find opportunities to generate separate versions of functions that take advantage of features of the specified Intel® processor.

If the compiler finds such an opportunity, it first checks whether generating a processor-specific version of a function is likely to result in a performance gain. If this is the case, the compiler generates both a processor-specific version of a function and a baseline version of the function. At run time, one of the versions is chosen to execute, depending on the Intel processor in use. In this way, the program can benefit from performance gains on more advanced Intel processors, while still working properly on older processors.

You can use more than one of the processor values by combining them. For example, you can specify `-axSSE4.1,SSSE3` (Linux and Mac OS X) or `/QaxSSE4.1,SSSE3` (Windows). You cannot combine the old style, deprecated options and the new options. For example, you cannot specify `-axSSE4.1,T` (Linux and Mac OS X) or `/QaxSSE4.1,T` (Windows).

Previous values W and K are deprecated. The details on replacements are as follows:

- Mac OS X systems: On these systems, there is no exact replacement for W or K. You can upgrade to the default option `-msse3` (IA-32 architecture) or option `-mssse3` (Intel® 64 architecture).
Windows and Linux systems: The replacement for W is -msse2 (Linux) or /arch:SSE2 (Windows). There is no exact replacement for K. However, on Windows systems, /QaxK is interpreted as /arch:IA32; on Linux systems, -axK is interpreted as -mia32. You can also do one of the following:

- Upgrade to option -msse2 (Linux) or option /arch:SSE2 (Windows). This will produce one code path that is specialized for Intel® SSE2. It will not run on earlier processors
- Specify the two option combination -mia32 -axSSE2 (Linux) or /arch:IA32 /QaxSSE2 (Windows). This combination will produce an executable that runs on any processor with IA-32 architecture but with an additional specialized Intel® SSE2 code path.

The -ax and /Qax options enable additional optimizations not enabled with option -m or option /arch.

**Alternate Options**
None

**See Also**
- x, Qx
- m
- arch
- Targeting IA-32 and Intel 64 Architecture Processors Manually

**B**
Specifies a directory that can be used to find include files, libraries, and executables.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
-Bdir

Windows: None

Arguments
dir Is the directory to be used. If necessary, the compiler adds a directory separator character at the end of dir.

Default
OFF The compiler looks for files in the directories specified in your PATH environment variable.

Description
This option specifies a directory that can be used to find include files, libraries, and executables. The compiler uses dir as a prefix.

For include files, the dir is converted to -I/dir/include. This command is added to the front of the includes passed to the preprocessor.

For libraries, the dir is converted to -L/dir. This command is added to the front of the standard -L inclusions before system libraries are added.

For executables, if dir contains the name of a tool, such as ld or as, the compiler will use it instead of those found in the default directories.

The compiler looks for include files in dir/include while library files are looked for in dir.

Another way to get the behavior of this option is to use the environment variable GCC_EXEC_PREFIX.

Alternate Options
None
**Bdynamic**

*Enables dynamic linking of libraries at run time.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

- **Linux:**
  - `-Bdynamic`

- **Mac OS X:**
  - None

- **Windows:**
  - None

**Arguments**
None

**Default**
OFF  Limited dynamic linking occurs.

**Description**

This option enables dynamic linking of libraries at run time. Smaller executables are created than with static linking.

This option is placed in the linker command line corresponding to its location on the user command line. It controls the linking behavior of any library that is passed using the command line.

All libraries on the command line following option `-Bdynamic` are linked dynamically until the end of the command line or until a `-Bstatic` option is encountered. The `-Bstatic` option enables static linking of libraries.
**Alternate Options**

None

**See Also**

- Bstatic

**bigobj**

*Increases the number of sections that an object file can contain.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/bigobj

**Arguments**

None

**Default**

OFF

An object file can hold up to 65,536 (2**16) addressable sections.

**Description**

This option increases the number of sections that an object file can contain. It increases the address capacity to 4,294,967,296(2**32).
An .obj file produced with this option can only be effectively passed to a linker that shipped in Microsoft Visual C++* 2005 or later. Linkers shipped with earlier versions of the product cannot read .obj files of this size.

This option may be helpful for .obj files that can hold more sections, such as machine generated code or code that makes heavy use of template libraries.

**Alternate Options**

None

**Bstatic**

*Enables static linking of a user's library.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**

-Bstatic

**Mac OS X:**

None

**Windows:**

None

**Arguments**

None

**Default**

OFF Default static linking occurs.

**Description**

This option enables static linking of a user's library.
This option is placed in the linker command line corresponding to its location on the user command line. It controls the linking behavior of any library that is passed using the command line.

All libraries on the command line following option `-Bstatic` are linked statically until the end of the command line or until a `-Bdynamic` option is encountered. The `-Bdynamic` option enables dynamic linking of libraries.

**Alternate Options**

None

**See Also**

- `-Bdynamic`

**c**

Prevents linking.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-c`

Windows:

/`c`

**Arguments**

None
**Default**
OFF Linking is performed.

**Description**
This option prevents linking. Compilation stops after the object file is generated.

The compiler generates an object file for each C or C++ source file or preprocessed source file. It also takes an assembler file and invokes the assembler to generate an object file.

**Alternate Options**
None

**C**
Places comments in preprocessed source output.

**IDE Equivalent**
Windows: Preprocessor > Keep Comments
Linux: None
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

`-C`

Windows:

`/C`

**Arguments**
None

**Default**
OFF No comments are placed in preprocessed source output.
**Description**
This option places (or preserves) comments in preprocessed source output. Comments following preprocessing directives, however, are not preserved.

**Alternate Options**
None

**See Also**
- Building Applications: About Preprocessor Options

**c99, Qc99**
*Determines whether C99 support is enabled for C programs. This is a deprecated option.*

**IDE Equivalent**
- Windows: **Language > Enable C99 Support**
- Linux: **Language > Disable C99 Support**
- Mac OS X: **Language > Disable C99 Support**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
- Linux and Mac OS X:
  - `-c99`
  - `-no-c99`
- Windows:
  - `/Qc99`
  - `/Qc99-`

**Arguments**
None
Default

-no-c99
or /Qc99-

C99 support is disabled for C programs on Linux.

Description

This option determines whether C99 support is enabled for C programs. One of the features enabled by -c99 (Linux and Mac OS X) or /Qc99 (Windows), restricted pointers, is available by using option restrict. For more information, see restrict.

Alternate Options

-std, /Qstd

See Also

• restrict, /Qrestrict

check-uninit

Determines whether checking occurs for uninitialized variables.

IDE Equivalent

Windows: None

Linux: Runtime > Check Uninitialized Variables

Mac OS X: Runtime > Check Uninitialized Variables

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-check-uninit
-no-check-uninit
Windows:
None

Arguments
None

Default
-no-check-uninit

Description
Enables run-time checking for uninitialized variables. If a variable is read before it is written, a run-time error routine will be called. Run-time checking of undefined variables is only implemented on local, scalar variables. It is not implemented on dynamically allocated variables, extern variables or static variables. It is not implemented on structs, classes, unions or arrays.

Alternate Options
None

complex-limited-range, Qcomplex-limited-range
Determined whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled.

IDE Equivalent
Windows: Floating Point > Limit COMPLEX Range
Linux: None
Mac OS X: Floating Point > Limit COMPLEX Range

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-complex-limited-range
-no-complex-limited-range

Windows:
/Qcomplex-limited-range
/Qcomplex-limited-range-

Arguments
None

Default
-no-complex-limited-range  or/Qcomplex-limited-range-

Description
This option determines whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled.

When the option is enabled, this can cause performance improvements in programs that use a lot of COMPLEX arithmetic. However, values at the extremes of the exponent range may not compute correctly.

Alternate Options
None

cxxlib
Determines whether the compile links using the C++ run-time libraries and header files provided by gcc.

IDE Equivalent
Windows: None
Linux: Preprocessor > gcc Compatibility Options
Mac OS X: Preprocessor > gcc Compatibility Options

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
- cxxlib[=dir]
- cxxlib-nostd
- no-cxxlib

Windows:
None

Arguments

dir Is an optional top-level location for the gcc binaries and libraries.

Default

C++: -cxxlib
C: -no-cxxlib

For C++, the compiler uses the run-time libraries and headers provided by gcc.
For C, the compiler uses the default run-time libraries and headers and does not link to any additional C++ run-time libraries and headers. However, if you specify compiler option -std=gnu++98, the default is -cxxlib.

Description

This option determines whether the compile links using the C++ run-time libraries and header files provided by gcc.

Option -cxxlib-nostd prevents the compiler from linking with the standard C++ library.

Alternate Options

None

See Also

• Building Applications: Options for Interoperability
D
Defines a macro name that can be associated with an optional value.

**IDE Equivalent**
Windows: **Preprocessor > Preprocessor Definitions**
Linux: **Preprocessor > Preprocessor Definitions**
Mac OS X: **Preprocessor > Preprocessor Definitions**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
-D name[=value]

Windows:
/D name[=value]

**Arguments**
- **name**: Is the name of the macro.
- **value**: Is an optional integer or an optional character string delimited by double quotes; for example, Dname=string.

**Default**
OFF Only default symbols or macros are defined.

**Description**
Defines a macro name that can be associated with an optional value.
This option is equivalent to a #define preprocessor directive.
If a value is not specified, name is defined as "1".
If you specify noD, all preprocessor definitions apply only to fpp and not to Intel® Fortran conditional compilation directives. To use this option, you must also specify option fpp.

**CAUTION.** On Linux and Mac OS X systems, if you are not specifying a value, do not use D for name, because it will conflict with the -DD option.

**Alternate Options**

None

**See Also**

- Building Applications: Predefined Preprocessor Symbols

**dD, QdD**

*Same as -dM, but outputs #define directives in preprocessed source.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- dD

**Windows:**

/ QdD

**Arguments**

None
Default
OFF

The compiler does not output `#define` directives.

Description
Same as `-dM`, but outputs `#define` directives in preprocessed source. To use this option, you must also specify the `E` option.

Alternate Options
None

default (Linux* OS and Mac OS* X)

Enables or disables generation of debugging information.

IDE Equivalent
Windows:
Linux: `Debug > Enable Parallel Debug Checks` (-debug parallel)

`Debug > Enable Expanded Line Number Information` (-debug expr-source-pos)

Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- `debug [keyword]`

Windows:
None

Arguments

`keyword` Is the type of debugging information to be generated.
Possible values are:
Disables generation of debugging information.

full or all
Generates complete debugging information.

minimal
Generates line number information for debugging.

[no]emit_column
Determines whether the compiler generates column number information for debugging.

[no]expr-source-pos
Determines whether the compiler generates source position information at the expression level of granularity.

[no]inline-debug-info
Determines whether the compiler generates enhanced debug information for inlined code.

[no]semantic-stepping
Determines whether the compiler generates enhanced debug information useful for breakpoints and stepping.

[no]variable-locations
Determines whether the compiler generates enhanced debug information useful in finding scalar local variables.

extended
Sets keyword values semantic-stepping and variable-locations.

[no]parallel
(\text{Linux only}; \text{i32, i64em})
Determines whether the compiler generates parallel debug code instrumentations useful for thread data sharing and reentrant call detection.

For information on the non-default settings for these keywords, see the Description section.

\textbf{Default}

\texttt{-debug none}
No debugging information is generated.

\textbf{Description}
This option enables or disables generation of debugging information.
Note that if you turn debugging on, optimization is turned off.

**Keywords** semantic-stepping, inline-debug-info, variable-locations, and extended can be used in combination with each other. If conflicting keywords are used in combination, the last one specified on the command line has precedence.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-debug none</td>
<td>Disables generation of debugging information.</td>
</tr>
<tr>
<td>-debug full or -debug all</td>
<td>Generates complete debugging information. It is the same as specifying -debug with no keyword.</td>
</tr>
<tr>
<td>-debug minimal</td>
<td>Generates line number information for debugging.</td>
</tr>
<tr>
<td>-debug emit_column</td>
<td>Generates column number information for debugging.</td>
</tr>
<tr>
<td>-debug expr-source-pos</td>
<td>Generates source position information at the statement level of granularity.</td>
</tr>
<tr>
<td>-debug inline-debug-info</td>
<td>Generates enhanced debug information for inlined code. It provides more information to debuggers for function call traceback.</td>
</tr>
<tr>
<td>-debug semantic-stepping</td>
<td>Generates enhanced debug information useful for breakpoints and stepping. It tells the debugger to stop only at machine instructions that achieve the final effect of a source statement. For example, in the case of an assignment statement, this might be a store instruction that assigns a value to a program variable; for a function call, it might be the machine instruction that executes the call. Other instructions generated for those source statements are not displayed during stepping. This option has no impact unless optimizations have also been enabled.</td>
</tr>
<tr>
<td>-debug variable-locations</td>
<td>Generates enhanced debug information useful in finding scalar local variables. It uses a feature of the Dwarf object module known as &quot;location lists&quot;.</td>
</tr>
</tbody>
</table>
**Option** | **Description**
--- | ---
This feature allows the run-time locations of local scalar variables to be specified more accurately; that is, whether, at a given position in the code, a variable value is found in memory or a machine register.

*-debug extended* | Sets keyword values *semantic-stepping* and *variable-locations*. It also tells the compiler to include column numbers in the line information.

*-debug parallel*  
(Windows only) | Generates parallel debug code instrumentations needed for the thread data sharing and reentrant call detection of the Intel Parallel Debugger Extension. This option is only available on IA-32 and Intel® 64 architectures.

On Linux* systems, debuggers read debug information from executable images. As a result, information is written to object files and then added to the executable by the linker. On Mac OS* X systems, debuggers read debug information from object files. As a result, the executables don't contain any debug information. Therefore, if you want to be able to debug on these systems, you must retain the object files.

**Alternate Options**

For *debug full, -debug all, or -debug*  
Linux and Mac OS X: `-g`  
Windows: `/debug:full, /debug:all, or /debug`

For *-debug inline-debug-info*  
Linux and Mac OS X: `-inline-debug-info` (this is a deprecated option)  
Windows: None

**debug (Windows* OS)**

*Enables or disables generation of debugging information.*

**IDE Equivalent**

Windows: *Debug > Enable Parallel Debug Checks* (/debug:parallel)

*Debug > Enable Expanded Line Number Information* (/debug:expr-source-pos)

Linux: None

Mac OS X: None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/debug[:keyword]

Arguments

- **keyword**
  - Is the type of debugging information to be generated. Possible values are:
    - *none* Generates no symbol table information.
    - *full* or *all* Generates complete debugging information.
    - *minimal* Generates line numbers and minimal debugging information.
    - *partial* Deprecated. Generates global symbol table information needed for linking.

- **[no]expr-source-pos**
  - Determines whether the compiler generates source position information at the expression level of granularity.

- **[no]parallel** *(i32, i64em)*
  - Determines whether the compiler generates parallel debug code instrumentations useful for thread data sharing and reentrant call detection. For shared data and reentrancy detection, option `/Qopenmp` must be set.

For information on the non-default settings for these keywords, see the Description section.

Default

- `/debug:none`
  - This is the default on the command line and for a release configuration in the IDE.
This is the default for a debug configuration in the IDE.

**Description**

This option enables or disables generation of debugging information. It is passed to the linker. Note that if you turn debugging on, optimization is turned off.

If conflicting keywords are used in combination, the last one specified on the command line has precedence.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/debug:none</td>
<td>Disables generation of debugging information.</td>
</tr>
<tr>
<td>/debug:full or /debug:all</td>
<td>Generates complete debugging information. It produces symbol table information needed for full symbolic debugging of unoptimized code and global symbol information needed for linking. It is the same as specifying /debug with no keyword.</td>
</tr>
<tr>
<td>/debug:minimal</td>
<td>Generates line number information for debugging. It produces global symbol information needed for linking, but not local symbol table information needed for debugging.</td>
</tr>
<tr>
<td>/debug:partial</td>
<td>Generates global symbol table information needed for linking, but not local symbol table information needed for debugging. This option is deprecated and is not available in the IDE.</td>
</tr>
<tr>
<td>/debug:expr-source-pos</td>
<td>Generates source position information at the statement level of granularity.</td>
</tr>
<tr>
<td>/debug:parallel</td>
<td>Generates parallel debug code instrumentations needed for the thread data sharing and reentrant call detection of the Intel® Parallel Debugger Extension. This option is only available on IA-32 and Intel® 64 architectures.</td>
</tr>
</tbody>
</table>
Alternate Options

For /debug:minimal  Linux and Mac OS X: None
Windows: /Zd (this is a deprecated option)

For /debug:full or /debug
Linux and Mac OS X: None
Windows: /Zi, /Z7

diag, Qdiag
Controls the display of diagnostic information.

IDE Equivalent

Windows: Diagnostics > Disable Specific Diagnostics (/Qdiag-disable:id)
Diagnostics > Level of Source Code Analysis (/Qdiag-enable[:sc1,sc2,sc3])

Linux: Compilation Diagnostics > Disable Specific Diagnostics (-diag-disable id)
Compilation Diagnostics > Level of Source Code Analysis (-diag-enable [sc1,sc2,sc3]
or -diag-disable sv)

Mac OS X: Diagnostics > Disable Specific Diagnostics (-diag-disable id)
Diagnostics > Level of Source Code Analysis (-diag-enable [sc1,sc2,sc3])

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-diag-type diag-list

Windows:
/Qdiag-type:diag-list

Arguments

type  Is an action to perform on diagnostics. Possible values are:
enable  Enables a diagnostic message or a group of messages.
disable  Enables a diagnostic message or a group of messages.

error  Tells the compiler to change diagnostics to errors.

warning  Tells the compiler to change diagnostics to warnings.

remark  Tells the compiler to change diagnostics to remarks (comments).

diag-list  Is a diagnostic group or ID value. Possible values are:

driver  Specifies diagnostic messages issued by the compiler driver.

port-linux  Specifies diagnostic messages for language features that may cause errors when porting to Linux. This diagnostic group is only available on Windows systems.

port-win  Specifies diagnostic messages for GNU extensions that may cause errors when porting to Windows. This diagnostic group is only available on Linux and Mac OS X systems.

thread  Specifies diagnostic messages that help in thread-enabling a program.

vec  Specifies diagnostic messages issued by the vectorizer.

par  Specifies diagnostic messages issued by the auto-parallelizer (parallel optimizer).

openmp  Specifies diagnostic messages issued by the OpenMP* parallelizer.

sc[n]  Specifies diagnostic messages issued by the Source Checker. n can be any of the following: 1, 2, 3. For more details on these values, see below. This value is equivalent to deprecated value sv[n].

warn  Specifies diagnostic messages that have a "warning" severity level.
Specifies diagnostic messages that have an "error" severity level.

Specifies diagnostic messages that are remarks or comments.

Specifies the CPU dispatch remarks for diagnostic messages. These remarks are enabled by default. This diagnostic group is only available on IA-32 architecture and Intel® 64 architecture.

Specifies the ID number of one or more messages. If you specify more than one message number, they must be separated by commas. There can be no intervening white space between each id.

Specifies the mnemonic name of one or more messages. If you specify more than one mnemonic name, they must be separated by commas. There can be no intervening white space between each tag.

**Default**

**OFF**

The compiler issues certain diagnostic messages by default.

**Description**

This option controls the display of diagnostic information. Diagnostic messages are output to stderr unless compiler option `-diag-file` (Linux and Mac OS X) or `/Qdiag-file` (Windows) is specified.

When `diag-list` value "warn" is used with the Source Checker (sc) diagnostics, the following behavior occurs:

- Option `-diag-enable warn` (Linux and Mac OS X) and `/Qdiag-enable:warn` (Windows) enable all Source Checker diagnostics except those that have an "error" severity level. They enable all Source Checker warnings, cautions, and remarks.
- Option `-diag-disable warn` (Linux and Mac OS X) and `/Qdiag-disable:warn` (Windows) disable all Source Checker diagnostics except those that have an "error" severity level. They suppress all Source Checker warnings, cautions, and remarks.
The following table shows more information on values you can specify for diag-list item sc.

<table>
<thead>
<tr>
<th><strong>diag-list</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Item</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>sc[n]</td>
<td>The value of (n) for Source Checker messages can be any of the following:</td>
</tr>
<tr>
<td>1</td>
<td>Produces the diagnostics with severity level set to all critical errors.</td>
</tr>
<tr>
<td>2</td>
<td>Produces the diagnostics with severity level set to all errors. This is the default if (n) is not specified.</td>
</tr>
<tr>
<td>3</td>
<td>Produces the diagnostics with severity level set to all errors and warnings.</td>
</tr>
</tbody>
</table>

To control the diagnostic information reported by the vectorizer, use the `-vec-report` (Linux and Mac OS X) or `/Qvec-report` (Windows) option.

To control the diagnostic information reported by the auto-parallelizer, use the `-par-report` (Linux and Mac OS X) or `/Qpar-report` (Windows) option.

**Alternate Options**

| **enable vec** | **Linux and Mac OS X:** `-vec-report`<br>**Windows:** `/Qvec-report` |
| **disable vec** | **Linux and Mac OS X:** `-vec-report0`<br>**Windows:** `/Qvec-report0` |
| **enable par** | **Linux and Mac OS X:** `-par-report`<br>**Windows:** `/Qpar-report` |
| **disable par** | **Linux and Mac OS X:** `-par-report0`<br>**Windows:** `/Qpar-report0` |

**Example**

The following example shows how to enable diagnostic IDs 117, 230 and 450:

```bash
-dl-diag-enable 117,230,450 ! Linux and Mac OS X systems
/Qdiag-enable:117,230,450 ! Windows systems
```

The following example shows how to change vectorizer diagnostic messages to warnings:

```bash
-dl-diag-enable vec -dl-diag-warning vec ! Linux and Mac OS X systems
/Qdiag-enable:vec /Qdiag-warning:vec ! Windows systems
```

Note that you need to enable the vectorizer diagnostics before you can change them to warnings.
The following example shows how to disable all auto-parallelizer diagnostic messages:
-diag-disable par  ! Linux and Mac OS X systems
/Qdiag-disable:par  ! Windows systems

The following example shows how to produce Source Checker diagnostic messages for all critical errors:
-diag-enable sc1  ! Linux and Mac OS X systems
/Qdiag-enable:sc1  ! Windows system

The following example shows how to cause Source Checker diagnostics (and default diagnostics) to be sent to a file:
-diag-enable sc -diag-file=stat_ver_msg  ! Linux and Mac OS X systems
/Qdiag-enable:sc /Qdiag-file:stat_ver_msg ! Windows systems

Note that you need to enable the Source Checker diagnostics before you can send them to a file. In this case, the diagnostics are sent to file stat_ver_msg.diag. If a file name is not specified, the diagnostics are sent to name-of-the-first-source-file.diag.

The following example shows how to change all diagnostic warnings and remarks to errors:
-diag-error warn,remark  ! Linux and Mac OS X systems
/Qdiag-error:warn,remark ! Windows systems

See Also

- diag-dump, Qdiag-dump
- diag-id-numbers, Qdiag-id-numbers
- diag-file, Qdiag-file
- par-report, Qpar-report
- vec-report, Qvec-report
**diag-dump, Qdiag-dump**

*Tells the compiler to print all enabled diagnostic messages and stop compilation.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

- `diag-dump`

*Windows:*

- `/Qdiag-dump`

**Arguments**

None

**Default**

OFF

The compiler issues certain diagnostic messages by default.

**Description**

This option tells the compiler to print all enabled diagnostic messages and stop compilation. The diagnostic messages are output to `stdout`.

This option prints the enabled diagnostics from all possible diagnostics that the compiler can issue, including any default diagnostics.

If `-diag-enable diag-list` *(Linux and Mac OS X)* or `/Qdiag-enable diag-list` *(Windows)* is specified, the print out will include the `diag-list` diagnostics.

**Alternate Options**

None
Example

The following example adds vectorizer diagnostic messages to the printout of default diagnostics:
- `diag-enable vec -diag-dump`       ! Linux and Mac OS X systems

/Qdiag-enable:vec /Qdiag-dump       ! Windows systems

See Also

•
•
• `diag`, `Qdiag`

diag, Qdiag

*Controls the display of diagnostic information.*

IDE Equivalent

Windows: `Diagnostics > Disable Specific Diagnostics` (/Qdiag-disable:id)

Diagnostics > Level of Source Code Analysis (/Qdiag-enable[:sc1,sc2,sc3])

Linux: `Compilation Diagnostics > Disable Specific Diagnostics` (-diag-disable id)

Compilation Diagnostics > Level of Source Code Analysis (-diag-enable [sc1,sc2,sc3] or -diag-disable sv)

Mac OS X: `Diagnostics > Disable Specific Diagnostics` (-diag-disable id)

Diagnostics > Level of Source Code Analysis (-diag-enable [sc1,sc2,sc3])

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- `diag-type diag-list`

Windows:

/Qdiag-type:diag-list
Arguments

**type**

Is an action to perform on diagnostics. Possible values are:

- **enable** Enables a diagnostic message or a group of messages.
- **disable** Disables a diagnostic message or a group of messages.
- **error** Tells the compiler to change diagnostics to errors.
- **warning** Tells the compiler to change diagnostics to warnings.
- **remark** Tells the compiler to change diagnostics to remarks (comments).

**diag-list**

Is a diagnostic group or ID value. Possible values are:

- **driver** Specifies diagnostic messages issued by the compiler driver.
- **port-linux** Specifies diagnostic messages for language features that may cause errors when porting to Linux. This diagnostic group is only available on Windows systems.
- **port-win** Specifies diagnostic messages for GNU extensions that may cause errors when porting to Windows. This diagnostic group is only available on Linux and Mac OS X systems.
- **thread** Specifies diagnostic messages that help in thread-enabling a program.
- **vec** Specifies diagnostic messages issued by the vectorizer.
- **par** Specifies diagnostic messages issued by the auto-parallelizer (parallel optimizer).
- **openmp** Specifies diagnostic messages issued by the OpenMP® parallelizer.
Specifies diagnostic messages issued by the Source Checker. \( n \) can be any of the following: 1, 2, 3. For more details on these values, see below. This value is equivalent to deprecated value sv\[n\].

Specifies diagnostic messages that have a "warning" severity level.

Specifies diagnostic messages that have an "error" severity level.

Specifies diagnostic messages that are remarks or comments.

Specifies the CPU dispatch remarks for diagnostic messages. These remarks are enabled by default. This diagnostic group is only available on IA-32 architecture and Intel® 64 architecture.

Specifies the ID number of one or more messages. If you specify more than one message number, they must be separated by commas. There can be no intervening white space between each id.

Specifies the mnemonic name of one or more messages. If you specify more than one mnemonic name, they must be separated by commas. There can be no intervening white space between each tag.

**Default**

OFF

The compiler issues certain diagnostic messages by default.

**Description**

This option controls the display of diagnostic information. Diagnostic messages are output to stderr unless compiler option `-diag-file` (Linux and Mac OS X) or `/Qdiag-file` (Windows) is specified.
When \textit{diag-list} value "warn" is used with the Source Checker (sc) diagnostics, the following behavior occurs:

- Option \texttt{-diag-enable warn} (Linux and Mac OS X) and \texttt{/Qdiag-enable:warn} (Windows) enable all Source Checker diagnostics except those that have an "error" severity level. They enable all Source Checker warnings, cautions, and remarks.
- Option \texttt{-diag-disable warn} (Linux and Mac OS X) and \texttt{/Qdiag-disable:warn} (Windows) disable all Source Checker diagnostics except those that have an "error" severity level. They suppress all Source Checker warnings, cautions, and remarks.

The following table shows more information on values you can specify for diag-list item sc.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{sc[n]}</td>
<td>The value of \texttt{n} for Source Checker messages can be any of the following:</td>
</tr>
<tr>
<td>1</td>
<td>Produces the diagnostics with severity level set to all critical errors.</td>
</tr>
<tr>
<td>2</td>
<td>Produces the diagnostics with severity level set to all errors. This is the default if \texttt{n} is not specified.</td>
</tr>
<tr>
<td>3</td>
<td>Produces the diagnostics with severity level set to all errors and warnings.</td>
</tr>
</tbody>
</table>

To control the diagnostic information reported by the vectorizer, use the \texttt{-vec-report} (Linux and Mac OS X) or \texttt{/Qvec-report} (Windows) option.

To control the diagnostic information reported by the auto-parallelizer, use the \texttt{-par-report} (Linux and Mac OS X) or \texttt{/Qpar-report} (Windows) option.

**Alternate Options**

- \texttt{enable vec} (Linux and Mac OS X): \texttt{-vec-report}  
  (Windows): \texttt{/Qvec-report}
- \texttt{disable vec} (Linux and Mac OS X): \texttt{-vec-report0}  
  (Windows): \texttt{/Qvec-report0}
- \texttt{enable par} (Linux and Mac OS X): \texttt{-par-report}  
  (Windows): \texttt{/Qpar-report}
- \texttt{disable par} (Linux and Mac OS X): \texttt{-par-report0}  
  (Windows): \texttt{/Qpar-report0}
Example

The following example shows how to enable diagnostic IDs 117, 230 and 450:

```-diag-enable 117,230,450 ! Linux and Mac OS X systems
/Qdiag-enable:117,230,450 ! Windows systems```

The following example shows how to change vectorizer diagnostic messages to warnings:

```-diag-enable vec -diag-warning vec ! Linux and Mac OS X systems
/Qdiag-enable:vec /Qdiag-warning:vec ! Windows systems```

Note that you need to enable the vectorizer diagnostics before you can change them to warnings.

The following example shows how to disable all auto-parallelizer diagnostic messages:

```-diag-disable par ! Linux and Mac OS X systems
/Qdiag-disable:par ! Windows systems```

The following example shows how to produce Source Checker diagnostic messages for all critical errors:

```-diag-enable sc1 ! Linux and Mac OS X systems
/Qdiag-enable:sc1 ! Windows system```

The following example shows how to cause Source Checker diagnostics (and default diagnostics) to be sent to a file:

```-diag-enable sc -diag-file=stat_ver_msg ! Linux and Mac OS X systems
/Qdiag-enable:sc /Qdiag-file:stat_ver_msg ! Windows systems```

Note that you need to enable the Source Checker diagnostics before you can send them to a file. In this case, the diagnostics are sent to file stat_ver_msg.diag. If a file name is not specified, the diagnostics are sent to name-of-the-first-source-file.diag.

The following example shows how to change all diagnostic warnings and remarks to errors:

```-diag-error warn,remark ! Linux and Mac OS X systems
/Qdiag-error:warn,remark ! Windows systems```

See Also

•
•
•
•
• diag-dump, Qdiag-dump
• diag-id-numbers, Qdiag-id-numbers
• diag-file, Qdiag-file
• par-report, Qpar-report
• vec-report, Qvec-report

diag-enable sc-include, Qdiag-enable:sc-include

*Tell a source code analyzer to process include files and source files when issuing diagnostic messages.*

**IDE Equivalent**

Windows: Diagnostics > Analyze Include Files

Linux: Compilation Diagnostics > Analyze Include Files

Mac OS X: Diagnostics > Analyze Include Files

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X:

-diag-enable sc-include

Windows:

/Qdiag-enable:sc-include

**Arguments**

None

**Default**

OFF

The compiler issues certain diagnostic messages by default. If the Source Checker is enabled, include files are not analyzed by default.
Description
This option tells a source code analyzer (Source Checker) to process include files and source files when issuing diagnostic messages. Normally, when Source Checker diagnostics are enabled, only source files are analyzed.

To use this option, you must also specify -diag-enable sc (Linux and Mac OS X) or /Qdiag-enable:sc (Windows) to enable the Source Checker diagnostics, or -diag-enable sc-parallel (Linux and Mac OS X) or /Qdiag-enable:sc-parallel (Windows) to enable parallel lint.

Alternate Options
Linux and Mac OS X: -diag-enable sv-include (this is a deprecated option)
Windows: /Qdiag-enable:sv-include (this is a deprecated option)

Example
The following example shows how to cause include files to be analyzed as well as source files:

-diag-enable sc -diag-enable sc-include       ! Linux and Mac OS systems
/Qdiag-enable:sc /Qdiag-enable:sc-include       ! Windows systems

In the above example, the first compiler option enables Source Checker messages. The second compiler option causes include files referred to by the source file to be analyzed also.

See Also
•
•
• diag-enable sc-parallel, Qdiag-enable:sc-parallel
• diag, Qdiag

diag-enable sc-parallel, Qdiag-enable:sc-parallel
Enables analysis of parallelization in source code (parallel lint diagnostics).

IDE Equivalent
Windows: Diagnostics > Level of Source Code Parallelization Analysis
Linux: None
Mac OS X: None
Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:
-diag-enable sc-parallel[n]

Windows:
/Qdiag-enable:sc-parallel[n]

Arguments

$n$ is the level of analysis to perform. Possible values are:

1. Produces the diagnostics with severity level set to all critical errors.

2. Tells the compiler to generate a report with the medium level of detail. Produces the diagnostics with severity level set to all errors. This is the default if $n$ is not specified.

3. Produces the diagnostics with severity level set to all errors and warnings.

Default
OFF

The compiler does not analyze parallelization in source code.

Description

This option enables analysis of parallelization in source code (parallel lint diagnostics). Currently, this analysis uses OpenMP pragmas, so this option has no effect unless option /Qopenmp (Windows) or option -openmp (Linux and Mac OS X) is set.

Parallel lint performs interprocedural source code analysis to identify mistakes when using parallel pragmas. It reports various problems that are difficult to find, including data dependency and potential deadlocks.
Source Checker diagnostics (enabled by /Qdiag-enable:sc on Windows* OS or -diag-enable sc on Linux* OS and Mac OS* X) are a superset of parallel lint diagnostics. Therefore, if Source Checker diagnostics are enabled, the parallel lint option is not taken into account.

Alternate Options
None

See Also
•
•
• diag, Qdiag

diag-error-limit, Qdiag-error-limit
Specifies the maximum number of errors allowed before compilation stops.

IDE Equivalent
Windows: Compilation Diagnostics > Error Limit
Linux: Compilation Diagnostics > Set Error Limit
Mac OS X: Compilation Diagnostics > Error Limit

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-diag-error-limit n
-no-diag-error-limit

Windows:
/Qdiag-error-limit:n
/Qdiag-error-limit-
Arguments

\( n \)  
Is the maximum number of error-level or fatal-level compiler errors allowed.

Default

30  
A maximum of 30 error-level and fatal-level messages are allowed.

Description

This option specifies the maximum number of errors allowed before compilation stops. It indicates the maximum number of error-level or fatal-level compiler errors allowed for a file specified on the command line.

If you specify `-no-diag-error-limit` (Linux and Mac OS X) or `/Qdiag-error-limit-` (Windows) on the command line, there is no limit on the number of errors that are allowed.

If the maximum number of errors is reached, a warning message is issued and the next file (if any) on the command line is compiled.

Alternate Options

Linux and Mac OS X: `–wn` (this is a deprecated option)

Windows: `/Qwn` (this is a deprecated option)

diag-file, Qdiag-file

Causes the results of diagnostic analysis to be output to a file.

IDE Equivalent

Windows: Diagnostics > Diagnostics File

Linux: Compilation Diagnostics > Diagnostics File

Mac OS X: Diagnostics > Diagnostics File

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
-diag-file=file

Windows:
/Qdiag-file:file

Arguments

file Is the name of the file for output.

Default

OFF Diagnostic messages are output to stderr.

Description

This option causes the results of diagnostic analysis to be output to a file. The file is placed in the current working directory.

If file is specified, the name of the file is file.diag. The file can include a file extension; for example, if file.ext is specified, the name of the file is file.ext.

If file is not specified, the name of the file is name-of-the-first-source-file.diag. This is also the name of the file if the name specified for file conflicts with a source file name provided in the command line.

NOTE. If you specify -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows) and you also specify -diag-file-append (Linux and Mac OS X) or /Qdiag-file-append (Windows), the last option specified on the command line takes precedence.

Alternate Options

None
Example
The following example shows how to cause diagnostic analysis to be output to a file named my_diagnostics.diag:

- diag-file=my_diagnostics ! Linux and Mac OS X systems
/Qdiag-file:my_diagnostics ! Windows systems

See Also

- diag, Qdiag
- diag-file-append, Qdiag-file-append

diag-file-append, Qdiag-file-append
Causes the results of diagnostic analysis to be appended to a file.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- diag-file-append[=file]

Windows:
/Qdiag-file-append[:file]

Arguments
file Is the name of the file to be appended to. It can include a path.

Default
OFF Diagnostic messages are output to stderr.
**Description**

This option causes the results of diagnostic analysis to be appended to a file. If you do not specify a path, the driver will look for file in the current working directory.

If file is not found, then a new file with that name is created in the current working directory. If the name specified for file conflicts with a source file name provided in the command line, the name of the file is name-of-the-first-source-file.diag.

**NOTE.** If you specify -diag-file-append (Linux and Mac OS X) or /Qdiag-file-append (Windows) and you also specify -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows), the last option specified on the command line takes precedence.

**Alternate Options**

None

**Example**

The following example shows how to cause diagnostic analysis to be appended to a file named my_diagnostics.txt:

-diag-file-append=my_diagnostics.txt ! Linux and Mac OS X systems
/Qdiag-file-append:my_diagnostics.txt ! Windows systems

**See Also**

- diag, Qdiag
- diag-file, Qdiag-file

**diag-id-numbers, Qdiag-id-numbers**

Determines whether the compiler displays diagnostic messages by using their ID number values.

**IDE Equivalent**

None
**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `-diag-id-numbers`
- `-no-diag-id-numbers`

**Windows:**
- `/Qdiag-id-numbers`
- `/Qdiag-id-numbers-`

**Arguments**
None

**Default**

- `-diag-id-numbers`  The compiler displays diagnostic messages by using their ID number values.

- `/Qdiag-id-numbers`  (Windows), mnemonic names are output for driver diagnostics only.

**Description**

This option determines whether the compiler displays diagnostic messages by using their ID number values. If you specify `-no-diag-id-numbers` (Linux and Mac OS X) or `/Qdiag-id-numbers-` (Windows), mnemonic names are output for driver diagnostics only.

**Alternate Options**
None

**See Also**
- `diag, Qdiag`
**diag-once, Qdiag-once**

*Tells the compiler to issue one or more diagnostic messages only once.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

`-diag-once id[,id,...]`

**Windows:**

`/Qdiag-once: id[,id,...]`

**Arguments**

`id` Is the ID number of the diagnostic message. If you specify more than one message number, they must be separated by commas. There can be no intervening white space between each `id`.

**Default**

OFF The compiler issues certain diagnostic messages by default.

**Description**

This option tells the compiler to issue one or more diagnostic messages only once.

**Alternate Options**

Linux: `–wo` (this is a deprecated option)

Windows: `/Qwo` (this is a deprecated option)
**dM, QdM**

*Tells the compiler to output macro definitions in effect after preprocessing.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

```bash
-dM
```

**Windows:**

```cmd
/QdM
```

**Arguments**
None

**Default**
OFF

The compiler does not output macro definitions after preprocessing.

**Description**

This option tells the compiler to output macro definitions in effect after preprocessing. To use this option, you must also specify the `E` option.

**Alternate Options**
None

**See Also**

- 
- 
- `E`
**dN, QdN**

*Same as \( -dD \), but output \#define directives contain only macro names.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

\(-dN\)

*Windows:*

\(/QdN\)

**Arguments**

None

**Default**

OFF

The compiler does not output \#define directives.

**Description**

Same as \( -dD \), but output \#define directives contain only macro names. To use this option, you must also specify the \( E \) option.

**Alternate Options**

None
dryrun
Specifies that driver tool commands should be shown but not executed.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-dryrun
Windows:
None

Arguments
None

Default
OFF No tool commands are shown, but they are executed.

Description
This option specifies that driver tool commands should be shown but not executed.

Alternate Options
None

See Also
•
• v
dumpmachine

Displays the target machine and operating system configuration.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-dumpmachine

Windows:

None

Arguments

None

Default

OFF

The compiler does not display target machine or operating system information.

Description

This option displays the target machine and operating system configuration. No compilation is performed.

Alternate Options

None

See Also

•

• dumpversion
dumpversion
Displays the version number of the compiler.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-dumpversion

Windows:
None

Arguments
None

Default
OFF The compiler does not display the compiler version number.

Description
This option displays the version number of the compiler. It does not compile your source files.

Alternate Options
None

Example
Consider the following command:
icc -dumpversion

If it is specified when using the Intel C++ Compiler 10.1, the compiler displays "10.1".
See Also

•
  • dumpmachine

dynamic-linker

Specifies a dynamic linker other than the default.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux:
-dynamic-linker file

Mac OS X:
None

Windows:
None

Arguments

file

Is the name of the dynamic linker to be used.

Default

OFF

The default dynamic linker is used.

Description

This option lets you specify a dynamic linker other than the default.

Alternate Options

None
**dynamiclib**

*Invokes the libtool command to generate dynamic libraries.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux:**

None

**Mac OS X:**

-`-dynamiclib`

**Windows:**

None

**Arguments**

None

**Default**

OFF  
The compiler produces an executable.

**Description**

This option invokes the *libtool* command to generate dynamic libraries.

When passed this option, the compiler uses the *libtool* command to produce a dynamic library instead of an executable when linking.

To build static libraries, you should specify option `-staticlib` or `libtool -static <objects>.

**Alternate Options**

None
See Also

- staticlib

E

Causes the preprocessor to send output to stdout.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-\n
Windows:

/\n
Arguments

None

Default

OFF

Preprocessed source files are output to the compiler.

Description

This option causes the preprocessor to send output to stdout. Compilation stops when the files have been preprocessed.

When you specify this option, the compiler's preprocessor expands your source module and writes the result to stdout. The preprocessed source contains #line directives, which the compiler uses to determine the source file and line number.
Alternate Options
None

early-template-check
Lets you semantically check template function template prototypes before instantiation.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-early-template-check
-no-early-template-check

Windows:
None

Arguments
None

Default
-no-early-template-check  The prototype instantiation of function templates and function members of class templates is deferred.

Description
Lets you semantically check template function template prototypes before instantiation. On Linux® OS platforms, gcc 3.4 (or newer) compatibility modes (i.e. -gcc-version=340 and later) must be in effect. For all Mac OS* X platforms, gcc 4.0 (or newer) is required.

Alternate Options
None
**EH**

*Enables different models of exception handling.*

**IDE Equivalent**

*Windows:* **Code Generation > Enable C++ Exceptions**

*Linux:* None

*Mac OS X:* None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

None

*Windows:*

/\EH\type

**Arguments**

\textit{type} \hspace{1cm} Possible values are: \textit{a}, \textit{s}, or \textit{c}

**Default**

OFF

**Description**

This option enables different models of exception handling:

- /\EH\a -- enable asynchronous C++ exception handling model
- /\EH\s -- enable synchronous C++ exception handling model
- /\EH\c -- assume extern "C" functions do not throw exceptions

**Alternate Options**

None
**See Also**

- Qsafe4eh

**EP**

*Causes the preprocessor to send output to stdout, omitting #line directives.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

`-EP`

Windows:

`/EP`

**Arguments**

None

**Default**

OFF

Preprocessed source files are output to the compiler.

**Description**

This option causes the preprocessor to send output to stdout, omitting #line directives.

If you also specify option `P` or Linux option `F`, the preprocessor will write the results (without #line directives) to a file instead of stdout.

**Alternate Options**

None
**export**

*Enables support for the C++ export template feature.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

```
-export
```

*Windows:*

None

**Arguments**

None

**Default**

```
OFF
```

The export template feature is not enabled.

**Description**

This option enables support for the C++ export template feature. This option is supported only in C++ mode.

**Alternate Options**

None

**See Also**

- `export-dir`
**export-dir**

Specifies a directory name for the exported template search path.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-export-dir dir`

**Windows:**

None

**Arguments**

`dir` Is the directory name to add to the search path.

**Default**

OFF The compiler does not recognize exported templates.

**Description**

This option specifies a directory name for the exported template search path. To use this option, you must also specify the -export option.

Directories in the search path are used to find the definitions of exported templates and are searched in the order in which they are specified on the command-line. The current directory is always the first entry in the search path.

**Alternate Options**

None

**See Also**

•
• export

**F (Mac OS* X)**
*Add framework directory to head of include file search path.*

**IDE Equivalent**
None

**Architectures**
IA-32 architecture

**Syntax**

**Linux:**
None

**Mac OS X:**
-Fdir

**Windows:**
None

**Arguments**

`dir` Is the name for the framework directory.

**Default**

OFF The compiler does add a framework directory to head of include file search path.

**Description**

Add framework directory to head of include file search path.

**Alternate Options**
None
**F (Windows*)**

Specifies the stack reserve amount for the program.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/F\n
**Arguments**

\( n \)

Is the stack reserve amount. It can be specified as a decimal integer or by using a C-style convention for constants (for example, /F0x1000).

**Default**

OFF

The stack size default is chosen by the operating system.

**Description**

This option specifies the stack reserve amount for the program. The amount \( (n) \) is passed to the linker.

Note that the linker property pages have their own option to do this.

**Alternate Options**

None
Fa
Specifies the contents of an assembly listing file.

IDE Equivalent
Windows: Output Files > ASM List Location
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None
Windows:
/Fa[/file|dir]

Arguments
/file Is the name of the assembly listing file.
dir Is the directory where the file should be placed. It can include file.

Default
OFF No assembly listing file is produced.

Description
This option specifies that an assembly listing file should be generated (optionally named file).

Alternate Options
Linux and Mac OS X: -S
Windows: None
**FA**

*Specifies the contents of an assembly listing file.*

**IDE Equivalent**

Windows: **Output Files > Assembler Output**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

None

Windows:

/FA{specifier}

**Arguments**

*specifier*  
Denotes the contents of the assembly listing file. Possible values are $c$, $s$, or $cs$.

**Default**

OFF  
No additional information appears in the assembly listing file, if one is produced.

**Description**

These options specify what information, in addition to the assembly code, should be generated in the assembly listing file:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/FAc</td>
<td>Produces an assembly listing with machine code.</td>
</tr>
<tr>
<td>/FAs</td>
<td>Produces an assembly listing with source code.</td>
</tr>
</tbody>
</table>
### Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/FAcs</td>
<td>Produces an assembly listing with machine code and source code.</td>
</tr>
</tbody>
</table>

### Alternate Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Linux and Mac OS X:</th>
<th>Windows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>/FAc</td>
<td>-fcode-asm</td>
<td>None</td>
</tr>
</tbody>
</table>

### fabi-version

*Instructs the compiler to select a specific ABI implementation.*

### IDE Equivalent

- **Linux**: None
- **Mac OS X**: None

### Architectures

- IA-32, Intel® 64, IA-64 architectures

### Syntax

- **Linux and Mac OS X**:
  
  - `fabi-version=n`

- **Windows**:
  
  None

### Arguments

- **n**
  
  Is the ABI implementation. Possible values are:

  - 0  
    
    Requests the latest ABI implementation.

  - 1  
    
    Requests the ABI implementation used in gcc 3.2 and gcc 3.3.
Requests the ABI implementation used in gcc 3.4 and higher.

**Default**

Varies

The compiler uses the ABI implementation that corresponds to the installed version of gcc.

**Description**

This option tells the compiler to select a specific ABI implementation. This option is compatible with gcc option `-fabi-version`. If you have multiple versions of gcc installed, the compiler may change the value of `n` depending on which gcc is detected in your path.

---

**NOTE.** gcc 3.2 and 3.3 are not fully ABI-compliant, but gcc 3.4 is highly ABI-compliant.

---

**CAUTION.** Do not mix different values for `-fabi-version` in one link.

**Alternate Options**

None

**falias**

*Determine whether aliasing should be assumed in the program.*

**IDE Equivalent**

Windows: None

Linux: **Data > Assume No Aliasing in Program**

Mac OS X: **Data > Assume No Aliasing in Program**

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax
Linux and Mac OS X:
- -falias
- -fno-alias
Windows:
None

Arguments
None

Default
-falias
Aliasing is assumed in the program.

Description
This option determines whether aliasing should be assumed in the program.
If you do not want aliasing to be assumed in the program, specify -fno-alias.

Alternate Options
Linux and Mac OS X: None
Windows: /Oa[-]

See Also
- ffnalias

falign-functions, Qfnalign
Tells the compiler to align functions on an optimal byte boundary.

IDE Equivalent
None
Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:
-falign-functions[=n]
-fno-align-functions

Windows:
/Qfnalign[:n]
/Qfnalign-

Arguments

n Is the byte boundary for function alignment. Possible values are 2 or 16.

Default

-fno-align-functions or The compiler aligns functions on 2-byte boundaries. This is the same as specifying -falign-functions=2 (Linux and Mac OS X) or /Qfnalign:2 (Windows).

Description
This option tells the compiler to align functions on an optimal byte boundary. If you do not specify n, the compiler aligns the start of functions on 16-byte boundaries.

Alternate Options
None

falign-stack
Tells the compiler the stack alignment to use on entry to routines.

IDE Equivalent
None
Architectures
IA-32 architecture

Syntax
Linux and Mac OS X:
-falign-stack=mode
Windows:
None

Arguments
mode Is the method to use for stack alignment. Possible values are:
default Tells the compiler to use default heuristics for stack alignment. If alignment is required, the compiler dynamically aligns the stack.
maintain-16-byte Tells the compiler to not assume any specific stack alignment, but attempt to maintain alignment in case the stack is already aligned. If alignment is required, the compiler dynamically aligns the stack. This setting is compatible with GCC.
assemble-16-byte Tells the compiler to assume the stack is aligned on 16-byte boundaries and continue to maintain 16-byte alignment. This setting is compatible with GCC.

Default
-falign-stack=default The compiler uses default heuristics for stack alignment.

Description
This option tells the compiler the stack alignment to use on entry to routines.
Alternate Options
None

fargument-alias, Qalias-args
Determines whether function arguments can alias each other.

IDE Equivalent
Windows: None
Linux: Data > Enable Argument Aliasing
Mac OS X: Data > Enable Argument Aliasing

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fargument-alias
-fargument-noalias

Windows:
/Qalias-args
/Qalias-args-

Arguments
None

Default
-fargument-alias or /Qalias-args Function arguments can alias each other and can alias global storage.
**Description**

This option determines whether function arguments can alias each other. If you specify `-fargument-noalias` or `/Qalias-args-`, function arguments cannot alias each other, but they can alias global storage.

On Linux and Mac OS X systems, you can also disable aliasing for global storage, by specifying option `-fargument-noalias-global`.

**Alternate Options**

Linux and Mac OS X: `-[no-]alias-args` (this is a deprecated option)

Windows: None

**See Also**

- `fargument-noalias-global`

**fargument-noalias-global**

*Tells the compiler that function arguments cannot alias each other and cannot alias global storage.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

- `-fargument-noalias-global`

*Windows:*

None
Arguments
None

Default
OFF
Function arguments can alias each other and can alias global storage.

Description
This option tells the compiler that function arguments cannot alias each other and they cannot alias global storage.

If you only want to prevent function arguments from being able to alias each other, specify option -fargument-noalias.

Alternate Options
None

See Also
•
•  fargument-alias, Qalias-args

fasm-blocks
Enables the use of blocks and entire functions of assembly code within a C or C++ file.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax

Linux:
None
Mac OS X:
-fasm-blocks

Windows:
None

Arguments
None

Default
OFF   You cannot use these features within a C or C++ file.

Description
This option enables the use of blocks and entire functions of assembly code within a C or C++ file.

Note that this option enables a Microsoft*-style assembly block not a GNU*-style assembly block.

This option is provided for compatibility with the Apple* GNU compiler.

Alternate Options
None

fast
Maximizes speed across the entire program.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:

-fast
Windows:
/fast

Arguments
None

Default
OFF

The optimizations that maximize speed are not enabled.

Description
This option maximizes speed across the entire program.

It sets the following options:

- On systems using IA-64 architecture:
  Windows: /O3 and /Qipo
  Linux: -ipo, -O3, and -static

- On systems using IA-32 architecture and Intel® 64 architecture:
  Mac OS X: -ipo, -mdynamic-no-pic, -O3, -no-prec-div, -static, and -xHost
  Windows: /O3, /Qipo, /Qprec-div-, and /QxHost
  Linux: -ipo, -O3, -no-prec-div, -static, and -xHost

When option fast is specified on systems using IA-32 architecture or Intel® 64 architecture, you can override the -xHost or /QxHost setting by specifying a different processor-specific -x or /Qx option on the command line. However, the last option specified on the command line takes precedence.

For example, if you specify -fast -xSSE3 (Linux) or /fast /QxSSE3 (Windows), option -xSSE3 or /QxSSE3 takes effect. However, if you specify -xSSE3 -fast (Linux) or /QxSSE3 /fast (Windows), option -xHost or /QxHost takes effect.

NOTE. The options set by option fast may change from release to release.

Alternate Options
None
See Also

• $x$, $Qx$

**fast-transcendentals, Qfast-transcendentals**

*Enables the compiler to replace calls to transcendental functions with faster but less precise implementations.*

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

- `-fast-transcendentals`
- `-no-fast-transcendentals`

**Windows:**

/`Qfast-transcendentals`

/`Qfast-transcendentals-`

Default

- `-fast-transcendentals` or `/Qfast-transcendentals`

The default depends on the setting of `-fp-model` (Linux and Mac OS X) or `/fp` (Windows). The default is ON if default setting `-fp-model fast` or `/fp:fast` is in effect. However, if a value-safe option such as `-fp-model precise` or `/fp:precise` is specified, the default is OFF.

Description

This option enables the compiler to replace calls to transcendental functions with implementations that may be faster but less precise.
It tells the compiler to perform certain optimizations on transcendental functions, such as replacing individual calls to sine in a loop with a single call to a less precise vectorized sine library routine.

This option has an effect only when specified with one of the following options:

- **Windows* OS:** /fp:except or /fp:precise
- **Linux* OS and Mac OS* X:** -fp-model except or -fp-model precise

You cannot use this option with option -fp-model strict (Linux and Mac OS X) or /fp:strict (Windows).

**Alternate Options**

None

**See Also**

- fp-model, fp

**fbuiltin, Oi**

*Enables or disables inline expansion of intrinsic functions.*

**IDE Equivalent**

Windows: **Optimization > Enable Intrinsic Functions**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- fbuiltin[[-func]]
- fno-builtin[[-func]]
Windows: 
/Oi[-]

Arguments

\textit{func} \hspace{1cm} \text{A comma-separated list of intrinsic functions.}

Default

OFF \hspace{1cm} \text{Inline expansion of intrinsic functions disabled.}

Description

This option enables or disables inline expansion of one or more intrinsic functions. If \textit{-func} is not specified, \textit{-fno-built-in} disables inline expansion for all intrinsic functions.

For a list of built-in functions affected by \textit{-fbuilt-in}, search for "built-in functions" in the appropriate gcc* documentation.

For a list of built-in functions affected by \textit{/Oi}, search for "/Oi" in the appropriate Microsoft* Visual C/C++* documentation.

Alternate Options

None

\textbf{FC}

\textit{Displays the full path of source files passed to the compiler in diagnostics.}

IDE Equivalent

Windows: Advanced > Use Full Paths

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

\textbf{Linux and Mac OS X:}

None
Windows:
/FC

Arguments
None

Default
OFF
The compiler does not display the full path of source files passed to the compiler in diagnostics.

Description
Displays the full path of source files passed to the compiler in diagnostics. This option is supported with Microsoft Visual Studio .NET 2003* or newer.

Alternate Options
None

fcode-asm
Produces an assembly listing with machine code annotations.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fcode-asm
Windows:
None

Arguments
None
Default

OFF No machine code annotations appear in the assembly listing file, if one is produced.

Description
This option produces an assembly listing file with machine code annotations.
The assembly listing file shows the hex machine instructions at the beginning of each line of assembly code. The file cannot be assembled; the filename is the name of the source file with an extension of .cod.
To use this option, you must also specify option -S, which causes an assembly listing to be generated.

Alternate Options
Linux and Mac OS X: None
Windows: /FAc

See Also
•
• S

fcommon
Determines whether the compiler treats common symbols as global definitions.

IDE Equivalent
Windows: None
Linux: Data > Allow gprel Addressing of Common Data Variables
Mac OS X: Data > Allow gprel Addressing of Common Data Variables

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
- `fcommon`
- `fnocommon`

Windows:
None

Arguments
None

Default
- `fcommon` The compiler does not treat common symbols as global definitions.

Description
This option determines whether the compiler treats common symbols as global definitions and to allocate memory for each symbol at compile time.

Option `-fno-common` tells the compiler to treat common symbols as global definitions. When using this option, you can only have a common variable declared in one module; otherwise, a link time error will occur for multiple defined symbols.

On IA-64 architecture, this option allows the use of gp-relative(gprel) addressing of common data variables.

Normally, a file-scope declaration with no initializer and without the `extern` or `static` keyword "int i;" is represented as a common symbol. Such a symbol is treated as an external reference. However, if no other compilation unit has a global definition for the name, the linker allocates memory for it.

Alternate Options
None
FD
Generates file dependencies related to the Microsoft® C/C++ compiler.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/FD

Arguments
None

Default
OFF
The compiler does not generate Microsoft C/C++-related file dependencies.

Description
This option generates file dependencies related to the Microsoft C/C++ compiler. It invokes the Microsoft C/C++ compiler and passes the option to it.

Alternate Options
None
**Fe**

Specifies the name for a built program or dynamic-link library.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Fe[file|dir]

**Arguments**

- *file*: Is the name for the built program or dynamic-link library.
- *dir*: Is the directory where the built program or dynamic-link library should be placed. It can include *file*.

**Default**

OFF

The name of the file is the name of the first source file on the command line with file extension .exe, so file.f becomes file.exe.

**Description**

This option specifies the name for a built program (.EXE) or a dynamic-link library (.DLL).

You can use this option to specify an alternate name for an executable file. This is especially useful when compiling and linking a set of input files. You can use the option to give the resulting file a name other than that of the first input file (source or object) on the command line.

**Alternate Options**

**Linux and Mac OS X:** –o
Example
In the following example, the command produces an executable file named outfile.exe as a result of compiling and linking three files: one object file and two C++ source files.

prompt> icl /Feoutfile.exe file1.obj file2.cpp file3.cpp

By default, this command produces an executable file named file1.exe.

See Also

fexceptions
Enables exception handling table generation.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-fexceptions
-fno-exceptions

Windows:
None

Arguments
None

Default

-fexceptions Exception handling table generation is enabled. Default for C++.
-fno-exceptions Exception handling table generation is disabled. Default for C.
Description
This option enables exception handling table generation. The -fno-exceptions option disables exception handling table generation, resulting in smaller code. When this option is used, any use of exception handling constructs (such as try blocks and throw statements) will produce an error. Exception specifications are parsed but ignored. It also undefines the preprocessor symbol __EXCEPTIONS.

Alternate Options
None

ffnalias
Specifies that aliasing should be assumed within functions.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-ffnalias
-fno-ffnalias

Windows:
None

Arguments
None

Default
-ffnalias Aliasing is assumed within functions.
Description
This option specifies that aliasing should be assumed within functions.
The `-fno-fnalias` option specifies that aliasing should not be assumed within functions, but should be assumed across calls.

Alternate Options
Linux and Mac OS X: None
Windows: `/Ow[-]`

See Also
•
  • `falias`

**ffreestanding, Qfreestanding**
*Ensures that compilation takes place in a freestanding environment.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
**Linux and Mac OS X:**
- `ffreestanding`

**Windows:**
/`Qfreestanding`

Arguments
None

Default
OFF Standard libraries are used during compilation.
Description
This option ensures that compilation takes place in a freestanding environment. The compiler assumes that the standard library may not exist and program startup may not necessarily be at main. This environment meets the definition of a freestanding environment as described in the C and C++ standard.

An example of an application requiring such an environment is an OS kernel.

**NOTE.** When you specify this option, the compiler will not assume the presence of compiler-specific libraries. It will only generate calls that appear in the source code.

Alternate Options
None

**ffunction-sections**
Placed each function in its own COMDAT section.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and MacOS X:
-ffunction-sections

Windows:
None

Arguments
None

Default
OFF
Description
Places each function in its own COMDAT section.

Alternate Options
-fdata-sections

FI
*Tells the preprocessor to include a specified filename as the header file.*

IDE Equivalent
Windows: **Advanced > Force Includes**
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/FI file

Arguments
*file* Is the file name to be included as the header file.

Default
OFF The compiler uses default header files.

Description
This option tells the preprocessor to include a specified file name as the header file.
The file specified with `#FI` is included in the compilation before the first line of the primary source file.

Alternate Options
None

`finline`
*Tells the compiler to inline functions declared with `__inline` and perform C++ inlining.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
*Linux and Mac OS X:*
- `-finline`
- `-fno-inline`

*Windows:*
None

Arguments
None

Default
- `-fno-inline` The compiler does not inline functions declared with `__inline`.

Description
This option tells the compiler to inline functions declared with `__inline` and perform C++ inlining.

Alternate Options
None
finline-functions

Enables function inlining for single file compilation.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-finline-functions
-fno-inline-functions

Windows:
None

Arguments
None

Default
-finline-functions
Interprocedural optimizations occur. However, if you specify -O0, the default is OFF.

Description
This option enables function inlining for single file compilation.
It enables the compiler to perform inline function expansion for calls to functions defined within the current source file.
The compiler applies a heuristic to perform the function expansion. To specify the size of the function to be expanded, use the -finline-limit option.

Alternate Options
Linux and Mac OS X: -inline-level=2
Windows: /Ob2
See Also
• ip, Qip
• finline-limit
• Compiler Directed Inline Expansion of User Functions
• Inline Function Expansion

finline-limit
*Lets you specify the maximum size of a function to be inlined.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-finline-limit=n

Windows:
None

Arguments

n
Must be an integer greater than or equal to zero. It is the maximum number of lines the function can have to be considered for inlining.

Default

OFF
The compiler uses default heuristics when inlining functions.
Description

This option lets you specify the maximum size of a function to be inlined. The compiler inlines smaller functions, but this option lets you inline large functions. For example, to indicate a large function, you could specify 100 or 1000 for \( n \).

Note that parts of functions cannot be inlined, only whole functions.

This option is a modification of the `-finline-functions` option, whose behavior occurs by default.

Alternate Options

None

See Also

- `finline-functions`

`finstrument-functions, Qinstrument-functions`

Determines whether function entry and exit points are instrumented.

IDE Equivalent

None

Architectures

IA-32, Intel\(^{\circledR}\) 64, IA-64 architectures

Syntax

Linux and Mac OS X:

- `finstrument-functions`
- `fno-instrument-functions`

Windows:

/`Qinstrument-functions`
/`Qinstrument-functions-`
Arguments
None

Default
-fno-instrument-functions  Function entry and exit points are not instrumented.
or/ Qinstrument-functions-

Description
This option determines whether function entry and exit points are instrumented. It may increase execution time.

The following profiling functions are called with the address of the current function and the address of where the function was called (its "call site"):

- This function is called upon function entry:
  - On IA-32 architecture and Intel® 64 architecture:
    ```c
    void __cyg_profile_func_enter (void *this_fn,
                                 void *call_site);
    ```
  - On IA-64 architecture:
    ```c
    void __cyg_profile_func_enter (void **this_fn,
                                    void *call_site);
    ```

- This function is called upon function exit:
  - On IA-32 architecture and Intel® 64 architecture:
    ```c
    void __cyg_profile_func_exit (void *this_fn,
                                   void *call_site);
    ```
  - On IA-64 architecture:
    ```c
    void __cyg_profile_func_exit (void **this_fn,
                                   void *call_site);
    ```
On IA-64 architecture, the additional de-reference of the function pointer argument is required to obtain the function entry point contained in the first word of the function descriptor for indirect function calls. The descriptor is documented in the Intel® Itanium® Software Conventions and Runtime Architecture Guide, section 8.4.2. You can find this design guide at web site http://www.intel.com.

These functions can be used to gather more information, such as profiling information or timing information. Note that it is the user's responsibility to provide these profiling functions.

If you specify `-finstrument-functions` (Linux and Mac OS X) or `/Qinstrument-functions` (Windows), function inlining is disabled. If you specify `-fno-instrument-functions` or `/Qinstrument-functions-`, inlining is not disabled.

On Linux and Mac OS X systems, you can use the following attribute to stop an individual function from being instrumented:

```c
__attribute__((__no_instrument_function__))
```

It also stops inlining from being disabled for that individual function.

This option is provided for compatibility with gcc.

**Alternate Options**

None

**fixed**

_Causes the linker to create a program that can be loaded only at its preferred base address._

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

_/fixed_
**Arguments**
None

**Default**
OFF
The compiler uses default methods to load programs.

**Description**
This option is passed to the linker, causing it to create a program that can be loaded only at its preferred base address.

**Alternate Options**
None

**fjump-tables**
*Determines whether jump tables are generated for switch statements.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- *fjump-tables
- *fno-jump-tables

Windows:
None

**Arguments**
None
The compiler uses jump tables for switch statements.

This option determines whether jump tables are generated for switch statements.

Option `-fno-jump-tables` prevents the compiler from generating jump tables for switch statements. This action is performed unconditionally and independent of any generated code performance consideration.

Option `-fno-jump-tables` also prevents the compiler from creating switch statements internally as a result of optimizations.

Use `-fno-jump-tables` with `-fpic` when compiling objects that will be loaded in a way where the jump table relocation cannot be resolved.

Alternate Options
None

See Also

- `fpic`

`fkeep-static-consts`, `Qkeep-static-consts`

Tells the compiler to preserve allocation of variables that are not referenced in the source.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
`-fkeep-static-consts`
`-fno-keep-static-consts`
Windows:
/Qkeep-static-consts
/Qkeep-static-consts-

Arguments
None

Default
-fno-keep-static-consts or /Qkeep-static-consts-

If a variable is never referenced in a routine, the variable is discarded unless optimizations are disabled by option -00 (Linux and Mac OS X) or /Od (Windows).

Description
This option tells the compiler to preserve allocation of variables that are not referenced in the source.
The negated form can be useful when optimizations are enabled to reduce the memory usage of static data.

Alternate Options
None

Fm
Tells the linker to generate a link map file.
This option has been deprecated.

IDE Equivalent
None

Architectures
IA-32 architecture, Intel® 64 architecture, IA-64 architecture

Syntax
Linux and Mac OS X:
None
**Windows:**

/Fm[<file>|<dir>]

**Arguments**

- **file**: Is the name for the link map file.
- **dir**: Is the directory where the link map file should be placed. It can include `<file>`.

**Default**

OFF  No link map is generated.

**Description**

This option tells the linker to generate

**Alternate Options**

None

**fma, Qfma**

Enables the combining or contraction of floating-point multiplications and add or subtract operations.

**IDE Equivalent**

Windows: None

Linux: **Floating Point > Floating-point Operation Contraction**

Mac OS X: None

**Architectures**

IA-64 architecture

**Syntax**

- **Linux:**
  - `fma`
  - `-no-fma`
Mac OS X:
None

Windows:
/Qfma
/Qfma-

Arguments
None

Default
-fma
or /Qfma

Floating-point multiplications and add/subtract operations are combined. However, if you specify -fp-model strict (Linux) or /fp:strict (Windows), but do not explicitly specify -fma or /Qfma, the default is -no-fma or /Qfma-.

Description
This option enables the combining or contraction of floating-point multiplications and add or subtract operations into a single operation.

Alternate Options
Linux: -IPF-fma (this is a deprecated option)
Windows: /QIPF-fma (this is a deprecated option)

See Also
• fp-model, fp

Floating-point Operations: Floating-point Options Quick Reference
**fmath-errno**
*Tells the compiler that `errno` can be reliably tested after calls to standard math library functions.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
  - `fmath-errno`
  - `fno-math-errno`

**Windows:**
None

**Arguments**
None

**Default**
`fno-math-errno`  The compiler assumes that the program does not test `errno` after calls to standard math library functions.

**Description**

This option tells the compiler to assume that the program tests `errno` after calls to math library functions. This restricts optimization because it causes the compiler to treat most math functions as having side effects.

Option `-fno-math-errno` tells the compiler to assume that the program does not test `errno` after calls to math library functions. This frequently allows the compiler to generate faster code. Floating-point code that relies on IEEE exceptions instead of `errno` to detect errors can safely use this option to improve performance.
Alternate Options
None

**fminshared**
Specifies that a compilation unit is a component of a main program and should not be linked as part of a shareable object.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
**Linux and Mac OS X:**
-fminshared

**Windows:**
None

Arguments
None

Default
OFF Source files are compiled together to form a single object file.

Description
This option specifies that a compilation unit is a component of a main program and should not be linked as part of a shareable object.

This option allows the compiler to optimize references to defined symbols without special visibility settings. To ensure that external and common symbol references are optimized, you need to specify visibility hidden or protected by using the `-fvisibility`, `-fvisibility-hidden`, or `-fvisibility-protected` option.
Also, the compiler does not need to generate position-independent code for the main program. It can use absolute addressing, which may reduce the size of the global offset table (GOT) and may reduce memory traffic.

**Alternate Options**

None

**See Also**

- \textbf{fvisibility}

**fmudflap**

“The compiler instruments risky pointer operations to prevent buffer overflows and invalid heap use.”

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**

\texttt{-fmudflap}

**Mac\ OS\ X:**

None

**Windows:**

None

**Arguments**

None

**Default**

OFF \hspace{1cm} The compiler does not instruments risky pointer operations.
**Description**

The compiler instruments risky pointer operations to prevent buffer overflows and invalid heap use. Requires gcc 4.0 or newer.

When using this compiler option, you must specify linker option `-lmudflap` in the link command line to resolve references to the `libmudflap` library.

**Alternate Options**

None

**fno-gnu-keywords**

*Do not recognize `typeof` as keyword.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `fno-gnu-keywords`

**Windows:**

None

**Arguments**

None

**Default**

OFF

**Description**

Do not recognize `typeof` as keyword.
Alternate Options

None

**fno-implicit-inline-templates**
* Tells the compiler to not emit code for implicit instantiations of inline templates.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-fno-implicit-inline-templates

**Windows:**

None

**Arguments**

None

**Default**

OFF

The compiler handles inlines so that compilations, with and without optimization, will need the same set of explicit instantiations.

**Description**

This option tells the compiler to not emit code for implicit instantiations of inline templates.

Alternate Options

None
**fno-implicit-templates**
*Tells the compiler to not emit code for non-inline templates that are instantiated implicitly.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
-fno-implicit-templates

**Windows:**
None

**Arguments**
None

**Default**
OFF

The compiler handles inlines so that compilations, with and without optimization, will need the same set of explicit instantiations.

**Description**
This option tells the compiler to not emit code for non-inline templates that are instantiated implicitly, but to only emit code for explicit instantiations.

**Alternate Options**
None
fno-operator-names

Disables support for the operator names specified in the standard.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-fno-operator-names

Windows:

None

Arguments

None

Default

OFF

Description

Disables support for the operator names specified in the standard.

Alternate Options

None

fno-rtti

Disables support for run-time type information (RTTI).

IDE Equivalent

None
Architectures
IA-32, IA-64 architectures

Syntax
Linux and Mac OS X:
-急于实现了
Windows:
None

Arguments
None

Default
OFF

Description
This option disables support for run-time type information (RTTI).

Alternate Options
None

fnon-call-exceptions

Allows trapping instructions to throw C++ exceptions.

IDEEquivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-急于实现了
-fno-non-call-exceptions

Windows:
None

Arguments
None

Default
-fno-non-call-exceptions  C++ exceptions are not thrown from trapping instructions.

Description
This option allows trapping instructions to throw C++ exceptions. It allows hardware signals generated by trapping instructions to be converted into C++ exceptions and caught using the standard C++ exception handling mechanism. Examples of such signals are SIGFPE (floating-point exception) and SIGSEGV (segmentation violation).

You must write a signal handler that catches the signal and throws a C++ exception. After that, any occurrence of that signal within a C++ try block can be caught by a C++ catch handler of the same type as the C++ exception thrown within the signal handler.

Only signals generated by trapping instructions (that is, memory access instructions and floating-point instructions) can be caught. Signals that can occur at any time, such as SIGALRM, cannot be caught in this manner.

Alternate Options
None

fnon-lvalue-assign
Determines whether casts and conditional expressions can be used as lvalues.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:

-fnon-lvalue-assign
-fno-non-lvalue-assign

Windows:

None

Arguments

None

Default

-fnon-lvalue-assign  The compiler allows casts and conditional expressions to be used as lvalues.

Description

This option determines whether casts and conditional expressions can be used as lvalues.

Alternate Options

None

fnsplit, Qfnsplit

Enables function splitting.

IDE Equivalent

Windows: Code Generation > Disable Function Splitting
Linux: None
Mac OS X: None

Architectures

/Qfnsplit[-]: IA-32 architecture, Intel® 64 architecture
-[no-]fnsplit: IA-64 architecture
Syntax

Linux:
- fnsplit
- no-fnsplit

Mac OS X:
None

Windows:
/Qfnsplit
/Qfnsplit-

Arguments
None

Default
-no-fnsplit  or /Qfnsplit-  Function splitting is not enabled unless -prof-use (Linux) or /Qprof-use (Windows) is also specified.

Description
This option enables function splitting if -prof-use (Linux) or /Qprof-use (Windows) is also specified. Otherwise, this option has no effect.

It is enabled automatically if you specify -prof-use or /Qprof-use. If you do not specify one of those options, the default is -no-fnsplit (Linux) or /Qfnsplit- (Windows), which disables function splitting but leaves function grouping enabled.

To disable function splitting when you use -prof-use or /Qprof-use, specify -no-fnsplit or /Qfnsplit-.

Alternate Options
None
Fo
Specifies the name for an object file.

IDE Equivalent
Windows: Output Files > Object File Name

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/Fo[file|dir]

Arguments
file Is the name for the object file.

dir Is the directory where the object file should be placed. It can include file.

Default
OFF An object file has the same name as the name of the first source file and a file extension of .obj.

Description
This option specifies the name for an object file.

Alternate Options
None
fomit-frame-pointer, Oy

Determines whether EBP is used as a general-purpose register in optimizations.

IDE Equivalent

Windows: Optimization > Omit Frame Pointers
Linux: Optimization > Provide Frame Pointer
Mac OS X: Optimization > Provide Frame Pointer

Architectures

-f[no-]omit-frame-pointer: IA-32 architecture, Intel® 64 architecture
/Oy[-]: IA-32 architecture

Syntax

Linux and Mac OS X:
-fomit-frame-pointer
-fno-omit-frame-pointer

Windows:
/o y
/o y-

Arguments

None

Default

-fomit-frame-pointer or /Oy

EBP is used as a general-purpose register in optimizations. However, on Linux* and Mac OS X systems, the default is -fno-omit-frame-pointer if option -00 or -g is specified. On Windows* systems, the default is /Oy- if option /Od is specified.
Description

These options determine whether EBP is used as a general-purpose register in optimizations. Options `-fomit-frame-pointer` and `/Oy` allow this use. Options `-fno-omit-frame-pointer` and `/Oy-` disallow it.

Some debuggers expect EBP to be used as a stack frame pointer, and cannot produce a stack backtrace unless this is so. The `-fno-omit-frame-pointer` and `/Oy-` options direct the compiler to generate code that maintains and uses EBP as a stack frame pointer for all functions so that a debugger can still produce a stack backtrace without doing the following:

- For `-fno-omit-frame-pointer`: turning off optimizations with `-O0`
- For `/Oy-`: turning off `/O1`, `/O2`, or `/O3` optimizations

The `-fno-omit-frame-pointer` option is set when you specify option `-O0` or the `-g` option. The `-fomit-frame-pointer` option is set when you specify option `-O1`, `-O2`, or `-O3`.

The `/Oy` option is set when you specify the `/O1`, `/O2`, or `/O3` option. Option `/Oy-` is set when you specify the `/Od` option.

Using the `-fno-omit-frame-pointer` or `/Oy-` option reduces the number of available general-purpose registers by 1, and can result in slightly less efficient code.

NOTE. There is currently an issue with GCC 3.2 exception handling. Therefore, the Intel compiler ignores this option when GCC 3.2 is installed for C++ and exception handling is turned on (the default).

Alternate Options

Linux and Mac OS X: `-fp` (this is a deprecated option)

Windows: None

`fp-model, fp`

Controls the semantics of floating-point calculations.

IDE Equivalent

Windows: Code Generation>Floating Point Model

Code Generation>Enable Floating Point Exceptions
Code Generation> Floating Point Expression Evaluation

Linux: Floating Point > Floating Point Model
Mac OS X: Floating Point > Floating Point Model

Floating Point > Reliable Floating Point Exceptions Model

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fp-model keyword

Windows:
/fp:keyword

Arguments

keyword

Specifies the semantics to be used. Possible values are:

- **precise** Enables value-safe optimizations on floating-point data.
- **fast[1|2]** Enables more aggressive optimizations on floating-point data.
- **strict** Enables precise and except, disables contractions, and enables pragma stdc fenv_access.
- **source** Rounds intermediate results to source-defined precision and enables value-safe optimizations.
- **double** Rounds intermediate results to 53-bit (double) precision.
- **extended** Rounds intermediate results to 64-bit (extended) precision.
[no-]except  Determines whether floating-point exception semantics are used. (Linux and Mac OS X) or except[=] (Windows)

Default

-fp-model fast=1 or /fp:fast=1  The compiler uses more aggressive optimizations on floating-point calculations.

Description

This option controls the semantics of floating-point calculations.

The *keywords* can be considered in groups:

- **Group A**: precise, fast, strict
- **Group B**: source, double, extended
- **Group C**: except (or the negative form)

You can use more than one *keyword*. However, the following rules apply:

- You cannot specify fast and except together in the same compilation. You can specify any other combination of group A, group B, and group C. Since fast is the default, you must not specify except without a group A or group B *keyword*.
- You should specify only one *keyword* from group A and only one keyword from group B. If you try to specify more than one *keyword* from either group A or group B, the last (rightmost) one takes effect.
- If you specify except more than once, the last (rightmost) one takes effect.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fp-model precise or /fp:precise</td>
<td>Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations. It disables optimizations that can change the result of floating-point calculations, which is required for strict ANSI conformance. These semantics</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>ensure the accuracy of floating-point computations, but they may slow performance.</td>
</tr>
<tr>
<td></td>
<td>The compiler assumes the default floating-point environment; you are not allowed to modify it.</td>
</tr>
<tr>
<td></td>
<td>Intermediate results are computed with the precision shown in the following table, unless it is overridden by a keyword from Group B:</td>
</tr>
<tr>
<td>Windows</td>
<td>Linux</td>
</tr>
<tr>
<td>IA-32 architecture</td>
<td>Double</td>
</tr>
<tr>
<td>Intel® 64 architecture</td>
<td>Source</td>
</tr>
<tr>
<td>IA-64 architecture</td>
<td>Extended</td>
</tr>
<tr>
<td></td>
<td>Floating-point exception semantics are disabled by default. To enable these semantics, you must also specify -fp-model except or /fp:except.</td>
</tr>
<tr>
<td></td>
<td>For information on the semantics used to interpret floating-point calculations in the source code, see precise in Floating-point Operations: Using the -fp-model (/fp) Option.</td>
</tr>
<tr>
<td>-fp-model fast[=1</td>
<td>2] or /fp:fast[=1</td>
</tr>
</tbody>
</table>
### Option

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying fast is the same as specifying \texttt{fast=1}. \texttt{fast=2} may produce faster and less accurate results. Floating-point exception semantics are disabled by default and they cannot be enabled because you cannot specify fast and except together in the same compilation. To enable exception semantics, you must explicitly specify another keyword (see other keyword descriptions for details). For information on the semantics used to interpret floating-point calculations in the source code, see \texttt{fast} in \textit{Floating-point Operations: Using the -fp-model (/fp) Option}.</td>
</tr>
</tbody>
</table>

- **\texttt{-fp-model strict} or \texttt{/fp:strict}**
  
  Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations and enables floating-point exception semantics. This is the strictest floating-point model. The compiler does not assume the default floating-point environment; you are allowed to modify it. Floating-point exception semantics can be disabled by explicitly specifying \texttt{-fp-model no-except} or \texttt{/fp:except-}. For information on the semantics used to interpret floating-point calculations in the source code, see \texttt{strict} in \textit{Floating-point Operations: Using the -fp-model (/fp) Option}. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-fp-model source</code> or <code>/fp:source</code></td>
<td>This option causes intermediate results to be rounded to the precision defined in the source code. It also implies keyword <code>precise</code> unless it is overridden by a keyword from Group A. Intermediate expressions use the precision of the operand with higher precision, if any.</td>
</tr>
<tr>
<td>long</td>
<td>64-bit precision 80-bit data type 15-bit exponent</td>
</tr>
<tr>
<td><code>double</code></td>
<td>53-bit precision 64-bit data type 11-bit exponent; on Windows systems using IA-32 architecture, the exponent may be 15-bit if an x87 register is used to hold the value.</td>
</tr>
<tr>
<td><code>float</code></td>
<td>24-bit precision 32-bit data type 8-bit exponent</td>
</tr>
</tbody>
</table>

The compiler assumes the default floating-point environment; you are not allowed to modify it.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For information on the semantics used to interpret floating-point calculations in the source code, see source in <em>Floating-point Operations: Using the -fp-model (/fp) Option</em>.</td>
</tr>
<tr>
<td>-fp-model double or /fp:double</td>
<td>This option causes intermediate results to be rounded as follows: 53-bit (double) precision 64-bit data type 11-bit exponent; on Windows systems using IA-32 architecture, the exponent may be 15-bit if an x87 register is used to hold the value. This option also implies keyword precise unless it is overridden by a keyword from Group A. The compiler assumes the default floating-point environment; you are not allowed to modify it. For information on the semantics used to interpret floating-point calculations in the source code, see double in <em>Floating-point Operations: Using the -fp-model (/fp) Option</em>.</td>
</tr>
<tr>
<td>-fp-model extended or /fp:extended</td>
<td>This option causes intermediate results to be rounded as follows: 64-bit (extended) precision 80-bit data type 15-bit exponent</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-fp-model except or /fp:except</td>
<td>Tells the compiler to use floating-point exception semantics.</td>
</tr>
</tbody>
</table>

**NOTE.** On Windows and Linux operating systems on IA-32 architecture, the compiler, by default, implements floating-point (FP) arithmetic using SSE2 and SSE instructions. This can cause differences in floating-point results when compared to previous x87 implementations.

**Alternate Options**
None

**Example**
For examples of how to use this option, see *Floating-point Operations: Using the -fp-model (/fp) Option*.

**See Also**
-  
-  
- O
- Od
- mp1, Qprec
Floating-point Environment

The MSDN article Microsoft Visual C++ Floating-Point Optimization, which discusses concepts that apply to this option.

Fp

*Let you specify an alternate path or file name for precompiled header files.*

IDE Equivalent

Windows: **Precompiled Headers > Precompiled Header File**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

None

**Windows:**

/Fp{file|dir}

Arguments

file 

Is the name for the precompiled header file.

dir 

Is the directory where the precompiled header file should be placed. It can include file.

Default

OFF 

The compiler does not create or use precompiled headers unless you tell it to do so.

Description

This option lets you specify an alternate path or file name for precompiled header files.

Alternate Options

None
fp-model, fp
Controls the semantics of floating-point calculations.

IDE Equivalent
Windows: Code Generation>Floating Point Model
Code Generation>Enable Floating Point Exceptions
Code Generation> Floating Point Expression Evaluation
Linux: Floating Point > Floating Point Model
Mac OS X: Floating Point > Floating Point Model
Floating Point > Reliable Floating Point Exceptions Model

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fp-model keyword

Windows:
/fp:keyword

Arguments
keyword Specifies the semantics to be used. Possible values are:
precise Enables value-safe optimizations on floating-point data.
fast[=1|2] Enables more aggressive optimizations on floating-point data.
strict Enables precise and except, disables contractions, and enables pragma stdc fenv_access.
source        Rounds intermediate results to source-defined precision and enables value-safe optimizations.

double       Rounds intermediate results to 53-bit (double) precision.

extended     Rounds intermediate results to 64-bit (extended) precision.

[no-]except (Linux and Mac OS X) or except[-] (Windows) Determines whether floating-point exception semantics are used.

Default

-fp-model fast=1 or /fp:fast=1 The compiler uses more aggressive optimizations on floating-point calculations.

Description

This option controls the semantics of floating-point calculations.

The *keywords* can be considered in groups:

- **Group A:** precise, fast, strict
- **Group B:** source, double, extended
- **Group C:** except (or the negative form)

You can use more than one *keyword*. However, the following rules apply:

- You cannot specify fast and except together in the same compilation. You can specify any other combination of group A, group B, and group C. Since fast is the default, you must not specify except without a group A or group B *keyword*.
- You should specify only one *keyword* from group A and only one *keyword* from group B. If you try to specify more than one *keyword* from either group A or group B, the last (rightmost) one takes effect.
- If you specify except more than once, the last (rightmost) one takes effect.
Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations. It disables optimizations that can change the result of floating-point calculations, which is required for strict ANSI conformance. These semantics ensure the accuracy of floating-point computations, but they may slow performance.

The compiler assumes the default floating-point environment; you are not allowed to modify it.

Intermediate results are computed with the precision shown in the following table, unless it is overridden by a keyword from Group B:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Windows</th>
<th>Linux</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-32</td>
<td>Double</td>
<td>Extended</td>
<td>Extended</td>
</tr>
<tr>
<td>Intel® 64</td>
<td>Source</td>
<td>Source</td>
<td>Source</td>
</tr>
<tr>
<td>IA-64</td>
<td>Extended</td>
<td>Extended</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Floating-point exception semantics are disabled by default. To enable these semantics, you must also specify `-fp-model except` or `/fp:except`.

For information on the semantics used to interpret floating-point calculations in the source code, see `precise` in *Floating-point Operations: Using the -fp-model (/fp) Option*. 

### Option Description

- `-fp-model precise` or `/fp:precise`
  
  Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations. It disables optimizations that can change the result of floating-point calculations, which is required for strict ANSI conformance. These semantics ensure the accuracy of floating-point computations, but they may slow performance.

  The compiler assumes the default floating-point environment; you are not allowed to modify it.

  Intermediate results are computed with the precision shown in the following table, unless it is overridden by a keyword from Group B:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Windows</th>
<th>Linux</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-32</td>
<td>Double</td>
<td>Extended</td>
<td>Extended</td>
</tr>
<tr>
<td>Intel® 64</td>
<td>Source</td>
<td>Source</td>
<td>Source</td>
</tr>
<tr>
<td>IA-64</td>
<td>Extended</td>
<td>Extended</td>
<td>N/A</td>
</tr>
</tbody>
</table>

  Floating-point exception semantics are disabled by default. To enable these semantics, you must also specify `-fp-model except` or `/fp:except`.

  For information on the semantics used to interpret floating-point calculations in the source code, see `precise` in *Floating-point Operations: Using the -fp-model (/fp) Option*. 


<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fp-model fast[=1</td>
<td>2]</td>
</tr>
<tr>
<td>/fp:fast[=1</td>
<td>2]</td>
</tr>
<tr>
<td>-fp-model strict</td>
<td>Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations and enables floating-point exception semantics. This is the strictest floating-point model. The compiler does not assume the default floating-point environment; you are allowed to modify it. Floating-point exception semantics can be disabled by explicitly specifying -fp-model no-except or /fp:except-.</td>
</tr>
<tr>
<td>/fp:strict</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-fp-model source or /fp:source</td>
<td>This option causes intermediate results to be rounded to the precision defined in the source code. It also implies keyword precise unless it is overridden by a keyword from Group A. Intermediate expressions use the precision of the operand with higher precision, if any.</td>
</tr>
<tr>
<td>long double precision</td>
<td>64-bit 80-bit data type 15-bit exponent</td>
</tr>
<tr>
<td>double precision</td>
<td>53-bit 64-bit data type 11-bit exponent; on Windows systems using IA-32 architecture, the exponent may be 15-bit if an x87 register is used to hold the value.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>float</code></td>
<td>24-bit precision 32-bit data type 8-bit exponent</td>
</tr>
</tbody>
</table>

The compiler assumes the default floating-point environment; you are not allowed to modify it.

For information on the semantics used to interpret floating-point calculations in the source code, see `source` in *Floating-point Operations: Using the `–fp-model (/fp)` Option*.

`–fp-model double` or `/fp:double`  
This option causes intermediate results to be rounded as follows:

- 53-bit (double) precision
- 64-bit data type
- 11-bit exponent; on Windows systems using IA-32 architecture, the exponent may be 15-bit if an x87 register is used to hold the value.

This option also implies keyword `precise` unless it is overridden by a keyword from Group A.

The compiler assumes the default floating-point environment; you are not allowed to modify it.

For information on the semantics used to interpret floating-point calculations in the source code, see `double` in *Floating-point Operations: Using the `–fp-model (/fp)` Option*.
This option causes intermediate results to be rounded as follows:
- 64-bit (extended) precision
- 80-bit data type
- 15-bit exponent

This option also implies keyword `precise` unless it is overridden by a keyword from Group A.

The compiler assumes the default floating-point environment; you are not allowed to modify it.

For information on the semantics used to interpret floating-point calculations in the source code, see `double` in Floating-point Operations: Using the `-fp-model (/fp)` Option.

Tells the compiler to use floating-point exception semantics.

**NOTE.** On Windows and Linux operating systems on IA-32 architecture, the compiler, by default, implements floating-point (FP) arithmetic using SSE2 and SSE instructions. This can cause differences in floating-point results when compared to previous x87 implementations.

**Alternate Options**
None

**Example**
For examples of how to use this option, see Floating-point Operations: Using the `-fp-model (/fp)` Option.
See Also

- o
- Od
- mp1, Qprec
- Floating-point Environment

The MSDN article Microsoft Visual C++ Floating-Point Optimization, which discusses concepts that apply to this option.

fp-port, Qfp-port

Rounds floating-point results after floating-point operations.

IDE Equivalent

Windows: Optimization > Floating-point Precision Improvements
Linux: Floating Point > Round Floating-Point Results
Mac OS X: Floating Point > Round Floating-Point Results

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- -fp-port
- -no-fp-port

Windows:
/ Qfp-port
/ Qfp-port-

Arguments

None
Default
-no-fp-port  or /Qfp-port-
The default rounding behavior depends on the compiler's code generation decisions and the precision parameters of the operating system.

Description
This option rounds floating-point results after floating-point operations. Rounding to user-specified precision occurs at assignments and type conversions. This has some impact on speed.
The default is to keep results of floating-point operations in higher precision. This provides better performance but less consistent floating-point results.

Alternate Options
None

fp-relaxed, Qfp-relaxed
Enables use of faster but slightly less accurate code sequences for math functions.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux:
-fp-relaxed
-no-fp-relaxed
Mac OS X:
None
Windows:
/Qfp-relaxed
/Qfp-relaxed-

Arguments
None

Default
-no-fp-relaxed
or /Qfp-relaxed-

Default code sequences are used for math functions.

Description
This option enables use of faster but slightly less accurate code sequences for math functions, such as divide and sqrt. When compared to strict IEEE* precision, this option slightly reduces the accuracy of floating-point calculations performed by these functions, usually limited to the least significant digit.

This option also enables the performance of more aggressive floating-point transformations, which may affect accuracy.

Alternate Options
Linux: -IPF-fp-relaxed (this is a deprecated option)
Windows: /QIPF-fp-relaxed (this is a deprecated option)

See Also
•
•
• fp-model, fp

fp-speculation, Qfp-speculation
Tells the compiler the mode in which to speculate on floating-point operations.

IDE Equivalent
Windows: Optimization > Floating-Point Speculation
Linux: Floating Point > Floating-Point Speculation
Mac OS X: Floating Point > Floating-Point Speculation
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fp-speculation=mode

Windows:
/Qfp-speculation:mode

Arguments

mode Is the mode for floating-point operations. Possible values are:

- fast Tells the compiler to speculate on floating-point operations.
- safe Tells the compiler to disable speculation if there is a possibility that the speculation may cause a floating-point exception.
- strict Tells the compiler to disable speculation on floating-point operations.
- off This is the same as specifying strict.

Default

-fp-speculation=fast or /Qfp-speculation:fast  The compiler speculates on floating-point operations. This is also the behavior when optimizations are enabled. However, if you specify no optimizations (-O0 on Linux; /Od on Windows), the default is -fp-speculation=safe (Linux) or /Qfp-speculation:safe (Windows).

Description
This option tells the compiler the mode in which to speculate on floating-point operations.

Alternate Options
None
fp-stack-check, Qfp-stack-check

Tells the compiler to generate extra code after every function call to ensure that the floating-point stack is in the expected state.

IDE Equivalent

Windows: None
Linux: Floating Point > Check Floating-point Stack
Mac OS X: Floating Point > Check Floating-point Stack

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:
-fp-stack-check

Windows:
/Qfp-stack-check

Arguments

None

Default

OFF

There is no checking to ensure that the floating-point (FP) stack is in the expected state.

Description

This option tells the compiler to generate extra code after every function call to ensure that the floating-point (FP) stack is in the expected state.

By default, there is no checking. So when the FP stack overflows, a NaN value is put into FP calculations and the program's results differ. Unfortunately, the overflow point can be far away from the point of the actual bug. This option places code that causes an access violation exception immediately after an incorrect call occurs, thus making it easier to locate these issues.
**Alternate Options**
None

**See Also**
- Floating-point Operations:
  - Checking the Floating-point Stack State

**fpack-struct**
Specifies that structure members should be packed together.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

- **Linux and Mac OS X:**
  - `-fpack-struct`

- **Windows:**
  - None

**Arguments**
None

**Default**
OFF

**Description**
Specifies that structure members should be packed together. Note: Using this option may result in code that is not usable with standard (system) c and C++ libraries.
Alternate Options

-zp1

**fpascal-strings**

*Allow for Pascal-style string literals.*

**IDE Equivalent**

Windows: None

Linux: **Data > Recognize Pascal Strings**

Mac OS X: None

**Architectures**

IA-32 architecture

**Syntax**

Linux:
None

Mac OS X:

- **fpascal-strings**

Windows:
None

**Arguments**

None

**Default**

OFF  

The compiler does not allow for Pascal-style string literals.

**Description**

Allow for Pascal-style string literals.

**Alternate Options**

None
**fpermissive**
*Allow for non-conformant code.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `fpermissive`

**Windows:**
None

**Arguments**
None

**Default**
OFF

**Description**
Allow for non-conformant code.

**Alternate Options**
None

**fpic**
*Determines whether the compiler generates position-independent code.*

**IDE Equivalent**
Windows: None
Linux: **Code Generation > Generate Position Independent Code**
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- `-fpic`
- `-fno-pic`

Windows:
None

**Arguments**
None

**Default**
- `-fno-pic` or `-fpic`

On systems using IA-32 or Intel® 64 architecture, the compiler does not generate position-independent code. On systems using IA-64 architecture, the compiler generates position-independent code.

**Description**

This option determines whether the compiler generates position-independent code.

Option `-fpic` specifies full symbol preemption. Global symbol definitions as well as global symbol references get default (that is, preemptable) visibility unless explicitly specified otherwise.

Option `-fno-pic` is only valid on systems using IA-32 or Intel® 64 architecture.

On systems using IA-32 or Intel® 64 architecture, `-fpic` must be used when building shared objects.

This option can also be specified as `-fPIC`.

**Alternate Options**

None
**fpie**

*Tells the compiler to generate position-independent code. The generated code can only be linked into executables.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux:*
- `-fpie`

*Mac OS X:*
None

*Windows:*
None

**Arguments**
None

**Default**
OFF

The compiler does not generate position-independent code for an executable-only object.

**Description**

This option tells the compiler to generate position-independent code. It is similar to `-fpic`, but code generated by `-fpie` can only be linked into an executable.

Because the object is linked into an executable, this option causes better optimization of some symbol references.

To ensure that run-time libraries are set up properly for the executable, you should also specify option `-pie` to the compiler driver on the link command line.
Option -fpie can also be specified as -fPIE.

Alternate Options
None

See Also
•
• fpic
• pie

Fr
Invokes the Microsoft C/C++ compiler and tells it to produce a BSCMAKE .sbr file without information on local variables. This is a deprecated option.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/Fr[file|dir]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Is the name for the BSCMAKE .sbr file.</td>
</tr>
<tr>
<td>dir</td>
<td>Is the directory where the file should be placed. It can include file.</td>
</tr>
</tbody>
</table>
Default

OFF The compiler does not invoke the Microsoft C/C++ compiler to produce a .sbr file.

Description

This option invokes the Microsoft C/C++ compiler and tells it to produce a BSCMAKE .sbr file without information on local variables.

You can provide a name for the file. If you do not specify a file name, the .sbr file gets the same base name as the source file.

Alternate Options

None

See Also

• FR

FR

Invokes the Microsoft C/C++ compiler and tells it to produce a BSCMAKE .sbr file with complete symbolic information.

IDE Equivalent

Windows: Browse Information > Enable Browse Information

Linux: None

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

None
**Windows:**

FR[file|dir]

**Arguments**

replace file Is the name for the BSCMAKE .sbr file.  
dir Is the directory where the file should be placed. It can include file.

**Default**

OFF The compiler does not invoke the Microsoft C/C++ compiler to produce a .sbr file.

**Description**

This option invokes the Microsoft C/C++ compiler and tells it to produce a BSCMAKE .sbr file with complete symbolic information.

You can provide a name for the file. If you do not specify a file name, the .sbr file gets the same base name as the source file.

**Alternate Options**

None

**See Also**

•  
•  [fr]

**fr32**

Disables the use of the high floating-point registers.

**IDE Equivalent**

Windows: None

Linux: **Floating Point > Disable Use of High Floating-point Registers**

Mac OS X: None
Architectures
IA-64 architecture

Syntax
Linux:
-fr32
Mac OS X:
None
Windows:
None

Arguments
None

Default
OFF The use of the high floating-point registers is enabled.

Description
This option disables the use of the high floating-point registers. Only the lower 32 floating-point registers are used.

Alternate Options
None

freg-struct-return
Return struct and union values in registers when possible.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
-freg-struct-return

Windows:
None

Arguments

None

Default

OFF

Description

Return struct and union values in registers when possible.

Alternate Options

None

fshort-enums

Tells the compiler to allocate as many bytes as needed for enumerated types.

IDE Equivalent

Windows: None

Linux: Data > Associate as Many Bytes as Needed for Enumerated Types

Mac OS X: Data > Allocate enumerated types

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-fshort-enums
Windows: None

Arguments None

Default OFF

The compiler allocates a default number of bytes for enumerated types.

Description
This option tells the compiler to allocate as many bytes as needed for enumerated types.

Alternate Options
None

fsource-asm
Produces an assembly listing with source code annotations.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fsource-asm

Windows:
None
**Default**

OFF  
No source code annotations appear in the assembly listing file, if one is produced.

**Description**

This option produces an assembly listing file with source code annotations. The assembly listing file shows the source code as interspersed comments.

To use this option, you must also specify option \(-s\), which causes an assembly listing to be generated.

**Alternate Options**

None

**See Also**

- \(s\)
- \texttt{fstack-security-check, GS}

Determines whether the compiler generates code that detects some buffer overruns.

**IDE Equivalent**

Windows: \textbf{Code Generation > Buffer Security Check}

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel\textsuperscript{®} 64 architectures

**Syntax**

\texttt{Linux and Mac OS X:}

- \texttt{-fstack-security-check}
- \texttt{-fno-stack-security-check}
Windows:
/\texttt{GS}
\texttt{/GS-}

**Arguments**
None

**Default**
\texttt{-fno-stack-security-check} \quad \text{The compiler does not detect buffer overruns.}
\texttt{or /GS-}

**Description**
This option determines whether the compiler generates code that detects some buffer overruns that overwrite the return address. This is a common technique for exploiting code that does not enforce buffer size restrictions.

The \texttt{/GS} option is supported with Microsoft Visual Studio .NET 2003* and Microsoft Visual Studio 2005*.

**Alternate Options**
Linux and Mac OS X: \texttt{-f[no-]stack-protector}
Windows: None

\texttt{fstack-security-check, GS}

\textit{Determines whether the compiler generates code that detects some buffer overruns.}

**IDE Equivalent**
Windows: \textbf{Code Generation > Buffer Security Check}
Linux: None
Mac OS X: None

**Architectures**
IA-32, Intel\textsuperscript{	extregistered} 64 architectures
Syntax

Linux and Mac OS X:
- `fstack-security-check`
- `fno-stack-security-check`

Windows:
`/GS`
`/GS-`

Arguments
None

Default
- `fno-stack-security-check` or `/GS-`

The compiler does not detect buffer overruns.

Description
This option determines whether the compiler generates code that detects some buffer overruns that overwrite the return address. This is a common technique for exploiting code that does not enforce buffer size restrictions.

The `/GS` option is supported with Microsoft Visual Studio .NET 2003* and Microsoft Visual Studio 2005*.

Alternate Options

Linux and Mac OS X: `-f[no-]stack-protector`

Windows: None

`fsyntax-only`

*Tells the compiler to check only for correct syntax.*

IDE Equivalent

None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fsyntax-only

Windows:
None

Arguments
None

Default
OFF
Normal compilation is performed.

Description
For details, see option syntax.

Alternate Options
/Zs

-ftemplate-depth, Qtemplate-depth
Control the depth in which recursive templates are expanded.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-ftemplate-depth-n
Windows:
/QTtemplate-depth-n

Arguments

\(n\)  The number of recursive templates that are expanded.

Default
OFF

Description
Control the depth in which recursive templates are expanded. On Linux*, this option is supported only by invoking the compiler with icpc.

Alternate Options
None

\texttt{ftls-model}

\textit{Change thread local storage model.}

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-\texttt{ftls-model=model}

Windows:
None

Arguments

\texttt{model}  Possible values are:
global-dynamic
local-dynamic
initial-exec
local-exec

Default
OFF

Description
Change thread local storage model.

Alternate Options
None

ftrapuv, Qtrapuv
*Initializes stack local variables to an unusual value to aid error detection.*

IDE Equivalent
Windows: Code Generation > Initialize Local Variables to NaN
Linux: Code Generation > Initialize Local Variables to NaN
Mac OS X: Code Generation > Initialize Local Variables to NaN

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-ftrapuv

Windows:
/Qtrapuv
Arguments
None

Default
OFF

The compiler does not initialize local variables.

Description
This option initializes stack local variables to an unusual value to aid error detection. Normally, these local variables should be initialized in the application.

The option sets any uninitialized local variables that are allocated on the stack to a value that is typically interpreted as a very large integer or an invalid address. References to these variables are then likely to cause run-time errors that can help you detect coding errors.

This option sets option -g (Linux and Mac OS X) and /zi or /Z7 (Windows).

Alternate Options
None

See Also
•
•
• $g, zi, Z7$

ftz, Qftz
Flushes denormal results to zero.

IDE Equivalent
Windows: Optimization > Flush Denormal Results to Zero
Linux: Floating-Point > Flush Denormal Results to Zero
Mac OS X: Floating-Point > Flush Denormal Results to Zero

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
- `-ftz`
- `-no-ftz`

Windows:
- `/Qftz`
- `/Qftz-`

Arguments

None

Default

On systems using IA-64 architecture, the compiler lets results gradually underflow. On systems using IA-32 architecture and Intel® 64 architecture, denormal results are flushed to zero.

Systems using IA-64 architecture:
- `-no-ftz`
- `/Qftz-`

Systems using IA-32 architecture and Intel® 64 architecture:
- `-ftz`
- `/Qftz`

Description

This option flushes denormal results to zero when the application is in the gradual underflow mode. It may improve performance if the denormal values are not critical to your application’s behavior.

This option sets or resets the FTZ and the DAZ hardware flags. If FTZ is ON, denormal results from floating-point calculations will be set to the value zero. If FTZ is OFF, denormal results remain as is. If DAZ is OFF, denormal values used as input to floating-point instructions will be treated as zero. If DAZ is ON, denormal instruction inputs remain as is. Systems using IA-64 architecture have FTZ but not DAZ. Systems using Intel® 64 architecture have both FTZ and DAZ. FTZ and DAZ are not supported on all IA-32 architectures.

When `-ftz` (Linux and Mac OS X) or `/Qftz` (Windows) is used in combination with an SSE-enabling option on systems using IA-32 architecture (for example, xN or QxN), the compiler will insert code in the main routine to set FTZ and DAZ. When `-ftz` or `/Qftz` is used without
such an option, the compiler will insert code to conditionally set FTZ/DAZ based on a run-time processor check. `-no-ftz` (Linux and Mac OS X) or `/Qftz-` (Windows) will prevent the compiler from inserting any code that might set FTZ or DAZ.

This option only has an effect when the main program is being compiled. It sets the FTZ/DAZ mode for the process. The initial thread and any threads subsequently created by that process will operate in FTZ/DAZ mode.

On systems using IA-64 architecture, optimization option `O3` sets `-ftz` and `/Qftz`; optimization option `O2` sets `-no-ftz` (Linux) and `/Qftz-` (Windows). On systems using IA-32 architecture and Intel® 64 architecture, every optimization option `O` level, except `O0`, sets `-ftz` and `/Qftz`.

If this option produces undesirable results of the numerical behavior of your program, you can turn the FTZ/DAZ mode off by using `-no-ftz` or `/Qftz-` in the command line while still benefiting from the `O3` optimizations.

**NOTE.** Options `-ftz` and `/Qftz` are performance options. Setting these options does not guarantee that all denormals in a program are flushed to zero. They only cause denormals generated at run time to be flushed to zero.

**Alternate Options**

None

**Example**

To see sample code showing the state of the FTZ and DAZ flags, see Reading the FTZ and DAZ Flags.

**See Also**

•

•

• x, Qx
**func-groups**

This is a deprecated option. See prof-func-groups.

**funroll-loops**

See unroll, Qunroll.

**funroll-all-loops**

Unroll all loops even if the number of iterations is uncertain when the loop is entered.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

-funroll-all-loops

Windows:

None

**Arguments**

None

**Default**

OFF  Do not unroll all loops.

**Description**

Unroll all loops, even if the number of iterations is uncertain when the loop is entered. There may a performance impact with this option.

**Alternate Options**

None
**funsigned-bitfields**

*Determines whether the default bitfield type is changed to* unsigned.

**IDE Equivalent**

Windows: None

Linux: **Data > Change Default Bitfield Type to unsigned**

Mac OS X: **Data > Unsigned bitfield Type**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `funsigned-bitfields`
- `fno-unsigned-bitfields`

**Windows:**

None

**Arguments**

None

**Default**

`fno-unsigned-bitfields` The default bitfield type is signed.

**Description**

This option determines whether the default bitfield type is changed to unsigned.

**Alternate Options**

None
**funsigned-char**

*Change default char type to unsigned.*

**IDE Equivalent**

Windows: None

Linux: Data > Change default char type to unsigned

Mac OS X: Data > Unsigned char Type

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

-funsigned-char

Windows:

None

**Arguments**

None

**Default**

OFF  

Do not change default char type to unsigned.

**Description**

Change default char type to unsigned.

**Alternate Options**

None
fverbose-asm

Produces an assembly listing with compiler comments, including options and version information.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fverbose-asm
-fno-verbose-asm

Windows:
None

Arguments
None

Default
-fno-verbose-asm

No source code annotations appear in the assembly listing file, if one is produced.

Description
This option produces an assembly listing file with compiler comments, including options and version information.

To use this option, you must also specify -S, which sets -fverbose-asm.

If you do not want this default when you specify -S, specify -fno-verbose-asm.

Alternate Options
None
See Also

- \( S \)

**fvisibility**

Specifies the default visibility for global symbols or the visibility for symbols in a file.

**IDE Equivalent**

Windows: None

Linux: **Data > Default Symbol Visibility**

Mac OS X: **Data > Default Symbol Visibility**

**Architectures**

IA-32, Intel\(^\text{®}\) 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

\[-fvisibility=\text{keyword}\]

\[-fvisibility-\text{keyword}=\text{file}\]

**Windows:**

None

**Arguments**

\( \text{keyword} \)

Specifies the visibility setting. Possible values are:

- **default**
  - Sets visibility to default.
- **extern**
  - Sets visibility to extern.
- **hidden**
  - Sets visibility to hidden.
- **internal**
  - Sets visibility to internal.
- **protected**
  - Sets visibility to protected.
Is the pathname of a file containing the list of symbols whose visibility you want to set. The symbols must be separated by whitespace (spaces, tabs, or newlines).

**Default**

- `-fvisibility=default` The compiler sets visibility of symbols to default.

**Description**

This option specifies the default visibility for global symbols (syntax `-fvisibility=keyword`) or the visibility for symbols in a file (syntax `-fvisibility-keyword=file`). Visibility specified by `-fvisibility-keyword=file` overrides visibility specified by `-fvisibility=keyword` for symbols specified in a file.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-fvisibility=default</code></td>
<td>Sets visibility of symbols to default. This means other components can reference the symbol, and the symbol definition can be overridden (preempted) by a definition of the same name in another component.</td>
</tr>
<tr>
<td><code>-fvisibility-default=file</code></td>
<td></td>
</tr>
<tr>
<td><code>-fvisibility=extern</code></td>
<td>Sets visibility of symbols to extern. This means the symbol is treated as though it is defined in another component. It also means that the symbol can be overridden by a definition of the same name in another component.</td>
</tr>
<tr>
<td><code>-fvisibility-extern=file</code></td>
<td></td>
</tr>
<tr>
<td><code>-fvisibility=hidden</code></td>
<td>Sets visibility of symbols to hidden. This means that other components cannot directly reference the symbol. However, its address may be passed to other components indirectly.</td>
</tr>
<tr>
<td><code>-fvisibility-hidden=file</code></td>
<td></td>
</tr>
<tr>
<td><code>-fvisibility=internal</code></td>
<td>Sets visibility of symbols to internal. This means the symbol cannot be referenced outside its defining component, either directly or indirectly.</td>
</tr>
<tr>
<td><code>-fvisibility-internal=file</code></td>
<td></td>
</tr>
</tbody>
</table>
If an -fvisibility option is specified more than once on the command line, the last specification takes precedence over any others.

If a symbol appears in more than one visibility file, the setting with the least visibility takes precedence.

The following shows the precedence of the visibility settings (from greatest to least visibility):

- extern
- default
- protected
- hidden
- internal

Note that extern visibility only applies to functions. If a variable symbol is specified as extern, it is assumed to be default.

Alternate Options

None

Example

A file named prot.txt contains symbols a, b, c, d, and e. Consider the following:

-fvisibility-protected=prot.txt

This option sets protected visibility for all the symbols in the file. It has the same effect as specifying fvisibility=protected in the declaration for each of the symbols.

See Also

- Symbol Visibility Attribute Options (Linux* and Mac OS* X)
**fvisibility-inlines-hidden**

*Causes inline member functions (those defined in the class declaration) to be marked hidden.*

**IDE Equivalent**

None

**Architectures**

IA-32 architecture

**Syntax**

**Linux and Mac OS X:**

-fvisibility-inlines-hidden

**Windows:**

None

**Arguments**

None

**Default**

OFF The compiler does not cause inline member functions to be marked hidden.

**Description**

Causes inline member functions (those defined in the class declaration) to be marked hidden. This option is particularly useful for templates.

**Alternate Options**

None
**g, Zi, Z7**

Tells the compiler to generate full debugging information in the object file.

**IDE Equivalent**

Windows: **General > Debug Information Format**

Linux: **General > Include Debug Information**

Mac OS X: **General > Generate Debug Information**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-g
```

Windows:

```
/zi
/z7
```

**Arguments**

None

**Default**

OFF

No debugging information is produced in the object file.

**Description**

This option tells the compiler to generate symbolic debugging information in the object file for use by debuggers.

The compiler does not support the generation of debugging information in assemblable files. If you specify this option, the resulting object file will contain debugging information, but the assemblable file will not.
This option turns off `-O2` and makes `-O0` (Linux and Mac OS X) or `-Od` (Windows) the default unless `-O2` (or another `-O` option) is explicitly specified in the same command line.

On Linux systems using Intel® 64 architecture and Linux and Mac OS X systems using IA-32 architecture, specifying the `-g` or `-O0` option sets the `-fno-omit-frame-pointer` option.

Alternate Options

Linux: None

Windows: `/ZI`, `/debug`

`g0`

_Disables generation of symbolic debug information._

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

`-g0`

Windows:

None

Arguments

None

Default

OFF  The compiler generates symbolic debug information.

Description

This option disables generation of symbolic debug information.
Alternate Options

None

G2, G2-p9000

Optimizes application performance for systems using IA-64 architecture.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux and Mac OS X:

None

Windows:

/G2
/G2-p9000

Arguments

None

Default

/G2-p9000

Performance is optimized for Dual-Core Intel® Itanium® 2 processor 9000 series.

Description

These options optimize application performance for a particular Intel® processor or family of processors. The compiler generates code that takes advantage of features of IA-64 architecture.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2</td>
<td>Optimizes for Intel® Itanium® 2 processors.</td>
</tr>
</tbody>
</table>
Option | Description
--- | ---
G2-p9000 | Optimizes for Dual-Core Intel® Itanium® 2 processor 9000 series. This option affects the order of the generated instructions, but the generated instructions are limited to Intel® Itanium® 2 processor instructions unless the program specifies and executes intrinsics specific to the Dual-Core Intel® Itanium® 2 processor 9000 series.

The resulting executable is backwards compatible and generated code is optimized for specific processors. For example, code generated with /G2-p9000 will run correctly on single-core Itanium® 2 processors, but it might not run as fast as if it had been generated using /G2.

Alternate Options

/G2 | Linux: -mtune=itanium2
Mac OS X: None
Windows: None
/G2-p9000 | Linux: -mtune=itanium2-p9000, -mcpu=itanium2-p9000
(-mcpu is a deprecated option)
Mac OS X: None
Windows: None

Example

In the following example, the compiled binary of the source program prog.c is optimized for the Dual-Core Intel® Itanium® 2 processor 9000 series by default. The same binary will also run on single-core Itanium® 2 processors (unless the program specifies and executes intrinsics specific to the Dual-Core Intel® Itanium® 2 processor 9000 series). All lines in the code example are equivalent.

icl prog.c
icl /G2-p9000 prog.c

In the following example, the compiled binary is optimized for single-core Itanium® 2 processors:

icl /G2 prog.c

See Also
•
G5, G6, G7
Optimize application performance for systems using IA-32 architecture and Intel® 64 architecture. These are deprecated options.

IDE Equivalent
Windows: Optimization > Optimize for Processor
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/G5
/G6
/G7

Arguments
None

Default
/G7

On systems using IA-32 architecture and Intel® 64 architecture, performance is optimized for Intel® Pentium® 4 processors, Intel® Xeon® processors, Intel® Pentium® M processors, and Intel® Pentium® 4 processors with Streaming SIMD Extensions 3 (SSE3) instruction support.
These options optimize application performance for a particular Intel® processor or family of processors. The compiler generates code that takes advantage of features of the specified processor.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G5</td>
<td>Optimizes for Intel® Pentium® and Pentium® with MMX™ technology processors.</td>
</tr>
<tr>
<td>G6</td>
<td>Optimizes for Intel® Pentium® Pro, Pentium® II and Pentium® III processors.</td>
</tr>
<tr>
<td>G7</td>
<td>Optimizes for Intel® Core™ Duo processors, Intel® Core™ Solo processors, Intel® Pentium® 4 processors, Intel® Xeon® processors based on the Intel® Core microarchitecture, Intel® Pentium® M processors, and Intel® Pentium® 4 processors with Streaming SIMD Extensions 3 (SSE3) instruction support.</td>
</tr>
</tbody>
</table>

On systems using Intel® 64 architecture, only option G7 is valid.

These options always generate code that is backwards compatible with Intel processors of the same architecture. For example, code generated with the G7 option runs correctly on Pentium III processors, although performance may be faster on Pentium III processors when compiled using or G6.

**Alternate Options**

Windows: /GB (an alternate for /G6; this option is also deprecated)

Linux: None

**Example**

In the following example, the compiled binary of the source program prog.c is optimized, by default, for Intel® Pentium® 4 processors, Intel® Xeon® processors, Intel® Pentium® M processors, and Intel® Pentium® 4 processors with Streaming SIMD Extensions 3 (SSE3). The same binary will also run on Pentium, Pentium Pro, Pentium II, and Pentium III processors. All lines in the code example are equivalent.

icl prog.c

icl /G7 prog.c
In the following example, the compiled binary is optimized for Pentium processors and Pentium processors with MMX technology:
ifort /G5 prog.f
icl /G5 prog.c

**See Also**

- mtune

**GA**

*Enables faster access to certain thread-local storage (TLS) variables.*

**IDE Equivalent**

Windows: **Optimization > Optimize for Windows Applications**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/GA

**Arguments**

None

**Default**

OFF Default access to TLS variables is in effect.
**Description**

This option enables faster access to certain thread-local storage (TLS) variables. When you compile your main executable (.EXE) program with this option, it allows faster access to TLS variables declared with the `__declspec(thread)` specification.

Note that if you use this option to compile .DLLs, you may get program errors.

**Alternate Options**

None

**gcc**

_Determines whether certain GNU macros are defined or undefined._

**IDE Equivalent**

Windows: None

Linux: Preprocessor > gcc Predefined Macro Enablement

Mac OS X: Preprocessor > Predefine gcc Macros

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `gcc`
- `no-gcc`
- `gcc-sys`

Windows:

None

**Arguments**

None
Default

-gcc

The compiler defines the GNU macros __GNUC__, __GNUC_MINOR__, and __GNUC_PATCHLEVEL__

Description

This option determines whether the GNU macros __GNUC__, __GNUC_MINOR__, and __GNUC_PATCHLEVEL__ are defined or defined.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-gcc</td>
<td>Defines GNU macros</td>
</tr>
<tr>
<td>-no-gcc</td>
<td>Undefines GNU macros</td>
</tr>
<tr>
<td>-gcc-sys</td>
<td>Defines GNU macros only during compilation of system headers</td>
</tr>
</tbody>
</table>

Alternate Options

None

gcc

Determines whether certain GNU macros are defined or undefined.

IDE Equivalent

Windows: None

Linux: Preprocessor > gcc Predefined Macro Enablement

Mac OS X: Preprocessor > Predefine gcc Macros

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

- gcc
- no-gcc
-gcc-sys

Windows:
None

Arguments
None

Default
-gcc

The compiler defines the GNU macros \_\_GNUC\_, \_\_GNUC_MINOR\_,
and \_\_GNUC_PATCHLEVEL\_

Description
This option determines whether the GNU macros \_\_GNUC\_, \_\_GNUC_MINOR\_, and
\_\_GNUC_PATCHLEVEL\_ are defined or defined.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-gcc</td>
<td>Defines GNU macros</td>
</tr>
<tr>
<td>-no-gcc</td>
<td>Undefines GNU macros</td>
</tr>
<tr>
<td>-gcc-sys</td>
<td>Defines GNU macros only during compilation of system headers</td>
</tr>
</tbody>
</table>

Alternate Options
None

gcc-name
Specifies the location of the gcc compiler when the
compiler cannot locate the gcc C++ libraries.

IDE Equivalent
Windows: None
Linux: Preprocessor > Nonstandard gcc Installation
Mac OS X: Preprocessor > gcc Installed to Non-standard Location
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-gcc-name=dir

Windows:
None

Arguments
dir
Is the full path location of the gcc compiler.

Default
OFF
The compiler locates the gcc libraries in the gcc install directory.

Description
This option specifies the location of the gcc compiler when the compiler cannot locate the gcc C++ libraries.
This option is helpful when you are referencing a non-standard gcc installation.
The C++ equivalent to option -gcc-name is -gxx-name.

Alternate Options
None

See Also
•
  • gxx-name
  • cxlib
**gcc-version**

*Provides compatible behavior with gcc.*

**IDE Equivalent**

Windows: None

Linux: **Preprocessor > gcc Compatibility Options**

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-gcc-version=n`

**Windows:**

None

**Arguments**

Is the gcc compatibility. Possible values are:

- `n=320` Specifies gcc 3.2 compatibility.
- `n=330` Specifies gcc 3.3 compatibility.
- `n=340` Specifies gcc 3.4 compatibility.
- `n=400` Specifies gcc 4.0 compatibility.
- `n=410` Specifies gcc 4.1 compatibility.
- `n=411` Specifies gcc 4.11 compatibility.
- `n=420` Specifies gcc 4.2 compatibility.
- `n=430` Specifies gcc 4.3 compatibility.

**Default**

OFF This option defaults to the installed version of gcc.
**Description**
This option provides compatible behavior with gcc. It selects the version of gcc with which you achieve ABI interoperability.

**Alternate Options**
None

**See Also**
- Compiler Options for Interoperability

**Gd**
Makes __cdecl the default calling convention.

**IDE Equivalent**
Windows: Advanced > Calling Convention
Linux: None
Mac OS X: None

**Architectures**
IA-32 architecture

**Syntax**

**Linux and Mac OS X:**
None

**Windows:**
/Gd

**Arguments**
None

**Default**
ON

The default calling convention is __cdecl.
**Description**
This option makes `__cdecl` the default calling convention.

**Alternate Options**
None

**gdwarf-2**
*Enables generation of debug information using the DWARF2 format.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

- **Linux and Mac OS X:**
  - `-gdwarf-2`

- **Windows:**
  - None

**Arguments**
None

**Default**
OFF

No debug information is generated. However, if compiler option `-g` is specified, debug information is generated in the latest DWARF format, which is currently DWARF2.

**Description**
This option enables generation of debug information using the DWARF2 format. This is currently the default when compiler option `-g` is specified.
Alternate Options
None

See Also
•
  • \$g

Ge
Enables stack-checking for all functions.
This option has been deprecated.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/Ge

Arguments
None

Default
OFF
Stack-checking for all functions is disabled.

Description
This option enables stack-checking for all functions.

Alternate Options
None
Gf
Enables read/write string-pooling optimization.
This is a deprecated option.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/Gf

Arguments
None

Default
ON Read/write string-pooling optimization is enabled.

Description
This option enables read/write string-pooling optimization.
You should not use /GF if you write to your strings because it can result in unexpected behavior.
If you are not writing to your strings, you should use /GF.

Alternate Options
None

See Also
•
• GF
GF
Enables read-only string-pooling optimization.

IDE Equivalent
Windows: Code Generation > Enable String Pooling
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/GF

Arguments
None

Default
OFF
Read/write string-pooling optimization is enabled.

Description
This option enables read only string-pooling optimization.

Alternate Options
None
**Gh**  
Calls a function to aid custom user profiling.

**IDE Equivalent**  
None

**Architectures**  
IA-32, Intel® 64 architectures

**Syntax**  

**Linux and Mac OS X:**  
None

**Windows:**  

/Gh

**Arguments**  
None

**Default**  
The compiler uses the default libraries.

**Description**  
This option calls the __penter function to aid custom user profiling. The prototype for __penter is not included in any of the standard libraries or Intel-provided libraries. You do not need to provide a prototype unless you plan to explicitly call __penter.

**Alternate Options**  
None

**See Also**  
•
• GH
GH

*Calls a function to aid custom user profiling.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64 architectures

**Syntax**

*Linux and Mac OS X:*
None

*Windows:*

/\GH

**Arguments**
None

**Default**
OFF

The compiler uses the default libraries.

**Description**
This option calls the __pexit function to aid custom user profiling. The prototype for __pexit is not included in any of the standard libraries or Intel-provided libraries. You do not need to provide a prototype unless you plan to explicitly call __pexit.

**Alternate Options**
None

**See Also**

•

• Gh
**Gm**

*Enables a minimal rebuild.*

**IDE Equivalent**

Windows: **Code Generation > Enable Minimal Rebuild**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

None

Windows:

/\Gm

**Arguments**

None

**Default**

OFF

Minimal rebuilds are disabled.

**Description**

This option enables a minimal rebuild.

**Alternate Options**

None
global-hoist, Qglobal-hoist

Enables certain optimizations that can move memory loads to a point earlier in the program execution than where they appear in the source.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- global-hoist
- no-global-hoist

Windows:
/Qglobal-hoist
/Qglobal-hoist-

Arguments
None

Default
- global-hoist
or/Qglobal-hoist

Description
This option enables certain optimizations that can move memory loads to a point earlier in the program execution than where they appear in the source. In most cases, these optimizations are safe and can improve performance.

The -no-global-hoist (Linux and Mac OS X) or /Qglobal-hoist- (Windows) option is useful for some applications, such as those that use shared or dynamically mapped memory, which can fail if a load is moved too early in the execution stream (for example, before the memory is mapped).
Alternate Options
None

Gr
Makes __fastcall the default calling convention.

IDE Equivalent
Windows: Advanced > Calling Convention
Linux: None
Mac OS X: None

Architectures
IA-32 architecture

Syntax
Linux and Mac OS X:
None
Windows:
/Gr

Arguments
None

Default
OFF The default calling convention is __cdecl

Description
This option makes __fastcall the default calling convention.

Alternate Options
None
GR
Enables C++ Run Time Type Information (RTTI).

IDE Equivalent
Windows: Language > Enable Run-Time Type Info
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/GR
/GR-

Arguments
None

Default
/GR When using Microsoft Visual Studio* 2005
/GR- When using Microsoft Visual Studio .NET 2003* (or earlier)

Description
This option enables C++ Run Time Type Information (RTTI). /Qvc8 implies /GR, while /Qvc7.1 (or lower) implies /GR-.

Alternate Options
None
**Gs**

Disables stack-checking for routines with more than a specified number of bytes of local variables and compiler temporaries.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
None

Windows:
/Gs[n]

**Arguments**

n
Is the number of bytes of local variables and compiler temporaries.

**Default**

4096
Stack checking is disabled for routines with more than 4KB of stack space allocated.

**Description**

This option disables stack-checking for routines with n or more bytes of local variables and compiler temporaries. If you do not specify n, you get the default of 4096.

**Alternate Options**

None
fstack-security-check, GS
Determines whether the compiler generates code that detects some buffer overruns.

IDE Equivalent
Windows: Code Generation > Buffer Security Check
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-fstack-security-check
-fno-stack-security-check

Windows:
/GS
/GS-

Arguments
None

Default
-fno-stack-security-check The compiler does not detect buffer overruns.
or /GS-

Description
This option determines whether the compiler generates code that detects some buffer overruns that overwrite the return address. This is a common technique for exploiting code that does not enforce buffer size restrictions.

The /GS option is supported with Microsoft Visual Studio .NET 2003* and Microsoft Visual Studio 2005*.
Alternate Options
Linux and Mac OS X: -f[no-]stack-protector
Windows: None

GT
_Enables fiber-safe thread-local storage of data._

IDE Equivalent
Windows: Optimization > Enable Fiber-safe Optimizations
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/GT

Arguments
None

Default
OFF There is no fiber-safe thread-local storage.

Description
This option enables fiber-safe thread-local storage (TLS) of data.

Alternate Options
None
GX
Enables C++ exception handling. This option is deprecated.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/GX
/GX-

Arguments
None

Default
/GX When using Microsoft Visual Studio® 2005
/GX- When using Microsoft Visual Studio .NET 2003* (or earlier)

Description
This option enables C++ exception handling. /Qvc8 implies /GX, while /Qvc7.1 (or lower) implies /GX-.

Alternate Options
None
**gxx-name**
Specifies that the g++ compiler should be used to set up the environment for C++ compilations.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-gxx-name=dir

**Windows:**

None

**Arguments**

dir Is the full path location of the g++ compiler.

**Default**

OFF The compiler uses the PATH setting to find the g++ compiler and resolve environment settings.

**Description**

This option specifies that the g++ compiler should be used to set up the environment for C++ compilations.

The C equivalent to option -gxx-name is -gcc-name.

NOTE. When compiling a C++ file with icc, g++ is used to get the environment.

**Alternate Options**

None
See Also

- gcc-name

**Gy**

Separates functions into COMDATs for the linker.
This option is deprecated.

**IDE Equivalent**

Windows: **Code Generation > Enable Function-Level Linking**
Linux: None
Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Gy
/Gy-

**Arguments**

None

**Default**

ON The compiler separates functions into COMDATs.

**Description**

This option tells the compiler to separate functions into COMDATs for the linker.

**Alternate Options**

None
Gz
_Makes _stdcall the default calling convention._

**IDE Equivalent**

Windows: *Advanced > Calling Convention*

Linux: None

Mac OS X: None

**Architectures**

IA-32 architecture

**Syntax**

Linux and Mac OS X:
None

Windows:
/Gz

**Arguments**

None

**Default**

OFF  

The default calling convention is __cdecl._

**Description**

This option makes _stdcall the default calling convention.

**Alternate Options**

None

**GZ**

_Initializes all local variables._
This option is deprecated.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
None

Windows:
/GZ

Arguments
None

Default
OFF

The compiler does not initialize local variables.

Description
This option initializes all local variables to a non-zero value. To use this option, you must also specify option /Od.

Alternate Options
None

H, QH

Tells the compiler to display the include file order and continue compilation.

IDE Equivalent
None
**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*
- `–H`

*Windows:*
- `/QH`

**Arguments**
None

**Default**
OFF Compilation occurs as usual.

**Description**
This option tells the compiler to display the include file order and continue compilation.

**Alternate Options**
None

**H (Windows*)**  
*Causes the compiler to limit the length of external symbol names. This is a deprecated option.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*
None
Windows:
/Hn
Arguments

n Is the maximum number of characters for external symbol names.

Default
OFF The compiler follows default rules for the length of external symbol names.

Description
This option causes the compiler to limit the length of external symbol names to a maximum of n characters.

Alternate Options
None

help
Displays all available compiler options or a category of compiler options.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-help[category]

Windows:
/help[category]
Arguments

category

Is a category or class of options to display. Possible values are:

- advanced: Displays advanced optimization options that allow fine tuning of compilation or allow control over advanced features of the compiler.
- codegen: Displays Code Generation options.
- compatibility: Displays options affecting language compatibility.
- component: Displays options for component control.
- data: Displays options related to interpretation of data in programs or the storage of data.
- deprecated: Displays options that have been deprecated.
- diagnostics: Displays options that affect diagnostic messages displayed by the compiler.
- float: Displays options that affect floating-point operations.
- help: Displays all the available help categories.
- inline: Displays options that affect inlining.
- ipo: Displays Interprocedural Optimization (IPO) options.
- language: Displays options affecting the behavior of the compiler language features.
- link: Displays linking or linker options.
- misc: Displays miscellaneous options that do not fit within other categories.
- openmp: Displays OpenMP and parallel processing options.
- opt: Displays options that help you optimize code.
output Displays options that provide control over compiler output.

pgo Displays Profile Guided Optimization (PGO) options.

preproc Displays options that affect preprocessing operations.

reports Displays options for optimization reports.

Default
OFF No list is displayed unless this compiler option is specified.

Description
This option displays all available compiler options or a category of compiler options. If category is not specified, all available compiler options are displayed. This option can also be specified as --help.

Alternate Options
Linux and Mac OS X: None
Windows: /?

help-pragma, Qhelp-pragma
Displays all supported pragmas.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-help-pragma

Windows:
/Qhelp-pragma
Arguments
None

Default
OFF  No list is displayed unless this compiler option is specified.

Description
This option displays all supported pragmas and shows their syntaxes.

Alternate Options
None

homeparams
Tells the compiler to store parameters passed in registers to the stack.

IDEEquivalent
None

Architectures
IA-64 architecture

Syntax
Linux and Mac OS X:
None

Windows:
/homeparams

Arguments
None

Default
OFF  Register parameters are not written to the stack.
Description
This option tells the compiler to store parameters passed in registers to the stack.

Alternate Options
None

hotpatch
*Tells the compiler to prepare a routine for hotpatching.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/hotpatch[:n]

Arguments

\( n \)
An integer specifying the number of bytes the compiler should add before the function entry point.

Default
OFF
The compiler does not prepare routines for hotpatching.

Description
This option tells the compiler to prepare a routine for hotpatching. The compiler inserts nop padding around function entry points so that the resulting image is hot patchable.

Specifically, the compiler adds nop bytes after each function entry point and enough nop bytes before the function entry point to fit a direct jump instruction on the target architecture.
If $n$ is specified, it overrides the default number of bytes that the compiler adds before the function entry point.

**Alternate Options**
None

\[-I\]

*Specifies an additional directory to search for include files.*

**IDE Equivalent**
- **Windows:** General > Additional Include Directories
- **Linux:** Preprocessor > Additional Include Directories
- **Mac OS X:** Preprocessor > Additional Include Directories

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
- **Linux and Mac OS X:**
  \[-Idir\]
- **Windows:**
  \[/Idir\]

**Arguments**
- **dir**
  Is the additional directory for the search.

**Default**
- **OFF**
  The default directory is searched for include files.

**Description**
This option specifies an additional directory to search for include files. To specify multiple directories on the command line, repeat the include option for each directory.
Alternate Options

None

i-dynamic

*This is a deprecated option. See shared-intel.*

i-static

*This is a deprecated option. See static-intel.*

icc

*Determines whether certain Intel compiler macros are defined or undefined.*

IDE Equivalent

Windows: None
Linux: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-icc
-no-icc

Windows:

None

Arguments

None

Default

-icc

The __ICC and __INTEL_COMPILER macros are set to represent the current version of the compiler.
Description
This option determines whether certain Intel compiler macros are defined or undefined.
If you specify -no-icc, the compiler undefines the __ICC and __INTEL_COMPILER macros. These macros are defined by default or by specifying -icc.

Alternate Options
None

idirafter
Adds a directory to the second include file search path.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-idirafterdir

Windows:
None

Arguments

dir Is the name of the directory to add.

Default
OFF Include file search paths include certain default directories.

Description
This option adds a directory to the second include file search path (after -I).
Alternate Options
None

**imacros**
*Allows a header to be specified that is included in front of the other headers in the translation unit.*

IDÉ Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

```
-include file
```

**Windows:**
None

Arguments

`file`  Name of header file.

Default
OFF

Description
Allows a header to be specified that is included in front of the other headers in the translation unit.

Alternate Options
None
**inline-calloc, Qinline-calloc**  
*Tells the compiler to inline calls to calloc() as calls to malloc() and memset().*

**IDE Equivalent**  
None

**Architectures**  
IA-32, Intel® 64 architectures

**Syntax**  

**Linux and Mac OS X:**
- `inline-calloc`
- `no-inline-calloc`

**Windows:**
- `/Qinline-calloc`
- `/Qinline-calloc-`

**Arguments**  
None

**Default**  
- `no-inline-calloc`  
  The compiler inlines calls to calloc() as calls to malloc().

  or  
  `/Qinline-calloc-`

**Description**  
This option tells the compiler to inline calls to calloc() as calls to malloc() and memset(). This enables additional memset() optimizations. For example, it can enable inlining as a sequence of store operations when the size is a compile time constant.

**Alternate Options**  
None
inline-debug-info, Qinline-debug-info

*Produces enhanced source position information for inlined code. This is a deprecated option.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `inline-debug-info`

**Windows:**

/`Qinline-debug-info`

**Arguments**

None

**Default**

OFF

No enhanced source position information is produced for inlined code.

**Description**

This option produces enhanced source position information for inlined code. This leads to greater accuracy when reporting the source location of any instruction. It also provides enhanced debug information useful for function call traceback.

To use this option for debugging, you must also specify a debug enabling option, such as `-g` (Linux) or `/debug` (Windows).

**Alternate Options**

**Linux and Mac OS X:** `-debug inline-debug-info`

**Windows:** None
**inline-factor, Qinline-factor**

*Specifies the percentage multiplier that should be applied to all inlining options that define upper limits.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `-inline-factor=n`
- `-no-inline-factor`

**Windows:**
- `/Qinline-factor=n`
- `/Qinline-factor-`

**Arguments**

\( n \)  

Is a positive integer specifying the percentage value. The default value is 100 (a factor of 1).

**Default**

- `-no-inline-factor` or `/Qinline-factor-`  
  
  The compiler uses default heuristics for inline routine expansion.

**Description**

This option specifies the percentage multiplier that should be applied to all inlining options that define upper limits:

- `-inline-max-size` and `/Qinline-max-size`
- `-inline-max-total-size` and `/Qinline-max-total-size`
- `inline-max-per-routine` and `/Qinline-max-per-routine`
- `inline-max-per-compile` and `/Qinline-max-per-compile`

This option takes the default value for each of the above options and multiplies it by \( n \) divided by 100. For example, if 200 is specified, all inlining options that define upper limits are multiplied by a factor of 2. This option is useful if you do not want to individually increase each option limit.

If you specify `-no-inline-factor` (Linux and Mac OS X) or `/Qinline-factor-` (Windows), the following occurs:
- Every function is considered to be a small or medium function; there are no large functions.
- There is no limit to the size a routine may grow when inline expansion is performed.
- There is no limit to the number of times some routine may be inlined into a particular routine.
- There is no limit to the number of times inlining can be applied to a compilation unit.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**CAUTION.** When you use this option to increase default limits, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**
None

**See Also**
- `inline-max-size`, `/Qinline-max-size`
- `inline-max-total-size`, `/Qinline-max-total-size`
- `inline-max-per-routine`, `/Qinline-max-per-routine`
- `inline-max-per-compile`, `/Qinline-max-per-compile`
- `opt-report`, `/Qopt-report`
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions
**inline-forceinline, Qinline-forceinline**

Specifies that an inline routine should be inlined whenever the compiler can do so.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `-inline-forceinline`

**Windows:**
- `/Qinline-forceinline`

**Default**

OFF  

The compiler uses default heuristics for inline routine expansion.

**Description**

This option specifies that a inline routine should be inlined whenever the compiler can do so. This causes the routines marked with an inline keyword or attribute to be treated as if they were "forceinline".

**NOTE.** Because C++ member functions whose definitions are included in the class declaration are considered inline functions by default, using this option will also make these member functions "forceinline" functions.

The "forceinline" condition can also be specified by using the keyword `__forceinline`.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS) or `/Qopt-report` (Windows).
CAUTION. When you use this option to change the meaning of inline to "forceinline", the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options
None

See Also
- opt-report, Qopt-report
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions

**inline-level, Ob**
Specifies the level of inline function expansion.

IDE Equivalent
Windows: Optimization > Inline Function Expansion
Linux: Optimization > Inline Function Expansion
Mac OS X: Optimization > Inline Function Expansion

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- -inline-level=n

Windows:
/Obn
Arguments

$n$  
Is the inline function expansion level. Possible values are 0, 1, and 2.

Default

-`-inline-level=2` or `/Ob2`  This is the default if option `-O2` is specified or is in effect by default. On Windows systems, this is also the default if option `-O3` is specified.

-`-inline-level=0` or `/Ob0`  This is the default if option `-O0` (Linux and Mac OS) or `/Od` (Windows) is specified.

Description

This option specifies the level of inline function expansion. Inlining procedures can greatly improve the run-time performance of certain programs.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-inline-level=0</code></td>
<td>Disables inlining of user-defined functions. Note that statement functions are always inlined.</td>
</tr>
<tr>
<td><code>/Ob0</code></td>
<td></td>
</tr>
<tr>
<td><code>-inline-level=1</code></td>
<td>Enables inlining when an inline keyword or an inline attribute is specified. Also enables inlining according to the C++ language.</td>
</tr>
<tr>
<td><code>/Ob1</code></td>
<td></td>
</tr>
<tr>
<td><code>-inline-level=2</code></td>
<td>Enables inlining of any function at the compiler's discretion.</td>
</tr>
<tr>
<td><code>/Ob2</code></td>
<td></td>
</tr>
</tbody>
</table>

Alternate Options

Linux: `-Ob`  (this is a deprecated option)

Mac OS X: None

Windows: None
**inline-max-per-compile, Qinline-max-per-compile**

*Specifies the maximum number of times inlining may be applied to an entire compilation unit.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-inline-max-per-compile=n`
- `-no-inline-max-per-compile`

**Windows:**

- `/Qinline-max-per-compile=n`
- `/Qinline-max-per-compile-`

**Arguments**

`n`  
Is a positive integer that specifies the number of times inlining may be applied.

**Default**

- `-no-inline-max-per-compile` or `/Qinline-max-per-compile-`  
The compiler uses default heuristics for inline routine expansion.

**Description**

This option the maximum number of times inlining may be applied to an entire compilation unit. It limits the number of times that inlining can be applied.

For compilations using Interprocedural Optimizations (IPO), the entire compilation is a compilation unit. For other compilations, a compilation unit is a file.
If you specify `-no-inline-max-per-compile` (Linux and Mac OS X) or `/Qinline-max-per-compile-` (Windows), there is no limit to the number of times inlining may be applied to a compilation unit.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**CAUTION.** When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options
None

See Also

- `inline-factor`, `Qinline-factor`
- `opt-report`, `Qopt-report`
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions

**inline-max-per-routine**, `Qinline-max-per-routine`

*Specifies the maximum number of times the inliner may inline into a particular routine.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

```
-inline-max-per-routine=n
```
-no-inline-max-per-routine

**Windows:**
/Qinline-max-per-routine=n
/Qinline-max-per-routine-

**Arguments**

$n$ Is a positive integer that specifies the maximum number of times the inliner may inline into a particular routine.

**Default**

-no-inline-max-per-routine or /Qinline-max-per-routine- The compiler uses default heuristics for inline routine expansion.

**Description**

This option specifies the maximum number of times the inliner may inline into a particular routine. It limits the number of times that inlining can be applied to any routine.

If you specify -no-inline-max-per-routine (Linux and Mac OS X) or /Qinline-max-per-routine- (Windows), there is no limit to the number of times some routine may be inlined into a particular routine.

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

---

**CAUTION.** When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None

**See Also**

•
inline-factor, Qinline-factor
opt-report, Qopt-report
Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

**inline-max-size, Qinline-max-size**

Specifies the lower limit for the size of what the inliner considers to be a large routine.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `inline-max-size=n`
- `no-inline-max-size`

**Windows:**
/`Qinline-max-size=n`
/`Qinline-max-size-`

**Arguments**

`n`  
Is a positive integer that specifies the minimum size of what the inliner considers to be a large routine.

**Default**

`-no-inline-max-size`  
or/`Qinline-max-size-`  
The compiler uses default heuristics for inline routine expansion.
Description

This option specifies the lower limit for the size of what the inliner considers to be a large routine (a function). The inliner classifies routines as small, medium, or large. This option specifies the boundary between what the inliner considers to be medium and large-size routines.

The inliner prefers to inline small routines. It has a preference against inlining large routines. So, any large routine is highly unlikely to be inlined.

If you specify `-no-inline-max-size` (Linux and Mac OS X) or `/Qinline-max-size-` (Windows), there are no large routines. Every routine is either a small or medium routine.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

---

**CAUTION.** When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None

See Also

- `inline-min-size, Qinline-min-size`
- `inline-factor, Qinline-factor`
- `opt-report, Qopt-report`
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions
**inline-max-total-size, Qinline-max-total-size**

Specifies how much larger a routine can normally grow when inline expansion is performed.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `inline-max-total-size=n`
- `-no-inline-max-total-size`

**Windows:**

- `/Qinline-max-total-size=n`
- `/Qinline-max-total-size-`

**Arguments**

`n` is a positive integer that specifies the permitted increase in the routine's size when inline expansion is performed.

**Default**

- `-no-inline-max-total-size`
- `/Qinline-max-total-size-`

The compiler uses default heuristics for inline routine expansion.

**Description**

This option specifies how much larger a routine can normally grow when inline expansion is performed. It limits the potential size of the routine. For example, if 2000 is specified for `n`, the size of any routine will normally not increase by more than 2000.

If you specify `-no-inline-max-total-size (Linux and Mac OS X) or `/Qinline-max-total-size- (Windows), there is no limit to the size a routine may grow when inline expansion is performed.
To see compiler values for important inlining limits, specify compiler option \texttt{-opt-report} (Linux and Mac OS X) or \texttt{/Qopt-report} (Windows).

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

\textbf{Alternate Options}

None

\textbf{See Also}

\begin{itemize}
  \item \texttt{inline-factor}, \texttt{Qinline-factor}
  \item \texttt{opt-report}, \texttt{Qopt-report}
  \item Developer Directed Inline Expansion of User Functions
  \item Compiler Directed Inline Expansion of User Functions
\end{itemize}

\texttt{inline-min-size}, \texttt{Qinline-min-size}

\textit{Specifies the upper limit for the size of what the inliner considers to be a small routine.}

\textbf{IDE Equivalent}

None

\textbf{Architectures}

IA-32, Intel\textsuperscript{®} 64, IA-64 architectures

\textbf{Syntax}

\texttt{Linux and Mac OS X:}

\begin{verbatim}
  -inline-min-size=n
  -no-inline-min-size
\end{verbatim}
Windows:
/Qinline-min-size=n
/Qinline-min-size-

Arguments

\( n \)  
Is a positive integer that specifies the maximum size of what the inliner considers to be a small routine.

Default

-no-inline-min-size  
/or/Qinline-min-size-
The compiler uses default heuristics for inline routine expansion.

Description

This option specifies the upper limit for the size of what the inliner considers to be a small routine (a function). The inliner classifies routines as small, medium, or large. This option specifies the boundary between what the inliner considers to be small and medium-size routines.

The inliner has a preference to inline small routines. So, when a routine is smaller than or equal to the specified size, it is very likely to be inlined.

If you specify -no-inline-min-size (Linux and Mac OS X) or /Qinline-min-size- (Windows), there is no limit to the size of small routines. Every routine is a small routine; there are no medium or large routines.

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

CAUTION. When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None
See Also

- inline-min-size, Qinline-min-size
- opt-report, Qopt-report
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions

ip, Qip

*Determines whether additional interprocedural optimizations for single-file compilation are enabled.*

IDE Equivalent

Windows: **Optimization > Interprocedural Optimization**

Linux: **Optimization > Enable Interprocedural Optimizations for Single File Compilation**

Mac OS X: **Optimization > Enable Interprocedural Optimization for Single File Compilation**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

- -ip
- -no-ip

Windows:

/ Qip
/ Qip-

Arguments

None
**Default**

OFF

Some limited interprocedural optimizations occur, including inline function expansion for calls to functions defined within the current source file. These optimizations are a subset of full intra-file interprocedural optimizations. Note that this setting is not the same as `-no-ip` (Linux and Mac OS X) or `/Qip-` (Windows).

**Description**

This option determines whether additional interprocedural optimizations for single-file compilation are enabled.

Options `-ip` (Linux and Mac OS X) and `/Qip` (Windows) enable additional interprocedural optimizations for single-file compilation.

Options `-no-ip` (Linux and Mac OS X) and `/Qip-` (Windows) may not disable inlining. To ensure that inlining of user-defined functions is disabled, specify `-inline-level=0` or `-fno-inline` (Linux and Mac OS X), or specify `/O0` (Windows). To ensure that inlining of compiler intrinsic functions is disabled, specify `-fno-builtin` (Linux and MacOS X) or `/Oi-` (Windows).

**Alternate Options**

None

**See Also**

- `finline-functions`

**ip-no-inlining, Qip-no-inlining**

Disables full and partial inlining enabled by interprocedural optimization options.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
- ip-no-inlining

Windows:
/Qip-no-inlining

Arguments
None

Default
OFF  Inlining enabled by interprocedural optimization options is performed.

Description
This option disables full and partial inlining enabled by the following interprocedural optimization options:

- On Linux and Mac OS X systems: -ip or -ipo
- On Windows systems: /Qip, /Qipo, or /Ob2

It has no effect on other interprocedural optimizations.

On Windows systems, this option also has no effect on user-directed inlining specified by option /Ob1.

Alternate Options
None

ip-no-pinlining, Qip-no-pinlining
Disables partial inlining enabled by interprocedural optimization options.

IDE Equivalent
None
Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-ip-no-pinlining

Windows:
/Qip-no-pinlining

Arguments
None

Default
OFF

Description
Inlining enabled by interprocedural optimization options is performed.

This option disables partial inlining enabled by the following interprocedural optimization options:

- On Linux and Mac OS X systems: -ip or -ipo
- On Windows systems: /Qip or /Qipo

It has no effect on other interprocedural optimizations.

Alternate Options
None

IPF-flt-eval-method0, QIPF-flt-eval-method0
Tells the compiler to evaluate the expressions involving floating-point operands in the precision indicated by the variable types declared in the program. This is a deprecated option.

IDE Equivalent
None
Architectures
IA-64 architecture

Syntax
Linux:
-IPF-flt-eval-method0

Mac OS X:
None

Windows:
/QIPF-flt-eval-method0

Arguments
None

Default
OFF
Expressions involving floating-point operands are evaluated by default rules.

Description
This option tells the compiler to evaluate the expressions involving floating-point operands in the precision indicated by the variable types declared in the program.

By default, intermediate floating-point expressions are maintained in higher precision.

The recommended method to control the semantics of floating-point calculations is to use option -fp-model (Linux) or /fp (Windows).

Instead of using -IPF-flt-eval-method0 (Linux) or /QIPF-flt-eval-method0 (Windows), you can use -fp-model source (Linux) or /fp:source (Windows).

Alternate Options
None

See Also
•
•
• fp-model, fp

**IPF-fltacc, QIPF-fltacc**  
*Disables optimizations that affect floating-point accuracy. This is a deprecated option.*

**IDE Equivalent**
None

**Architectures**
IA-64 architecture

**Syntax**

**Linux:**
- `IPF-fltacc`
- `no-IPF-fltacc`

**Mac OS X:**
None

**Windows:**
/QIPF-fltacc  
/QIPF-fltacc-

**Arguments**
None

**Default**

- `no-IPF-fltacc`  
  Optimizations are enabled that affect floating-point accuracy.  
  or/QIPF-fltacc-

**Description**
This option disables optimizations that affect floating-point accuracy.

If the default setting is used, the compiler may apply optimizations that reduce floating-point accuracy.
You can use this option to improve floating-point accuracy, but at the cost of disabling some optimizations.

The recommended method to control the semantics of floating-point calculations is to use option `-fp-model` (Linux) or `/fp` (Windows).

Instead of using `-IPF-fltacc` (Linux) or `/QIPF-fltacc` (Windows), you can use `-fp-model precise` (Linux) or `/fp:precise` (Windows).

Instead of using `-no-IPF-fltacc` (Linux) or `/QIPF-fltacc-` (Windows), you can use `-fp-model fast` (Linux) or `/fp:fast` (Windows).

**Alternate Options**

None

**See Also**

- `-IPF-fltacc`, `-no-IPF-fltacc`
- `-fp-model`, `-fp`
- `fp-model, fp`

**IPF-fma, QIPF-fma**

See `fma`, `Qfma`.

**IPF-fp-relaxed, QIPF-fp-relaxed**

See `fp-relaxed`, `Qfp-relaxed`.

**ipo, Qipo**

Enables interprocedural optimization between files.

**IDE Equivalent**

Windows: `Optimization > Interprocedural Optimization`

Linux: `Optimization > Enable Whole Program Optimization`

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:

-ipo\[n\]

Windows:

/Qipo\[n\]

Arguments

\(n\)  Is an optional integer that specifies the number of object files the compiler should create. The integer must be greater than or equal to 0.

Default

OFF  Multifile interprocedural optimization is not enabled.

Description

This option enables interprocedural optimization between files. This is also called multifile interprocedural optimization (multifile IPO) or Whole Program Optimization (WPO).

When you specify this option, the compiler performs inline function expansion for calls to functions defined in separate files.

You cannot specify the names for the files that are created.

If \(n\) is 0, the compiler decides whether to create one or more object files based on an estimate of the size of the application. It generates one object file for small applications, and two or more object files for large applications.

If \(n\) is greater than 0, the compiler generates \(n\) object files, unless \(n\) exceeds the number of source files, in which case the compiler generates only the number of source files.

If you do not specify \(n\), the default is 0.

Alternate Options

None
ipo-c, Qipo-c

*Tells the compiler to optimize across multiple files and generate a single object file.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*
-ipo-c

*Windows:*
/Qipo-c

**Arguments**
None

**Default**
OFF The compiler does not generate a multifile object file.

**Description**
This option tells the compiler to optimize across multiple files and generate a single object file (named ipo_out.o on Linux and Mac OS X systems; ipo_out.obj on Windows systems).

It performs the same optimizations as -ipo (Linux and Mac OS X) or /Qipo (Windows), but compilation stops before the final link stage, leaving an optimized object file that can be used in further link steps.

**Alternate Options**
None

**See Also**
•
ipo-jobs, Qipo-jobs
Specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO).

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-ipo-jobs n
Windows:
/Qipo-jobs:n

Arguments
n Is the number of commands (jobs) to run simultaneously.
The number must be greater than or equal to 1.

Default
-ipo-jobs1 or /Qipo-jobs:1 One command (job) is executed in an interprocedural optimization parallel build.

Description
This option specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO). It should only be used if the link-time compilation is generating more than one object. In this case, each object is generated by a separate compilation, which can be done in parallel.

This option can be affected by the following compiler options:
• \texttt{-ipo} (Linux and Mac OS X) or /Qipo (Windows) when applications are large enough that the compiler decides to generate multiple object files.
• \texttt{-ipon} (Linux and Mac OS X) or /Qipon (Windows) when \( n \) is greater than 1.
• \texttt{-ipo-separate} (Linux) or /Qipo-separate (Windows)

\textbf{CAUTION.} Be careful when using this option. On a multi-processor system with lots of memory, it can speed application build time. However, if \( n \) is greater than the number of processors, or if there is not enough memory to avoid thrashing, this option can increase application build time.

\subsection*{Alternate Options}
None

\subsection*{See Also}
•
  
  • \texttt{ipo, Qipo}
  
  • \texttt{ipo-separate, Qipo-separate}

\texttt{ipo-S, Qipo-S}
\textit{Tells the compiler to optimize across multiple files and generate a single assembly file.}

\subsection*{IDE Equivalent}
None

\subsection*{ Architectures}
IA-32, Intel® 64, IA-64 architectures

\subsection*{Syntax}
\textbf{Linux and Mac OS X:}
\texttt{-ipo-S}
Windows:
/Qipo-S

Arguments
None

Default
OFF The compiler does not generate a multifile assembly file.

Description
This option tells the compiler to optimize across multiple files and generate a single assembly file (named ipo_out.s on Linux and Mac OS X systems; ipo_out.asm on Windows systems).

It performs the same optimizations as -ipo (Linux and Mac OS X) or /Qipo (Windows), but compilation stops before the final link stage, leaving an optimized assembly file that can be used in further link steps.

Alternate Options
None

See Also
•
•
• ipo, Qipo

ipo-separate, Qipo-separate
Trust the compiler to generate one object file for every source file.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux:
-ipo-separate

Mac OS X:
None

Windows:
/Qipo-separate

Arguments
None

Default
OFF

The compiler decides whether to create one or more object files.

Description
This option tells the compiler to generate one object file for every source file. It overrides any -ipo (Linux) or /Qipo (Windows) specification.

Alternate Options
None

See Also

- ipo, Qipo

ipp, Qipp
Tells the compiler to link to the some or all of the Intel® Integrated Performance Primitives (Intel® IPP) libraries.

IDE Equivalent
Windows: None
Linux: **Performance Library Build Components > Use Intel(R) Integrated Performance Primitives Libraries**

Mac OS X: **Libraries > Use Intel(R) Integrated Performance Primitives Libraries**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
-ipp[=lib]

Windows:
/Qipp[:lib]

**Arguments**

- **lib**
  Indicates the Intel® IPP libraries that the compiler should link to. Possible values are:
  - **common**
    Tells the compiler to link using the main libraries set. This is the default if the option is specified with no lib.
  - **crypto**
    Tells the compiler to link using the main libraries set and the crypto library.
  - **nonpic** (Linux only)
    Tells the compiler to link using the version of the libraries that do not have position-independent code.
  - **nonpic_crypto** (Linux only)
    Tells the compiler to link using the crypto library and the version of the libraries that do not have position-independent code.

**Default**
OFF
The compiler does not link to the Intel® IPP libraries.

**Description**
The option tells the compiler to link to the some or all of the Intel® Integrated Performance Primitives (Intel® IPP) libraries and include the Intel® IPP headers.
Alternate Options
None

-iprefix
Option for indicating the prefix for referencing directories containing header files.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-iprefix <prefix>

Windows:
None

Arguments
None

Default
OFF

Description
Options for indicating the prefix for referencing directories containing header files. Use <prefix> with -Iwithprefix as a prefix.

Alternate Options
None
iquote
Add directory to the front of the include file search path for files included with quotes but not brackets.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-iquote dir

Windows:
None

Arguments
dir Is the name of the directory to add.

Default
OFF The compiler does not add a directory to the front of the include file search path.

Description
Add directory to the front of the include file search path for files included with quotes but not brackets.

Alternate Options
None
**isystem**

Specifies a directory to add to the start of the system include path.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `isystem dir`

Windows:
None

**Arguments**

dir

Is the directory to add to the system include path.

**Default**
OFF

The default system include path is used.

**Description**

This option specifies a directory to add to the system include path. The compiler searches the specified directory for include files after it searches all directories specified by the `-I` compiler option but before it searches the standard system directories. This option is provided for compatibility with gcc.

**Alternate Options**

None
ivdep-parallel, Qivdep-parallel
*Tells the compiler that there is no loop-carried memory dependency in the loop following an IVDEP pragma.*

**IDE Equivalent**

Windows: None

Linux: **Optimization > IVDEP Directive Memory Dependency**

Mac OS X: None

**Architectures**

IA-64 architecture

**Syntax**

Linux:

- `ivdep-parallel`

Mac OS X:

None

Windows:

/Qivdep-parallel

**Arguments**

None

**Default**

OFF  
*There may be loop-carried memory dependency in a loop that follows an IVDEP pragma.*

**Description**

This option tells the compiler that there is no loop-carried memory dependency in the loop following an IVDEP. There may be loop-carried memory dependency in a loop that follows an IVDEP pragma.
Alternate Options
None

iwithprefix
_Appends a directory to the prefix passed in by -iprefix and puts it on the include search path at the end of the include directories._

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-iwithprefix<dir>

Windows:
None

Arguments
None

Default
OFF

Description
This option appends a directory to the prefix passed in by -iprefix and puts it on the include search path at the end of the include directories.

Alternate Options
None
iwithprefixbefore

Similar to -iwithprefix except the include directory is placed in the same place as -I command line include directories.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-iwithprefixbefore <dir>

Windows:
None

Arguments
None

Default
OFF

Description
Similar to -iwithprefix except the include directory is placed in the same place as -I command line include directories.

Alternate Options
None
Sets the default character type to unsigned.

IDE Equivalent
Windows: Language > Default Char Unsigned
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None
Windows: /J

Arguments
None

Default
OFF The default character type is signed

Description
This option sets the default character type to unsigned. This option has no effect on character values that are explicitly declared signed. This option sets _CHAR_UNSIGNED = 1.

Alternate Options
None
**Kc++, TP**
*Tells the compiler to process all source or unrecognized file types as C++ source files. This option is deprecated.*

**IDE Equivalent**

Windows: **Advanced > Compile As**
Linux: None
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-Kc++`

Windows:

 `/TP`

**Arguments**

None

**Default**

OFF The compiler uses default rules for determining whether a file is a C++ source file.

**Description**

This option tells the compiler to process all source or unrecognized file types as C++ source files.

**Alternate Options**

None
kernel
Generates code for inclusion in the kernel.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux:
- kernel

Mac OS X:
None

Windows:
None

Arguments
None

Default
OFF The restrictions on kernel code are not enforced.

Description
This option generates code for inclusion in the kernel. It prevents generation of speculation because support may not be available when the code runs.

This option also suppresses software pipelining.

Alternate Options
None
Tells the linker to search for a specified library when linking.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-lstring

Windows:
None

Arguments
string Specifies the library (libstring) that the linker should search.

Default
OFF The linker searches for standard libraries in standard directories.

Description
This option tells the linker to search for a specified library when linking.

When resolving references, the linker normally searches for libraries in several standard directories, in directories specified by the l option, then in the library specified by the l option.

The linker searches and processes libraries and object files in the order they are specified. So, you should specify this option following the last object file it applies to.

Alternate Options
None
See Also

•
• L

L
_Tells the linker to search for libraries in a specified directory before searching the standard directories._

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-ldir

Windows:

None

Arguments

dir

Is the name of the directory to search for libraries.

Default

OFF

The linker searches the standard directories for libraries.

Description

This option tells the linker to search for libraries in a specified directory before searching for them in the standard directories.

Alternate Options

None
See Also
- 1

LD
Specifies that a program should be linked as a dynamic-link (DLL) library.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/LD
/LDd

Arguments
None

Default
OFF The program is not linked as a dynamic-link (DLL) library.

Description
This option specifies that a program should be linked as a dynamic-link (DLL) library instead of an executable (.exe) file. You can also specify /LDd, where d indicates a debug version.

Alternate Options
None
**link**

*Passes user-specified options directly to the linker at compile time.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel<sup>®</sup> 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

`/link`

**Arguments**

None

**Default**

OFF

No user-specified options are passed directly to the linker.

**Description**

This option passes user-specified options directly to the linker at compile time.

All options that appear following `/link` are passed directly to the linker.

**Alternate Options**

None

**See Also**

- None

- Xlinker
\texttt{m}

*Tells the compiler to generate optimized code specialized for the processor that executes your program.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

*Linux and Mac OS X:*

\texttt{-m[processor]}

*Windows:*

None

**Arguments**

\textit{processor} Indicates the processor for which code is generated. Possible values are:

- \texttt{ia32} Generates code that will run on any Pentium or later processor. Disables any default extended instruction settings, and any previously set extended instruction settings. This value is only available on Linux systems using IA-32 architecture.

- \texttt{sse} This is the same as specifying \texttt{ia32}.

- \texttt{sse2} Generates code for Intel® Streaming SIMD Extensions 2 (Intel® SSE2). This value is only available on Linux systems.

- \texttt{sse3} Generates code for Intel® Streaming SIMD Extensions 3 (Intel® SSE3).
ssse3 Generates code for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3).

dsse4.1 Generates code for Intel® Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators.

Default

Linux systems: -msse2 For more information on the default values, see Arguments above.
Mac OS X systems using IA-32 architecture: -msse3
Mac OS X systems using Intel® 64 architecture: -mssse3

Description

This option tells the compiler to generate optimized code specialized for the processor that executes your program.

Code generated with the values ia32, sse, sse2, or sse3 should execute on any compatible non-Intel processor with support for the corresponding instruction set.

Options -x and -m are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning.

Alternate Options

Linux and Mac OS X: None

Windows: /arch

See Also

• x, Qx
• ax, Qax
• arch
**M, QM**

*Tells the compiler to generate makefile dependency lines for each source file.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

  `-M`

*Windows:*

  `/QM`

**Arguments**

None

**Default**

OFF  The compiler does not generate makefile dependency lines for each source file.

**Description**

This option tells the compiler to generate makefile dependency lines for each source file, based on the `#include` lines found in the source file.

**Alternate Options**

None
**m32, m64**
*Tells the compiler to generate code for a specific architecture.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**
- `-m32`
- `-m64`

**Windows:**
None

**Arguments**
None

**Default**
OFF

The compiler's behavior depends on the host system.

**Description**
These options tell the compiler to generate code for a specific architecture.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-m32</code></td>
<td>Tells the compiler to generate code for IA-32 architecture.</td>
</tr>
<tr>
<td><code>-m64</code></td>
<td>Tells the compiler to generate code for Intel® 64 architecture.</td>
</tr>
</tbody>
</table>
The -m32 and -m64 options are the same as Mac OS* X options -arch i386 and -arch x86_64, respectively. Note that these options are provided for compatibility with gcc. They are not related to the Intel® C++ compiler option arch.

Alternate Options
None

m32, m64
Tells the compiler to generate code for a specific architecture.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:
-m32
-m64

Windows:
None

Arguments
None

Default
OFF The compiler's behavior depends on the host system.

Description
These options tell the compiler to generate code for a specific architecture.
**Option** | **Description**
--- | ---
-m32 | Tells the compiler to generate code for IA-32 architecture.
-m64 | Tells the compiler to generate code for Intel® 64 architecture.

The -m32 and -m64 options are the same as Mac OS® X options -arch i386 and -arch x86_64, respectively. Note that these options are provided for compatibility with gcc. They are not related to the Intel® C++ compiler option arch.

**Alternate Options**

None

**malign-double**

Aligns double, long double, and long long types for better performance for systems based on IA-32 architecture.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

-malign-double

**Windows:**

None

**Arguments**

None
**Default**
OFF

**Description**
Aligns double, long double, and long long types for better performance for systems based on IA-32 architecture.

**Alternate Options**
-align

**malign-mac68k**
Aligns structure fields on 2-byte boundaries (m68k compatible).

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64 architectures

**Syntax**

*Linux:*
None

*Mac OS X:*
-malign-mac68k

*Windows:*
None

**Arguments**
None

**Default**
The compiler does not align structure fields on 2-byte boundaries.

OFF
Description
This option aligns structure fields on 2-byte boundaries (m68k compatible).

Alternate Options
None

malign-natural
Aligns larger types on natural size-based boundaries (overrides ABI).

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux:
None

Mac OS X:
-malign-natural

Windows:
None

Arguments
None

Default
OFF The compiler does not align larger types on natural size-based boundaries.

Description
This option aligns larger types on natural size-based boundaries (overrides ABI).
**Alternate Options**
None

**malign-power**
*Aligns based on ABI-specified alignment rules.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64 architectures

**Syntax**

**Linux:**
None

**Mac OS X:**
-malign-power

**Windows:**
None

**Arguments**
None

**Default**
ON
The compiler aligns based on ABI-specified alignment rules.

**Description**
Aligns based on ABI-specified alignment rules.

**Alternate Options**
None
map-opts, Qmap-opts
Maps one or more compiler options to their equivalent on a different operating system.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:
-map-opts
Mac OS X:
None
Windows:
/Qmap-opts

Arguments
None

Default
OFF No platform mappings are performed.

Description
This option maps one or more compiler options to their equivalent on a different operating system. The result is output to stdout.

On Windows systems, the options you provide are presumed to be Windows options, so the options that are output to stdout will be Linux equivalents.
On Linux systems, the options you provide are presumed to be Linux options, so the options that are output to stdout will be Windows equivalents.

The tool can be invoked from the compiler command line or it can be used directly.
No compilation is performed when the option mapping tool is used.
This option is useful if you have both compilers and want to convert scripts or makefiles.

**NOTE.** Compiler options are mapped to their equivalent on the architecture you are using.
For example, if you are using a processor with IA-32 architecture, you will only see equivalent options that are available on processors with IA-32 architecture.

**Alternate Options**

None

**Example**

The following command line invokes the option mapping tool, which maps the Linux options to Windows-based options, and then outputs the results to `stdout`:

```
icc -map-opts -xP -O2
```

The following command line invokes the option mapping tool, which maps the Windows options to Linux-based options, and then outputs the results to `stdout`:

```
icl /Qmap-opts /QxP /O2
```

**See Also**

- Building Applications: Using the Option Mapping Tool

**march**

*Tells the compiler to generate code for a specified processor.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures
Syntax

Linux:
-march=processor

Mac OS X:
None

Windows:
None

Arguments

processor

Is the processor for which the compiler should generate code. Possible values are:

- pentium3: Generates code for Intel® Pentium® III processors.
- pentium4: Generates code for Intel® Pentium® 4 processors.
- core2: Generates code for the Intel® Core 2™ processor family.

Default

OFF or -march=pentium4

On IA-32 architecture, the compiler does not generate processor-specific code unless it is told to do so. On systems using Intel® 64 architecture, the compiler generates code for Intel Pentium 4 processors.

Description

This option tells the compiler to generate code for a specified processor.

Specifying -march=pentium4 sets -mtune=pentium4.

For compatibility, a number of historical processor values are also supported, but the generated code will not differ from the default.

Alternate Options

None
**mcmodel**
*Tells the compiler to use a specific memory model to generate code and store data.*

**IDE Equivalent**
None

**Architectures**
Intel® 64 architecture

**Syntax**

**Linux:**
-mcmodel=mem_model

**Mac OS X:**
None

**Windows:**
None

**Arguments**

*mem_model*

Is the memory model to use. Possible values are:

- **small**
  Tells the compiler to restrict code and data to the first 2GB of address space. All accesses of code and data can be done with Instruction Pointer (IP)-relative addressing.

- **medium**
  Tells the compiler to restrict code to the first 2GB; it places no memory restriction on data. Accesses of code can be done with IP-relative addressing, but accesses of data must be done with absolute addressing.

- **large**
  Places no memory restriction on code or data. All accesses of code and data must be done with absolute addressing.
Default

-mcmodel=small

On systems using Intel® 64 architecture, the compiler restricts code and data to the first 2GB of address space. Instruction Pointer (IP)-relative addressing can be used to access code and data.

Description

This option tells the compiler to use a specific memory model to generate code and store data. It can affect code size and performance. If your program has global and static data with a total size smaller than 2GB, -mcmodel=small is sufficient. Global and static data larger than 2GB requires -mcmodel=medium or -mcmodel=large. Allocation of memory larger than 2GB can be done with any setting of -mcmodel.

IP-relative addressing requires only 32 bits, whereas absolute addressing requires 64-bits. IP-relative addressing is somewhat faster. So, the small memory model has the least impact on performance.

NOTE. When you specify -mcmodel=medium or -mcmodel=large, you must also specify compiler option -shared-intel to ensure that the correct dynamic versions of the Intel run-time libraries are used.

Alternate Options

None

Example

The following example shows how to compile using -mcmodel:

icl -shared-intel -mcmodel=medium -o prog prog.c

See Also

- shared-intel
- fpic
mcpu
This is a deprecated option. See mtune.

MD, QMD
Preprocess and compile, generating output file containing dependency information ending with extension .d.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- MD

Windows:
/QMD

Arguments
None

Default
OFF The compiler does not generate dependency information.

Description
Preprocess and compile, generating output file containing dependency information ending with extension .d.

Alternate Options
None
**MD**

*Tells the linker to search for unresolved references in a multithreaded, dynamic-link run-time library.*

**IDE Equivalent**

Windows: **Code Generation > Runtime Library**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/MD

/MDd

**Arguments**

None

**Default**

OFF  The linker searches for unresolved references in a single-threaded, static run-time library.

**Description**

This option tells the linker to search for unresolved references in a multithreaded, dynamic-link (DLL) run-time library. You can also specify /MDd, where d indicates a debug version.

**Alternate Options**

None
mdynamic-no-pic
Generates code that is not position-independent but has position-independent external references.

IDE Equivalent
None

Architectures
IA-32 architecture

Syntax
Linux:
None

Mac OS X:
-mdynamic-no-pic

Windows:
None

Arguments
None

Default
OFF All references are generated as position independent.

Description
This option generates code that is not position-independent but has position-independent external references.

The generated code is suitable for building executables, but it is not suitable for building shared libraries.

This option may reduce code size and produce more efficient code. It overrides the -fpic compiler option.
Alternate Options
None

See Also
•
• fpic

MF, QMF
*Tells the compiler to generate makefile dependency information in a file.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-MF file

Windows:
/QMF file

Arguments

file
Is the name of the file where the makefile dependency information should be placed.

Default

OFF
The compiler does not generate makefile dependency information in files.

Description
This option tells the compiler to generate makefile dependency information in a file. To use this option, you must also specify /QM or /QMM.
Alternate Options
None

See Also
- QM
- QMM

mfixed-range
*Reserves certain registers (f12-f15, f32-f127) for use by the Linux* kernel.*

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax

Linux:
- \(-mfixed-range=f12-f15,f32-f127\)

Mac OS X:
None

Windows:
None

Arguments
None

Default
OFF
Description
Reserves certain registers (f12-f15,f32-f127) for use by the Linux* kernel.

Alternate Options
None

MG, QMG
Tells the compiler to generate makefile dependency lines for each source file.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- MG

Windows:
/ QMG

Arguments
None

Default
OFF The compiler does not generate makefile dependency information in files.

Description
This option tells the compiler to generate makefile dependency lines for each source file. It is similar to /QM, but it treats missing header files as generated files.
Alternate Options
None

See Also
•
•
• QM

minstruction, Qinstruction
Determines whether MOVBE instructions are generated for Intel processors.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-minstruction=[no]movbe

Windows:
/Qinstruction:[no]movbe

Arguments
None

Default
-minstruction=nomovbe The compiler does not generate MOVBE instructions for Intel® or/Qinstruction:nomovbe Atom™ processors.
Description

This option determines whether MOVBE instructions are generated for Intel processors. To use this option, you must also specify `-xSSE3_ATOM` (Linux and Mac OS X) or `/QxSSE3_ATOM` (Windows).

If `-minstruction=movbe` or `/Qinstruction:movbe` is specified, the following occurs:

- MOVBE instructions are generated that are specific to the Intel® Atom™ processor.
- The options are ON by default when `-xSSE3_ATOM` or `/QxSSE3_ATOM` is specified.
- Generated executables can only be run on Intel® Atom™ processors or processors that support Intel® Streaming SIMD Extensions 3 (Intel® SSE3) and MOVBE.

If `-minstruction=nomovbe` or `/Qinstruction:nomovbe` is specified, the following occurs:

- The compiler optimizes code for the Intel® Atom™ processor, but it does not generate MOVBE instructions.
- Generated executables can be run on non-Intel® Atom™ processors that support Intel® SSE3.

Alternate Options

None

See Also

- `x`, `/x`
- `mkl`, `/Qmkl`

Tell the compiler to link to certain parts of the Intel® Math Kernel Library (Intel® MKL).

IDE Equivalent

Windows: None

Linux: Performance Library Build Components > Use Intel(R) Math Kernel Library

Mac OS X: Libraries > Use Intel(R) Math Kernel Library

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:

- mkl[=lib]

Windows:

/Qmkl[:lib]

Arguments

lib

Indicates the part of the library that the compiler should link to. Possible values are:

parallel

Tells the compiler to link using the threaded part of the Intel® MKL. This is the default if the option is specified with no lib.

sequential

Tells the compiler to link using the non-threaded part of the Intel® MKL.

cluster

Tells the compiler to link using the cluster part and the sequential part of the Intel® MKL.

Default

OFF

The compiler does not link to the Intel® MKL.

Description

This option tells the compiler to link to certain parts of the Intel® Math Kernel Library (Intel® MKL).

Alternate Options

None

ML

Tells the linker to search for unresolved references in a single-threaded, static run-time library.
This option has been deprecated.

**IDE Equivalent**

Windows: **Code Generation > Runtime Library**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

None

Windows:

/ML

/MLd

**Arguments**

None

**Default**

Systems using Microsoft Visual Studio 2003: /ML

Systems using Microsoft Visual Studio 2005 or later: OFF

If Microsoft* Visual Studio* 2003 is being used, the linker searches for unresolved references in a single-threaded, static run-time library.

If Microsoft* Visual Studio* 2005 or greater is being used, or Microsoft* Visual Studio* Premier Partner Edition (VSPPE) has been installed, the linker searches for unresolved references in a multithreaded, static run-time library.

**Description**

This option tells the linker to search for unresolved references in a single-threaded, static run-time library. It is only valid with Microsoft Visual Studio 2003, and is deprecated with later versions.

You can also specify /MLd, where d indicates a debug version.
**Alternate Options**
None

**MM, QMM**
*Tells the compiler to generate makefile dependency lines for each source file.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

`-MM`

*Windows:*

`/QMM`

**Arguments**
None

**Default**
OFF The compiler does not generate makefile dependency information in files.

**Description**
This option tells the compiler to generate makefile dependency lines for each source file. It is similar to `/QM`, but it does not include system header files.

**Alternate Options**
None

**See Also**
•
MMD, QMMD
*Tells the compiler to generate an output file containing dependency information.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
-MMD

Windows:
/QMMD

**Arguments**
None

**Default**
OFF  The compiler does not generate an output file containing dependency information.

**Description**
This option tells the compiler to preprocess and compile a file, then generate an output file (with extension .d) containing dependency information.

It is similar to /QMD, but it does not include system header files.

**Alternate Options**
None
**mp**
Maintains floating-point precision while disabling some optimizations. This is a deprecated option.

**IDE Equivalent**

Windows: None
Linux: **Floating Point > Improve Floating-point Consistency**
Mac OS X: **Floating Point > Improve Floating-point Consistency**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-mp
```

Windows:

None

**Arguments**

None

**Default**

OFF The compiler provides good accuracy and run-time performance at the expense of less consistent floating-point results.

**Description**

This option maintains floating-point precision while disabling some optimizations, such as -fma (Linux) and /Qfma (Windows). It restricts optimization to maintain declared precision and to ensure that floating-point arithmetic conforms more closely to the ANSI* language and IEEE* arithmetic standards.

For most programs, specifying this option adversely affects performance. If you are not sure whether your application needs this option, try compiling and running your program both with and without it to evaluate the effects on both performance and precision.
The recommended method to control the semantics of floating-point calculations is to use option -fp-model.

Alternate Options

Linux and Mac OS X: -mieee-fp

Windows: None

See Also

- m1, Qprec
- fp-model, fp

MP

* Tells the compiler to add a phony target for each dependency.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

- MP

Windows:

None (see below)

Arguments

None

Default

OFF

The compiler does not generate dependency information unless it is told to do so.
**Description**
This option tells the compiler to add a phony target for each dependency.
Note that this option is not related to Windows* option /MP.

**Alternate Options**
None

**multiple-processes, MP**
* Creates multiple processes that can be used to compile large numbers of source files at the same time.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

* **Linux and Mac OS X:**
  - `multiple-processes [=n]`

* **Windows:**
  - `/MP [:n]`

**Arguments**

*`n` is the maximum number of processes that the compiler should create.

**Default**
OFF A single process is used to compile source files.
Description

This option creates multiple processes that can be used to compile large numbers of source files at the same time. It can improve performance by reducing the time it takes to compile source files on the command line.

This option causes the compiler to create one or more copies of itself, each in a separate process. These copies simultaneously compile the source files.

If \( n \) is not specified for this option, the default value is as follows:

- On Windows OS, the value is based on the setting of the NUMBER_OF_PROCESSORS environment variable.
- On Linux OS and Mac OS X, the value is 2.

This option applies to compilations, but not to linking or link-time code generation.

Alternate Options

None

mp1, Qprec

*Improves floating-point precision and consistency.*

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

```
-mp1
```

**Windows:**

```
/Qprec
```

Arguments

None
**Default**
OFF

The compiler provides good accuracy and run-time performance at the expense of less consistent floating-point results.

**Description**
This option improves floating-point consistency. It ensures the out-of-range check of operands of transcendental functions and improves the accuracy of floating-point compares.

This option prevents the compiler from performing optimizations that change NaN comparison semantics and causes all values to be truncated to declared precision before they are used in comparisons. It also causes the compiler to use library routines that give better precision results compared to the X87 transcendental instructions.

This option disables fewer optimizations and has less impact on performance than option mp or Op.

**Alternate Options**
None

**See Also**
•
•
• mp

**MQ**
*Changes the default target rule for dependency generation.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
-MQ target
Windows:
None

Arguments

target Is the target rule to use.

Default
OFF The default target rule applies to dependency generation.

Description
This option changes the default target rule for dependency generation. It is similar to `-MT`, but quotes special Make characters.

Alternate Options
None

mregparm
Control the number registers used to pass integer arguments.

IDE Equivalent
Windows: None
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-mregparm=value

Windows:
None
**Arguments**

None

**Default**

OFF

The compiler does not use registers to pass arguments.

**Description**

Control the number registers used to pass integer arguments.

**Alternate Options**

None

**mrelax**

*Tells the compiler to pass linker option* `-relax` *to the linker.*

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

**Linux and Mac OS X:**

- `-mrelax`
- `-mno-relax`

**Mac OS X:**

None

**Windows:**

None

**Arguments**

None
**Default**

- `mno-relax`  
  The compiler does not pass `-relax` to the linker.

**Description**

This option tells the compiler to pass linker option `-relax` to the linker.

**Alternate Options**

None

**mserialize-volatile, Qserialize-volatile**

Determines whether strict memory access ordering is imposed for volatile data object references.

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

**Linux:**

- `mserialize-volatile`
  - `mno-serialize-volatile`

**Mac OS X:**

None

**Windows:**

- `/Qserialize-volatile`
  - `/Qserialize-volatile-`

**Arguments**

None
**Default**

-mno-serialize-volatile or /Qserialize-volatile

**Description**

This option determines whether strict memory access ordering is imposed for volatile data object references.

If you specify -mno-serialize-volatile, the compiler may suppress both run-time and compile-time memory access ordering for volatile data object references. Specifically, the .rel/.acq completers will not be issued on referencing loads and stores.

**Alternate Options**

None

**MT, QMT**

Changes the default target rule for dependency generation.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- MTtarget

Windows:

/QTtarget

**Arguments**

*target* Is the target rule to use.
**Default**

OFF

The default target rule applies to dependency generation.

**Description**

This option changes the default target rule for dependency generation.

**Alternate Options**

None

**MT**

*Tells the linker to search for unresolved references in a multithreaded, static run-time library.*

**IDE Equivalent**

Windows: **Code Generation > Runtime Library**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/MT

/MTd

**Arguments**

None
**Default**

Systems using Intel® 64 architecture: `/MT`  
IA-32 architecture and IA-64 architecture: `OFF`  

On systems using Intel® 64 architecture, the linker searches for unresolved references in a multithreaded, static run-time library. On systems using IA-32 architecture and IA-64 architecture, the linker searches for unresolved references in a single-threaded, static run-time library. However, on systems using IA-32 architecture, if option `Qvc8` is in effect, the linker searches for unresolved references in threaded libraries.

**Description**

This option tells the linker to search for unresolved references in a multithreaded, static run-time library. You can also specify `/MTd`, where `d` indicates a debug version.

**Alternate Options**

None

**See Also**

- `Qvc`

**mtune**

*Performs optimizations for specific processors.*

**IDE Equivalent**

Windows: None

Linux: **Optimization > Optimize for Intel processor**

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-mtune=processor
```
Windows:
None

**Arguments**

`processor`  
Is the processor for which the compiler should perform optimizations. Possible values are:

- **generic**: Generates code for the compiler’s default behavior.
- **core2**: Optimizes for the Intel® Core™ 2 processor family, including support for MMX™, Intel® SSE, SSE2, SSE3 and SSSE3 instruction sets.
- **pentium**: Optimizes for Intel® Pentium® processors.
- **pentium-mmx**: Optimizes for Intel® Pentium® with MMX technology.
- **pentiumpro**: Optimizes for Intel® Pentium® Pro, Intel Pentium II, and Intel Pentium III processors.
- **pentium4**: Optimizes for Intel® Pentium® 4 processors.
- **pentium4m**: Optimizes for Intel® Pentium® 4 processors with MMX technology.
- **itanium2**: Optimizes for Intel® Itanium® 2 processors.
- **itanium2-p9000**: Optimizes for the Dual-Core Intel® Itanium® 2 processor 9000 series. This option affects the order of the generated instructions, but the generated instructions are limited to Intel® Itanium® 2 processor instructions unless the program uses (executes) intrinsics specific to the Dual-Core Intel® Itanium® 2 processor 9000 series.
**Default**

**generic**
On systems using IA-32 and Intel® 64 architectures, code is generated for the compiler's default behavior.

**itanium2-p9000**
On systems using IA-64 architecture, the compiler optimizes for the Dual-Core Intel® Itanium® 2 processor 9000 series.

**Description**
This option performs optimizations for specific processors.

The resulting executable is backwards compatible and generated code is optimized for specific processors. For example, code generated with `-mtune=itanium2-p9000` will run correctly on single-core Itanium® 2 processors, but it might not run as fast as if it had been generated using `-mtune=itanium2`.

The following table shows on which architecture you can use each value.

<table>
<thead>
<tr>
<th>processor Value</th>
<th>IA-32 architecture</th>
<th>Intel® 64 architecture</th>
<th>IA-64 architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>core2</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pentium</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pentium-mmx</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pentiumpro</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pentium4</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pentium4m</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>itanium2</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>itanium2-p9000</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Alternate Options

- *-mtune*  
  Linux: `-mcpu` *(this is a deprecated option)*  
  Mac OS X: None  
  Windows: None

- *-mtune=itanium2*  
  Linux: `-mcpu=itanium2` *(–mcpu is a deprecated option)*  
  Mac OS X: None  
  Windows: `/G2`

- *-mtune=itanium2-p9000*  
  Linux: `-mcpu=itanium2-p9000` *(–mcpu is a deprecated option)*  
  Mac OS X: None  
  Windows: `/G2-p9000`

**multibyte-chars, Qmultibyte-chars**

*Determine whether multi-byte characters are supported.*

**IDE Equivalent**

Windows: None

Linux: **Language > Support Multibyte Characters in Source**  
Mac OS X: **Language > Support Multibyte Characters in Source**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `multibyte-chars`
- `no-multibyte-chars`

**Windows:**

`/Qmultibyte-chars`  
`/Qmultibyte-chars-`

**Arguments**

None
**Default**

- `multibyte-chars` or `/Qmultibyte-chars` Multi-byte characters are supported.

**Description**

This option determines whether multi-byte characters are supported.

**Alternate Options**

None

**multiple-processes, MP**

*Creates multiple processes that can be used to compile large numbers of source files at the same time.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `multiple-processes [=n]`

**Windows:**

- `/MP[:n]`

**Arguments**

`n` Is the maximum number of processes that the compiler should create.

**Default**

OFF A single process is used to compile source files.
**Description**

This option creates multiple processes that can be used to compile large numbers of source files at the same time. It can improve performance by reducing the time it takes to compile source files on the command line.

This option causes the compiler to create one or more copies of itself, each in a separate process. These copies simultaneously compile the source files.

If \( n \) is not specified for this option, the default value is as follows:

- On Windows OS, the value is based on the setting of the NUMBER_OF_PROCESSORS environment variable.
- On Linux OS and Mac OS X, the value is 2.

This option applies to compilations, but not to linking or link-time code generation.

**Alternate Options**

None

**noBool**

Disables the `bool` keyword.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

None

Windows:

/noBool
Arguments
None

Default
OFF

The `bool` keyword is enabled.

Description
This option disables the `bool` keyword.

Alternate Options
None

`no-bss-init`, `Qnobss-init`
 Tells the compiler to place in the DATA section any variables explicitly initialized with zeros.

IDE Equivalent
Windows: None
Linux: **Data > Disable Placement of Zero-initialized Variables in .bss - use .data**
Mac OS X: **Data > Allocate Zero-initialized Variables to .data**

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
`-no-bss-init`

Windows:
`/Qnobss-init`

Arguments
None
Default

OFF

Variables explicitly initialized with zeros are placed in the BSS section.

Description

This option tells the compiler to place in the DATA section any variables explicitly initialized with zeros.

Alternate Options

Linux and Mac OS X: -nobss-init (this is a deprecated option)
Windows: None

nodefaultlibs

Prevents the compiler from using standard libraries when linking.

IDE Equivalent

Windows: None

Linux: Libraries > Use no system libraries
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- -nodefaultlibs

Windows:
None

Arguments

None
Default
OFF
The standard libraries are linked.

Description
This option prevents the compiler from using standard libraries when linking. It is provided for GNU compatibility.

Alternate Options
None

See Also
• nostdlib

nolib-inline
Disables inline expansion of standard library or intrinsic functions.

IDE Equivalent
Windows: None
Linux: Optimization > Disable Intrinsic Inline Expansion
Mac OS X: Optimization > Disable Intrinsic Inline Expansion

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-nolib-inline

Windows:
None

Arguments
None
Default

OFF

The compiler inlines many standard library and intrinsic functions.

Description

This option disables inline expansion of standard library or intrinsic functions. It prevents the unexpected results that can arise from inline expansion of these functions.

Alternate Options

None

Do not display compiler version information.

IDE Equivalent

Windows: General > Suppress Startup Banner
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

None

Windows:

/nologo

Arguments

None

Default

OFF
**Description**
Do not display compiler version information.

**Alternate Options**
None

**nostartfiles**
Prevents the compiler from using standard startup files when linking.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
-nostartfiles

Windows:
None

**Arguments**
None

**Default**
OFF

The compiler uses standard startup files when linking.

**Description**
This option prevents the compiler from using standard startup files when linking.

**Alternate Options**
None
See Also

- nostdlib

**nstdinc++**

*Do not search for header files in the standard directories for C++, but search the other standard directories.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `nstdinc++`

**Windows:**

None

**Arguments**

None

**Default**

OFF

**Description**

Do not search for header files in the standard directories for C++, but search the other standard directories.

**Alternate Options**

None
n ostdlib

Prevents the compiler from using standard libraries and startup files when linking.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- nostdlib

Windows:
None

Arguments

None

Default

OFF

The compiler uses standard startup files and standard libraries when linking.

Description

This option prevents the compiler from using standard libraries and startup files when linking. It is provided for GNU compatibility.

Alternate Options

None

See Also

•
  • n odefaul tlibs
  • n ostartfiles
Specifies the name for an output file.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-ofile

Windows:
None

Arguments
file Is the name for the output file. The space before file is optional.

Default
OFF The compiler uses the default file name for an output file.

Description
This option specifies the name for an output file as follows:
- If -c is specified, it specifies the name of the generated object file.
- If -S is specified, it specifies the name of the generated assembly listing file.
- If -P is specified, it specifies the name of the generated preprocessor file.

Otherwise, it specifies the name of the executable file.
NOTE. If you misspell a compiler option beginning with "o", such as -openmp, -opt-report, etc., the compiler interprets the misspelled option as an -o file option. For example, say you misspell "-opt-report" as "-opt-reprt"; in this case, the compiler interprets the misspelled option as "-o pt-reprt", where pt-reprt is the output file name.

Alternate Options
Linux and Mac OS X: None
Windows: /Fe

See Also
• Fe

0
Specifies the code optimization for applications.

IDE Equivalent
Windows: Optimization > Optimization
Linux: General > Optimization Level
Mac OS X: General > Optimization Level

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-0[n]
Windows:
/O[n]

Arguments
n Is the optimization level. Possible values are 1, 2, or 3. On Linux and Mac OS X systems, you can also specify 0.
Default

O2

Optimizes for code speed. This default may change depending on which other compiler options are specified. For details, see below.

Description

This option specifies the code optimization for applications.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O (Linux and Mac OS X)</td>
<td>This is the same as specifying O2.</td>
</tr>
<tr>
<td>O0 (Linux and Mac OS X)</td>
<td>Disables all optimizations. On systems using IA-32 architecture and Intel® 64 architecture, this option sets option -fno-omit-frame-pointer and option -fmath-errno.</td>
</tr>
<tr>
<td>O1</td>
<td>Enables optimizations for speed and disables some optimizations that increase code size and affect speed. To limit code size, this option:</td>
</tr>
<tr>
<td></td>
<td>• Enables global optimization; this includes data-flow analysis, code motion, strength reduction and test replacement, split-lifetime analysis, and instruction scheduling.</td>
</tr>
<tr>
<td></td>
<td>• Disables intrinsic recognition and intrinsics inlining.</td>
</tr>
<tr>
<td></td>
<td>• On systems using IA-64 architecture, it disables software pipelining, loop unrolling, and global code scheduling.</td>
</tr>
</tbody>
</table>

On systems using IA-64 architecture, this option also enables optimizations for server applications (straight-line and branch-like code with a flat profile). The O1 option sets the following options:

- On Linux and Mac OS X systems using IA-32 architecture and Intel® 64 architecture:
  -funroll-loops0,-fno-builtin,-mno-ieee-fp,-fomit-frame-pointer,-ffunction-sections,-ftz
- On Linux systems using IA-64 architecture:
  -funroll-loops0,-fbuiltin,-mno-ieee-fp,-fomit-frame-pointer,-ffunction-sections,-ftz
- On Windows systems using IA-32 architecture:
On Windows systems using Intel® 64 architecture and IA-64 architecture:
/Qunroll0, /Oi-, /Op-, /Oy, /Oy, /Os, /GF (/Qvc7 and above), /Gf (/Qvc6 and below), /Ob2, /Og, /Gftz

- The O1 option may improve performance for applications with very large code size, many branches, and execution time not dominated by code within loops.

O2

Enables optimizations for speed. This is the generally recommended optimization level. Vectorization is enabled at O2 and higher levels. On systems using IA-64 architecture, this option enables optimizations for speed, including global code scheduling, software pipelining, predication, and speculation. This option also enables:

- Inlining of intrinsics
- Intra-file interprocedural optimization, which includes:
  - inlining
  - constant propagation
  - forward substitution
  - routine attribute propagation
  - variable address-taken analysis
  - dead static function elimination
  - removal of unreferenced variables
- The following capabilities for performance gain:
  - constant propagation
  - copy propagation
  - dead-code elimination
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| • global register allocation  
• global instruction scheduling and control speculation  
• loop unrolling  
• optimized code selection  
• partial redundancy elimination  
• strength reduction/induction variable simplification  
• variable renaming  
• exception handling optimizations  
• tail recursions  
• peephole optimizations  
• structure assignment lowering and optimizations  
• dead store elimination |  

The **O2** option sets the following options:

- On Windows systems using IA-32 architecture:
  - /Og, /Oi-, /Os, /Gy, /Ob2, /GF (/Qvc7 and above), /GF (/Qvc6 and below), /Gs, /Gy, and /Qftz
- On Windows systems using Intel® 64 architecture:
  - /Og, /Oi-, /Os, /Ob2, /GF (/Qvc7 and above), /GF (/Qvc6 and below), /Gs, /Gy, and /Qftz

This option sets other options that optimize for code speed. The options set are determined by the compiler depending on which architecture and operating system you are using.

Enables **O2** optimizations plus more aggressive optimizations, such as prefetching, scalar replacement, and loop and memory access transformations. Enables optimizations for maximum speed, such as:

- Loop unrolling, including instruction scheduling  
- Code replication to eliminate branches
Padding the size of certain power-of-two arrays to allow more efficient cache use.

On Windows systems, the O3 option sets the /GF (/Qvc7 and above), /GF (/Qvc6 and below), and /Ob2 option.

On Linux and Mac OS X systems, the O3 option sets option -fomit-frame-pointer.

On systems using IA-32 architecture or Intel® 64 architecture, when O3 is used with options -ax or -x (Linux) or with options /Qax or /Qx (Windows), the compiler performs more aggressive data dependency analysis than for O2, which may result in longer compilation times.

On systems using IA-64 architecture, the O3 option enables optimizations for technical computing applications (loop-intensive code): loop optimizations and data prefetch.

The O3 optimizations may not cause higher performance unless loop and memory access transformations take place. The optimizations may slow down code in some cases compared to O2 optimizations.

The O3 option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets.

The last option specified on the command line takes precedence over any others.

NOTE. The options set by the O option may change from release to release.

Alternate Options

O1 Linux and Mac OS X: None
Windows: /Od

See Also

- Od
**Oa**
*Tells the compiler to assume there is no aliasing.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
None

**Windows:**
/Oa
/Oa-

**Arguments**
None

**Default**
OFF The compiler assumes there is aliasing.

**Description**
This option tells the compiler to assume there is no aliasing.

**Alternate Options**
None

**inline-level, Ob**
*Specifies the level of inline function expansion.*

**IDE Equivalent**
Windows: **Optimization > Inline Function Expansion**
Linux: Optimization > Inline Function Expansion
Mac OS X: Optimization > Inline Function Expansion

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-`-inline-level=n`

Windows:
-/Obn

Arguments

`n`

Is the inline function expansion level. Possible values are 0, 1, and 2.

Default

`-inline-level=2` or `/Ob2` This is the default if option `-O2` is specified or is in effect by default. On Windows systems, this is also the default if option `-O3` is specified.

`-inline-level=0` or `/Ob0` This is the default if option `-O0` (Linux and Mac OS) or `/Od` (Windows) is specified.

Description

This option specifies the level of inline function expansion. Inlining procedures can greatly improve the run-time performance of certain programs.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-inline-level=0</code> or</td>
<td>Disables inlining of user-defined functions. Note that statement functions are always inlined.</td>
</tr>
<tr>
<td><code>/Ob0</code></td>
<td></td>
</tr>
<tr>
<td><code>-inline-level=1</code> or</td>
<td>Enables inlining when an inline keyword or an inline attribute is specified. Also enables inlining according to the C++ language.</td>
</tr>
<tr>
<td><code>/Ob1</code></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-inline-level=2 or</td>
<td>Enables inlining of any function at the compiler’s discretion.</td>
</tr>
<tr>
<td>Ob2</td>
<td></td>
</tr>
</tbody>
</table>

**Alternate Options**

- **Linux**: `-Ob` (this is a deprecated option)
- **Mac OS X**: None
- **Windows**: None

**Od**

Disables all optimizations.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

- **Linux and Mac OS X**: None
- **Windows**: `/Od`

**Arguments**

None

**Default**

OFF The compiler performs default optimizations.
Description
This option disables all optimizations. It can be used for selective optimizations, such as a combination of /Od and /Og (disables all optimizations except global optimizations), or /Od and /Ob1 (disables all optimizations, but enables inlining).

On IA-32 architecture, this option sets the /Oy- option.

Alternate Options
Linux and Mac OS X: -00
Windows: None

See Also
•
• o

Og
Enables global optimizations.

IDE Equivalent
Windows: Optimization > Global Optimization
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/Og
/Og-
Arguments
None

Default
/Og
Global optimizations are enabled unless /Od is specified.

Description
This option enables global optimizations.

Alternate Options
None

fbuiltin, Oi
Enables or disables inline expansion of intrinsic functions.

IDE Equivalent
Windows: Optimization > Enable Intrinsic Functions
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-xfbuiltin[-func]
-fno-builtin[-func]

Windows:
/Oi[-]

Arguments
func A comma-separated list of intrinsic functions.
**Default**

**OFF**

Inline expansion of intrinsic functions disabled.

**Description**

This option enables or disables inline expansion of one or more intrinsic functions. If `-func` is not specified, `-fno-builtin` disables inline expansion for all intrinsic functions.

For a list of built-in functions affected by `-fbuiltin`, search for "built-in functions" in the appropriate gcc* documentation.

For a list of built-in functions affected by `/Oi`, search for "/Oi" in the appropriate Microsoft* Visual C/C++* documentation.

**Alternate Options**

None

**Op**

Enables conformance to the ANSI C and IEEE 754 standards for floating-point arithmetic. This is a deprecated option.

**IDE Equivalent**

Windows: Optimization > Floating-point Precision Improvement

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Op

/Op-
Arguments

None

Default

OFF

Description

This option enables conformance to the ANSI C and IEEE 754 standards for floating-point arithmetic.

It restricts some optimizations to maintain declared precision and to ensure that floating-point arithmetic conforms more closely to the ANSI and IEEE standards. Floating point intermediate results are kept in full 10-byte internal precision. All spills and reloads of the x87 floating-point registers utilize this internal format to prevent accidental loss of precision.

For most programs, specifying this option adversely affects performance. If you are not sure whether your application needs this option, try compiling and running your program both with and without it to evaluate the effects on performance versus precision. Alternatives to /Op include /QxN (for the Intel® Pentium™ 4 processor or newer) and /Qprec.

Specifying the /Op option has the following effects on program compilation:

- User variables declared as floating-point types are not assigned to registers.
- Whenever an expression is spilled (moved from a register to memory), it is spilled as 80 bits (extended precision), not 64 bits (double precision).
- Floating-point arithmetic comparisons conform to the IEEE 754 specification except for NaN behavior.
- The exact operations specified in the code are performed. For example, division is never changed to multiplication by the reciprocal.
- The compiler performs floating-point operations in the order specified without re-association.
- The compiler does not perform the constant-folding optimization on floating-point values. Constant folding also eliminates any multiplication by 1, division by 1, and addition or subtraction of 0. For example, code that adds 0.0 to a number is executed exactly as written. Compile-time floating-point arithmetic is not performed to ensure that floating-point exceptions are also maintained.
Floating-point operations conform to ANSI C. When assignments to type float and double are made, the precision is rounded from 80 bits (extended) down to 32 bits (float) or 64 bits (double). When you do not specify /Op, the extra bits of precision are not always rounded before the variable is reused.

It sets the /Oi- option, which disables inline functions expansion.

The recommended method to control the semantics of floating-point calculations is to use option /fp.

**Alternate Options**

None

**See Also**

- fp-model, fp

**openmp, Qopenmp**

*Enables the parallelizer to generate multi-threaded code based on the OpenMP* directives.

**IDE Equivalent**

Windows: *Language > OpenMP* Support

Linux: *Language > Process OpenMP Directives*

Mac OS X: *Language > Process OpenMP Directives*

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `openmp`

**Windows:**

- `/Qopenmp`
Arguments
None

Default
OFF  No OpenMP multi-threaded code is generated by the compiler.

Description
This option enables the parallelizer to generate multi-threaded code based on the OpenMP* directives. The code can be executed in parallel on both uniprocessor and multiprocessor systems.

This option works with any optimization level. Specifying no optimization (-O0 on Linux or /Od on Windows) helps to debug OpenMP applications.

NOTE. On Mac OS X systems, when you enable OpenMP*, you must also set the DYLD_LIBRARY_PATH environment variable within Xcode or an error will be displayed.

Alternate Options
None

See Also
•
  •
  •  openmp-stubs, Qopenmp-stubs

donmp-lib, Qopenmp-lib
Let you specify an OpenMP* run-time library to use for linking.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax
Linux:
-openmp-lib type

Mac OS X:
None

Windows:
/Qopenmp-lib:type

Arguments

type

Specifies the type of library to use; it implies compatibility levels. Possible values are:

 legacy
 Tells the compiler to use the legacy OpenMP run-time library (libguide). This setting does not provide compatibility with object files created using other compilers. This is a deprecated option.

 compat
 Tells the compiler to use the compatibility OpenMP run-time library (libiomp). This setting provides compatibility with object files created using Microsoft* and GNU* compilers.

Default

-openmp-lib compat
or/Qopenmp-lib:compat

The compiler uses the compatibility OpenMP run-time library (libiomp).

Description

This option lets you specify an OpenMP run-time library to use for linking.

The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers.

The compatibility OpenMP run-time library is compatible with object files created using the Microsoft* OpenMP run-time library (vcomp) and GNU OpenMP run-time library (libgomp).
To use the compatibility OpenMP run-time library, compile and link your application using the
–openmp-lib compat (Linux) or /Qopenmp-lib:compat (Windows) option. To use this option,
you must also specify one of the following compiler options:

- Linux OS: –openmp, –openmp-profile, or –openmp-stubs
- Windows OS: /Qopenmp, /Qopenmp-profile, or /Qopenmp-stubs

On Windows* systems, the compatibility OpenMP* run-time library lets you combine OpenMP*
object files compiled with the Microsoft* C/C++ compiler with OpenMP* object files compiled
with the Intel C/C++ or Fortran compilers. The linking phase results in a single, coherent copy
of the run-time library.

On Linux* systems, the compatibility Intel OpenMP* run-time library lets you combine OpenMP*
object files compiled with the GNU* gcc or gfortran compilers with similar OpenMP* object files
compiled with the Intel C/C++ or Fortran compilers. The linking phase results in a single,
coherent copy of the run-time library.

**NOTE.** The compatibility OpenMP run-time library is not compatible with object files
created using versions of the Intel compiler earlier than 10.0.

**Alternate Options**
None

**See Also**

- openmp, Qopenmp
- openmp-stubs, Qopenmp-stubs
- openmp-profile, Qopenmp-profile

**openmp-link, Qopenmp-link**

*Controls whether the compiler links to static or
dynamic OpenMP run-time libraries.*

**IDE Equivalent**
None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
-openmp-link library

**Windows:**
/Qopenmp-link:library

Arguments

library Specifies the OpenMP library to use. Possible values are:

- static Tells the compiler to link to static OpenMP run-time libraries.
- dynamic Tells the compiler to link to dynamic OpenMP run-time libraries.

Default

-openmp-link dynamic or /Qopenmp-link:dynamic The compiler links to dynamic OpenMP run-time libraries. However, if option static is specified, the compiler links to static OpenMP run-time libraries.

Description

This option controls whether the compiler links to static or dynamic OpenMP run-time libraries.

To link to the static OpenMP run-time library (RTL) and create a purely static executable, you must specify -openmp-link static (Linux and Mac OS X) or /Qopenmp-link:static (Windows). However, we strongly recommend you use the default setting, -openmp-link dynamic (Linux and Mac OS X) or /Qopenmp-link:dynamic (Windows).

**NOTE.** Compiler options -static-intel and -shared-intel (Linux and Mac OS X) have no effect on which OpenMP run-time library is linked.
Alternate Options
None

**openmp-profile, Qopenmp-profile**

*Enables analysis of OpenMP* applications if Intel® Thread Profiler is installed.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux:**
- `openmp-profile`

**Mac OS X:**
None

**Windows:**
`/Qopenmp-profile`

Arguments
None

Default
OFF

OpenMP applications are not analyzed.

Description
This option enables analysis of OpenMP* applications. To use this option, you must have previously installed Intel® Thread Profiler, which is one of the Intel® Threading Analysis Tools.

This option can adversely affect performance because of the additional profiling and error checking invoked to enable compatibility with the threading tools. Do not use this option unless you plan to use the Intel® Thread Profiler.
For more information about Intel® Thread Profiler, open the page associated with threading tools at Intel® Software Development Products.

**Alternate Options**

None

**openmp-report, Qopenmp-report**

*Controls the OpenMP® parallelizer's level of diagnostic messages.*

**IDE Equivalent**

Windows: None

Linux: **Compilation Diagnostics > OpenMP Report**

Mac OS X: **Diagnostics > OpenMP Report**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

\[-openmp-report\][n]

**Windows:**

\[/Qopenmp-report\][n]

**Arguments**

\(n\)

Is the level of diagnostic messages to display. Possible values are:

- **0**: No diagnostic messages are displayed.
- **1**: Diagnostic messages are displayed indicating loops, regions, and sections successfully parallelized.
- **2**: The same diagnostic messages are displayed as specified by openmp_report1 plus diagnostic messages indicating...
successful handling of MASTER constructs, SINGLE constructs, CRITICAL constructs, ORDERED constructs, ATOMIC directives, and so forth.

**Default**

- `openmp-report1`
  - `or/Qopenmp-report1`

  If you do not specify \( n \), the compiler displays diagnostic messages indicating loops, regions, and sections successfully parallelized. If you do not specify the option on the command line, the default is to display no messages.

**Description**

This option controls the OpenMP* parallelizer's level of diagnostic messages. To use this option, you must also specify `-openmp` (Linux and Mac OS X) or `/Qopenmp` (Windows).

If this option is specified on the command line, the report is sent to stdout.

**Alternate Options**

None

**See Also**

- `openmp`, `Qopenmp`

  Optimizing Applications:
  - Using Parallelism
  - OpenMP* Report

**openmp-stubs, Qopenmp-stubs**

*Enables compilation of OpenMP programs in sequential mode.*

**IDE Equivalent**

Windows: Language > Process OpenMP Directives

Linux: Language > Process OpenMP Directives
Mac OS X: Language > Process OpenMP Directives

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- -openmp-stubs
Windows:
/Qopenmp-stubs

Arguments
None

Default
OFF  The library of OpenMP function stubs is not linked.

Description
This option enables compilation of OpenMP programs in sequential mode. The OpenMP directives are ignored and a stub OpenMP library is linked.

Alternate Options
None

See Also
- 
- 
- openmp, Qopenmp

openmp-task, Qopenmp-task
Lets you choose an OpenMP* tasking model.

IDE Equivalent
None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-oopenmp-task model

Windows:
/Qopenmp-task:model

Arguments
model
Is an OpenMP tasking model. Possible values are:

*intel*
Tells the compiler to accept Intel® taskqueuing pragmas (#pragma intel_omp_taskq and #pragma intel_omp_task). When this value is specified, OpenMP 3.0 tasking pragmas are ignored; if they are specified, warnings are issued.

*omp*
Tells the compiler to accept OpenMP 3.0 tasking pragmas (#pragma omp_task). When this value is specified, Intel taskqueuing pragmas are ignored; if they are specified, warnings are issued.

Default
-oopenmp-task omp or /Qopenmp-task:omp
The compiler accepts OpenMP 3.0 tasking pragmas.

Description
The option lets you choose an OpenMP tasking model.
To use this option, you must also specify option -openmp (Linux and Mac OS X) or /Qopenmp (Windows).
Alternate Options
None

**openmp-threadprivate, Qopenmp-threadprivate**

*Let you specify an OpenMP* threadprivate implementation.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**

`-openmp-threadprivate type`

**Mac OS X:**

None

**Windows:**

`/Qopenmp-threadprivate:type`

**Arguments**

`type`  
Specifies the type of threadprivate implementation. Possible values are:

- `legacy`  
  Tells the compiler to use the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel® compiler. This setting does not provide compatibility with the implementation used by other compilers.

- `compat`  
  Tells the compiler to use the compatibility OpenMP* threadprivate implementation based on applying the __declspec(thread) attribute to each threadprivate variable.
The limitations of the attribute on a given platform also apply to the threadprivate implementation. This setting provides compatibility with the implementation provided by the Microsoft* and GNU* compilers.

**Default**

- `openmp-threadprivate legacy` or `/Qopenmp-threadprivate:legacy`

The compiler uses the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel® compiler.

**Description**

This option lets you specify an OpenMP* threadprivate implementation.

The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers.

To use this option, you must also specify one of the following compiler options:

- **Linux OS:** `-openmp`, `-openmp-profile`, or `-openmp-stubs`
- **Windows OS:** `/Qopenmp`, `/Qopenmp-profile`, or `/Qopenmp-stubs`

The value specified for this option is independent of the value used for option `-openmp-lib` (Linux) or `/Qopenmp-lib` (Windows).

**Alternate Options**

None

**opt-block-factor, Qopt-block-factor**

*Let you specify a loop blocking factor.*

**IDE Equivalent**

Windows: Diagnostics > Optimization Diagnostic File

Diagnostics > Emit Optimization Diagnostics to File

Linux: None

Mac OS X: None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and MacOS X:
- opt-block-factor=n

Windows:
/Qopt-block-factor:n

Arguments
n Is the blocking factor. It must be an integer. The compiler may ignore the blocking factor if the value is 0 or 1.

Default
OFF The compiler uses default heuristics for loop blocking.

Description
This option lets you specify a loop blocking factor.

Alternate Options
None

opt-calloc
Tells the compiler to substitute a call to _intel_fast_calloc() for a call to calloc().

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures
Syntax

Linux:
- `opt-calloc`
- `no-opt-calloc`

Mac OS X:
None

Windows:
None

Arguments
None

Default

- `no-opt-calloc` The compiler does not substitute a call to `_intel_fast_calloc()` for a call to `calloc()`.

Description
This option tells the compiler to substitute a call to `_intel_fast_calloc()` for a call to `calloc()`.

This option may increase the performance of long-running programs that use `calloc()` frequently. It is recommended for these programs over combinations of options `-inline-calloc` and `-opt-malloc-options=3` because this option causes less memory fragmentation.

Alternate Options
None

`opt-class-analysis, Qopt-class-analysis`

Determines whether C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time.

IDE Equivalent
None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- `-opt-class-analysis`
- `-no-opt-class-analysis`

Windows:
/`Qopt-class-analysis`
/`Qopt-class-analysis-`

Arguments
None

Default

- `-no-opt-class-analysis` C++ class hierarchy information is not used to analyze and resolve C++ virtual function calls at compile time.

Description
This option determines whether C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time. The option is turned on by default with the `-ipo` compiler option, enabling improved C++ optimization. If a C++ application contains non-standard C++ constructs, such as pointer down-casting, it may result in different behaviors.

Alternate Options
None

`opt-jump-tables, Qopt-jump-tables`
Enables or disables generation of jump tables for switch statements.

IDEEquivalent
None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- opt-jump-tables=keyword
- no-opt-jump-tables

Windows:
/Qopt-jump-tables:keyword
/Qopt-jump-tables-

Arguments
keyword Is the instruction for generating jump tables. Possible values are:

never Tells the compiler to never generate jump tables. All switch statements are implemented as chains of if-then-elses. This is the same as specifying -no-opt-jump-tables (Linux and Mac OS) or /Qopt-jump-tables- (Windows).

default The compiler uses default heuristics to determine when to generate jump tables.

large Tells the compiler to generate jump tables up to a certain pre-defined size (64K entries).

n Must be an integer. Tells the compiler to generate jump tables up to n entries in size.

Default
-opt-jump-tables=default The compiler uses default heuristics to determine when to generate jump tables for switch statements.
or/Qopt-jump-tables:default

Description
This option enables or disables generation of jump tables for switch statements. When the option is enabled, it may improve performance for programs with large switch statements.

Alternate Options
None

opt-loadpair, Qopt-loadpair
Enables or disables loadpair optimization.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux:
-oct-loadpair
-noct-loadpair

Mac OS X:
None

Windows:
/Qopt-loadpair
/Qopt-loadpair-

Arguments
None
Default
-no-opt-loadpair  Loadpair optimization is disabled unless option 03 is specified.
or/Qopt-loadpair-

Description
This option enables or disables loadpair optimization.
When -O3 is specified on IA-64 architecture, loadpair optimization is enabled by default. To disable loadpair generation, specify -no-opt-loadpair (Linux) or /Qopt-loadpair- (Windows).

Alternate Options
None

opt-malloc-options
Let you specify an alternate algorithm for malloc().

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-opt-malloc-options=n

Windows:
None

Arguments
n  Specifies the algorithm to use for malloc(). Possible values are:
0  Tells the compiler to use the default algorithm for malloc(). This is the default.
Causes the following adjustments to the malloc() algorithm: \texttt{M\_MMAP\_MAX}=2 and \texttt{M\_TRIM\_THRESHOLD}=0x10000000.

Causes the following adjustments to the malloc() algorithm: \texttt{M\_MMAP\_MAX}=2 and \texttt{M\_TRIM\_THRESHOLD}=0x40000000.

Causes the following adjustments to the malloc() algorithm: \texttt{M\_MMAP\_MAX}=0 and \texttt{M\_TRIM\_THRESHOLD}=-1.

Causes the following adjustments to the malloc() algorithm: \texttt{M\_MMAP\_MAX}=0, \texttt{M\_TRIM\_THRESHOLD}=-1, \texttt{M\_TOP\_PAD}=4096.

Default

\texttt{-opt-malloc-options=0} The compiler uses the default algorithm when malloc() is called. No call is made to mallopt().

Description

This option lets you specify an alternate algorithm for malloc().

If you specify a non-zero value for \( n \), it causes alternate configuration parameters to be set for how malloc() allocates and frees memory. It tells the compiler to insert calls to mallopt() to adjust these parameters to malloc() for dynamic memory allocation. This may improve speed.

Alternate Options

None

See Also

• malloc(3) man page

mallopt function (defined in malloc.h)
**opt-mem-bandwidth, Qopt-mem-bandwidth**

*Enables performance tuning and heuristics that control memory bandwidth use among processors.*

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

**Linux:**
- `-opt-mem-bandwidth n`

**Mac OS X:**
None

**Windows:**
- `/Qopt-mem-bandwidth n`

**Arguments**

`n` is the level of optimizing for memory bandwidth usage. Possible values are:

- **0** Enables a set of performance tuning and heuristics in compiler optimizations that is optimal for serial code.
- **1** Enables a set of performance tuning and heuristics in compiler optimizations for multithreaded code generated by the compiler.
- **2** Enables a set of performance tuning and heuristics in compiler optimizations for parallel code such as Windows Threads, pthreads, and MPI code, besides multithreaded code generated by the compiler.
Default

- `opt-mem-bandwidth0` For serial (non-parallel) compilation, a set of performance tuning and heuristics in compiler optimizations is enabled that is optimal for serial code.

- `opt-mem-bandwidth1` If you specify compiler option `-parallel` (Linux) or `/Qparallel` (Windows), or `-openmp` (Linux) or `/Qopenmp` (Windows), a set of performance tuning and heuristics in compiler optimizations for multithreaded code generated by the compiler is enabled.

Description

This option enables performance tuning and heuristics that control memory bandwidth use among processors. It allows the compiler to be less aggressive with optimizations that might consume more bandwidth, so that the bandwidth can be well-shared among multiple processors for a parallel program.

For values of \( n \) greater than 0, the option tells the compiler to enable a set of performance tuning and heuristics in compiler optimizations such as prefetching, privatization, aggressive code motion, and so forth, for reducing memory bandwidth pressure and balancing memory bandwidth traffic among threads.

This option can improve performance for threaded or parallel applications on multiprocessors or multicore processors, especially when the applications are bounded by memory bandwidth.

Alternate Options

None

See Also

- parallel, Qparallel
- openmp, Qopenmp

opt-mod-versioning, Qopt-mod-versioning

Enables or disables versioning of modulo operations for certain types of operands.

IDE Equivalent

None
Architectures
IA-64 architecture

Syntax
Linux:
- opt-mod-versioning
- no-opt-mod-versioning

Mac OS X:
None

Windows:
/Q opt-mod-versioning
/Q opt-mod-versioning-

Arguments
None

Default
-no-opt-mod-versioning Versioning of modulo operations is disabled.
or/Qopt-mod-versioning-

Description
This option enables or disables versioning of modulo operations for certain types of operands. It is used for optimization tuning.

Versioning of modulo operations may improve performance for x mod y when modulus y is a power of 2.

Alternate Options
None
opt-multi-version-aggressive, Qopt-multi-version-aggressive

Tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-opt-multi-version-aggressive
-no-opt-multi-version-aggressive

Windows:
/Qopt-multi-version-aggressive
/Qopt-multi-version-aggressive-

Arguments
None

Default
-no-opt-multi-version- The compiler uses default heuristics when checking for pointer aliasing and scalar replacement. or/Qopt-multi-version-
aggressive-

Description
This option tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement. This option may improve performance.

Alternate Options
None
**opt-prefetch, Qopt-prefetch**

*Enables or disables prefetch insertion optimization.*

**IDE Equivalent**

Windows: None

Linux: **Optimization > Enable Prefetch Insertion**

Mac OS X: **Optimization > Enable Prefetch Insertion**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `--opt-prefetch[=n]`
- `--no-opt-prefetch`

Windows:

- `/Qopt-prefetch[:n]`
- `/opt-prefetch-`

**Arguments**

- `n` is the level of detail in the report. Possible values are:
  - `0` Disables software prefetching. This is the same as specifying `--no-opt-prefetch` (Linux and Mac OS X) or `/Qopt-prefetch-` (Windows).
  - `1 to 4` Enables different levels of software prefetching. If you do not specify a value for `n`, the default is 2 on IA-32 and Intel® 64 architecture; the default is 3 on IA-64 architecture. Use lower values to reduce the amount of prefetching.
**Default**

IA-64 architecture: `-opt-prefetch` or `/Qopt-prefetch`

IA-32 architecture and Intel®: On IA-32 architecture and Intel® 64 architecture, prefetch insertion optimization is disabled.

- `-no-opt-prefetch`
- `/Qopt-prefetch-`

**Description**

This option enables or disables prefetch insertion optimization. The goal of prefetching is to reduce cache misses by providing hints to the processor about when data should be loaded into the cache.

On IA-64 architecture, this option is enabled by default if you specify option `O1` or higher. To disable prefetching at these optimization levels, specify `-no-opt-prefetch` (Linux and Mac OS X) or `/Qopt-prefetch-` (Windows).

On IA-32 architecture and Intel® 64 architecture, this option enables prefetching when higher optimization levels are specified.

**Alternate Options**

Linux and Mac OS X: `-prefetch` (this is a deprecated option)

Windows: `/Qprefetch` (this is a deprecated option)

**opt-prefetch-initial-values, Qopt-prefetch-initial-values**

Enables or disables prefetches that are issued before a loop is entered.

**IDE Equivalent**

None

**Architectures**

IA-64 architecture
Syntax

Linux:
- opt-prefetch-initial-values
- no-opt-prefetch-initial-values

Mac OS X:
None

Windows:
/Qopt-prefetch-initial-values
/Qopt-prefetch-initial-values-

Arguments
None

Default
-opt-prefetch-initial-values Prefetches are issued before a loop is entered. values
or /Qopt-prefetch-initial-values

Description
This option enables or disables prefetches that are issued before a loop is entered. These prefetches target the initial iterations of the loop.

When -O1 or higher is specified on IA-64 architecture, prefetches are issued before a loop is entered. To disable these prefetches, specify -no-opt-prefetch-initial-values (Linux) or /Qopt-prefetch-initial-values- (Windows).

Alternate Options
None
**opt-prefetch-issue-excl-hint, Qopt-prefetch-issue-excl-hint**

Determines whether the compiler issues prefetches for stores with exclusive hint.

**IDE Equivalent**
None

**Architectures**
IA-64 architecture

**Syntax**

**Linux:**
- `opt-prefetch-issue-excl-hint`
- `no-opt-prefetch-issue-excl-hint`

**Mac OS X:**
None

**Windows:**
/Qopt-prefetch-issue-excl-hint
/Qopt-prefetch-issue-excl-hint-

**Arguments**
None

**Default**
- `no-opt-prefetch-issue-excl-hint`
  The compiler does not issue prefetches for stores with exclusive hint.
- or/Qopt-prefetch-issue-excl-hint-
Description
This option determines whether the compiler issues prefetches for stores with exclusive hint. If option `-opt-prefetch-issue-excl-hint` (Linux) or `/Qopt-prefetch-issue-excl-hint` (Windows) is specified, the prefetches will be issued if the compiler determines it is beneficial to do so.

When prefetches are issued for stores with exclusive-hint, the cache-line is in "exclusive-mode". This saves on cache-coherence traffic when other processors try to access the same cache-line. This feature can improve performance tuning.

Alternate Options
None

`opt-prefetch-next-iteration`, `Qopt-prefetch-next-iteration`
Enables or disables prefetches for a memory access in the next iteration of a loop.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux:
- `opt-prefetch-next-iteration`
- `no-opt-prefetch-next-iteration`

Mac OS X:
None

Windows:
/`Qopt-prefetch-next-iteration`
/`Qopt-prefetch-next-iteration-`
Arguments
None

Default
-opt-prefetch-next-iteration Prefetches are issued for a memory access in the next iteration of a loop.
or/Qopt-prefetch-next-iteration

Description
This option enables or disables prefetches for a memory access in the next iteration of a loop. It is typically used in a pointer-chasing loop.
When -O1 or higher is specified on IA-64 architecture, prefetches are issued for a memory access in the next iteration of a loop. To disable these prefetches, specify -no-opt-prefetch-next-iteration (Linux) or /Qopt-prefetch-next-iteration- (Windows).

Alternate Options
None

opt-ra-region-strategy, Qopt-ra-region-strategy
Selects the method that the register allocator uses to partition each routine into regions.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-opt-ra-region-strategy[=keyword]

Windows:
/Qopt-ra-region-strategy[=keyword]
Arguments

*keyword*  Is the method used for partitioning. Possible values are:

- **routine**: Creates a single region for each routine.
- **block**: Partitions each routine into one region per basic block.
- **trace**: Partitions each routine into one region per trace.
- **region**: Partitions each routine into one region per loop.
- **default**: The compiler determines which method is used for partitioning.

Default

`-opt-ra-region-strategy=default`  The compiler determines which method is used for partitioning. This is also the default if *keyword* is not specified.

`or/Qopt-ra-region-strategy=default`

Description

This option selects the method that the register allocator uses to partition each routine into regions.

When setting `default` is in effect, the compiler attempts to optimize the tradeoff between compile-time performance and generated code performance.

This option is only relevant when optimizations are enabled (`O1` or higher).

Alternate Options

None

See Also

- •
- •
- 0

548
**opt-report, Qopt-report**

*Tells the compiler to generate an optimization report to stderr.*

**IDE Equivalent**

Windows:  **Diagnostics > Optimization Diagnostic Level**

Linux:  **Compilation Diagnostics > Optimization Diagnostic Level**

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `opt-report [n]`

**Windows:**

/`Qopt-report[:n]`

**Arguments**

`n`  
Is the level of detail in the report. On Linux OS and Mac OS X systems, a space must appear before the `n`. Possible values are:

0  
Tells the compiler to generate no optimization report.

1  
Tells the compiler to generate a report with the minimum level of detail.

2  
Tells the compiler to generate a report with the medium level of detail.

3  
Tells the compiler to generate a report with the maximum level of detail.
**Default**

- `opt-report 2` or `/Qopt-report:2`
  If you do not specify `n`, the compiler generates a report with medium detail. If you do not specify the option on the command line, the compiler does not generate an optimization report.

**Description**

This option tells the compiler to generate an optimization report to `stderr`.

**Alternate Options**

None

**See Also**

- `opt-report-file`, `Qopt-report-file`
  Optimizing Applications: Optimizer Report Generation

**opt-report-file, Qopt-report-file**

Specifies the name for an optimization report.

**IDE Equivalent**

Windows: Diagnostics > Optimization Diagnostic File

Diagnostics > Emit Optimization Diagnostics to File

Linux: Compilation Diagnostics > Emit Optimization Diagnostics to File

Compilation Diagnostics > Optimization Diagnostics File

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `opt-report-file=file`
Windows:
/Qopt-report-file:file

Arguments

file Is the name for the optimization report.

Default
OFF No optimization report is generated.

Description
This option specifies the name for an optimization report. If you use this option, you do not have to specify -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

Alternate Options
None

See Also
- opt-report, Qopt-report

Optimizing Applications: Optimizer Report Generation

opt-report-help, Qopt-report-help
Displays the optimizer phases available for report generation.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
- `opt-report-help`

Windows:
/`Qopt-report-help`

Arguments
None

Default
OFF
No optimization reports are generated.

Description
This option displays the optimizer phases available for report generation using `-opt-report-phase` (Linux and Mac OS X) or `/Qopt-report-phase` (Windows). No compilation is performed.

Alternate Options
None

See Also
•
•
• `opt-report, Qopt-report`
• `opt-report-phase, Qopt-report-phase`

`opt-report-phase, Qopt-report-phase`
Specifies an optimizer phase to use when optimization reports are generated.

IDE Equivalent
Windows: Diagnostics > Optimization Diagnostic Phase
Linux: Compilation Diagnostics > Optimization Diagnostic Phase
Mac OS X: None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
  -opt-report-phase=phase
Windows:
  /Qopt-report-phase:phase

Arguments
phase
  Is the phase to generate reports for. Some of the possible values are:
  
  * ipo: The Interprocedural Optimizer phase
  * hlo: The High Level Optimizer phase
  * hpo: The High Performance Optimizer phase
  * ilo: The Intermediate Language Scalar Optimizer phase
  * ecg: The Code Generator phase (Windows and Linux systems using IA-64 architecture only)
  * ecg_swp: The software pipelining component of the Code Generator phase (Windows and Linux systems using IA-64 architecture only)
  * pgo: The Profile Guided Optimization phase
  * all: All optimizer phases

Default
OFF
  No optimization reports are generated.
Description
This option specifies an optimizer phase to use when optimization reports are generated. To use this option, you must also specify -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).
This option can be used multiple times on the same command line to generate reports for multiple optimizer phases.
When one of the logical names for optimizer phases is specified for phase, all reports from that optimizer phase are generated.
To find all phase possibilities, use option -opt-report-help (Linux and Mac OS X) or /Qopt-report-help (Windows).

Alternate Options
None

See Also

- opt-report, Qopt-report

opt-report-routine, Qopt-report-routine
Tells the compiler to generate reports on the routines containing specified text.

IDE Equivalent
Windows: Diagnostics > Optimization Diagnostic Routine
Linux: Compilation Diagnostics > Optimization Diagnostic Routine
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-opt-report-routine=string
Windows:
/Qopt-report-routine:string

Arguments

string Is the text (string) to look for.

Default

OFF No optimization reports are generated.

Description

This option tells the compiler to generate reports on the routines containing specified text as part of their name.

Alternate Options

None

See Also

- opt-report, Qopt-report

opt-streaming-stores, Qopt-streaming-stores
Enables generation of streaming stores for optimization.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:

/opt-streaming-stores keyword
Windows:
/Qopt-streaming-stores:keyword

Arguments

keyword Specifies whether streaming stores are generated. Possible values are:
always Enables generation of streaming stores for optimization. The compiler optimizes under the assumption that the application is memory bound.
never Disables generation of streaming stores for optimization. Normal stores are performed.
auto Lets the compiler decide which instructions to use.

Default

-opt-streaming-stores auto or /Qopt-streaming-stores:auto

Description

This option enables generation of streaming stores for optimization. This method stores data with instructions that use a non-temporal buffer, which minimizes memory hierarchy pollution.

For this option to be effective, the compiler must be able to generate SSE2 (or higher) instructions. For more information, see compiler option x or ax.

This option may be useful for applications that can benefit from streaming stores.

Alternate Options

None

See Also

•

•
• ax, Qax
• x, Qx
• opt-mem-bandwidth, Qopt-mem-bandwidth, Qx

**opt-subscript-in-range, Qopt-subscript-in-range**

_Determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops._

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

- opt-subscript-in-range
- no-opt-subscript-in-range

**Windows:**

/Qopt-subscript-in-range
/Qopt-subscript-in-range-

**Arguments**

None

**Default**

- no-opt-subscript-in-range
- or/Qopt-subscript-in-range-

The compiler assumes overflows in the intermediate computation of subscript expressions in loops.
Description

This option determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.

If you specify -opt-subscript-in-range (Linux and Mac OS X) or /Qopt-subscript-in-range (Windows), the compiler ignores any data type conversions used and it assumes no overflows in the intermediate computation of subscript expressions. This feature can enable more loop transformations.

Alternate Options

None

Example

The following shows an example where these options can be useful. m is declared as type long (64-bits) and all other variables inside the subscript are declared as type int (32-bits):

\[ A[i + j + (n + k) * m] \]

Os

Enables optimizations that do not increase code size and produces smaller code size than O2.

IDE Equivalent

Windows: Optimization > Favor Size or Speed
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-Os

Windows:

/Os
Arguments
None

Default
Off
Optimizations are made for code speed. However, if O1 is specified, O2 is the default.

Description
This option enables optimizations that do not increase code size and produces smaller code size than O2. It disables some optimizations that increase code size for a small speed benefit.

This option tells the compiler to favor transformations that reduce code size over transformations that produce maximum performance.

Alternate Options
None

See Also
- O
- Ot

Ot
Enables all speed optimizations.

IDE Equivalent
Windows: Optimization > Favor Size or Speed
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:

None

Windows:

/ot

Arguments

None

Default

/ot

Optimizations are made for code speed. If /Od is specified, all optimizations are disabled. If /O1 is specified, Os is the default.

Description

This option enables all speed optimizations.

Alternate Options

None

See Also

- Od
- Os

Ow

Tells the compiler to assume there is no cross-function aliasing.

IDÉ Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax
Linux and Mac OS X:
None
Windows:
/Ow
/Ow-

Arguments
None

Default
OFF The compiler assumes cross-function aliasing occurs.

Description
This option tells the compiler to assume there is no cross-function aliasing.

Alternate Options
None

/Ox
Enables maximum optimizations.

IDE Equivalent
Windows: Optimization > Optimization
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/0x

Arguments
None

Default
OFF
The compiler does not enable optimizations.

Description
The compiler enables maximum optimizations by combining the following options:

• /Ob2
• /Og
• /Oy
• /Ot
• /Oi

Alternate Options
None

fomit-frame-pointer, Oy
Determines whether EBP is used as a general-purpose register in optimizations.

IDE Equivalent
Windows: Optimization > Omit Frame Pointers
Linux: Optimization > Provide Frame Pointer
Mac OS X: Optimization > Provide Frame Pointer

Architectures
-f[no-]omit-frame-pointer: IA-32 architecture, Intel® 64 architecture
/Oy[-]: IA-32 architecture
Syntax

Linux and Mac OS X:
- `fomit-frame-pointer`
- `fno-omit-frame-pointer`

Windows:
`/Oy`
`/Oy-`

Arguments
None

Default
- `fomit-frame-pointer`
  or `/Oy`

EBP is used as a general-purpose register in optimizations. However, on Linux* and Mac OS X systems, the default is `-fno-omit-frame-pointer` if option `-O0` or `-g` is specified. On Windows* systems, the default is `/Oy-` if option `/Od` is specified.

Description

These options determine whether EBP is used as a general-purpose register in optimizations. Options `-fomit-frame-pointer` and `/Oy` allow this use. Options `-fno-omit-frame-pointer` and `/Oy-` disallow it.

Some debuggers expect EBP to be used as a stack frame pointer, and cannot produce a stack backtrace unless this is so. The `-fno-omit-frame-pointer` and `/Oy-` options direct the compiler to generate code that maintains and uses EBP as a stack frame pointer for all functions so that a debugger can still produce a stack backtrace without doing the following:

- For `-fno-omit-frame-pointer`: turning off optimizations with `-O0`
- For `/Oy-`: turning off `/O1`, `/O2`, or `/O3` optimizations

The `-fno-omit-frame-pointer` option is set when you specify option `-O0` or the `-g` option. The `-fomit-frame-pointer` option is set when you specify option `-O1`, `-O2`, or `-O3`. The `/Oy` option is set when you specify the `/O1`, `/O2`, or `/O3` option. Option `/Oy-` is set when you specify the `/Od` option.
Using the `-fno-omit-frame-pointer` or `/Oy-` option reduces the number of available general-purpose registers by 1, and can result in slightly less efficient code.

**NOTE.** There is currently an issue with GCC 3.2 exception handling. Therefore, the Intel compiler ignores this option when GCC 3.2 is installed for C++ and exception handling is turned on (the default).

**Alternate Options**

Linux and Mac OS X: `-fp` (this is a deprecated option)

Windows: None

**p**

*Compiles and links for function profiling with gprof(1).*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-p
```

Windows:

None

**Arguments**

None

**Default**

OFF Files are compiled and linked without profiling.
**Description**

This option compiles and links for function profiling with gprof(1).

**Alternate Options**

Linux and Mac OS X: `-qp` (this is a deprecated option)

Windows: None

**P**

*Tells the compiler to stop the compilation process and write the results to a file.*

**IDE Equivalent**

Windows: **Preprocessor > Generate Preprocessed File**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-P
```

Windows:

```
/P
```

**Arguments**

None

**Default**

OFF  

Normal compilation is performed.
**Description**

This option tells the compiler to stop the compilation process after C or C++ source files have been preprocessed and write the results to files named according to the compiler's default file-naming conventions.

On Linux systems, this option causes the preprocessor to expand your source module and direct the output to a `.i` file instead of `stdout`. Unlike the `-E` option, the output from `-F` on Linux does not include #line number directives. By default, the preprocessor creates the name of the output file using the prefix of the source file name with a `.i` extension. You can change this by using the `-o` option.

**Alternate Options**

- `-F`

**See Also**

- Building Applications: About Preprocessor Options

**par-affinity, Qpar-affinity**

Specifies thread affinity.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**

`-par-affinity=[modifier,...]type[,permute][,offset]`

**Mac OS X:**

None

**Windows:**

`/Qpar-affinity:[modifier,...]type[,permute][,offset]`
Arguments

modifier
Is one of the following values:
granularity={fine|thread|core}, [no]respect, [no]verbose, [no]warnings, proclist=proc_list. The default is granularity=core, respect, and noverbose. For information on value proclist, see Thread Affinity Interface in Optimizing Applications.

type
Indicates the thread affinity. This argument is required and must be one of the following values: compact, disabled, explicit, none, scatter, logical, physical. The default is none. Values logical and physical are deprecated. Use compact and scatter, respectively, with no permute value.

permute
Is a positive integer. You cannot use this argument with type setting explicit, none, or disabled. The default is 0.

offset
Is a positive integer. You cannot use this argument with type setting explicit, none, or disabled. The default is 0.

Default
OFF
The thread affinity is determined by the run-time environment.

Description

This option specifies thread affinity, which binds threads to physical processing units. It has the same effect as environment variable KMP_AFFINITY.

This option overrides the environment variable when both are specified.

This option only has an effect if the following is true:

- Linux* OS: You have specified option -parallel or -openmp (or both).
- Windows* OS: You have specified option /Qparallel or /Qopenmp (or both).
- You are compiling the main program.

Alternate Options

None
See Also

- Thread Affinity Interface

par-num-threads, Qpar-num-threads

Specifies the number of threads to use in a parallel region.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-par-num-threads=n

Windows:

/Qpar-num-threads:n

Arguments

n

Is the number of threads to use. It must be a positive integer.

Default

OFF

The number of threads to use is determined by the run-time environment.

Description

This option specifies the number of threads to use in a parallel region. It has the same effect as environment variable OMP_NUM_THREADS.

This option overrides the environment variable when both are specified.

This option only has an effect if the following is true:
• Linux* OS and Mac OS* X: You have specified option `-parallel` or `-openmp` (or both).
  Windows* OS: You have specified option `/Qparallel` or `/Qopenmp` (or both).
• You are compiling the main program.

Alternate Options
None

**par-report, Qpar-report**
*Controls the diagnostic information reported by the auto-parallelizer.*

IDE Equivalent
Windows: None
Linux: **Compilation Diagnostics > Auto-Parallelizer Report**
Mac OS X: **Diagnostics > Auto-Parallelizer Report**

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
-`par-report[n]`

**Windows:**
/Qpar-report[n]

Arguments

- `n`  
  Is a value denoting which diagnostic messages to report. Possible values are:
  - `0`  
    Tells the auto-parallelizer to report no diagnostic information.
Tells the auto-parallelizer to report diagnostic messages for loops successfully auto-parallelized. The compiler also issues a "LOOP AUTO-PARALLELIZED" message for parallel loops.

2 Tells the auto-parallelizer to report diagnostic messages for loops successfully and unsuccessfully auto-parallelized.

3 Tells the auto-parallelizer to report the same diagnostic messages specified by 2 plus additional information about any proven or assumed dependencies inhibiting auto-parallelization (reasons for not parallelizing).

**Default**

- `par-report1`
  
  or `/Qpar-report1`

  If you do not specify `n`, the compiler displays diagnostic messages for loops successfully auto-parallelized. If you do not specify the option on the command line, the default is to display no messages.

**Description**

This option controls the diagnostic information reported by the auto-parallelizer (parallel optimizer). To use this option, you must also specify `-parallel` (Linux and Mac OS X) or `/Qparallel` (Windows).

If this option is specified on the command line, the report is sent to `stdout`.

**Alternate Options**

None

**par-runtime-control, Qpar-runtime-control**

*Generates code to perform run-time checks for loops that have symbolic loop bounds.*

**IDE Equivalent**

None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-par-runtime-control
-no-par-runtime-control

Windows:
/Qpar-runtime-control
/Qpar-runtime-control-

Arguments
None

Default
-no-par-runtime-control The compiler uses default heuristics when checking loops.
or/Qpar-runtime-con-
trol-

Description
This option generates code to perform run-time checks for loops that have symbolic loop bounds.
If the granularity of a loop is greater than the parallelization threshold, the loop will be executed in parallel.
If you do not specify this option, the compiler may not parallelize loops with symbolic loop bounds if the compile-time granularity estimation of a loop can not ensure it is beneficial to parallelize the loop.

Alternate Options
None
par-schedule, Qpar-schedule

*Lets you specify a scheduling algorithm or a tuning method for loop iterations.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-par-schedule-keyword[=n]

**Windows:**

/Qpar-schedule-keyword[:]=n

**Arguments**

**keyword**

Specifies the scheduling algorithm or tuning method.

Possible values are:

- **auto**
  
  Lets the compiler or run-time system determine the scheduling algorithm.

- **static**
  
  Divides iterations into contiguous pieces.

- **static-balanced**
  
  Divides iterations into even-sized chunks.

- **static-steal**
  
  Divides iterations into even-sized chunks, but allows threads to steal parts of chunks from neighboring threads.

- **dynamic**
  
  Gets a set of iterations dynamically.

- **guided**
  
  Specifies a minimum number of iterations.

- **guided-analytical**
  
  Divides iterations by using exponential distribution or dynamic distribution.

- **runtime**
  
  Defers the scheduling decision until run time.
Is the size of the chunk or the number of iterations for each chunk. This setting can only be specified for static, dynamic, and guided. For more information, see the descriptions of each keyword below.

**Default**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static-balanced</td>
<td>Iterations are divided into even-sized chunks and the chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.</td>
</tr>
</tbody>
</table>

**Description**

This option lets you specify a scheduling algorithm or a tuning method for loop iterations. It specifies how iterations are to be divided among the threads of the team.

This option affects performance tuning and can provide better performance during auto-parallelization.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-par-schedule-auto or /Qpar-schedule-auto</td>
<td>Lets the compiler or run-time system determine the scheduling algorithm. Any possible mapping may occur for iterations to threads in the team.</td>
</tr>
<tr>
<td>-par-schedule-static or /Qpar-schedule-static</td>
<td>Divides iterations into contiguous pieces (chunks) of size n. The chunks are assigned to threads in the team in a round-robin fashion in the order of the thread number. Note that the last chunk to be assigned may have a smaller number of iterations. If no n is specified, the iteration space is divided into chunks that are approximately equal in size, and each thread is assigned at most one chunk.</td>
</tr>
<tr>
<td>-par-schedule-static-balanced or /Qpar-schedule-static-balanced</td>
<td>Divides iterations into even-sized chunks. The chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-par-schedule-static-steal or /Qpar-schedule-static-steal</td>
<td>Divides iterations into even-sized chunks, but when a thread completes its chunk, it can steal parts of chunks assigned to neighboring threads. Each thread keeps track of L and U, which represent the lower and upper bounds of its chunks respectively. Iterations are executed starting from the lower bound, and simultaneously, L is updated to represent the new lower bound.</td>
</tr>
<tr>
<td>-par-schedule-dynamic or /Qpar-schedule-dynamic</td>
<td>Can be used to get a set of iterations dynamically. Assigns iterations to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests another chunk, until no chunks remain to be assigned. As each thread finishes a piece of the iteration space, it dynamically gets the next set of iterations. Each chunk contains n iterations, except for the last chunk to be assigned, which may have fewer iterations. If no n is specified, the default is 1.</td>
</tr>
<tr>
<td>-par-schedule-guided or /Qpar-schedule-guided</td>
<td>Can be used to specify a minimum number of iterations. Assigns iterations to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests another chunk, until no chunks remain to be assigned. For a chunk of size 1, the size of each chunk is proportional to the number of unassigned iterations divided by the number of threads, decreasing to 1.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-par-schedule-guided-analytical or /Qpar-schedule-guided-analytical</td>
<td>For an $n$ with value $k$ (greater than 1), the size of each chunk is determined in the same way with the restriction that the chunks do not contain fewer than $k$ iterations (except for the last chunk to be assigned, which may have fewer than $k$ iterations). If no $n$ is specified, the default is 1.</td>
</tr>
<tr>
<td>-par-schedule-runtime or /Qpar-schedule-runtime</td>
<td>Divides iterations by using exponential distribution or dynamic distribution. The method depends on run-time implementation. Loop bounds are calculated with faster synchronization and chunks are dynamically dispatched at run time by threads in the team.</td>
</tr>
<tr>
<td></td>
<td>Defers the scheduling decision until run time. The scheduling algorithm and chunk size are then taken from the setting of environment variable OMP_SCHEDULE.</td>
</tr>
</tbody>
</table>

**Alternate Options**

None

**par-threshold, Qpar-threshold**

Sets a threshold for the auto-parallelization of loops.

**IDE Equivalent**

Windows: None

Linux: **Optimization > Auto-Parallelization Threshold**

Mac OS X: **Optimization > Auto-Parallelization Threshold**

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
-par-threshold[n]

Windows:
/Qpar-threshold[[:]]n

Arguments

$n$

Is an integer whose value is the threshold for the auto-parallelization of loops. Possible values are 0 through 100.
If $n$ is 0, loops get auto-parallelized always, regardless of computation work volume.
If $n$ is 100, loops get auto-parallelized when performance gains are predicted based on the compiler analysis data.
Loops get auto-parallelized only if profitable parallel execution is almost certain.
The intermediate 1 to 99 values represent the percentage probability for profitable speed-up. For example, $n=50$ directs the compiler to parallelize only if there is a 50% probability of the code speeding up if executed in parallel.

Default

-par-threshold100 or /Qpar-threshold100

Loops get auto-parallelized only if profitable parallel execution is almost certain. This is also the default if you do not specify $n$.

Description

This option sets a threshold for the auto-parallelization of loops based on the probability of profitable execution of the loop in parallel. To use this option, you must also specify -parallel (Linux and Mac OS X) or /Qparallel (Windows).

This option is useful for loops whose computation work volume cannot be determined at compile-time. The threshold is usually relevant when the loop trip count is unknown at compile-time.

The compiler applies a heuristic that tries to balance the overhead of creating multiple threads versus the amount of work available to be shared amongst the threads.
Alternate Options
None

parallel, Qparallel
_Tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel._

IDE Equivalent
Windows: Optimization > Parallelization
Linux: Optimization > Parallelization
Mac OS X: Optimization > Parallelization

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-parallel
Windows:
/Qparallel

Arguments
None

Default
OFF

Multithreaded code is not generated for loops that can be safely executed in parallel.

Description
This option tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.

To use this option, you must also specify option _O2_ or _O3_.

NOTE. On Mac OS X systems, when you enable automatic parallelization, you must also set the `DYLD_LIBRARY_PATH` environment variable within Xcode or an error will be displayed.

Alternate Options
None

See Also
•
•
• `par-report`, `Qpar-report`
• `par-affinity`, `Qpar-affinity`
• `par-num-threads`, `Qpar-num-threads`
• `par-runtime-control`, `Qpar-runtime-control`
• `par-schedule`, `Qpar-schedule`

`pc`, `Qpc`
Enables control of floating-point significand precision.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:
```bash
-pcn
```

Windows:
```bash
/Qpcn
```
Arguments

Is the floating-point significand precision. Possible values are:

- 32: Rounds the significand to 24 bits (single precision).
- 64: Rounds the significand to 53 bits (double precision).
- 80: Rounds the significand to 64 bits (extended precision).

Default

On Linux* and Mac OS* X systems, the floating-point significand is rounded to 64 bits. On Windows* systems, the floating-point significand is rounded to 53 bits.

Description

This option enables control of floating-point significand precision.

Some floating-point algorithms are sensitive to the accuracy of the significand, or fractional part of the floating-point value. For example, iterative operations like division and finding the square root can run faster if you lower the precision with the this option.

Note that a change of the default precision control or rounding mode, for example, by using the -pc32 (Linux and Mac OS X) or /Qpc32 (Windows) option or by user intervention, may affect the results returned by some of the mathematical functions.

Alternate Options

None

pch

Tells the compiler to use appropriate precompiled header files.

IDE Equivalent

Windows: None

Linux: Precompiled Headers > Automatic Processing for Precompiled Headers
Mac OS X: **Precompiled Headers > Precompile Prefix Header**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```bash
-pch
```

Windows:

None

**Arguments**

None

**Default**

OFF

The compiler does not create or use precompiled headers unless you tell it to do so.

**Description**

This option tells the compiler to use appropriate precompiled header (PCH) files. If none are available, they are created as `sourcefile.pchi`. This option is supported for multiple source files.

The `-pch` option will use PCH files created from other sources if the headers files are the same. For example, if you compile `source1.cpp` using `-pch`, then `source1.pchi` is created. If you then compile `source2.cpp` using `-pch`, the compiler will use `source1.pchi` if it detects the same headers.

---

**CAUTION.** Depending on how you organize the header files listed in your sources, this option may increase compile times. To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

**Alternate Options**

None
Example

Consider the following command line:
```bash
icpc -pch source1.cpp source2.cpp
```

It produces the following output when `.pchi` files do not exist:
```
"source1.cpp": creating precompiled header file "source1.pchi"
"source2.cpp": creating precompiled header file "source2.pchi"
```

It produces the following output when `.pchi` files do exist:
```
"source1.cpp": using precompiled header file "source1.pchi"
"source2.cpp": using precompiled header file "source2.pchi"
```

See Also

- `-pch-create`
- `-pch-dir`
- `-pch-use`

**pch-create, Yc**

*Let you create and specify a name for a precompiled header file.*

IDE Equivalent

Windows: **Precompiled Headers > Create–Use Precompiled Header / Create–Use PCH Through File**

Linux: None

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

```bash
-pch-createfile
```
Windows:
/Ycfile

**Arguments**

file Is the name for the precompiled header file.

**Default**

OFF The compiler does not create or use precompiled headers unless you tell it to do so.

**Description**

This option lets you specify a name for a precompiled header (PCH) file. It is supported only for single source file compilations.

The .pchi extension is not automatically appended to the file name.

This option cannot be used in the same compilation as the -pch-use option.

Depending on how you organize the header files listed in your sources, this option may increase compile times.

To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

**Alternate Options**

None

**Example**

Consider the following command line:

```bash
icpc -pch-create /pch/source32.pchi source.cpp
```

It produces the following output:

"source.cpp": creating precompiled header file "/pch/source32.pchi"

**See Also**

•

•

Using Precompiled Header Files
**pch-dir**
*Tells the compiler where to find or create a file for precompiled headers.*

**IDE Equivalent**
Windows: None
Linux: Precompiled Headers > Precompiled Headers' File Directory
Mac OS X: Precompiled Headers > Prefix Header

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
- `pch-dir dir`

Windows:
None

**Arguments**

*dir*  
*Is the path where the file is located or should be created. The path must exist.*

**Default**
OFF  
*The compiler does not create or use precompiled headers unless you tell it to do so.*

**Description**
This option tells the compiler where to find or create a file (PCH) for precompiled headers.
This option can be used with the `-pch`, `-pch-create`, and `-pch-use` options.

---

**CAUTION.** Depending on how you organize the header files listed in your sources, this option may increase compile times. To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.
Alternate Options

None

Example

Consider the following command line:

```bash
icpc -pch -pch-dir /pch source32.cpp
```

It produces the following output:

"source32.cpp": creating precompiled header file /pch/source32.pchi

See Also

- `-pch`
- `-pch-create`
- `-pch-use`

`-pch-use`

*Let you use a specific precompiled header file.*

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

```
-pch-use {file | dir}
```

**Windows:**

None

Arguments

`file`

Is the name of the precompiled header file to use.
dir

Is the path where the file is located, including . The path must exist.

Default

OFF

The compiler does not create or use precompiled headers unless you tell it to do so.

Description

This option lets you use a specific precompiled header (PCH) file. It is supported for multiple source files when all source files use the same .pchi file.

This option cannot be used in the same compilation as the -pch-create option.

CAUTION. Depending on how you organize the header files listed in your sources, this option may increase compile times. To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

Alternate Options

None

Example

Consider the following command line:

icpc -pch-use /pch/source32.pchi source.cpp

It produces the following output:

"source.cpp": using precompiled header file /pch/source32.pchi

See Also

•

• -pch-create
pie
 Produces a position-independent executable on processors that support it.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:
-pie

Mac OS X:
None

Windows:
None

Arguments
None

Default
OFF The driver does not set up special run-time libraries and the linker
does not perform the optimizations on executables.

Description
This option produces a position-independent executable on processors that support it. It is both
a compiler option and a linker option. When used as a compiler option, this option ensures the
linker sets up run-time libraries correctly.

Normally the object linked has been compiled with option -fpie.

When you specify -pie, it is recommended that you specify the same options that were used
during compilation of the object.
Alternate Options
None

See Also
•
  • fpie

pragma-optimization-level
Specifies which interpretation of the optimization_level pragma should be used if no prefix is specified.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-pragma-optimization-level=interpretation

Windows:
None

Arguments
interpretation Compiler-specific interpretation of optimization_level pragma. Possible values are:
  Intel Specify the Intel interpretation.
  GCC Specify the GCC interpretation.

Default
-pragma-optimization-level=Intel
Description
Specifies which interpretation of the `optimization_level` pragma should be used if no prefix is specified.

Alternate Options
None

See Also
• Compiler Reference: pragma optimization_level

prec-div, Qprec-div
Improves precision of floating-point divides.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-prec-div
-no-prec-div

Windows:
/Qprec-div
/Qprec-div-

Arguments
None

Default
-prec-div or/Qprec-div The compiler uses this method for floating-point divides.
Description

This option improves precision of floating-point divides. It has a slight impact on speed.

With some optimizations, such as \(-xSSE2\) (Linux) or /QSSE2 (Windows), the compiler may change floating-point division computations into multiplication by the reciprocal of the denominator. For example, \(A/B\) is computed as \(A \times (1/B)\) to improve the speed of the computation.

However, sometimes the value produced by this transformation is not as accurate as full IEEE division. When it is important to have fully precise IEEE division, use this option to disable the floating-point division-to-multiplication optimization. The result is more accurate, with some loss of performance.

If you specify \(-no-prec-div\) (Linux and Mac OS X) or /Qprec-div- (Windows), it enables optimizations that give slightly less precise results than full IEEE division.

Alternate Options

None

\textit{prec-sqrt, Qprec-sqrt}

\textit{Improves precision of square root implementations.}

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

\textbf{Linux and Mac OS X:}

-\texttt{prec-sqrt}
-\texttt{-no-prec-sqrt}

\textbf{Windows:}

/\texttt{Qprec-sqrt}
/\texttt{Qprec-sqrt-}
Arguments
None

Default
-no-prec-sqrt or /Qprec-sqrt-
The compiler uses a faster but less precise implementation of square root.
However, the default is -prec-sqrt or /Qprec-sqrt if any of the following options are specified: /Od, /Op, or /Qprec on Windows systems; -O0, -mp, or -mp1 on Linux and Mac OS X systems.

Description
This option improves precision of square root implementations. It has a slight impact on speed.
This option inhibits any optimizations that can adversely affect the precision of a square root computation. The result is fully precise square root implementations, with some loss of performance.

Alternate Options
None

print-multi-lib
Prints information about where system libraries should be found.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-print-multi-lib
Windows:
None
Arguments
None

Default
OFF

No information is printed unless the option is specified.

Description
This option prints information about where system libraries should be found, but no compilation occurs. It is provided for compatibility with gcc.

Alternate Options
None

prof-data-order, Qprof-data-order
Enables or disables data ordering if profiling information is enabled.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux:
-prof-data-order
-no-prof-data-order

Mac OS X:
None

Windows:
/Qprof-data-order
/Qprof-data-order-
Arguments

None

Default

-no-prof-data-order   Data ordering is disabled.
or/Qprof-data-order-

Description

This option enables or disables data ordering if profiling information is enabled. It controls the use of profiling information to order static program data items.

For this option to be effective, you must do the following:

- For instrumentation compilation, you must specify -prof-gen=globdata (Linux) or /Qprof-gen:globdata (Windows).
- For feedback compilation, you must specify -prof-use (Linux) or /Qprof-use (Windows). You must not use multi-file optimization by specifying options such as option -ipo (Linux) or /Qipo (Windows), or option -ipo-c (Linux) or /Qipo-c (Windows).

Alternate Options

None

See Also

- prof-gen, Qprof-gen
- prof-use, Qprof-use
- prof-func-order, Qprof-func-order

prof-dir, Qprof-dir

Specifies a directory for profiling information output files.

IDE Equivalent

Windows: General > Profile Directory
Linux: Compiler > Profile Directory
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `-prof-dir dir`

**Windows:**
- `/Qprof-dir dir`

**Arguments**

`dir` 
Is the name of the directory.

**Default**

OFF 
Profiling output files are placed in the directory where the program is compiled.

**Description**

This option specifies a directory for profiling information output files (*.dyn and *.dpi). The specified directory must already exist.

You should specify this option using the same directory name for both instrumentation and feedback compilations. If you move the .dyn files, you need to specify the new path.

**Alternate Options**

None

**prof-file, Qprof-file**

Specifies an alternate file name for the profiling summary files.

**IDE Equivalent**

None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-prof-file file

Windows:
/Qprof-file file

Arguments

file Is the name of the profiling summary file.

Default

OFF The profiling summary files have the file name pgopti.*

Description
This option specifies an alternate file name for the profiling summary files. The file is used as the base name for files created by different profiling passes.

If you add this option to profmerge, the .dpi file will be named file.dpi instead of pgopti.dpi.

If you specify -prof-genx (Linux and Mac OS X) or /Qprof-genx (Windows) with this option, the .spi and .spl files will be named file.spi and file.spl instead of pgopti.spi and pgopti.spl.

If you specify -prof-use (Linux and Mac OS X) or /Qprof-use (Windows) with this option, the .dpi file will be named file.dpi instead of pgopti.dpi.

Alternate Options
None

See Also

•
•
• prof-gen, Qprof-gen
• prof-use, Qprof-use
**prof-func-groups**

*Enables or disables function grouping if profiling information is enabled.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux:**

- `-prof-func-groups`
- `-no-prof-func-groups`

**Mac OS X:**

None

**Windows:**

None

**Arguments**

None

**Default**

- `-no-prof-func-groups` Function grouping is disabled.

**Description**

This option enables or disables function grouping if profiling information is enabled.

A "function grouping" is a profiling optimization in which entire routines are placed either in the cold code section or the hot code section.

If profiling information is enabled by option `-prof-use`, option `-prof-func-groups` is set and function grouping is enabled. However, if you explicitly enable `-prof-func-order` (Linux) or `/Qprof-func-order` (Windows), function ordering is performed instead of function grouping.
If you want to disable function grouping when profiling information is enabled, specify `-no-prof-func-groups`.

To set the hotness threshold for function grouping, use option `-prof-hotness-threshold` (Linux) or `/Qprof-hotness-threshold` (Windows).

**Alternate Options**
- `func-groups` (this is a deprecated option)

**See Also**
- `prof-use, Qprof-use`
- `prof-func-order, Qprof-func-order`
- `prof-hotness-threshold, Qprof-hotness-threshold`

**prof-func-order, Qprof-func-order**

*Enables or disables function ordering if profiling information is enabled.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**
- `-prof-func-order`
- `-no-prof-func-order`

**Mac OS X:**

None

**Windows:**

- `/Qprof-func-order`
- `/Qprof-func-order-`
Arguments
None

Default
-no-prof-func-order Function ordering is disabled.
or /Qprof-func-order-

Description
This option enables or disables function ordering if profiling information is enabled.

For this option to be effective, you must do the following:

- For instrumentation compilation, you must specify -prof-gen=srcpos (Linux) or /Qprof-gen:srcpos (Windows).
- For feedback compilation, you must specify -prof-use (Linux) or /Qprof-use (Windows).
  You must not use multi-file optimization by specifying options such as option -ipo (Linux) or /Qipo (Windows), or option -ipo-c (Linux) or /Qipo-c (Windows).

If you enable profiling information by specifying option -prof-use (Linux) or /Qprof-use (Windows), -prof-func-groups (Linux) and /Qprof-func-groups (Windows) are set and function grouping is enabled. However, if you explicitly enable -prof-func-order (Linux) or /Qprof-func-order (Windows), function ordering is performed instead of function grouping.

On Linux* systems, this option is only available for Linux linker 2.15.94.0.1, or later.

To set the hotness threshold for function grouping and function ordering, use option -prof-hotness-threshold (Linux) or /Qprof-hotness-threshold (Windows).

Alternate Options
None

The following example shows how to use this option on a Windows system:

icl /Qprof-gen:globdata file1.c file2.c /Fe instrumented.exe
./instrumented.exe
icl /Qprof-use /Qprof-func-order file1.c file2.c /Fe feedback.exe
The following example shows how to use this option on a Linux system:

```plaintext
icl -prof-gen:globdata file1.c file2.c -o instrumented
./instrumented.exe
icl -prof-use -prof-func-order file1.c file2.c -o feedback
```

See Also

- prof-hotness-threshold, Qprof-hotness-threshold
- prof-gen, Qprof-gen
- prof-use, Qprof-use
- prof-data-order, Qprof-data-order
- prof-func-groups

prof-gen, Qprof-gen

*Produces an instrumented object file that can be used in profile-guided optimization.*

IDE Equivalent

Windows: **General > Profile Guided Optimization**

**Optimization > Profile Guided Optimization**

Linux: None

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

- `-prof-gen[=keyword]`
- `-no-prof-gen`

**Windows:**

- `/Qprof-gen[:keyword]`
/Qprof-gen-

**Arguments**

*keyword* Specifies details for the instrumented file. Possible values are:

- **default**: Produces an instrumented object file. This is the same as specifying `-prof-gen` (Linux* and Mac OS* X) or `/Qprof-gen` (Windows*) with no keyword.
- **srcpos**: Produces an instrumented object file that includes extra source position information. This option is the same as option `-prof-genx` (Linux* and Mac OS* X) or `/Qprof-genx` (Windows*), which are deprecated.
- **globdata**: Produces an instrumented object file that includes information for global data layout.

**Default**

-no-prof-gen or /Qprof-gen- Profile generation is disabled.

default

**Description**

This option produces an instrumented object file that can be used in profile-guided optimization. It gets the execution count of each basic block.

If you specify keyword srcpos or globdata, a static profile information file (.spi) is created. These settings may increase the time needed to do a parallel build using -prof-gen, because of contention writing the .spi file.

These options are used in phase 1 of the Profile Guided Optimizer (PGO) to instruct the compiler to produce instrumented code in your object files in preparation for instrumented execution.

**Alternate Options**

None
**proff-hotness-threshold, Qprof-hotness-threshold**  
*Lets you set the hotness threshold for function grouping and function ordering.*

**IDE Equivalent**  
None

**Architectures**  
IA-32, Intel® 64, IA-64 architectures

**Syntax**  
**Linux:**  
- `prof-hotness-threshold=n`

**Mac OS X:**  
None

**Windows:**  
/`Qprof-hotness-threshold:n`

**Arguments**

\[ n \]  
Is the hotness threshold. \( n \) is a percentage having a value between 0 and 100 inclusive. If you specify 0, there will be no hotness threshold setting in effect for function grouping and function ordering.

**Default**

OFF  
The compiler's default hotness threshold setting of 10 percent is in effect for function grouping and function ordering.

**Description**

This option lets you set the hotness threshold for function grouping and function ordering.
The "hotness threshold" is the percentage of functions in the application that should be placed in the application's hot region. The hot region is the most frequently executed part of the application. By grouping these functions together into one hot region, they have a greater probability of remaining resident in the instruction cache. This can enhance the application's performance.

For this option to take effect, you must specify option \texttt{-prof-use} (Linux) or \texttt{/Qprof-use} (Windows) and one of the following:

- On Linux systems: \texttt{-prof-func-groups} or \texttt{-prof-func-order}
- On Windows systems: \texttt{/Qprof-func-order}

**Alternate Options**

None

**See Also**

- prof-use, Qprof-use
- prof-func-groups
- prof-func-order, Qprof-func-order

**prof-src-dir, Qprof-src-dir**

Determines whether directory information of the source file under compilation is considered when looking up profile data records.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- \texttt{-prof-src-dir}
- \texttt{-no-prof-src-dir}
Windows:

/Qprof-src-dir
/Qprof-src-dir-

Arguments
None

Default

-prof-src-dir
or/Qprof-src-dir

Directory information is used when looking up profile data records in the .dpi file.

Description
This option determines whether directory information of the source file under compilation is considered when looking up profile data records in the .dpi file. To use this option, you must also specify option -prof-use (Linux and Mac OS X) or /Qprof-use (Windows).

If the option is enabled, directory information is considered when looking up the profile data records within the .dpi file. You can specify directory information by using one of the following options:

- Linux and Mac OS X: -prof-src-root or -prof-src-root-cwd
- Windows: /Qprof-src-root or /Qprof-src-root-cwd

If the option is disabled, directory information is ignored and only the name of the file is used to find the profile data record.

Note that options -prof-src-dir (Linux and Mac OS X) and /Qprof-src-dir (Windows) control how the names of the user’s source files get represented within the .dyn or .dpi files. Options -prof-dir (Linux and Mac OS X) and /Qprof-dir (Windows) specify the location of the .dyn or the .dpi files.

Alternate Options
None

See Also

- prof-use, Qprof-use
• prof-src-root, Qprof-src-root
• prof-src-root-cwd, Qprof-src-root-cwd

**prof-src-root, Qprof-src-root**

*Lets you use relative directory paths when looking up profile data and specifies a directory as the base.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `prof-src-root=dir`

**Windows:**

/QRprof-src-root:dir

**Arguments**

*dir* Is the base for the relative paths.

**Default**

OFF The setting of relevant options determines the path used when looking up profile data records.

**Description**

This option lets you use relative directory paths when looking up profile data in .dpi files. It lets you specify a directory as the base. The paths are relative to a base directory specified during the -prof-gen (Linux and Mac OS X) or /Qprof-gen (Windows) compilation phase.

This option is available during the following phases of compilation:

- Linux and Mac OS X: -prof-gen and -prof-use phases
- Windows: /Qprof-gen and /Qprof-use phases
When this option is specified during the -prof-gen or /Qprof-gen phase, it stores information into the .dyn or .dpi file. Then, when .dyn files are merged together or the .dpi file is loaded, only the directory information below the root directory is used for forming the lookup key.

When this option is specified during the -prof-use or /Qprof-use phase, it specifies a root directory that replaces the root directory specified at the -prof-gen or /Qprof-gen phase for forming the lookup keys.

To be effective, this option or option -prof-src-root-cwd (Linux and Mac OS X) or /Qprof-src-root-cwd (Windows) must be specified during the -prof-gen or /Qprof-gen phase. In addition, if one of these options is not specified, absolute paths are used in the .dpi file.

**Alternate Options**

None

Consider the initial -prof-gen compilation of the source file
c:\user1\feature_foo\myproject\common\glob.c:
icc -prof-gen -prof-src-root=c:\user1\feature_foo\myproject -c common\glob.c

For the -prof-use phase, the file glob.c could be moved into the directory
c:\user2\feature_bar\myproject\common\glob.c and profile information would be found from the .dpi when using the following:
icc -prof-use -prof-src-root=c:\user2\feature_bar\myproject -c common\glob.c

If you do not use option -prof-src-root during the -prof-gen phase, by default, the -prof-use compilation can only find the profile data if the file is compiled in the
c:\user1\feature_foo\my_project\common directory.

**See Also**

- prof-gen, Qprof-gen
- prof-use, Qprof-use
- prof-src-dir, Qprof-src-dir
- prof-src-root-cwd, Qprof-src-root-cwd
prof-src-root-cwd, Qprof-src-root-cwd

*Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `prof-src-root-cwd`

**Windows:**

/`Qprof-src-root-cwd`

**Arguments**

None

**Default**

OFF

The setting of relevant options determines the path used when looking up profile data records.

**Description**

This option lets you use relative directory paths when looking up profile data in .dpi files. It specifies the current working directory as the base. To use this option, you must also specify option `-prof-use` (Linux and Mac OS) or `/Qprof-use` (Windows).

This option is available during the following phases of compilation:

- Linux and Mac OS X: `-prof-gen` and `-prof-use` phases
- Windows: `/Qprof-gen` and `/Qprof-use` phases
When this option is specified during the -prof-gen or /Qprof-gen phase, it stores information into the .dyn or .dpi file. Then, when .dyn files are merged together or the .dpi file is loaded, only the directory information below the root directory is used for forming the lookup key.

When this option is specified during the -prof-use or /Qprof-use phase, it specifies a root directory that replaces the root directory specified at the -prof-gen or /Qprof-gen phase for forming the lookup keys.

To be effective, this option or option -prof-src-root (Linux and Mac OS X) or /Qprof-src-root (Windows) must be specified during the -prof-gen or /Qprof-gen phase. In addition, if one of these options is not specified, absolute paths are used in the .dpi file.

Alternate Options

None

See Also

•

• prof-gen, Qprof-gen
• prof-use, Qprof-use
• prof-src-dir, Qprof-src-dir
• prof-src-root, Qprof-src-root

prof-use, Qprof-use

Enables the use of profiling information during optimization.

IDE Equivalent

Windows: General > Profile Guided Optimization
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
- prof-use[="arg"]
- no-prof-use

Windows:
/Qprof-use[:arg]
/Qprof-use-

Arguments

arg  Specifies additional instructions. Possible values are:

  weighted  Tells the profmerge utility to apply a weighting to the .dyn file values when creating the .dpi file to normalize the data counts when the training runs have different execution durations. This argument only has an effect when the compiler invokes the profmerge utility to create the .dpi file. This argument does not have an effect if the .dpi file was previously created without weighting.

  [no]merge  Enables or disables automatic invocation of the profmerge utility. The default is merge. Note that you cannot specify both weighted and nomerge. If you try to specify both values, a warning will be displayed and nomerge takes precedence.

  default  Enables the use of profiling information during optimization. The profmerge utility is invoked by default. This value is the same as specifying -prof-use (Linux and Mac OS X) or /Qprof-use (Windows) with no argument.
Default

-no-prof-use or /Qprof-

Profiling information is not used during optimization.

Description

This option enables the use of profiling information (including function splitting and function
grouping) during optimization. It enables option -fnsplit (Linux) or /Qfnsplit (Windows).

This option instructs the compiler to produce a profile-optimized executable and it merges
available profiling output files into a pgopti.dpi file.

Note that there is no way to turn off function grouping if you enable it using this option.

To set the hotness threshold for function grouping and function ordering, use option
 prof-hotness-threshold (Linux) or /Qprof-hotness-threshold (Windows).

Alternate Options

None

See Also

- prof-hotness-threshold, Qprof-hotness-threshold

pthread

Tells the compiler to use pthreads library for
multithreading support.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-pthread
**Windows:**
None

**Arguments**
None

**Default**
OFF

The compiler does not use pthreads library for multithreading support.

**Description**
Tells the compiler to use pthreads library for multithreading support.

**Alternate Options**
None

**A, QA**
Specifies an identifier for an assertion.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

- `Aname[\(value\)]`

*Windows:*

/\QAnamex\(value\)\]

---

609
Arguments

name  Is the identifier for the assertion.
value  Is an optional value for the assertion. If a value is specified, it must be within quotes, including the parentheses delimiting it.

Default

OFF  Assertions have no identifiers or symbol names.

Description

This option specifies an identifier (symbol name) for an assertion. It is equivalent to an #assert preprocessing directive.

Note that this option is not the positive form of the C++ /QA- option.

Alternate Options

None

Example

To make an assertion for the identifier fruit with the associated values orange and banana use the following command.

On Windows* systems:
  icl /QA"fruit(orange,banana)" prog1.cpp

On Linux* and Mac OS* X systems:
  icpc -A"fruit(orange,banana)" prog1.cpp

A-, QA-

Disables all predefined macros. This is a deprecated option.

IDE Equivalent

Windows: None

Linux: Preprocessor > Undefine All Preprocessor Definitions

Mac OS X: Preprocessor > Undefine All Preprocessor Definitions
**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- A-

**Windows:**

/QA-

**Arguments**

None

**Default**

OFF Predefined macros remain enabled.

**Description**

This option disables all predefined macros. It causes all predefined macros and assertions to become inactive.

Note that this option is not the negative form of the C++ /QA option.

**Alternate Options**

None

**argument-alias, Qalias-args**

Determines whether function arguments can alias each other.

**IDE Equivalent**

Windows: None

Linux: **Data > Enable Argument Aliasing**

Mac OS X: **Data > Enable Argument Aliasing**
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fargument-alias
-fargument-noalias

Windows:
/Qalias-args
/Qalias-args-

Arguments
None

Default
-fargument-alias or /Qalias-args  Function arguments can alias each other and can alias global storage.

Description
This option determines whether function arguments can alias each other. If you specify -fargument-noalias or /Qalias-args-, function arguments cannot alias each other, but they can alias global storage.

On Linux and Mac OS X systems, you can also disable aliasing for global storage, by specifying option -fargument-noalias-global.

Alternate Options
Linux and Mac OS X: -[no-]alias-args (this is a deprecated option)

Windows: None

See Also
•
•
•
• fargument-noalias-global

**alias-const, Qalias-const**

*Determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const.*

**IDE Equivalent**

Windows: None

Linux: **Data > Assume Restrict Semantics for Const**

Mac OS X: **Data > Assume Restrict Semantics for Const**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-alias-const`
- `-no-alias-const`

Windows:

- `/Qalias-const`
- `/Qalias-const-`

**Arguments**

None

**Default**

- `/Qalias-const`
- `/Qalias-const-`

The compiler uses standard C/C++ rules for the interpretation of const.

**Description**

This option determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const. It implies an additional attribute for const.
This functionality complies with the input/output buffer rule, which assumes that input and output buffer arguments do not overlap. This option allows the compiler to do some additional optimizations with those parameters.

In C99, you can also get the same result if you additionally declare your pointer parameters with the restrict keyword.

**Alternate Options**

None

**ansi-alias, Qansi-alias**

*Enable use of ANSI aliasing rules in optimizations.*

**IDE Equivalent**

Windows: None

Linux: **Language > Enable Use of ANSI Aliasing Rules in Optimizations**

Mac OS X: **Language > Enable ANSI Aliasing**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `ansi-alias`
- `no-ansi-alias`

Windows:

/Qansi-alias
/Qansi-alias-

**Arguments**

None
Default
-no-ansi-alias or /Qan- Disable use of ANSI aliasing rules in optimizations.

Description
This option tells the compiler to assume that the program adheres to ISO C Standard aliasability rules.
If your program adheres to these rules, then this option allows the compiler to optimize more aggressively. If it doesn't adhere to these rules, then it can cause the compiler to generate incorrect code.

Alternate Options
None

auto-ilp32, Qauto-ilp32
Instructs the compiler to analyze the program to determine if there are 64-bit pointers which can be safely shrunk into 32-bit pointers.

IDE Equivalent
None

Architectures
Intel® 64 architecture, IA-64 architecture

Syntax
Linux and Mac OS X:
-auto-ilp32

Windows:
/Qauto-ilp32

Arguments
None
Default
OFF The optimization is not attempted.

Description
This option instructs the compiler to analyze and transform the program so that 64-bit pointers are shrunk to 32-bit pointers, and 64-bit longs (on Linux) are shrunk into 32-bit longs wherever it is legal and safe to do so. In order for this option to be effective the compiler must be able to optimize using the -ipo/-qipo option and must be able to analyze all library/external calls the program makes.

This option requires that the size of the program executable never exceeds $2^{32}$ bytes and all data values can be represented within 32 bits. If the program can run correctly in a 32-bit system, these requirements are implicitly satisfied. If the program violates these size restrictions, unpredictable behavior might occur.

Alternate Options
None

ax, Qax
Tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit.

IDE Equivalent
Windows: Code Generation > Add Processor-Optimized Code Path
Linux: Code Generation > Add Processor-Optimized Code Path
Mac OS X: Code Generation > Add Processor-Optimized Code Path

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-axprocessor
Windows:
/Qaxprocessor

Arguments

processor

Indicates the processor for which code is generated. The following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE). Possible values are:

SSE4.2
Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.

SSE4.1
Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture. This replaces value S, which is deprecated.

SSSE3
Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. For Mac OS* X systems, this value is only supported on Intel® 64 architecture. This replaces value T, which is deprecated.

SSE3
Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. For Mac OS*
X systems, this value is only supported on IA-32 architecture. This replaces value P, which is deprecated.

**SSE2**

Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2. This value is not available on Mac OS* X systems. This replaces value N, which is deprecated.

**Default**

OFF

No auto-dispatch code is generated. Processor-specific code is generated and is controlled by the setting of compiler option -m (Linux), compiler option /arch (Windows), or compiler option -x (Mac OS* X).

**Description**

This option tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit. It also generates a baseline code path. The baseline code is usually slower than the specialized code.

The baseline code path is determined by the architecture specified by the -x (Linux and Mac OS X) or /Qx (Windows) option. While there are defaults for the -x or /Qx option that depend on the operating system being used, you can specify an architecture for the baseline code that is higher or lower than the default. The specified architecture becomes the effective minimum architecture for the baseline code path.

If you specify both the -ax and -x options (Linux and Mac OS X) or the /Qax and /Qx options (Windows), the baseline code will only execute on processors compatible with the processor type specified by the -x or /Qx option.

This option tells the compiler to find opportunities to generate separate versions of functions that take advantage of features of the specified Intel® processor.

If the compiler finds such an opportunity, it first checks whether generating a processor-specific version of a function is likely to result in a performance gain. If this is the case, the compiler generates both a processor-specific version of a function and a baseline version of the function.
At run time, one of the versions is chosen to execute, depending on the Intel processor in use. In this way, the program can benefit from performance gains on more advanced Intel processors, while still working properly on older processors.

You can use more than one of the processor values by combining them. For example, you can specify `-axSSE4.1, SSSE3` (Linux and Mac OS X) or `/QaxSSE4.1, SSSE3` (Windows). You cannot combine the old style, deprecated options and the new options. For example, you cannot specify `-axSSE4.1, T` (Linux and Mac OS X) or `/QaxSSE4.1, T` (Windows).

Previous values W and K are deprecated. The details on replacements are as follows:

- **Mac OS X systems:** On these systems, there is no exact replacement for W or K. You can upgrade to the default option `-msse3` (IA-32 architecture) or option `-mssse3` (Intel® 64 architecture).

- **Windows and Linux systems:** The replacement for W is `-msse2` (Linux) or `/arch:SSE2` (Windows). There is no exact replacement for K. However, on Windows systems, `/QaxK` is interpreted as `/arch:IA32`; on Linux systems, `-axK` is interpreted as `-mia32`. You can also do one of the following:
  - Upgrade to option `-msse2` (Linux) or option `/arch:SSE2` (Windows). This will produce one code path that is specialized for Intel® SSE2. It will not run on earlier processors.
  - Specify the two option combination `-mia32` `-axSSE2` (Linux) or `/arch:IA32 /QaxSSE2` (Windows). This combination will produce an executable that runs on any processor with IA-32 architecture but with an additional specialized Intel® SSE2 code path.

The `-ax` and `/Qax` options enable additional optimizations not enabled with option `-m` or option `/arch`.

**Alternate Options**

None

**See Also**

- `x`, `Qx`
- `m`
- `arch`
- Targeting IA-32 and Intel 64 Architecture Processors Manually
**c99, Qc99**

Determines whether C99 support is enabled for C programs. This is a deprecated option.

**IDE Equivalent**

Windows: **Language > Enable C99 Support**

Linux: **Language > Disable C99 Support**

Mac OS X: **Language > Disable C99 Support**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `c99`
- `-no-c99`

**Windows:**

- `/Qc99`
- `/Qc99-`

**Arguments**

None

**Default**

- `-no-c99`
- `/Qc99-`

C99 support is disabled for C programs on Linux.

**Description**

This option determines whether C99 support is enabled for C programs. One of the features enabled by `-c99` (Linux and Mac OS X) or `/Qc99` (Windows), restricted pointers, is available by using option `restrict`. For more information, see `restrict`. 
Alternate Options
-std, /Qstd

See Also
•
•
• restrict, Qrestrict

Qchkstk
Enables stack probing when the stack is dynamically expanded at run-time.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux and Mac OS X:
None

Windows:
/Qchkstk
/Qchkstk-

Arguments
None

Default
/Qchkstk Stack probing is enabled when the stack is dynamically expanded at run-time.

Description
This option enables stack probing when the stack is dynamically expanded at run-time.
It instructs the compiler to generate a call to _chkstk. The call will probe the requested memory and detect possible stack overflow.

To cancel the call to _chkstk, specify /Qchkstk-.

**Alternate Options**

None

**complex-limited-range, Qcomplex-limited-range**

Determines whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled.

**IDE Equivalent**

Windows: **Floating Point > Limit COMPLEX Range**

Linux: None

Mac OS X: **Floating Point > Limit COMPLEX Range**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- complex-limited-range
- no-complex-limited-range

Windows:
/Qcomplex-limited-range
/Qcomplex-limited-range-

**Arguments**

None

**Default**

-no-complex-limited-range  Basic algebraic expansions of some arithmetic operations involving data of type COMPLEX are disabled.
or /Qcomplex-limited-range-

**Description**

This option determines whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled.

When the option is enabled, this can cause performance improvements in programs that use a lot of COMPLEX arithmetic. However, values at the extremes of the exponent range may not compute correctly.

**Alternate Options**

None

**Qcov-dir**

Specifies a directory for profiling information output files that can be used with the codecov or tselect tool.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/ Qcov-dir dir

**Arguments**

dir Is the name of the directory.
**Default**

OFF Profiling output files are placed in the directory where the program is compiled.

**Description**

This option specifies a directory for profiling information output files (*.dyn and *.dpi) that can be used with the code-coverage tool (codecov) or the test prioritization tool (tselect). The specified directory must already exist.

You should specify this option using the same directory name for both instrumentation and feedback compilations. If you move the .dyn files, you need to specify the new path.

**Alternate Options**

None

**See Also**

- Qcov-gen
- Qcov-file

**Qcov-file**

Specifies an alternate file name for the profiling summary files that can be used with the codecov or tselect tool.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Qcov-file file
Arguments

file

Is the name of the profiling summary file.

Default

OFF

The profiling summary files have the file name pgopti.*.

Description

This option specifies an alternate file name for the profiling summary files. The file name can be used with the code-coverage tool (codecov) or the test prioritization tool (tselect).

The file is used as the base name for files created by different profiling passes.

If you specify /Qcov-gen or equivalent option /Qprof-gen=srcpos with this option, the .spi and .spl files will be named file.spi and file.spl instead of pgopti.spi and pgopti.spl.

Alternate Options

None

See Also

•

• Qcov-gen

• Qcov-dir

Qcov-gen

Produces an instrumented object file that can be used with the codecov or tselect tool.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:

None
Windows:
/Qcov-gen
/Qcov-gen-

Arguments
None

Default
/Qcov-gen- The instrumented object file is not produced.

Description
This option produces an instrumented object file that can be used with the code-coverage tool (codecov) or the test prioritization tool (tselect). The instrumented code is included in the object file in preparation for instrumented execution.

This option also creates a static profile information file (.spi) that can be used with the codecov or tselect tool.

When using the codecov and tselect tools, you must specify option /Qcov-gen, which is the same as the option /Qprof-gen=srcpos.

Alternate Options
None

See Also
•
  • Qcov-dir
  • Qcov-file

Qcxx-features
Enables standard C++ features without disabling Microsoft features.

IDE Equivalent
None
**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Qcxx-features

**Arguments**

None

**Default**

OFF

The compiler enables standard C++ features.

**Description**

This option enables standard C++ features without disabling Microsoft features within the bounds of what is provided in the Microsoft headers and libraries.

This option has the same effect as specifying `/GX` `/GR`.

**Alternate Options**

None

**diag, Qdiag**

*Controls the display of diagnostic information.*

**IDE Equivalent**

Windows: **Diagnostics > Disable Specific Diagnostics** (`/Qdiag-disable:id`)

**Diagnostics > Level of Source Code Analysis** (`/Qdiag-enable[:sc1,sc2,sc3]`)

Linux: **Compilation Diagnostics > Disable Specific Diagnostics** (`-diag-disable id`)

**Compilation Diagnostics > Level of Source Code Analysis** (`-diag-enable [sc1,sc2,sc3]` or `-diag-disable sv`)
Mac OS X: **Diagnostics > Disable Specific Diagnostics** (-diag-disable id)

**Diagnostics > Level of Source Code Analysis** (-diag-enable [sc1,sc2,sc3])

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- `diag-type diag-list`

Windows:
- `/Qdiag-type:diag-list`

**Arguments**

- **type**
  - `enable` Enables a diagnostic message or a group of messages.
  - `disable` Disables a diagnostic message or a group of messages.
  - `error` Tells the compiler to change diagnostics to errors.
  - `warning` Tells the compiler to change diagnostics to warnings.
  - `remark` Tells the compiler to change diagnostics to remarks (comments).

- **diag-list**
  - `driver` Specifies diagnostic messages issued by the compiler driver.
  - `port-linux` Specifies diagnostic messages for language features that may cause errors when porting to Linux. This diagnostic group is only available on Windows systems.
**port-win**  Specifies diagnostic messages for GNU extensions that may cause errors when porting to Windows. This diagnostic group is only available on Linux and Mac OS X systems.

**thread**  Specifies diagnostic messages that help in thread-enabling a program.

**vec**  Specifies diagnostic messages issued by the vectorizer.

**par**  Specifies diagnostic messages issued by the auto-parallelizer (parallel optimizer).

**openmp**  Specifies diagnostic messages issued by the OpenMP* parallelizer.

**sc[n]**  Specifies diagnostic messages issued by the Source Checker. n can be any of the following: 1, 2, 3. For more details on these values, see below. This value is equivalent to deprecated value sv[n].

**warn**  Specifies diagnostic messages that have a "warning" severity level.

**error**  Specifies diagnostic messages that have an "error" severity level.

**remark**  Specifies diagnostic messages that are remarks or comments.

**cpu-dispatch**  Specifies the CPU dispatch remarks for diagnostic messages. These remarks are enabled by default. This diagnostic group is only available on IA-32 architecture and Intel® 64 architecture.

**id[,id,...]**  Specifies the ID number of one or more messages. If you specify more than one message number, they must be separated by commas. There can be no intervening white space between each id.

**tag[,tag,...]**  Specifies the mnemonic name of one or more messages. If you specify more than one mnemonic name, they must be
separated by commas. There can be no intervening white space between each tag.

**Default**

OFF  
The compiler issues certain diagnostic messages by default.

**Description**

This option controls the display of diagnostic information. Diagnostic messages are output to stderr unless compiler option `-diag-file` (Linux and Mac OS X) or `/Qdiag-file` (Windows) is specified.

When `diag-list` value "warn" is used with the Source Checker (sc) diagnostics, the following behavior occurs:

- Option `-diag-enable warn` (Linux and Mac OS X) and `/Qdiag-enable:warn` (Windows) enable all Source Checker diagnostics except those that have an "error" severity level. They enable all Source Checker warnings, cautions, and remarks.
- Option `-diag-disable warn` (Linux and Mac OS X) and `/Qdiag-disable:warn` (Windows) disable all Source Checker diagnostics except those that have an "error" severity level. They suppress all Source Checker warnings, cautions, and remarks.

The following table shows more information on values you can specify for `diag-list` item sc.

<table>
<thead>
<tr>
<th><code>diag-list Item</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sc[n]</code></td>
<td>The value of <code>n</code> for Source Checker messages can be any of the following:</td>
</tr>
<tr>
<td>1</td>
<td>Produces the diagnostics with severity level set to all critical errors.</td>
</tr>
<tr>
<td>2</td>
<td>Produces the diagnostics with severity level set to all errors. This is the default if <code>n</code> is not specified.</td>
</tr>
<tr>
<td>3</td>
<td>Produces the diagnostics with severity level set to all errors and warnings.</td>
</tr>
</tbody>
</table>

To control the diagnostic information reported by the vectorizer, use the `-vec-report` (Linux and Mac OS X) or `/Qvec-report` (Windows) option.

To control the diagnostic information reported by the auto-parallelizer, use the `-par-report` (Linux and Mac OS X) or `/Qpar-report` (Windows) option.
Alternate Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Linux and Mac OS X</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable vec</td>
<td>-vec-report</td>
<td>/Qvec-report</td>
</tr>
<tr>
<td>disable vec</td>
<td>-vec-report0</td>
<td>/Qvec-report0</td>
</tr>
<tr>
<td>enable par</td>
<td>-par-report</td>
<td>/Qpar-report</td>
</tr>
<tr>
<td>disable par</td>
<td>-par-report0</td>
<td>/Qpar-report0</td>
</tr>
</tbody>
</table>

Example

The following example shows how to enable diagnostic IDs 117, 230 and 450:

```bash
-diag-enable 117,230,450  ! Linux and Mac OS X systems
/Qdiag-enable:117,230,450  ! Windows systems
```

The following example shows how to change vectorizer diagnostic messages to warnings:

```bash
-diag-enable vec -diag-warning vec  ! Linux and Mac OS X systems
/Qdiag-enable:vec /Qdiag-warning:vec  ! Windows systems
```

Note that you need to enable the vectorizer diagnostics before you can change them to warnings.

The following example shows how to disable all auto-parallelizer diagnostic messages:

```bash
-diag-disable par  ! Linux and Mac OS X systems
/Qdiag-disable:par  ! Windows systems
```

The following example shows how to produce Source Checker diagnostic messages for all critical errors:

```bash
-diag-enable sc1  ! Linux and Mac OS X systems
/Qdiag-enable:sc1  ! Windows system
```

The following example shows how to cause Source Checker diagnostics (and default diagnostics) to be sent to a file:

```bash
-diag-enable sc -diag-file=stat_ver_msg  ! Linux and Mac OS X systems
/Qdiag-enable:sc /Qdiag-file:stat_ver_msg ! Windows systems
```
Note that you need to enable the Source Checker diagnostics before you can send them to a file. In this case, the diagnostics are sent to file stat_ver_msg.diag. If a file name is not specified, the diagnostics are sent to name-of-the-first-source-file.diag.

The following example shows how to change all diagnostic warnings and remarks to errors:

- `diag-error warn,remark` ! Linux and Mac OS X systems
- `/Qdiag-error:warn,remark` ! Windows systems

See Also

- `diag-dump`, `Qdiag-dump`
- `diag-id-numbers`, `Qdiag-id-numbers`
- `diag-file`, `Qdiag-file`
- `par-report`, `Qpar-report`
- `vec-report`, `Qvec-report`

**diag-dump, Qdiag-dump**

_Tells the compiler to print all enabled diagnostic messages and stop compilation._

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-diag-dump`

Windows:

- `/Qdiag-dump`
Arguments
None

Default
OFF

The compiler issues certain diagnostic messages by default.

Description
This option tells the compiler to print all enabled diagnostic messages and stop compilation. The diagnostic messages are output to stdout.

This option prints the enabled diagnostics from all possible diagnostics that the compiler can issue, including any default diagnostics.

If -diag-enable diag-list (Linux and Mac OS X) or /Qdiag-enable diag-list (Windows) is specified, the print out will include the diag-list diagnostics.

Alternate Options
None

Example
The following example adds vectorizer diagnostic messages to the printout of default diagnostics:
-diag-enable vec -diag-dump ! Linux and Mac OS X systems
/Qdiag-enable:vec /Qdiag-dump ! Windows systems

See Also
•
•
• diag, Qdiag

diag, Qdiag
Controls the display of diagnostic information.

IDE Equivalent
Windows: Diagnostics > Disable Specific Diagnostics (/Qdiag-disable:id)
Diagnostics > Level of Source Code Analysis (/Qdiag-enable[:sc1,sc2,sc3])
Linux: Compilation Diagnostics > Disable Specific Diagnostics (-diag-disable id)
**Compilation Diagnostics > Level of Source Code Analysis**

(-diag-enable [sc1,sc2,sc3] or -diag-disable sv)

Mac OS X: **Diagnostics > Disable Specific Diagnostics**

(-diag-disable id)

**Diagnostics > Level of Source Code Analysis**

(-diag-enable [sc1,sc2,sc3])

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

-diag-type diag-list

Windows:

/Qdiag-type:diag-list

**Arguments**

<table>
<thead>
<tr>
<th>type</th>
<th>Is an action to perform on diagnostics. Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Enables a diagnostic message or a group of messages.</td>
</tr>
<tr>
<td>disable</td>
<td>Disables a diagnostic message or a group of messages.</td>
</tr>
<tr>
<td>error</td>
<td>Tells the compiler to change diagnostics to errors.</td>
</tr>
<tr>
<td>warning</td>
<td>Tells the compiler to change diagnostics to warnings.</td>
</tr>
<tr>
<td>remark</td>
<td>Tells the compiler to change diagnostics to remarks (comments).</td>
</tr>
</tbody>
</table>

**diag-list**

<table>
<thead>
<tr>
<th>Is a diagnostic group or ID value. Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver</td>
</tr>
</tbody>
</table>
port-linux Specifies diagnostic messages for language features that may cause errors when porting to Linux. This diagnostic group is only available on Windows systems.

port-win Specifies diagnostic messages for GNU extensions that may cause errors when porting to Windows. This diagnostic group is only available on Linux and Mac OS X systems.

thread Specifies diagnostic messages that help in thread-enabling a program.

vec Specifies diagnostic messages issued by the vectorizer.

par Specifies diagnostic messages issued by the auto-parallelizer (parallel optimizer).

openmp Specifies diagnostic messages issued by the OpenMP* parallelizer.

sc[n] Specifies diagnostic messages issued by the Source Checker. $n$ can be any of the following: 1, 2, 3. For more details on these values, see below. This value is equivalent to deprecated value sv[n].

warn Specifies diagnostic messages that have a "warning" severity level.

eerror Specifies diagnostic messages that have an "error" severity level.

remark Specifies diagnostic messages that are remarks or comments.

cpu-dispatch Specifies the CPU dispatch remarks for diagnostic messages. These remarks are enabled by default. This diagnostic group is only available on IA-32 architecture and Intel® 64 architecture.
Specifies the ID number of one or more messages. If you specify more than one message number, they must be separated by commas. There can be no intervening white space between each id.

tag[,tag,...] Specifies the mnemonic name of one or more messages. If you specify more than one mnemonic name, they must be separated by commas. There can be no intervening white space between each tag.

**Default**

OFF

The compiler issues certain diagnostic messages by default.

**Description**

This option controls the display of diagnostic information. Diagnostic messages are output to stderr unless compiler option `-diag-file` (Linux and Mac OS X) or `/Qdiag-file` (Windows) is specified.

When `diag-list` value "warn" is used with the Source Checker (sc) diagnostics, the following behavior occurs:

- **Option** `-diag-enable warn` (Linux and Mac OS X) and `/Qdiag-enable:warn` (Windows) enable all Source Checker diagnostics except those that have an "error" severity level. They enable all Source Checker warnings, cautions, and remarks.

- **Option** `-diag-disable warn` (Linux and Mac OS X) and `/Qdiag-disable:warn` (Windows) disable all Source Checker diagnostics except those that have an "error" severity level. They suppress all Source Checker warnings, cautions, and remarks.

The following table shows more information on values you can specify for `diag-list` item sc.

<table>
<thead>
<tr>
<th><code>diag-list</code> Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sc[n]</code></td>
<td>The value of <code>n</code> for Source Checker messages can be any of the following:</td>
</tr>
<tr>
<td></td>
<td>Produces the diagnostics with severity level set to all critical errors.</td>
</tr>
</tbody>
</table>
### diag-list Description

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Produces the diagnostics with severity level set to all errors. This is the default if n is not specified. Produces the diagnostics with severity level set to all errors and warnings.</td>
</tr>
</tbody>
</table>

To control the diagnostic information reported by the vectorizer, use the `-vec-report` (Linux and Mac OS X) or `/Qvec-report` (Windows) option.

To control the diagnostic information reported by the auto-parallelizer, use the `-par-report` (Linux and Mac OS X) or `/Qpar-report` (Windows) option.

### Alternate Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| enable vec | Linux and Mac OS X: `-vec-report`  
Windows: `/Qvec-report` |
| disable vec | Linux and Mac OS X: `-vec-report0`  
Windows: `/Qvec-report0` |
| enable par | Linux and Mac OS X: `-par-report`  
Windows: `/Qpar-report` |
| disable par | Linux and Mac OS X: `-par-report0`  
Windows: `/Qpar-report0` |

### Example

The following example shows how to enable diagnostic IDs 117, 230 and 450:

```
-diag-enable 117,230,450  ! Linux and Mac OS X systems
/Qdiag-enable:117,230,450  ! Windows systems
```

The following example shows how to change vectorizer diagnostic messages to warnings:

```
-diag-enable vec -diag-warning vec  ! Linux and Mac OS X systems
/Qdiag-enable:vec /Qdiag-warning:vec  ! Windows systems
```

Note that you need to enable the vectorizer diagnostics before you can change them to warnings.

The following example shows how to disable all auto-parallelizer diagnostic messages:

```
-diag-disable par  ! Linux and Mac OS X systems
/Qdiag-disable:par  ! Windows systems
```
The following example shows how to produce Source Checker diagnostic messages for all critical errors:

```plaintext
-diaq-enable sc1 ! Linux and Mac OS X systems
/Qdiag-enable:sc1 ! Windows system
```

The following example shows how to cause Source Checker diagnostics (and default diagnostics) to be sent to a file:

```plaintext
-diaq-enable sc -diaq-file=stat_ver_msg ! Linux and Mac OS X systems
```

```plaintext
/Qdiag-enable:sc /Qdiag-file:stat_ver_msg ! Windows systems
```

Note that you need to enable the Source Checker diagnostics before you can send them to a file. In this case, the diagnostics are sent to file stat_ver_msg.diag. If a file name is not specified, the diagnostics are sent to name-of-the-first-source-file.diag.

The following example shows how to change all diagnostic warnings and remarks to errors:

```plaintext
-diaq-error warn,remark ! Linux and Mac OS X systems
```

```plaintext
/Qdiag-error:warn,remark ! Windows systems
```

**See Also**

- `diag-dump`, `Qdiag-dump`
- `diag-id-numbers`, `Qdiag-id-numbers`
- `diag-file`, `Qdiag-file`
- `par-report`, `Qpar-report`
- `vec-report`, `Qvec-report`

**diag-enable sc-include**, `Qdiag-enable:sc-include`

*Tells a source code analyzer to process include files and source files when issuing diagnostic messages.*

**IDE Equivalent**

Windows: *Diagnostics > Analyze Include Files*

Linux: *Compilation Diagnostics > Analyze Include Files*
Mac OS X: **Diagnostics > Analyze Include Files**

**Architectures**
IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X:
- `-diag-enable sc-include`

Windows:
- `/Qdiag-enable:sc-include`

**Arguments**
None

**Default**
OFF

The compiler issues certain diagnostic messages by default. If the Source Checker is enabled, include files are not analyzed by default.

**Description**
This option tells a source code analyzer (Source Checker) to process include files and source files when issuing diagnostic messages. Normally, when Source Checker diagnostics are enabled, only source files are analyzed.

To use this option, you must also specify `-diag-enable sc` (Linux and Mac OS X) or `/Qdiag-enable:sc` (Windows) to enable the Source Checker diagnostics, or `-diag-enable sc-parallel` (Linux and Mac OS X) or `/Qdiag-enable:sc-parallel` (Windows) to enable parallel lint.

**Alternate Options**

Linux and Mac OS X: `-diag-enable sv-include` *(this is a deprecated option)*

Windows: `/Qdiag-enable:sv-include` *(this is a deprecated option)*
**Example**

The following example shows how to cause include files to be analyzed as well as source files:

```
-diaenable:sc -diaenable:sc-include ! Linux and Mac OS systems
/Qdiag-enable:sc /Qdiag-enable:sc-include ! Windows systems
```

In the above example, the first compiler option enables Source Checker messages. The second compiler option causes include files referred to by the source file to be analyzed also.

**See Also**

- `diaenable sc-parallel, Qdiag-enable:sc-parallel`
- `dia, Qdiag`

**diag-enable sc-parallel, Qdiag-enable:sc-parallel**

*Enables analysis of parallelization in source code (parallel lint diagnostics).*

**IDE Equivalent**

Windows: **Diagnostics > Level of Source Code Parallelization Analysis**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

```
-diaenable sc-parallel[n]
```

**Windows:**

```
/Qdiag-enable:sc-parallel[n]
```

**Arguments**

```
n
```

Is the level of analysis to perform. Possible values are:
1 Produces the diagnostics with severity level set to all critical errors.

2 Tells the compiler to generate a report with the medium level of detail. Produces the diagnostics with severity level set to all errors. This is the default if \( n \) is not specified.

3 Produces the diagnostics with severity level set to all errors and warnings.

**Default**

OFF The compiler does not analyze parallelization in source code.

**Description**

This option enables analysis of parallelization in source code (parallel lint diagnostics). Currently, this analysis uses OpenMP pragmas, so this option has no effect unless option /Qopenmp (Windows) or option -openmp (Linux and Mac OS X) is set.

Parallel lint performs interprocedural source code analysis to identify mistakes when using parallel pragmas. It reports various problems that are difficult to find, including data dependency and potential deadlocks.

Source Checker diagnostics (enabled by /Qdiag-enable:sc on Windows* OS or -diag-enable sc on Linux* OS and Mac OS* X) are a superset of parallel lint diagnostics. Therefore, if Source Checker diagnostics are enabled, the parallel lint option is not taken into account.

**Alternate Options**

None

**See Also**

- 
- 
- diag, Qdiag
**diag-error-limit, Qdiag-error-limit**

Specifies the maximum number of errors allowed before compilation stops.

**IDE Equivalent**

Windows: Compilation Diagnostics > Error Limit

Linux: Compilation Diagnostics > Set Error Limit

Mac OS X: Compilation Diagnostics > Error Limit

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `diag-error-limit n`
- `no-diag-error-limit`

Windows:

/Qdiag-error-limit:n

/Qdiag-error-limit-

**Arguments**

$n$  
Is the maximum number of error-level or fatal-level compiler errors allowed.

**Default**

30  
A maximum of 30 error-level and fatal-level messages are allowed.

**Description**

This option specifies the maximum number of errors allowed before compilation stops. It indicates the maximum number of error-level or fatal-level compiler errors allowed for a file specified on the command line.
If you specify `-no-diag-error-limit` (Linux and Mac OS X) or `/Qdiag-error-limit-` (Windows) on the command line, there is no limit on the number of errors that are allowed. If the maximum number of errors is reached, a warning message is issued and the next file (if any) on the command line is compiled.

**Alternate Options**

Linux and Mac OS X: `-wn` (this is a deprecated option)

Windows: `/Qwn` (this is a deprecated option)

**diag-file, Qdiag-file**

*Causes the results of diagnostic analysis to be output to a file.*

**IDE Equivalent**

Windows: **Diagnostics > Diagnostics File**

Linux: **Compilation Diagnostics > Diagnostics File**

Mac OS X: **Diagnostics > Diagnostics File**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-diag-file [=file]
```

Windows:

```
/Qdiag-file [:file]
```

**Arguments**

`file` Is the name of the file for output.

**Default**

OFF Diagnostic messages are output to stderr.
Description

This option causes the results of diagnostic analysis to be output to a file. The file is placed in the current working directory.

If file is specified, the name of the file is file.diag. The file can include a file extension; for example, if file.ext is specified, the name of the file is file.ext.

If file is not specified, the name of the file is name-of-the-first-source-file.diag. This is also the name of the file if the name specified for file conflicts with a source file name provided in the command line.

NOTE. If you specify -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows) and you also specify -diag-file-append (Linux and Mac OS X) or /Qdiag-file-append (Windows), the last option specified on the command line takes precedence.

Alternate Options

None

Example

The following example shows how to cause diagnostic analysis to be output to a file named my_diagnostics.diag:

-diag-file=my_diagnostics ! Linux and Mac OS X systems
/Qdiag-file:my_diagnostics ! Windows systems

See Also

- diag, Qdiag
- diag-file-append, Qdiag-file-append

diag-file-append, Qdiag-file-append

Causes the results of diagnostic analysis to be appended to a file.

IDE Equivalent

None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-diag-file-append[=file]

Windows:
/Qdiag-file-append[:file]

Arguments

file
Is the name of the file to be appended to. It can include a path.

Default
OFF
Diagnostic messages are output to stderr.

Description
This option causes the results of diagnostic analysis to be appended to a file. If you do not specify a path, the driver will look for file in the current working directory.

If file is not found, then a new file with that name is created in the current working directory.
If the name specified for file conflicts with a source file name provided in the command line, the name of the file is name-of-the-first-source-file.diag.

NOTE. If you specify -diag-file-append (Linux and Mac OS X) or /Qdiag-file-append (Windows) and you also specify -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows), the last option specified on the command line takes precedence.

Alternate Options
None
Example

The following example shows how to cause diagnostic analysis to be appended to a file named my_diagnostics.txt:

- diag-file-append=my_diagnostics.txt  ! Linux and Mac OS X systems
/Qdiag-file-append:my_diagnostics.txt    ! Windows systems

See Also

- diag, Qdiag
- diag-file, Qdiag-file

**diag-id-numbers, Qdiag-id-numbers**

Determines whether the compiler displays diagnostic messages by using their ID number values.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

- diag-id-numbers
- no-diag-id-numbers

**Windows:**

/Qdiag-id-numbers
/Qdiag-id-numbers-

Arguments

None
**Default**

- `diag-id-numbers` or `/Qdiag-id-numbers`  
  The compiler displays diagnostic messages by using their ID number values.

**Description**

This option determines whether the compiler displays diagnostic messages by using their ID number values. If you specify `-no-diag-id-numbers` (Linux and Mac OS X) or `/Qdiag-id-numbers-` (Windows), mnemonic names are output for driver diagnostics only.

**Alternate Options**

None

**See Also**

- `diag`, `/Qdiag`

**diag-once, /Qdiag-once**

*Tells the compiler to issue one or more diagnostic messages only once.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-diag-once id[,id,...]`

**Windows:**

- `/Qdiag-once:init id[,id,...]`
Arguments

$id$  Is the ID number of the diagnostic message. If you specify more than one message number, they must be separated by commas. There can be no intervening white space between each $id$.

Default

OFF  The compiler issues certain diagnostic messages by default.

Description

This option tells the compiler to issue one or more diagnostic messages only once.

Alternate Options

Linux: $-W0$ (this is a deprecated option)
Windows: $/Qw0$ (this is a deprecated option)

$dD$, $QdD$

Same as $-DM$, but outputs $\#define$ directives in preprocessed source.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

$-dD$

Windows:

$/QdD$

Arguments

None
Default
OFF  The compiler does not output #define directives.

Description
Same as -dM, but outputs #define directives in preprocessed source. To use this option, you must also specify the E option.

Alternate Options
None

dM, QdM
*Tells the compiler to output macro definitions in effect after preprocessing.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:

-dM

Windows:

/QdM

Arguments
None

Default
OFF  The compiler does not output macro definitions after preprocessing.
Description
This option tells the compiler to output macro definitions in effect after preprocessing. To use this option, you must also specify the E option.

Alternate Options
None

See Also
•
•
• E

dN, QdN
Same as -dD, but output #define directives contain only macro names.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-dN

Windows:
/QdN

Arguments
None

Default
OFF The compiler does not output #define directives.
**Description**

Same as -dD, but output #define directives contain only macro names. To use this option, you must also specify the E option.

**Alternate Options**

None

**Weffc++, Qefc++**

This option enables warnings based on certain C++ programming guidelines.

**IDE Equivalent**

Windows: None

Linux: **Compilation Diagnostics > Enable Warnings for Style Guideline Violations**

Mac OS X: **Diagnostics > Report Effective C++ Violations**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-Weffc++

**Windows:**

/Qefc++

**Arguments**

None

**Default**

OFF Diagnostics are not enabled.
Description

This option enables warnings based on certain programming guidelines developed by Scott Meyers in his books on effective C++ programming. With this option, the compiler emits warnings for these guidelines:

- Use `const` and `inline` rather than `#define`. Note that you will only get this in user code, not system header code.
- Use `<iostream>` rather than `<stdio.h>`.
- Use `new` and `delete` rather than `malloc` and `free`.
- Use C++ style comments in preference to C style comments. C comments in system headers are not diagnosed.
- Use `delete` on pointer members in destructors. The compiler diagnoses any pointer that does not have a `delete`.
- Make sure you have a user copy constructor and assignment operator in classes containing pointers.
- Use initialization rather than assignment to members in constructors.
- Make sure the initialization list ordering matches the declaration list ordering in constructors.
- Make sure base classes have virtual destructors.
- Make sure `operator=` returns `*this`.
- Make sure prefix forms of increment and decrement return a `const` object.
- Never overload operators `&&`, `||`, and `,`.

NOTE. The warnings generated by this compiler option are based on the following books from Scott Meyers:

- Effective C++ Second Edition - 50 Specific Ways to Improve Your Programs and Designs
- More Effective C++ - 35 New Ways to Improve Your Programs and Designs

Alternate Options

None
fast-transcendentals, Qfast-transcendentals
 Enables the compiler to replace calls to transcendental functions with faster but less precise implementations.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-fast-transcendentals
-no-fast-transcendentals

Windows:
/Qfast-transcendentals
/Qfast-transcendentals-

Default
-fast-transcendentals or /Qfast-transcendentals
The default depends on the setting of -fp-model (Linux and Mac OS X) or /fp (Windows). The default is ON if default setting -fp-model fast or /fp:fast is in effect. However, if a value-safe option such as -fp-model precise or /fp:precise is specified, the default is OFF.

Description
This option enables the compiler to replace calls to transcendental functions with implementations that may be faster but less precise.

It tells the compiler to perform certain optimizations on transcendental functions, such as replacing individual calls to sine in a loop with a single call to a less precise vectorized sine library routine.

This option has an effect only when specified with one of the following options:
Windows* OS: /fp:except or /fp:precise
Linux* OS and Mac OS* X: -fp-model except or -fp-model precise

You cannot use this option with option -fp-model strict (Linux and Mac OS X) or /fp:strict (Windows).

**Alternate Options**

None

**See Also**

- fp-model
- fp

**fma, Qfma**

*Enables the combining or contraction of floating-point multiplications and add or subtract operations.*

**IDE Equivalent**

Windows: None
Linux: **Floating Point > Floating-point Operation Contraction**
Mac OS X: None

**Architectures**

IA-64 architecture

**Syntax**

Linux:
- fma
- no-fma

Mac OS X:
None
Windows:
/Qfma
/Qfma-

Arguments
None

Default
-fma
or /Qfma

Floating-point multiplications and add/subtract operations are combined. However, if you specify -fp-model strict (Linux) or /fp:strict (Windows), but do not explicitly specify -fma or /Qfma, the default is -no-fma or /Qfma-.

Description
This option enables the combining or contraction of floating-point multiplications and add or subtract operations into a single operation.

Alternate Options
Linux: -IPF-fma (this is a deprecated option)
Windows: /QIPF-fma (this is a deprecated option)

See Also
•
•
• fp-model, fp

Floating-point Operations: Floating-point Options Quick Reference

falign-functions, Qfnalign
* Tells the compiler to align functions on an optimal byte boundary.

IDE Equivalent
None
Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:
-falign-functions=[n]
-fno-align-functions

Windows:
/Qfnalign[:n]
/Qfnalign-

Arguments

n Is the byte boundary for function alignment. Possible values are 2 or 16.

Default

-fno-align-functions or The compiler aligns functions on 2-byte boundaries. This is the same as specifying -falign-functions=2 (Linux and Mac OS X) or /Qfnalign:2 (Windows).

Description

This option tells the compiler to align functions on an optimal byte boundary. If you do not specify \( n \), the compiler aligns the start of functions on 16-byte boundaries.

Alternate Options

None

fnsplit, Qfnsplit

Enables function splitting.

IDE Equivalent

Windows: Code Generation > Disable Function Splitting

Linux: None
Mac OS X: None

**Architectures**

/Qfnsplit[-]: IA-32 architecture, Intel® 64 architecture
- [no-]fnsplit: IA-64 architecture

**Syntax**

Linux:
- fnsplit
- no-fnsplit

Mac OS X:
None

Windows:
/Qfnsplit
/Qfnsplit-

**Arguments**

None

**Default**

-no-fnsplit
or/Qfnsplit-

Function splitting is not enabled unless -prof-use (Linux) or 
/Qprof-use (Windows) is also specified.

**Description**

This option enables function splitting if -prof-use (Linux) or /Qprof-use (Windows) is also specified. Otherwise, this option has no effect.

It is enabled automatically if you specify -prof-use or /Qprof-use. If you do not specify one of those options, the default is -no-fnsplit (Linux) or /Qfnsplit- (Windows), which disables function splitting but leaves function grouping enabled.

To disable function splitting when you use -prof-use or /Qprof-use, specify -no-fnsplit or /Qfnsplit-. 
Alternate Options

None

fp-port, Qfp-port

Rounds floating-point results after floating-point operations.

IDE Equivalent

Windows: Optimization > Floating-point Precision Improvements
Linux: Floating Point > Round Floating-Point Results
Mac OS X: Floating Point > Round Floating-Point Results

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- -fp-port
  - -no-fp-port

Windows:
/Qfp-port
/Qfp-port-

Arguments

None

Default

-no-fp-port or /Qfp-port-

The default rounding behavior depends on the compiler's code generation decisions and the precision parameters of the operating system.
**Description**

This option rounds floating-point results after floating-point operations. Rounding to user-specified precision occurs at assignments and type conversions. This has some impact on speed.

The default is to keep results of floating-point operations in higher precision. This provides better performance but less consistent floating-point results.

**Alternate Options**

None

**fp-relaxed, Qfp-relaxed**

*Enables use of faster but slightly less accurate code sequences for math functions.*

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

**Linux:**
- -fp-relaxed
- -no-fp-relaxed

**Mac OS X:**
None

**Windows:**
/ Qfp-relaxed
/ Qfp-relaxed-

**Arguments**

None
Default
-no-fp-relaxed or /Qfp-relaxed

Default code sequences are used for math functions.

Description
This option enables use of faster but slightly less accurate code sequences for math functions, such as divide and sqrt. When compared to strict IEEE* precision, this option slightly reduces the accuracy of floating-point calculations performed by these functions, usually limited to the least significant digit.

This option also enables the performance of more aggressive floating-point transformations, which may affect accuracy.

Alternate Options
Linux: -IPF-fp-relaxed (this is a deprecated option)
Windows: /QIPF-fp-relaxed (this is a deprecated option)

See Also
•
•
• fp-model, fp

fp-speculation, Qfp-speculation
Tells the compiler the mode in which to speculate on floating-point operations.

IDE Equivalent
Windows: Optimization > Floating-Point Speculation
Linux: Floating Point > Floating-Point Speculation
Mac OS X: Floating Point > Floating-Point Speculation

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
-`fp-speculation=mode`

Windows:
`/Qfp-speculation:mode`

Arguments

`mode` Is the mode for floating-point operations. Possible values are:

- `fast` Tells the compiler to speculate on floating-point operations.
- `safe` Tells the compiler to disable speculation if there is a possibility that the speculation may cause a floating-point exception.
- `strict` Tells the compiler to disable speculation on floating-point operations.
- `off` This is the same as specifying strict.

Default

-`fp-speculation=fast` or `/Qfp-speculation:fast`
The compiler speculates on floating-point operations. This is also the behavior when optimizations are enabled. However, if you specify no optimizations (`-00` on Linux; `/Od` on Windows), the default is `-fp-speculation=safe` (Linux) or `/Qfp-speculation:safe` (Windows).

Description

This option tells the compiler the mode in which to speculate on floating-point operations.

Alternate Options

None
**fp-stack-check, Qfp-stack-check**

*Tells the compiler to generate extra code after every function call to ensure that the floating-point stack is in the expected state.*

**IDE Equivalent**

Windows: None

Linux: **Floating Point > Check Floating-point Stack**

Mac OS X: **Floating Point > Check Floating-point Stack**

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

```
-fp-stack-check
```

**Windows:**

```
/Qfp-stack-check
```

**Arguments**

None

**Default**

OFF

There is no checking to ensure that the floating-point (FP) stack is in the expected state.

**Description**

This option tells the compiler to generate extra code after every function call to ensure that the floating-point (FP) stack is in the expected state.

By default, there is no checking. So when the FP stack overflows, a NaN value is put into FP calculations and the program’s results differ. Unfortunately, the overflow point can be far away from the point of the actual bug. This option places code that causes an access violation exception immediately after an incorrect call occurs, thus making it easier to locate these issues.
Alternate Options
None

See Also
•
•
Floating-point Operations:
• Checking the Floating-point Stack State

ffreestanding, Qfreestanding
Ensures that compilation takes place in a freestanding environment.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-ffreestanding

Windows:
/Qfreestanding

Arguments
None

Default
OFF Standard libraries are used during compilation.
**Description**

This option ensures that compilation takes place in a freestanding environment. The compiler assumes that the standard library may not exist and program startup may not necessarily be at main. This environment meets the definition of a freestanding environment as described in the C and C++ standard.

An example of an application requiring such an environment is an OS kernel.

**NOTE.** When you specify this option, the compiler will not assume the presence of compiler-specific libraries. It will only generate calls that appear in the source code.

**Alternate Options**

None

ftz, Qftz

*Flushes denormal results to zero.*

**IDE Equivalent**

Windows: **Optimization > Flush Denormal Results to Zero**

Linux: **Floating-Point > Flush Denormal Results to Zero**

Mac OS X: **Floating-Point > Flush Denormal Results to Zero**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `ftz`
- `no-ftz`

**Windows:**

- `/Qftz`
- `/Qftz-`
Arguments
None

Default
On systems using IA-64 architecture, the compiler lets results gradually underflow. On systems using IA-32 architecture and Intel® 64 architecture, denormal results are flushed to zero.

Systems using IA-64 architecture: -no-ftz or /Qftz-
Systems using IA-32 architecture and Intel® 64 architecture: -ftz or /Qftz

Description
This option flushes denormal results to zero when the application is in the gradual underflow mode. It may improve performance if the denormal values are not critical to your application's behavior.

This option sets or resets the FTZ and the DAZ hardware flags. If FTZ is ON, denormal results from floating-point calculations will be set to the value zero. If FTZ is OFF, denormal results remain as is. If DAZ is ON, denormal values used as input to floating-point instructions will be treated as zero. If DAZ is OFF, denormal instruction inputs remain as is. Systems using IA-64 architecture have FTZ but not DAZ. Systems using Intel® 64 architecture have both FTZ and DAZ. FTZ and DAZ are not supported on all IA-32 architectures.

When -ftz (Linux and Mac OS X) or /Qftz (Windows) is used in combination with an SSE-enabling option on systems using IA-32 architecture (for example, xN or QxN), the compiler will insert code in the main routine to set FTZ and DAZ. When -ftz or /Qftz is used without such an option, the compiler will insert code to conditionally set FTZ/DAZ based on a run-time processor check. -no-ftz (Linux and Mac OS X) or /Qftz- (Windows) will prevent the compiler from inserting any code that might set FTZ or DAZ.

This option only has an effect when the main program is being compiled. It sets the FTZ/DAZ mode for the process. The initial thread and any threads subsequently created by that process will operate in FTZ/DAZ mode.

On systems using IA-64 architecture, optimization option O3 sets -ftz and /Qftz; optimization option O2 sets -no-ftz (Linux) and /Qftz- (Windows). On systems using IA-32 architecture and Intel® 64 architecture, every optimization option O level, except O0, sets -ftz and /Qftz.

If this option produces undesirable results of the numerical behavior of your program, you can turn the FTZ/DAZ mode off by using -no-ftz or /Qftz- in the command line while still benefiting from the O3 optimizations.
NOTE. Options `-ftz` and `/Qftz` are performance options. Setting these options does not guarantee that all denormals in a program are flushed to zero. They only cause denormals generated at run time to be flushed to zero.

Alternate Options

None

Example

To see sample code showing the state of the FTZ and DAZ flags, see Reading the FTZ and DAZ Flags.

See Also

- 
- 
- `x, Qx`

`global-hoist, Qglobal-hoist`

*Enables certain optimizations that can move memory loads to a point earlier in the program execution than where they appear in the source.*

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

- `global-hoist`
- `no-global-hoist`

Windows:

/`Qglobal-hoist`
/Qglobal-hoist-

**Arguments**
None

**Default**
- `global-hoist` or `/Qglobal-hoist`
  Certain optimizations are enabled that can move memory loads.

**Description**
This option enables certain optimizations that can move memory loads to a point earlier in the program execution than where they appear in the source. In most cases, these optimizations are safe and can improve performance.

The `-no-global-hoist` (Linux and Mac OS X) or `/Qglobal-hoist-` (Windows) option is useful for some applications, such as those that use shared or dynamically mapped memory, which can fail if a load is moved too early in the execution stream (for example, before the memory is mapped).

**Alternate Options**
None

**H, QH**
*Tells the compiler to display the include file order and continue compilation.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- `H`
Windows:
/QH

Arguments
None

Default
OFF                      Compilation occurs as usual.

Description
This option tells the compiler to display the include file order and continue compilation.

Alternate Options
None

help-pragma, Qhelp-pragma
Displays all supported pragmas.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-depth-pragma

Windows:
/Qhelp-pragma

Arguments
None
**Default**
OFF

**Description**
No list is displayed unless this compiler option is specified.

**Alternate Options**
None

**QIA64-fr32**
_Disables use of high floating-point registers._

**IDE Equivalent**
None

**Architectures**
IA-64 architecture

**Syntax**

**Linux and Mac OS X:**
None

**Windows:**
/QIA64-fr32

**Arguments**
None

**Default**
OFF

**Description**
Use of high floating-point registers is enabled.

This option disables use of high floating-point registers.
Alternate Options

None

Qifist

See rcd, Qrcd.

inline-calloc, Qinline-calloc

Tells the compiler to inline calls to calloc() as calls to malloc() and memset().

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:

-inline-calloc
-no-inline-calloc

Windows:

/Qinline-calloc
/Qinline-calloc-

Arguments

None

Default

-no-inline-calloc or /Qinline-calloc-

The compiler inlines calls to calloc() as calls to calloc().
Description
This option tells the compiler to inline calls to `calloc()` as calls to `malloc()` and `memset()`. This enables additional `memset()` optimizations. For example, it can enable inlining as a sequence of store operations when the size is a compile time constant.

Alternate Options
None

`inline-debug-info, Qinline-debug-info`
*Produces enhanced source position information for inlined code. This is a deprecated option.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
**Linux and Mac OS X:**
```
--inline-debug-info
```

**Windows:**
```
/Qinline-debug-info
```

Arguments
None

Default
OFF

No enhanced source position information is produced for inlined code.

Description
This option produces enhanced source position information for inlined code. This leads to greater accuracy when reporting the source location of any instruction. It also provides enhanced debug information useful for function call traceback.
To use this option for debugging, you must also specify a debug enabling option, such as `-g` (Linux) or `/debug` (Windows).

**Alternate Options**

Linux and Mac OS X: `-debug inline-debug-info`

Windows: `None`

**Qinline-dllimport**

*Determines whether dllimport functions are inlined.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

`/Qinline-dllimport`

`/Qinline-dllimport-`

**Arguments**

None

**Default**

`/Qinline-dllimport`  The dllimport functions are inlined.

**Description**

This option determines whether dllimport functions are inlined. To disable dllimport functions from being inlined, specify `/Qinline-dllimport-`.  

672
Alternate Options
None

**inline-factor, Qinline-factor**
Specifies the percentage multiplier that should be applied to all inlining options that define upper limits.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
- `-inline-factor=n`
- `-no-inline-factor`

**Windows:**
- `/Qinline-factor=n`
- `/Qinline-factor-`

Arguments

$n$ Is a positive integer specifying the percentage value. The default value is 100 (a factor of 1).

Default

- `-no-inline-factor` The compiler uses default heuristics for inline routine expansion.
- `/Qinline-factor-`

Description
This option specifies the percentage multiplier that should be applied to all inlining options that define upper limits:
This option takes the default value for each of the above options and multiplies it by $n$ divided by 100. For example, if 200 is specified, all inlining options that define upper limits are multiplied by a factor of 2. This option is useful if you do not want to individually increase each option limit.

If you specify `-no-inline-factor` (Linux and Mac OS X) or `/Qinline-factor-` (Windows), the following occurs:

- Every function is considered to be a small or medium function; there are no large functions.
- There is no limit to the size a routine may grow when inline expansion is performed.
- There is no limit to the number of times some routine may be inlined into a particular routine.
- There is no limit to the number of times inlining can be applied to a compilation unit.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**CAUTION.** When you use this option to increase default limits, the compiler may do so much additional inlining that it runs out of memory and terminates with an “out of memory” message.

**Alternate Options**
None

**See Also**

- `-inline-max-size` and `/Qinline-max-size`
- `-inline-max-total-size` and `/Qinline-max-total-size`
- `-inline-max-per-routine` and `/Qinline-max-per-routine`
- `-inline-max-per-compile` and `/Qinline-max-per-compile`
inline-forceinline, Qinline-forceinline

Specifies that an inline routine should be inlined whenever the compiler can do so.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-inline-forceinline

Windows:
/Qinline-forceinline

Default
OFF The compiler uses default heuristics for inline routine expansion.

Description
This option specifies that a inline routine should be inlined whenever the compiler can do so. This causes the routines marked with an inline keyword or attribute to be treated as if they were "forceinline".

NOTE. Because C++ member functions whose definitions are included in the class declaration are considered inline functions by default, using this option will also make these member functions "forceinline" functions.

The "forceinline" condition can also be specified by using the keyword __forceinline.
To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS) or `/Qopt-report` (Windows).

**CAUTION.** When you use this option to change the meaning of inline to "forceinline", the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**
None

**See Also**
- opt-report, Qopt-report
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions

**inline-max-per-compile, Qinline-max-per-compile**
Specifies the maximum number of times inlining may be applied to an entire compilation unit.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `inline-max-per-compile=n`
- `no-inline-max-per-compile`

**Windows:**
- `/Qinline-max-per-compile=n`
Arguments

\( n \)  
Is a positive integer that specifies the number of times inlining may be applied.

Default

-no-inline-max-per-compile  
The compiler uses default heuristics for inline routine expansion.

-or/Qinline-max-per-compile-  

Description

This option the maximum number of times inlining may be applied to an entire compilation unit. It limits the number of times that inlining can be applied.

For compilations using Interprocedural Optimizations (IPO), the entire compilation is a compilation unit. For other compilations, a compilation unit is a file.

If you specify -no-inline-max-per-compile (Linux and Mac OS X) or /Qinline-max-per-compile- (Windows), there is no limit to the number of times inlining may be applied to a compilation unit.

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

CAUTION. When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None

See Also

- inline-factor, Qinline-factor
- opt-report, Qopt-report
- Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

**inline-max-per-routine, Qinline-max-per-routine**

Specifies the maximum number of times the inliner may inline into a particular routine.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

- `-inline-max-per-routine=n`
- `-no-inline-max-per-routine`

*Windows:*

- `/Qinline-max-per-routine=n`
- `/Qinline-max-per-routine-`

**Arguments**

`n`  
Is a positive integer that specifies the maximum number of times the inliner may inline into a particular routine.

**Default**

`-no-inline-max-per-routine`  
Or `/Qinline-max-per-routine-`  
The compiler uses default heuristics for inline routine expansion.

**Description**

This option specifies the maximum number of times the inliner may inline into a particular routine. It limits the number of times that inlining can be applied to any routine.

If you specify `-no-inline-max-per-routine` (*Linux and Mac OS X*) or `/Qinline-max-per-routine-` (*Windows*), there is no limit to the number of times some routine may be inlined into a particular routine.
To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**CAUTION.** When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None

**See Also**

- `inline-factor, Qinline-factor`
- `opt-report, Qopt-report`
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions

**inline-max-size, Qinline-max-size**

Specifies the lower limit for the size of what the inliner considers to be a large routine.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-inline-max-size=n`
- `-no-inline-max-size`
**Windows:**

/Quinline-max-size=\textit{n}
/Quinline-max-size-

**Arguments**

\textit{n} \hspace{1cm} Is a positive integer that specifies the minimum size of what the inliner considers to be a large routine.

**Default**

-no-inline-max-size \hspace{1cm} The compiler uses default heuristics for inline routine expansion.
or/Quinline-max-size-

**Description**

This option specifies the lower limit for the size of what the inliner considers to be a large routine (a function). The inliner classifies routines as small, medium, or large. This option specifies the boundary between what the inliner considers to be medium and large-size routines.

The inliner prefers to inline small routines. It has a preference against inlining large routines. So, any large routine is highly unlikely to be inlined.

If you specify -no-inline-max-size (Linux and Mac OS X) or /Quinline-max-size- (Windows), there are no large routines. Every routine is either a small or medium routine.

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

---

**CAUTION.** When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None
See Also

- inline-min-size, Qinline-min-size
- inline-factor, Qinline-factor
- opt-report, Qopt-report
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions

**inline-max-total-size, Qinline-max-total-size**

Specifies how much larger a routine can normally grow when inline expansion is performed.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- -inline-max-total-size=n
- -no-inline-max-total-size

Windows:

- /Qinline-max-total-size=n
- /Qinline-max-total-size-

**Arguments**

n

Is a positive integer that specifies the permitted increase in the routine's size when inline expansion is performed.
Default
-no-inline-max-total-size  or /Qinline-max-total-size-
The compiler uses default heuristics for inline routine expansion.

Description
This option specifies how much larger a routine can normally grow when inline expansion is performed. It limits the potential size of the routine. For example, if 2000 is specified for \( n \), the size of any routine will normally not increase by more than 2000.

If you specify -no-inline-max-total-size (Linux and Mac OS X) or /Qinline-max-total-size- (Windows), there is no limit to the size a routine may grow when inline expansion is performed.

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

To see compiler values for important inlining limits, specify compiler option -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

CAUTION. When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options
None

See Also
- inline-factor, Qinline-factor
- opt-report, Qopt-report
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions
**inline-min-size, Qinline-min-size**

Specifies the upper limit for the size of what the inliner considers to be a small routine.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-inline-min-size=n`
- `-no-inline-min-size`

**Windows:**

/`Qinline-min-size=n`

/`Qinline-min-size-`

**Arguments**

`n`  
Is a positive integer that specifies the maximum size of what the inliner considers to be a small routine.

**Default**

- `/Qno-inline-min-size`  
  The compiler uses default heuristics for inline routine expansion.

- `/Qinline-min-size-`

**Description**

This option specifies the upper limit for the size of what the inliner considers to be a small routine (a function). The inliner classifies routines as small, medium, or large. This option specifies the boundary between what the inliner considers to be small and medium-size routines.

The inliner has a preference to inline small routines. So, when a routine is smaller than or equal to the specified size, it is very likely to be inlined.
If you specify `-no-inline-min-size` (Linux and Mac OS X) or `/Qinline-min-size-` (Windows), there is no limit to the size of small routines. Every routine is a small routine; there are no medium or large routines.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**CAUTION.** When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None

**See Also**

- `inline-min-size, Qinline-min-size`
- `opt-report, Qopt-report`
- Developer Directed Inline Expansion of User Functions
- Compiler Directed Inline Expansion of User Functions

**Qinstall**

Specifies the root directory where the compiler installation was performed.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:

- Qinstalldir

Windows:

None

Arguments

dir

Is the root directory where the installation was performed.

Default

OFF

The default root directory for compiler installation is searched for the compiler.

Description

This option specifies the root directory where the compiler installation was performed. It is useful if you want to use a different compiler or if you did not use the iccvars shell script to set your environment variables.

Alternate Options

None

minstruction, Qinstruction

Determine whether MOVBE instructions are generated for Intel processors.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:

- minstruction=[no]movbe
Windows:
/Qinstruction:[no]movbe

Arguments
None

Default

-\minstruction=nomovbe  The compiler does not generate MOVBE instructions for Intel® or \Qinstruction:nomovbe Atom™ processors.

Description
This option determines whether MOVBE instructions are generated for Intel processors. To use this option, you must also specify -xSSE3_ATOM (Linux and Mac OS X) or /QxSSE3_ATOM (Windows).

If -\minstruction=movbe or \Qinstruction:movbe is specified, the following occurs:

• MOVBE instructions are generated that are specific to the Intel® Atom™ processor.
• The options are ON by default when -xSSE3_ATOM or /QxSSE3_ATOM is specified.
• Generated executables can only be run on Intel® Atom™ processors or processors that support Intel® Streaming SIMD Extensions 3 (Intel® SSE3) and MOVBE.

If -\minstruction=nomovbe or \Qinstruction:nomovbe is specified, the following occurs:

• The compiler optimizes code for the Intel® Atom™ processor, but it does not generate MOVBE instructions.
• Generated executables can be run on non-Intel® Atom™ processors that support Intel® SSE3.

Alternate Options
None

See Also
• x, Qx
**finstrument-functions, Qinstrument-functions**

_Determines whether function entry and exit points are instrumented._

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- `-finstrument-functions`
- `-fno-instrument-functions`

Windows:
- `/Qinstrument-functions`
- `/Qinstrument-functions-`

**Arguments**

None

**Default**

- `-fno-instrument-functions` Function entry and exit points are not instrumented.
- `/Qinstrument-functions-`

**Description**

This option determines whether function entry and exit points are instrumented. It may increase execution time.

The following profiling functions are called with the address of the current function and the address of where the function was called (its "call site"):

- This function is called upon function entry:
• On IA-32 architecture and Intel® 64 architecture:
  void __cyg_profile_func_enter (void *this_fn,
  void *call_site);

• On IA-64 architecture:
  void __cyg_profile_func_enter (void **this_fn,
  void *call_site);

• This function is called upon function exit:
  • On IA-32 architecture and Intel® 64 architecture:
    void __cyg_profile_func_exit (void *this_fn,
    void *call_site);
  • On IA-64 architecture:
    void __cyg_profile_func_exit (void **this_fn,
    void *call_site);

On IA-64 architecture, the additional de-reference of the function pointer argument is required to obtain the function entry point contained in the first word of the function descriptor for indirect function calls. The descriptor is documented in the Intel® Itanium® Software Conventions and Runtime Architecture Guide, section 8.4.2. You can find this design guide at web site http://www.intel.com.

These functions can be used to gather more information, such as profiling information or timing information. Note that it is the user's responsibility to provide these profiling functions.

If you specify -finstrument-functions (Linux and Mac OS X) or /Qinstrument-functions (Windows), function inlining is disabled. If you specify -fno-instrument-functions or /Qinstrument-functions-, inlining is not disabled.

On Linux and Mac OS X systems, you can use the following attribute to stop an individual function from being instrumented:

__attribute__((__no_instrument_function__))

It also stops inlining from being disabled for that individual function.

This option is provided for compatibility with gcc.
Alternate Options
None

**ip, Qip**
Determines whether additional interprocedural optimizations for single-file compilation are enabled.

**IDE Equivalent**
Windows: **Optimization > Interprocedural Optimization**
Linux: **Optimization > Enable Interprocedural Optimizations for Single File Compilation**
Mac OS X: **Optimization > Enable Interprocedural Optimization for Single File Compilation**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
- `-ip`
- `-no-ip`
Windows:
- `/Qip`
- `/Qip-`

**Arguments**
None

**Default**
OFF

Some limited interprocedural optimizations occur, including inline function expansion for calls to functions defined within the current source file. These optimizations are a subset of full intra-file interprocedural optimizations. Note that this setting is not the same as `-no-ip` (Linux and Mac OS X) or `/Qip-` (Windows).
Description

This option determines whether additional interprocedural optimizations for single-file compilation are enabled.

Options `-ip` (Linux and Mac OS X) and `/Qip` (Windows) enable additional interprocedural optimizations for single-file compilation.

Options `-no-ip` (Linux and Mac OS X) and `/Qip-` (Windows) may not disable inlining. To ensure that inlining of user-defined functions is disabled, specify `-inline-level=0` or `-fno-inline` (Linux and Mac OS X), or specify `/Ob0` (Windows). To ensure that inlining of compiler intrinsic functions is disabled, specify `-fno-built-in` (Linux and MacOS X) or `/Oi-` (Windows).

Alternate Options

None

See Also

- `finline-functions`

ip-no-inlining, Qip-no-inlining

Disables full and partial inlining enabled by interprocedural optimization options.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

- `-ip-no-inlining`

Windows:

- `/Qip-no-inlining`
Arguments
None

Default
OFF

Inlining enabled by interprocedural optimization options is performed.

Description
This option disables full and partial inlining enabled by the following interprocedural optimization options:

- On Linux and Mac OS X systems: -ip or -ipo
- On Windows systems: /Qip, /Qipo, or /Ob2

It has no effect on other interprocedural optimizations.
On Windows systems, this option also has no effect on user-directed inlining specified by option /Ob1.

Alternate Options
None

ip-no-pinlining, Qip-no-pinlining
Disables partial inlining enabled by interprocedural optimization options.

IDEEquivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-ip-no-pinlining
Windows:
/Qip-no-pinlining

Arguments
None

Default
OFF

Inlining enabled by interprocedural optimization options is performed.

Description
This option disables partial inlining enabled by the following interprocedural optimization options:

- On Linux and Mac OS X systems: -ip or -ipo
- On Windows systems: /Qip or /Qipo

It has no effect on other interprocedural optimizations.

Alternate Options
None

IPF-flt-eval-method0, QIPF-flt-eval-method0

Tells the compiler to evaluate the expressions involving floating-point operands in the precision indicated by the variable types declared in the program. This is a deprecated option.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax

Linux:
-IPF-flt-eval-method0
Mac OS X:
None

Windows:
/QIPF-flt-eval-method0

Arguments
None

Default
OFF

Expressions involving floating-point operands are evaluated by default rules.

Description
This option tells the compiler to evaluate the expressions involving floating-point operands in the precision indicated by the variable types declared in the program.

By default, intermediate floating-point expressions are maintained in higher precision.

The recommended method to control the semantics of floating-point calculations is to use option -fp-model (Linux) or /fp (Windows).

Instead of using -IPF-flt-eval-method0 (Linux) or /QIPF-flt-eval-method0 (Windows), you can use -fp-model source (Linux) or /fp:source (Windows).

Alternate Options
None

See Also
•
•
• fp-model, fp
IPF-fltacc, QIPF-fltacc

Disables optimizations that affect floating-point accuracy. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:
-IPF-fltacc
-no-IPF-fltacc

Mac OS X:
None

Windows:
/QIPF-fltacc
/QIPF-fltacc-

Arguments

None

Default

-no-IPF-fltacc
or/QIPF-fltacc-

Optimizations are enabled that affect floating-point accuracy.

Description

This option disables optimizations that affect floating-point accuracy.

If the default setting is used, the compiler may apply optimizations that reduce floating-point accuracy.
You can use this option to improve floating-point accuracy, but at the cost of disabling some optimizations.

The recommended method to control the semantics of floating-point calculations is to use option `-fp-model` (Linux) or `/fp` (Windows).

Instead of using `-IPF-fltacc` (Linux) or `/QIPF-fltacc` (Windows), you can use `-fp-model precise` (Linux) or `/fp:precise` (Windows).

Instead of using `-no-IPF-fltacc` (Linux) or `/QIPF-fltacc-` (Windows), you can use `-fp-model fast` (Linux) or `/fp:fast` (Windows).

**Alternate Options**

None

**See Also**

- `fp-model`, `fp`

**IPF-fma, QIPF-fma**

See `fma`, `Qfma`.

**IPF-fp-relaxed, QIPF-fp-relaxed**

See `fp-relaxed`, `Qfp-relaxed`.

**ipo, Qipo**

Enables interprocedural optimization between files.

**IDE Equivalent**

Windows: Optimization > Interprocedural Optimization

Linux: Optimization > Enable Whole Program Optimization

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
-ipo[n]

Windows:
/Qipo[n]

Arguments

n  Is an optional integer that specifies the number of object files the compiler should create. The integer must be greater than or equal to 0.

Default

OFF  Multifile interprocedural optimization is not enabled.

Description

This option enables interprocedural optimization between files. This is also called multifile interprocedural optimization (multifile IPO) or Whole Program Optimization (WPO).

When you specify this option, the compiler performs inline function expansion for calls to functions defined in separate files.

You cannot specify the names for the files that are created.

If n is 0, the compiler decides whether to create one or more object files based on an estimate of the size of the application. It generates one object file for small applications, and two or more object files for large applications.

If n is greater than 0, the compiler generates n object files, unless n exceeds the number of source files (m), in which case the compiler generates only m object files.

If you do not specify n, the default is 0.

Alternate Options

None
ipo-c, Qipo-c
*Tells the compiler to optimize across multiple files and generate a single object file.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
-ipo-c

**Windows:**
/Qipo-c

**Arguments**
None

**Default**
OFF The compiler does not generate a multifile object file.

**Description**
This option tells the compiler to optimize across multiple files and generate a single object file (named ipo_out.o on Linux and Mac OS X systems; ipo_out.obj on Windows systems).
It performs the same optimizations as -ipo (Linux and Mac OS X) or /Qipo (Windows), but compilation stops before the final link stage, leaving an optimized object file that can be used in further link steps.

**Alternate Options**
None

**See Also**
•
ipo-jobs, Qipo-jobs

Specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO).

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-ipo-jobs\(n\)

Windows:
/Qipo-jobs:n

Arguments

\(n\)

Is the number of commands (jobs) to run simultaneously. The number must be greater than or equal to 1.

Default

-ipo-jobs1
or/Qipo-jobs:1

One command (job) is executed in an interprocedural optimization parallel build.

Description

This option specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO). It should only be used if the link-time compilation is generating more than one object. In this case, each object is generated by a separate compilation, which can be done in parallel.

This option can be affected by the following compiler options:
- `ipo` (Linux and Mac OS X) or `/Qipo` (Windows) when applications are large enough that the compiler decides to generate multiple object files.
- `-ipon` (Linux and Mac OS X) or `/Qipon` (Windows) when n is greater than 1.
- `-ipo-separate` (Linux) or `/Qipo-separate` (Windows)

**CAUTION.** Be careful when using this option. On a multi-processor system with lots of memory, it can speed application build time. However, if n is greater than the number of processors, or if there is not enough memory to avoid thrashing, this option can increase application build time.

**Alternate Options**
None

**See Also**

- `ipo`, `Qipo`
- `ipo-separate`, `Qipo-separate`

**ipo-S, Qipo-S**
*Tells the compiler to optimize across multiple files and generate a single assembly file.*

**IDE Equivalent**
None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `ipo-S`
**Windows:**
/Qipo-S

**Arguments**
None

**Default**
OFF The compiler does not generate a multifile assembly file.

**Description**
This option tells the compiler to optimize across multiple files and generate a single assembly file (named ipo_out.s on Linux and Mac OS X systems; ipo_out.asm on Windows systems).

It performs the same optimizations as -ipo (Linux and Mac OS X) or /Qipo (Windows), but compilation stops before the final link stage, leaving an optimized assembly file that can be used in further link steps.

**Alternate Options**
None

**See Also**

- ipo, Qipo

**ipo-separate, Qipo-separate**
*Tells the compiler to generate one object file for every source file.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures
**Syntax**

Linux:
-ipo-separate

Mac OS X:
None

Windows:
/Qipo-separate

**Arguments**
None

**Default**
The compiler decides whether to create one or more object files.

**Description**
This option tells the compiler to generate one object file for every source file. It overrides any -ipo (Linux) or /Qipo (Windows) specification.

**Alternate Options**
None

**See Also**

- ipp, Qipp

**ipp, Qipp**
Tell the compiler to link to the some or all of the Intel® Integrated Performance Primitives (Intel® IPP) libraries.

**IDE Equivalent**
Windows: None
Linux: **Performance Library Build Components > Use Intel(R) Integrated Performance Primitives Libraries**

Mac OS X: **Libraries > Use Intel(R) Integrated Performance Primitives Libraries**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-ipp[:lib]
```

Windows:

```
/Qipp[:lib]
```

**Arguments**

```
lib
```

Indicates the Intel® IPP libraries that the compiler should link to. Possible values are:

- **common**
  - Tells the compiler to link using the main libraries set. This is the default if the option is specified with no `lib`.

- **crypto**
  - Tells the compiler to link using the main libraries set and the crypto library.

- **nonpic** (Linux only)
  - Tells the compiler to link using the version of the libraries that do not have position-independent code.

- **nonpic_crypto** (Linux only)
  - Tells the compiler to link using the crypto library and the version of the libraries that do not have position-independent code.

**Default**

```
OFF
```

The compiler does not link to the Intel® IPP libraries.

**Description**

The option tells the compiler to link to the some or all of the Intel® Integrated Performance Primitives (Intel® IPP) libraries and include the Intel® IPP headers.
**Alternate Options**

None

**ivdep-parallel, Qivdep-parallel**

* Tells the compiler that there is no loop-carried memory dependency in the loop following an IVDEP pragma.

**IDE Equivalent**

Windows: None

Linux: **Optimization > IVDEP Directive Memory Dependency**

Mac OS X: None

**Architectures**

IA-64 architecture

**Syntax**

Linux:

-ivdep-parallel

Mac OS X:

None

Windows:

/Qivdep-parallel

**Arguments**

None

**Default**

OFF

There may be loop-carried memory dependency in a loop that follows an IVDEP pragma.
Description
This option tells the compiler that there is no loop-carried memory dependency in the loop following an IVDEP There may be loop-carried memory dependency in a loop that follows an IVDEP pragma.

Alternate Options
None

fkeep-static-consts, Qkeep-static-consts
_Tells the compiler to preserve allocation of variables that are not referenced in the source._

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
_Linux and Mac OS X:_
- fkeep-static-consts
- fno-keep-static-consts

_Windows:_
/Qkeep-static-consts
/Qkeep-static-consts-

Arguments
None

Default
-fno-keep-static-consts _If a variable is never referenced in a routine, the variable is discarded unless optimizations are disabled by option -O0 (Linux and Mac OS X) or /Od (Windows)._
**Description**
This option tells the compiler to preserve allocation of variables that are not referenced in the source.

The negated form can be useful when optimizations are enabled to reduce the memory usage of static data.

**Alternate Options**
None

**Qlocation**
Specifies the directory for supporting tools.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
-`-Qlocation, string, dir`

**Windows:**
-/Qlocation, string, dir

**Arguments**

`string` Is the name of the tool.
`dir` Is the directory (path) where the tool is located.

**Default**
OFF The compiler looks for tools in a default area.

**Description**
This option specifies the directory for supporting tools.
string can be any of the following:

- c - Indicates the Intel C++ compiler.
- cpp (or fpp) - Indicates the Intel C++ preprocessor.
- cxxinc - Indicates C++ header files.
- cinc - Indicates C header files.
- asm - Indicates the assembler.
- link - Indicates the linker.
- prof - Indicates the profiler.
- On Windows systems, the following is also available:
  - masm - Indicates the Microsoft assembler.
- On Linux and Mac OS X systems, the following are also available:
  - as - Indicates the assembler.
  - gas - Indicates the GNU assembler.
  - ld - Indicates the loader.
  - gld - Indicates the GNU loader.
  - lib - Indicates an additional library.
  - crt - Indicates the crt%.o files linked into executables to contain the place to start execution.

**Alternate Options**

None

**See Also**

- Qoption
**Qlong-double**

Changes the default size of the long double data type.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
None

**Windows:**
/Qlong-double

**Arguments**
None

**Default**
The default size of the long double data type is 64 bits.

**Description**
This option changes the default size of the long double data type to 80 bits.

However, the alignment requirement of the data type is 16 bytes, and its size must be a multiple of its alignment, so the size of a long double on Windows is also 16 bytes. Only the lower 10 bytes (80 bits) of the 16 byte space will have valid data stored in it.

Note that the Microsoft compiler and Microsoft-provided library routines (such as printf) do not provide support for 80-bit floating-point values. As a result, this option should only be used when referencing symbols within parts of your application built with this option or symbols in libraries that were built with this option.

**Alternate Options**
None
**M, QM**
*Tells the compiler to generate makefile dependency lines for each source file.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

```
-M
```

**Windows:**

```
/QM
```

**Arguments**
None

**Default**
OFF

The compiler does not generate makefile dependency lines for each source file.

**Description**

This option tells the compiler to generate makefile dependency lines for each source file, based on the #include lines found in the source file.

**Alternate Options**
None
map-opts, Qmap-opts
Maps one or more compiler options to their equivalent on a different operating system.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:
-map-opts
Mac OS X:
None
Windows:
/Qmap-opts

Arguments
None

Default
OFF No platform mappings are performed.

Description
This option maps one or more compiler options to their equivalent on a different operating system. The result is output to stdout.

On Windows systems, the options you provide are presumed to be Windows options, so the options that are output to stdout will be Linux equivalents.
On Linux systems, the options you provide are presumed to be Linux options, so the options that are output to stdout will be Windows equivalents.

The tool can be invoked from the compiler command line or it can be used directly.
No compilation is performed when the option mapping tool is used.
This option is useful if you have both compilers and want to convert scripts or makefiles.

**NOTE.** Compiler options are mapped to their equivalent on the architecture you are using. For example, if you are using a processor with IA-32 architecture, you will only see equivalent options that are available on processors with IA-32 architecture.

**Alternate Options**

None

**Example**

The following command line invokes the option mapping tool, which maps the Linux options to Windows-based options, and then outputs the results to stdout:

```
icc -map-opts -xP -O2
```

The following command line invokes the option mapping tool, which maps the Windows options to Linux-based options, and then outputs the results to stdout:

```
icl /Qmap-opts /QxP /O2
```

**See Also**

•

•

Building Applications: Using the Option Mapping Tool

**MD, QMD**

*Preprocess and compile, generating output file containing dependency information ending with extension .d.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
**Syntax**

**Linux and Mac OS X:**

- `MD`

**Windows:**

/`QMD`

**Arguments**

None

**Default**

OFF

The compiler does not generate dependency information.

**Description**

Preprocess and compile, generating output file containing dependency information ending with extension `.d`.

**Alternate Options**

None

**MF, QMF**

* Tells the compiler to generate makefile dependency information in a file. 

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `MFfile`
Windows:

/QMF file

**Arguments**

_file_ Is the name of the file where the makefile dependency information should be placed.

**Default**

OFF The compiler does not generate makefile dependency information in files.

**Description**

This option tells the compiler to generate makefile dependency information in a file. To use this option, you must also specify /QM or /QMM.

**Alternate Options**

None

**See Also**

- QM
- QMM

**MG, QMG**

_Tells the compiler to generate makefile dependency lines for each source file._

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:

-<MG

Windows:

/QMG

Arguments

None

Default

OFF The compiler does not generate makefile dependency information in files.

Description

This option tells the compiler to generate makefile dependency lines for each source file. It is similar to /QM, but it treats missing header files as generated files.

Alternate Options

None

See Also

•

•

• QM

mkl, Qmkl

Tells the compiler to link to certain parts of the Intel® Math Kernel Library (Intel® MKL).

IDE Equivalent

Windows: None

Linux: Performance Library Build Components > Use Intel(R) Math Kernel Library

Mac OS X: Libraries > Use Intel(R) Math Kernel Library
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-mkl [=lib]

Windows:
/Qmkl [=lib]

Arguments
lib Indicates the part of the library that the compiler should link to. Possible values are:

parallel Tells the compiler to link using the threaded part of the Intel® MKL. This is the default if the option is specified with no lib.

sequential Tells the compiler to link using the non-threaded part of the Intel® MKL.

cluster Tells the compiler to link using the cluster part and the sequential part of the Intel® MKL.

Default
OFF The compiler does not link to the Intel® MKL.

Description
This option tells the compiler to link to certain parts of the Intel® Math Kernel Library (Intel® MKL).

Alternate Options
None
**MM, QMM**
*Tells the compiler to generate makefile dependency lines for each source file.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

`-MM`

**Windows:**

`/QMM`

**Arguments**
None

**Default**

OFF The compiler does not generate makefile dependency information in files.

**Description**
This option tells the compiler to generate makefile dependency lines for each source file. It is similar to `/QM`, but it does not include system header files.

**Alternate Options**
None

**See Also**

- 
- 
- `QM`
**MMD, QMMD**

Tells the compiler to generate an output file containing dependency information.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-MMD

**Windows:**

/QMMD

**Arguments**

None

**Default**

OFF The compiler does not generate an output file containing dependency information.

**Description**

This option tells the compiler to preprocess and compile a file, then generate an output file (with extension .d) containing dependency information.

It is similar to /QMD, but it does not include system header files.

**Alternate Options**

None
**Qms**
*Tells the compiler to emulate Microsoft compatibility bugs.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*
None

*Windows:*
/\texttt{Qms\textit{n}}

**Arguments**

Possible values are:

- **0**
  Instructs the compiler to disable some Microsoft compatibility bugs. It tells the compiler to emulate the fewest number of Microsoft compatibility bugs.

- **1**
  Instructs the compiler to enable most Microsoft compatibility bugs. It tells the compiler to emulate more Microsoft compatibility bugs than /\texttt{Qms\textit{0}}.

- **2**
  Instructs the compiler to generate code that is Microsoft compatible. The compiler emulates the largest number of Microsoft compatibility bugs.

**Default**
/\texttt{Qms\textit{1}}

The compiler emulates most Microsoft compatibility bugs.
**Description**

This option tells the compiler to emulate Microsoft compatibility bugs.

---

**CAUTION.** When using `/Qms0`, your program may not compile if it depends on Microsoft headers with compatibility bugs that are disabled with this option. Use `/Qms1` if your compilation fails.

---

**Alternate Options**

None

**Qmspp**

*Enables Microsoft Visual C++ 6.0 Processor Pack binary compatibility. This is a deprecated option.*

**IDE Equivalent**

None

**Architectures**

IA-32 architecture

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

`/Qmspp`

`/Qmspp-`

**Arguments**

None

**Default**

ON  The compiler is compatible with the Microsoft Visual C++ 6.0 Processor Pack binary.
**Description**

This option enables Microsoft Visual C++ 6.0 Processor Pack binary compatibility among modules using the SIMD data types.

The `/Qmspp` option is useful when you need to maintain compatibility with binaries that were built with earlier versions of the Intel® C++ Compiler.

**Alternate Options**

None

**MT, QMT**

Changes the default target rule for dependency generation.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

`-MT` target

**Windows:**

`/QMT` target

**Arguments**

`target`  
Is the target rule to use.

**Default**

OFF  
The default target rule applies to dependency generation.

**Description**

This option changes the default target rule for dependency generation.
Alternate Options
None

**multibyte-chars, Qmultibyte-chars**
Determines whether multi-byte characters are supported.

IDE Equivalent
Windows: None
Linux: Language > Support Multibyte Characters in Source
Mac OS X: Language > Support Multibyte Characters in Source

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- -multibyte-chars
- -no-multibyte-chars

Windows:
/Qtmultibyte-chars
/Qtmultibyte-chars-

Arguments
None

Default
- -multibyte-charsor Multi-byte characters are supported.
/Qtmultibyte-chars

Description
This option determines whether multi-byte characters are supported.
Alternate Options

None

**no-bss-init, Qnobss-init**
*Tells the compiler to place in the DATA section any variables explicitly initialized with zeros.*

**IDE Equivalent**
Windows: None
Linux: **Data > Disable Placement of Zero-initialized Variables in .bss - use .data**
Mac OS X: **Data > Allocate Zero-initialized Variables to .data**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
- **-no-bss-init**

Windows:
- **/Qnobss-init**

**Arguments**
None

**Default**
OFF Variables explicitly initialized with zeros are placed in the BSS section.

**Description**
This option tells the compiler to place in the DATA section any variables explicitly initialized with zeros.

**Alternate Options**
Linux and Mac OS X: **-nobss-init (this is a deprecated option)**
Windows: None

**Qnopic**

*Disables generation of position-independent code.*

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Qnopic

**Arguments**

None

**Default**

OFF The compiler can generate position-independent code.

**Description**

This option disables generation of position-independent code.

**Alternate Options**

None

**openmp, Qopenmp**

*Enables the parallelizer to generate multi-threaded code based on the OpenMP* directives.

**IDE Equivalent**

Windows: **Language > OpenMP* Support**
Linux: **Language > Process OpenMP Directives**  
Mac OS X: **Language > Process OpenMP Directives**  

**Architectures**  
IA-32, Intel® 64, IA-64 architectures  

**Syntax**  
Linux and Mac OS X:  
- `openmp`  

Windows:  
/`openmp`  

**Arguments**  
None  

**Default**  
OFF  
No OpenMP multi-threaded code is generated by the compiler.  

**Description**  
This option enables the parallelizer to generate multi-threaded code based on the OpenMP* directives. The code can be executed in parallel on both uniprocessor and multiprocessor systems.  
This option works with any optimization level. Specifying no optimization (`-00` on Linux or `/Od` on Windows) helps to debug OpenMP applications.  

**NOTE.** On Mac OS X systems, when you enable OpenMP*, you must also set the DYLD_LIBRARY_PATH environment variable within Xcode or an error will be displayed.  

**Alternate Options**  
None  

**See Also**  
•
openmp-stubs, Qopenmp-stubs

openmp-lib, Qopenmp-lib

 Lets you specify an OpenMP* run-time library to use for linking.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux:

-oopenmp-lib type

Mac OS X:

None

Windows:

/Qopenmp-lib:type

Arguments

type

Specifies the type of library to use; it implies compatibility levels. Possible values are:

legacy

Tells the compiler to use the legacy OpenMP* run-time library (libguide). This setting does not provide compatibility with object files created using other compilers. This is a deprecated option.

compat

Tells the compiler to use the compatibility OpenMP* run-time library (libiomp). This setting provides compatibility with object files created using Microsoft* and GNU* compilers.
Default

-openmp-lib compat
or /Qopenmp-lib:compat

The compiler uses the compatibility OpenMP* run-time library (libiomp).

Description

This option lets you specify an OpenMP* run-time library to use for linking.

The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers.

The compatibility OpenMP run-time library is compatible with object files created using the Microsoft* OpenMP run-time library (vcomp) and GNU OpenMP run-time library (libgomp).

To use the compatibility OpenMP run-time library, compile and link your application using the

-openmp-lib compat (Linux) or /Qopenmp-lib:compat (Windows) option. To use this option, you must also specify one of the following compiler options:

- Linux OS: -openmp, -openmp-profile, or -openmp-stubs
- Windows OS: /Qopenmp, /Qopenmp-profile, or /Qopenmp-stubs

On Windows* systems, the compatibility OpenMP* run-time library lets you combine OpenMP* object files compiled with the Microsoft* C/C++ compiler with OpenMP* object files compiled with the Intel C/C++ or Fortran compilers. The linking phase results in a single, coherent copy of the run-time library.

On Linux* systems, the compatibility Intel OpenMP* run-time library lets you combine OpenMP* object files compiled with the GNU* gcc or gfortran compilers with similar OpenMP* object files compiled with the Intel C/C++ or Fortran compilers. The linking phase results in a single, coherent copy of the run-time library.

NOTE. The compatibility OpenMP run-time library is not compatible with object files created using versions of the Intel compiler earlier than 10.0.

Alternate Options

None

See Also

•
openmp, Qopenmp
openmp-stubs, Qopenmp-stubs
openmp-profile, Qopenmp-profile

openmp-link, Qopenmp-link

Controls whether the compiler links to static or dynamic OpenMP run-time libraries.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-openmp-link library

Windows:
/Qopenmp-link:library

Arguments

library

Specifies the OpenMP library to use. Possible values are:

static
Tells the compiler to link to static OpenMP run-time libraries.
dynamic
Tells the compiler to link to dynamic OpenMP run-time libraries.

Default

-openmp-link dynamic or /Qopenmp-link:dynamic

The compiler links to dynamic OpenMP run-time libraries. However, if option static is specified, the compiler links to static OpenMP run-time libraries.
**Description**

This option controls whether the compiler links to static or dynamic OpenMP run-time libraries.

To link to the static OpenMP run-time library (RTL) and create a purely static executable, you must specify `-openmp-link static` (Linux and Mac OS X) or `/Qopenmp-link:static` (Windows). However, we strongly recommend you use the default setting, `-openmp-link dynamic` (Linux and Mac OS X) or `/Qopenmp-link:dyncamic` (Windows).

**NOTE.** Compiler options `-static-intel` and `-shared-intel` (Linux and Mac OS X) have no effect on which OpenMP run-time library is linked.

**Alternate Options**

None

**openmp-profile, Qopenmp-profile**

*Enables analysis of OpenMP* applications if Intel® Thread Profiler is installed.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**

- `-openmp-profile`

**Mac OS X:**

None

**Windows:**

- `/Qopenmp-profile`
Arguments
None

Default
OFF OpenMP applications are not analyzed.

Description
This option enables analysis of OpenMP* applications. To use this option, you must have previously installed Intel® Thread Profiler, which is one of the Intel® Threading Analysis Tools.
This option can adversely affect performance because of the additional profiling and error checking invoked to enable compatibility with the threading tools. Do not use this option unless you plan to use the Intel® Thread Profiler.
For more information about Intel® Thread Profiler, open the page associated with threading tools at Intel® Software Development Products.

Alternate Options
None

openmp-report, Qopenmp-report
Controls the OpenMP* parallelizer’s level of diagnostic messages.

IDE Equivalent
Windows: None
Linux: Compilation Diagnostics > OpenMP Report
Mac OS X: Diagnostics > OpenMP Report

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-openmp-report[n]
Windows:
/Qopenmp-report[n]

Arguments

Is the level of diagnostic messages to display. Possible values are:

0  No diagnostic messages are displayed.
1  Diagnostic messages are displayed indicating loops, regions, and sections successfully parallelized.
2  The same diagnostic messages are displayed as specified by openmp_report1 plus diagnostic messages indicating successful handling of MASTER constructs, SINGLE constructs, CRITICAL constructs, ORDERED constructs, ATOMIC directives, and so forth.

Default

If you do not specify \( n \), the compiler displays diagnostic messages indicating loops, regions, and sections successfully parallelized. If you do not specify the option on the command line, the default is to display no messages.

Description

This option controls the OpenMP* parallelizer's level of diagnostic messages. To use this option, you must also specify -openmp (Linux and Mac OS X) or /Qopenmp (Windows).

If this option is specified on the command line, the report is sent to stdout.

Alternate Options

None

See Also
openmp, Qopenmp

Optimizing Applications:
Using Parallelism
OpenMP* Report

openmp-stubs, Qopenmp-stubs

Enables compilation of OpenMP programs in sequential mode.

IDE Equivalent

Windows: Language > Process OpenMP Directives
Linux: Language > Process OpenMP Directives
Mac OS X: Language > Process OpenMP Directives

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- openmp-stubs

Windows:
/Qopenmp-stubs

Arguments
None

Default
OFF

The library of OpenMP function stubs is not linked.

Description

This option enables compilation of OpenMP programs in sequential mode. The OpenMP directives are ignored and a stub OpenMP library is linked.
Alternate Options

None

See Also

•
  •
  • `openmp`, `Qopenmp`

`openmp-task`, `Qopenmp-task`

*Lets you choose an OpenMP* tasking model.*

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

`-openmp-task model`

Windows:

`/Qopenmp-task:model`

Arguments

`model`  

Is an OpenMP tasking model. Possible values are:

`intel`  

Tells the compiler to accept Intel® taskqueuing pragmas (`#pragma intel_omp_taskq` and `#pragma intel_omp_task`). When this value is specified, OpenMP 3.0 tasking pragmas are ignored; if they are specified, warnings are issued.
Tells the compiler to accept OpenMP 3.0 tasking pragmas (#pragma omp_task).
When this value is specified, Intel taskqueueing pragmas are ignored; if they are specified, warnings are issued.

**Default**

- `-openmp-task omp` or `/Qopenmp-task:omp`

**Description**
The option lets you choose an OpenMP tasking model.
To use this option, you must also specify option `-openmp` (Linux and Mac OS X) or `/Qopenmp` (Windows).

**Alternate Options**
None

**openmp-threadprivate, Qopenmp-threadprivate**

*Let you specify an OpenMP* threadprivate *implementation.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**
- `-openmp-threadprivate type`

**Mac OS X:**
None
Windows:
/Qopenmp-threadprivate:type

Arguments

type Specifies the type of threadprivate implementation. Possible values are:

  legacy Tells the compiler to use the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel® compiler. This setting does not provide compatibility with the implementation used by other compilers.

  compat Tells the compiler to use the compatibility OpenMP* threadprivate implementation based on applying the __declspec(thread) attribute to each threadprivate variable. The limitations of the attribute on a given platform also apply to the threadprivate implementation. This setting provides compatibility with the implementation provided by the Microsoft* and GNU* compilers.

Default

-openmp-threadprivate legacy The compiler uses the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel® compiler.

or/Qopenmp-threadprivate:legacy

Description

This option lets you specify an OpenMP* threadprivate implementation.

The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers.

To use this option, you must also specify one of the following compiler options:

- Linux OS: -openmp, -openmp-profile, or -openmp-stubs
- Windows OS: /Qopenmp, /Qopenmp-profile, or /Qopenmp-stubs
The value specified for this option is independent of the value used for option -openmp-lib (Linux) or /Qopenmp-lib (Windows).

**Alternate Options**
None

**opt-block-factor, Qopt-block-factor**
*Let you specify a loop blocking factor.*

**IDE Equivalent**
Windows: Diagnostics > Optimization Diagnostic File
Diagnostics > Emit Optimization Diagnostics to File
Linux: None
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- opt-block-factor=n

**Windows:**
/ Qopt-block-factor:n

**Arguments**

\( n \)
Is the blocking factor. It must be an integer. The compiler may ignore the blocking factor if the value is 0 or 1.

**Default**
OFF
The compiler uses default heuristics for loop blocking.

**Description**
This option lets you specify a loop blocking factor.
Alternate Options
None

**opt-class-analysis, Qopt-class-analysis**

Determines whether C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
- -opt-class-analysis
- -no-opt-class-analysis

**Windows:**
/ Qopt-class-analysis
/ Qopt-class-analysis-

Arguments
None

Default
- -no-opt-class-analysis C++ class hierarchy information is not used to analyze and resolve
or / Qopt-class-analysis-C++ virtual function calls at compile time.

Description
This option determines whether C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time. The option is turned on by default with the -ipo compiler option, enabling improved C++ optimization. If a C++ application contains non-standard C++ constructs, such as pointer down-casting, it may result in different behaviors.
Alternate Options

None

**opt-jump-tables, Qopt-jump-tables**

*Enables or disables generation of jump tables for switch statements.*

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

- `opt-jump-tables=keyword`
- `no-opt-jump-tables`

**Windows:**

/`Qopt-jump-tables:keyword`

/`Qopt-jump-tables-`

Arguments

*keyword* is the instruction for generating jump tables. Possible values are:

- `never` tells the compiler to never generate jump tables. All switch statements are implemented as chains of if-then-elses. This is the same as specifying `-no-opt-jump-tables` (Linux and Mac OS) or `/Qopt-jump-tables-` (Windows).

- `default` tells the compiler to use default heuristics to determine when to generate jump tables.
large

Tells the compiler to generate jump tables up to a certain pre-defined size (64K entries).

n

Must be an integer. Tells the compiler to generate jump tables up to \( n \) entries in size.

**Default**

- `-opt-jump-tables=default`
- `or/Qopt-jump-tables:default`

**Description**

This option enables or disables generation of jump tables for switch statements. When the option is enabled, it may improve performance for programs with large switch statements.

**Alternate Options**

None

`opt-loadpair`, `Qopt-loadpair`

*Enables or disables loadpair optimization.*

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

- **Linux:**
  - `-opt-loadpair`
  - `-no-opt-loadpair`
Mac OS X:
None

Windows:
/Qopt-loadpair
/Qopt-loadpair-

Arguments
None

Default
-no-opt-loadpair Loadpair optimization is disabled unless option O3 is specified.
or/Qopt-loadpair-

Description
This option enables or disables loadpair optimization.
When -O3 is specified on IA-64 architecture, loadpair optimization is enabled by default. To disable loadpair generation, specify -no-opt-loadpair (Linux) or /Qopt-loadpair- (Windows).

Alternate Options
None

opt-mem-bandwidth, Qopt-mem-bandwidth
Enables performance tuning and heuristics that control memory bandwidth use among processors.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux:
-`opt-mem-bandwidth`:
Mac OS X:
None

Windows:
/Qopt-mem-bandwidth\n
Arguments

\( n \) Is the level of optimizing for memory bandwidth usage.
Possible values are:

- 0: Enables a set of performance tuning and heuristics in compiler optimizations that is optimal for serial code.
- 1: Enables a set of performance tuning and heuristics in compiler optimizations for multithreaded code generated by the compiler.
- 2: Enables a set of performance tuning and heuristics in compiler optimizations for parallel code such as Windows Threads, pthreads, and MPI code, besides multithreaded code generated by the compiler.

Default

- opt-mem-bandwidth0 or /Qopt-mem-bandwidth0
- opt-mem-bandwidth1 or /Qopt-mem-bandwidth1

For serial (non-parallel) compilation, a set of performance tuning and heuristics in compiler optimizations is enabled that is optimal for serial code.

If you specify compiler option -parallel (Linux) or /Qparallel (Windows), or -openmp (Linux) or /Qopenmp (Windows), a set of performance tuning and heuristics in compiler optimizations for multithreaded code generated by the compiler is enabled.
Description

This option enables performance tuning and heuristics that control memory bandwidth use among processors. It allows the compiler to be less aggressive with optimizations that might consume more bandwidth, so that the bandwidth can be well-shared among multiple processors for a parallel program.

For values of $n$ greater than 0, the option tells the compiler to enable a set of performance tuning and heuristics in compiler optimizations such as prefetching, privatization, aggressive code motion, and so forth, for reducing memory bandwidth pressure and balancing memory bandwidth traffic among threads.

This option can improve performance for threaded or parallel applications on multiprocessors or multicore processors, especially when the applications are bounded by memory bandwidth.

Alternate Options
None

See Also

- parallel, Qparallel
- openmp, Qopenmp

**opt-mod-versioning, Qopt-mod-versioning**

Enables or disables versioning of modulo operations for certain types of operands.

IDE Equivalent
None

Architectures

IA-64 architecture

Syntax

Linux:

- -opt-mod-versioning
- -no-opt-mod-versioning
Mac OS X:
None

Windows:
/Qopt-mod-versioning
/Qopt-mod-versioning-

Arguments
None

Default
-no-opt-mod-versioning Versioning of modulo operations is disabled.
or/Qopt-mod-versioning-

Description
This option enables or disables versioning of modulo operations for certain types of operands. It is used for optimization tuning.

Versioning of modulo operations may improve performance for x mod y when modulus y is a power of 2.

Alternate Options
None

opt-multi-version-aggressive, Qopt-multi-version-aggressive
Tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures
Syntax

**Linux and Mac OS X:**
- `opt-multi-version-aggressive`
- `no-opt-multi-version-aggressive`

**Windows:**
/`opt-multi-version-aggressive`
/`opt-multi-version-aggressive-`

Arguments

None

Default

`-no-opt-multi-version-aggressive`

or
/`opt-multi-version-aggressive-`

Description

This option tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement. This option may improve performance.

Alternate Options

None

**opt-prefetch, Qopt-prefetch**

*Enables or disables prefetch insertion optimization.*

IDE Equivalent

Windows: None

Linux: **Optimization > Enable Prefetch Insertion**

Mac OS X: **Optimization > Enable Prefetch Insertion**
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- `opt-prefetch[=n]`
- `no-opt-prefetch`

Windows:
/`Qopt-prefetch[:n]`
/`Qopt-prefetch-`

Arguments

`n` is the level of detail in the report. Possible values are:

- `0` Disables software prefetching. This is the same as specifying `-no-opt-prefetch` (Linux and Mac OS X) or `/Qopt-prefetch-` (Windows).
- `1` to `4` Enables different levels of software prefetching. If you do not specify a value for `n`, the default is 2 on IA-32 and Intel® 64 architecture; the default is 3 on IA-64 architecture. Use lower values to reduce the amount of prefetching.

Default

IA-64 architecture: `-opt-prefetch` On IA-64 architecture, prefetch insertion optimization is enabled.
/noopt-prefetch
or/`Qopt-prefetch`

IA-32 architecture and Intel® On IA-32 architecture and Intel® 64 architecture, prefetch insertion 64 architecture: optimization is disabled.
-no-opt-prefetch
or/`Qopt-prefetch-`
Description

This option enables or disables prefetch insertion optimization. The goal of prefetching is to reduce cache misses by providing hints to the processor about when data should be loaded into the cache.

On IA-64 architecture, this option is enabled by default if you specify option O1 or higher. To disable prefetching at these optimization levels, specify -no-opt-prefetch (Linux and Mac OS X) or /Qopt-prefetch- (Windows).

On IA-32 architecture and Intel® 64 architecture, this option enables prefetching when higher optimization levels are specified.

Alternate Options

Linux and Mac OS X: -prefetch (this is a deprecated option)

Windows: /Qprefetch (this is a deprecated option)

**opt-prefetch-initial-values, Qopt-prefetch-initial-values**

Enables or disables prefetches that are issued before a loop is entered.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:

- opt-prefetch-initial-values
- no-opt-prefetch-initial-values

Mac OS X:

None

Windows:

/Qopt-prefetch-initial-values
/Qopt-prefetch-initial-values-
Arguments
None

Default
- `opt-prefetch-initial-values` or `/Qopt-prefetch-initial-values`

Description
This option enables or disables prefetches that are issued before a loop is entered. These prefetches target the initial iterations of the loop.

When `-O1` or higher is specified on IA-64 architecture, prefetches are issued before a loop is entered. To disable these prefetches, specify `-no-opt-prefetch-initial-values (Linux)` or `/Qopt-prefetch-initial-values- (Windows)`.

Alternate Options
None

`opt-prefetch-issue-excl-hint, Qopt-prefetch-issue-excl-hint`

Determines whether the compiler issues prefetches for stores with exclusive hint.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux:
- `opt-prefetch-issue-excl-hint`
- `no-opt-prefetch-issue-excl-hint`
Mac OS X: None

Windows: 
/Qopt-prefetch-issue-excl-hint
/Qopt-prefetch-issue-excl-hint-

Arguments
None

Default
-no-opt-prefetch-issue- The compiler does not issue prefetches for stores with exclusive hint.
or/Qopt-prefetch-issue-
excl-hint-

Description
This option determines whether the compiler issues prefetches for stores with exclusive hint. If option -opt-prefetch-issue-excl-hint (Linux) or /Qopt-prefetch-issue-excl-hint (Windows) is specified, the prefetches will be issued if the compiler determines it is beneficial to do so.

When prefetches are issued for stores with exclusive-hint, the cache-line is in "exclusive-mode". This saves on cache-coherence traffic when other processors try to access the same cache-line. This feature can improve performance tuning.

Alternate Options
None

opt-prefetch-next-iteration, Qopt-prefetch-next-iteration
Enables or disables prefetches for a memory access in the next iteration of a loop.

IDE Equivalent
None
Architectures
IA-64 architecture

Syntax
Linux:
-opt-prefetch-next-iteration
-no-opt-prefetch-next-iteration

Mac OS X:
None

Windows:
/Qopt-prefetch-next-iteration
/Qopt-prefetch-next-iteration-

Arguments
None

Default
-opt-prefetch-next-iteration - Prefetches are issued for a memory access in the next iteration of a loop.
-or/Qopt-prefetch-next-
iteration

Description
This option enables or disables prefetches for a memory access in the next iteration of a loop. It is typically used in a pointer-chasing loop.

When -O1 or higher is specified on IA-64 architecture, prefetches are issued for a memory access in the next iteration of a loop. To disable these prefetches, specify -no-opt-prefetch-next-iteration (Linux) or /Qopt-prefetch-next-iteration- (Windows).

Alternate Options
None
**opt-ra-region-strategy, Qopt-ra-region-strategy**

Selects the method that the register allocator uses to partition each routine into regions.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

- `opt-ra-region-strategy[=keyword]`

**Windows:**

/`Qopt-ra-region-strategy[:keyword]`

**Arguments**

- `keyword`  
  Is the method used for partitioning. Possible values are:
  - `routine`  
    Creates a single region for each routine.
  - `block`  
    Partitions each routine into one region per basic block.
  - `trace`  
    Partitions each routine into one region per trace.
  - `region`  
    Partitions each routine into one region per loop.
  - `default`  
    The compiler determines which method is used for partitioning.

**Default**

- `opt-ra-region-strategy=default`
  The compiler determines which method is used for partitioning.
  This is also the default if `keyword` is not specified.
Description
This option selects the method that the register allocator uses to partition each routine into regions.

When setting default is in effect, the compiler attempts to optimize the tradeoff between compile-time performance and generated code performance.

This option is only relevant when optimizations are enabled (O1 or higher).

Alternate Options
None

See Also
•
•
•  o

opt-report, Qopt-report
Tells the compiler to generate an optimization report to stderr.

IDE Equivalent
Windows: Diagnostics > Optimization Diagnostic Level
Linux: Compilation Diagnostics > Optimization Diagnostic Level
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-opt-report [n]

Windows:
/Qopt-report[:n]
Arguments

\( n \)

Is the level of detail in the report. On Linux OS and Mac OS X systems, a space must appear before the \( n \). Possible values are:

- 0: Tells the compiler to generate no optimization report.
- 1: Tells the compiler to generate a report with the minimum level of detail.
- 2: Tells the compiler to generate a report with the medium level of detail.
- 3: Tells the compiler to generate a report with the maximum level of detail.

Default

-\opt-report 2 or /Qopt-report:2

If you do not specify \( n \), the compiler generates a report with medium detail. If you do not specify the option on the command line, the compiler does not generate an optimization report.

Description

This option tells the compiler to generate an optimization report to `stderr`.

Alternate Options

None

See Also

- opt-report-file, /Qopt-report-file

Optimizing Applications: Optimizer Report Generation
**opt-report-file, Qopt-report-file**

Specifies the name for an optimization report.

**IDE Equivalent**

Windows: Diagnostics > Optimization Diagnostic File

Diagnostics > Emit Optimization Diagnostics to File

Linux: Compilation Diagnostics > Emit Optimization Diagnostics to File

Compilation Diagnostics > Optimization Diagnostics File

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `opt-report-file=file`

Windows:

/Qopt-report-file=file

**Arguments**

*file* Is the name for the optimization report.

**Default**

OFF No optimization report is generated.

**Description**

This option specifies the name for an optimization report. If you use this option, you do not have to specify `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**Alternate Options**

None
See Also

- opt-report, Qopt-report

Optimizing Applications: Optimizer Report Generation

**opt-report-help, Qopt-report-help**

*Displays the optimizer phases available for report generation.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `opt-report-help`

**Windows:**

`/Qopt-report-help`

**Arguments**

None

**Default**

OFF

No optimization reports are generated.

**Description**

This option displays the optimizer phases available for report generation using `opt-report-phase` (Linux and Mac OS X) or `/Qopt-report-phase` (Windows). No compilation is performed.

**Alternate Options**

None
opt-report-phase, Qopt-report-phase

Specifies an optimizer phase to use when optimization reports are generated.

IDE Equivalent

Windows: Diagnostics > Optimization Diagnostic Phase
Linux: Compilation Diagnostics > Optimization Diagnostic Phase
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
- opt-report-phase=phase

Windows:
/Qopt-report-phase:phase

Arguments

phase  Is the phase to generate reports for. Some of the possible values are:

ipo  The Interprocedural Optimizer phase
hlo  The High Level Optimizer phase
hpo  The High Performance Optimizer phase
ilo  The Intermediate Language Scalar Optimizer phase
ecg  The Code Generator phase (Windows and Linux systems using IA-64 architecture only)
ecg_swpl  The software pipelining component of the Code Generator phase (Windows and Linux systems using IA-64 architecture only)
pgo  The Profile Guided Optimization phase
all  All optimizer phases

**Default**
OFF  No optimization reports are generated.

**Description**
This option specifies an optimizer phase to use when optimization reports are generated. To use this option, you must also specify -opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

This option can be used multiple times on the same command line to generate reports for multiple optimizer phases.

When one of the logical names for optimizer phases is specified for phase, all reports from that optimizer phase are generated.

To find all phase possibilities, use option -opt-report-help (Linux and Mac OS X) or /Qopt-report-help (Windows).

**Alternate Options**
None

**See Also**
- opt-report, Qopt-report
opt-report-routine, Qopt-report-routine

*Tells the compiler to generate reports on the routines containing specified text.*

**IDE Equivalent**

Windows: Diagnostics > Optimization Diagnostic Routine

Linux: [Compilation Diagnostics > Optimization Diagnostic Routine](#)

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-opt-report-routine=string
```

Windows:

```
/Qopt-report-routine:string
```

**Arguments**

`string` 

*Is the text (string) to look for.*

**Default**

OFF 

*No optimization reports are generated.*

**Description**

This option tells the compiler to generate reports on the routines containing specified text as part of their name.

**Alternate Options**

None

**See Also**

*
opt-report, Qopt-report

**opt-streaming-stores, Qopt-streaming-stores**

*Enables generation of streaming stores for optimization.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

*Linux and Mac OS X:*

```bash
-opt-streaming-stores keyword
```

*Windows:*

```bash
/Qopt-streaming-stores:keyword
```

**Arguments**

*keyword*

Specifies whether streaming stores are generated. Possible values are:

- **always**
  - Enables generation of streaming stores for optimization. The compiler optimizes under the assumption that the application is memory bound.

- **never**
  - Disables generation of streaming stores for optimization. Normal stores are performed.

- **auto**
  - Lets the compiler decide which instructions to use.
**Default**

- **-opt-streaming-stores auto** or **/Qopt-streaming-stores:auto**

**Description**

The compiler decides whether to use streaming stores or normal stores.

This option enables generation of streaming stores for optimization. This method stores data with instructions that use a non-temporal buffer, which minimizes memory hierarchy pollution.

For this option to be effective, the compiler must be able to generate SSE2 (or higher) instructions. For more information, see compiler option x or ax.

This option may be useful for applications that can benefit from streaming stores.

**Alternate Options**

None

**See Also**

- **ax, Qax**
- **x, Qx**
- **opt-mem-bandwidth, Qopt-mem-bandwidth, Qx**

**opt-subscript-in-range, Qopt-subscript-in-range**

Determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures
Syntax

Linux and Mac OS X:
- `opt-subscript-in-range`
- `no-opt-subscript-in-range`

Windows:
/`opt-subscript-in-range`
/`opt-subscript-in-range-`

Arguments
None

Default
`-no-opt-subscript-in-range` or /`opt-subscript-in-range-`

Description
This option determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.

If you specify `-opt-subscript-in-range` (Linux and Mac OS X) or /`opt-subscript-in-range` (Windows), the compiler ignores any data type conversions used and it assumes no overflows in the intermediate computation of subscript expressions. This feature can enable more loop transformations.

Alternate Options
None

Example
The following shows an example where these options can be useful. `m` is declared as type long (64-bits) and all other variables inside the subscript are declared as type int (32-bits):

```
A[ i + j + ( n + k) * m ]
```
**Qoption**

*Passes options to a specified tool.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-Qoption,string,options

**Windows:**

/Qoption,string,options

**Arguments**

- **string**
  - Is the name of the tool.

- **options**
  - Are one or more comma-separated, valid options for the designated tool.

**Default**

OFF

No options are passed to tools.

**Description**

This option passes options to a specified tool.

If an argument contains a space or tab character, you must enclose the entire argument in quotation marks (" "). You must separate multiple arguments with commas.

**string** can be any of the following:

- **asm** - Indicates the assembler.
- **link** - Indicates the linker.
- **prof** - Indicates the profiler.
- **On Windows systems, the following is also available:**
- masm - Indicates the Microsoft assembler.

- On Linux and Mac OS X systems, the following are also available:
  - as - Indicates the assembler.
  - gas - Indicates the GNU assembler.
  - ld - Indicates the loader.
  - gld - Indicates the GNU loader.
  - lib - Indicates an additional library.
  - crt - Indicates the crt%.o files linked into executables to contain the place to start execution.

**Alternate Options**

None

**See Also**

- `Qlocation`

**qp**

*See p.*

**Qpar-adjust-stack**

*Tells the compiler to generate code to adjust the stack size for a fiber-based main thread.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

None
Windows:
/Qpar-adjust-stack:n

Arguments

n Is the stack size (in bytes) for the fiber-based main thread. It must be a number equal to or greater than zero.

Default

/Qpar-adjust-stack:0 No adjustment is made to the main thread stack size.

Description

This option tells the compiler to generate code to adjust the stack size for a fiber-based main thread. This can reduce the stack size of threads.

For this option to be effective, you must also specify option /Qparallel.

Alternate Options

None

See Also

•
  • parallel, Qparallel

par-affinity, Qpar-affinity

Specifies thread affinity.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux:
-par-affinity=[modifier,...]type[,permute][,offset]
Mac OS X:
None

Windows:
/Qpar-affinity: [modifier,...] type[,permute][,offset]

Arguments

modifier
Is one of the following values:
granularity={fine|thread|core}, [no]respect,
[no]verbose,[no]warnings, proclist=proc_list. The
default is granularity=core, respect, and noverbose.
For information on value proclist, see Thread Affinity
Interface in Optimizing Applications.

type
Indicates the thread affinity. This argument is required and
must be one of the following values: compact, disabled,
explicit, none, scatter, logical, physical. The default
is none. Values logical and physical are deprecated.
Use compact and scatter, respectively, with no permute
value.

permute
Is a positive integer. You cannot use this argument with
type setting explicit, none, or disabled. The default is
0.

offset
Is a positive integer. You cannot use this argument with
type setting explicit, none, or disabled. The default is
0.

Default
OFF

The thread affinity is determined by the run-time environment.

Description

This option specifies thread affinity, which binds threads to physical processing units. It has
the same effect as environment variable KMP_AFFINITY.

This option overrides the environment variable when both are specified.

This option only has an effect if the following is true:

• Linux* OS: You have specified option -parallel or -openmp (or both).
Windows* OS: You have specified option \texttt{/Qparallel} or \texttt{/Qopenmp} (or both).

- You are compiling the main program.

**Alternate Options**

None

**See Also**

- 
- 
- Thread Affinity Interface

\texttt{par-num-threads, Qpar-num-threads}

*\textit{Specifies the number of threads to use in a parallel region.}*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel\textsuperscript{®} 64, IA-64 architectures

**Syntax**

\textbf{Linux and Mac OS X:}

- \texttt{-par-num-threads=n}

\textbf{Windows:}

- \texttt{/Qpar-num-threads:n}

**Arguments**

\begin{itemize}
  \item \textit{n} \hspace{2cm} Is the number of threads to use. It must be a positive integer.
\end{itemize}

**Default**

- \texttt{OFF} \hspace{2cm} The number of threads to use is determined by the run-time environment.
Description
This option specifies the number of threads to use in a parallel region. It has the same effect as environment variable OMP_NUM_THREADS.
This option overrides the environment variable when both are specified.
This option only has an effect if the following is true:

- Linux* OS and Mac OS* X: You have specified option -parallel or -openmp (or both).
  Windows* OS: You have specified option /Qparallel or /Qopenmp (or both).
- You are compiling the main program.

Alternate Options
None

par-report, Qpar-report
Controls the diagnostic information reported by the auto-parallelizer.

IDE Equivalent
Windows: None
Linux: Compilation Diagnostics > Auto-Parallelizer Report
Mac OS X: Diagnostics > Auto-Parallelizer Report

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
-par-report[n]

**Windows:**
/Qpar-report[n]
Arguments

\(n\)  
Is a value denoting which diagnostic messages to report. Possible values are:

0  
Tells the auto-parallelizer to report no diagnostic information.

1  
Tells the auto-parallelizer to report diagnostic messages for loops successfully auto-parallelized. The compiler also issues a "LOOP AUTO-PARALLELIZED" message for parallel loops.

2  
Tells the auto-parallelizer to report diagnostic messages for loops successfully and unsuccessfully auto-parallelized.

3  
Tells the auto-parallelizer to report the same diagnostic messages specified by 2 plus additional information about any proven or assumed dependencies inhibiting auto-parallelization (reasons for not parallelizing).

Default

-\texttt{par-report1} or /\texttt{Qpar-report1}  
If you do not specify \(n\), the compiler displays diagnostic messages for loops successfully auto-parallelized. If you do not specify the option on the command line, the default is to display no messages.

Description

This option controls the diagnostic information reported by the auto-parallelizer (parallel optimizer). To use this option, you must also specify -\texttt{parallel} (Linux and Mac OS X) or /\texttt{Qparallel} (Windows).

If this option is specified on the command line, the report is sent to stdout.

Alternate Options

None
**par-runtime-control, Qpar-runtime-control**

Generates code to perform run-time checks for loops that have symbolic loop bounds.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `par-runtime-control`
- `no-par-runtime-control`

**Windows:**
- `/Qpar-runtime-control`
- `/Qpar-runtime-control-`

**Arguments**

None

**Default**

`-no-par-runtime-control` The compiler uses default heuristics when checking loops.
`/Qpar-runtime-control`

**Description**

This option generates code to perform run-time checks for loops that have symbolic loop bounds. If the granularity of a loop is greater than the parallelization threshold, the loop will be executed in parallel.

If you do not specify this option, the compiler may not parallelize loops with symbolic loop bounds if the compile-time granularity estimation of a loop can not ensure it is beneficial to parallelize the loop.
Alternate Options
None

**par-schedule, Qpar-schedule**

*Lets you specify a scheduling algorithm or a tuning method for loop iterations.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

- **par-schedule:**
  - `keyword[=n]`

**Windows:**

- **Qpar-schedule:**
  - `[[::]n]`

Arguments

*keyword*

Specifies the scheduling algorithm or tuning method. Possible values are:

- **auto**: Lets the compiler or run-time system determine the scheduling algorithm.
- **static**: Divides iterations into contiguous pieces.
- **static-balanced**: Divides iterations into even-sized chunks.
- **static-steal**: Divides iterations into even-sized chunks, but allows threads to steal parts of chunks from neighboring threads.
- **dynamic**: Gets a set of iterations dynamically.
- **guided**: Specifies a minimum number of iterations.
- **guided-analytical**: Divides iterations by using exponential distribution or dynamic distribution.
runtime  Defers the scheduling decision until run

time.

\( n \)

Is the size of the chunk or the number of iterations for each
chunk. This setting can only be specified for static, dynamic,
and guided. For more information, see the descriptions of
each keyword below.

**Default**

static-balanced  Iterations are divided into even-sized chunks and the chunks are
assigned to the threads in the team in a round-robin fashion in
the order of the thread number.

**Description**

This option lets you specify a scheduling algorithm or a tuning method for loop iterations. It
specifies how iterations are to be divided among the threads of the team.

This option affects performance tuning and can provide better performance during
auto-parallelization.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-par-schedule-auto or /Qpar-schedule-auto</td>
<td>Lets the compiler or run-time system determine the scheduling algorithm. Any possible mapping may occur for iterations to threads in the team.</td>
</tr>
<tr>
<td>-par-schedule-static or /Qpar-sched-ule-static</td>
<td>Divides iterations into contiguous pieces (chunks) of size ( n ). The chunks are assigned to threads in the team in a round-robin fashion in the order of the thread number. Note that the last chunk to be assigned may have a smaller number of iterations. If no ( n ) is specified, the iteration space is divided into chunks that are approximately equal in size, and each thread is assigned at most one chunk.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-par-schedule-static-balanced</code> or <code>/Qpar-schedule-static-balanced</code></td>
<td>Divides iterations into even-sized chunks. The chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.</td>
</tr>
<tr>
<td><code>-par-schedule-static-steal</code> or <code>/Qpar-schedule-static-steal</code></td>
<td>Divides iterations into even-sized chunks, but when a thread completes its chunk, it can steal parts of chunks assigned to neighboring threads. Each thread keeps track of L and U, which represent the lower and upper bounds of its chunks respectively. Iterations are executed starting from the lower bound, and simultaneously, L is updated to represent the new lower bound.</td>
</tr>
<tr>
<td><code>-par-schedule-dynamic</code> or <code>/Qpar-schedule-dynamic</code></td>
<td>Can be used to get a set of iterations dynamically. Assigns iterations to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests another chunk, until no chunks remain to be assigned. As each thread finishes a piece of the iteration space, it dynamically gets the next set of iterations. Each chunk contains ( n ) iterations, except for the last chunk to be assigned, which may have fewer iterations. If no ( n ) is specified, the default is 1.</td>
</tr>
<tr>
<td><code>-par-schedule-guided</code> or <code>/Qpar-schedule-guided</code></td>
<td>Can be used to specify a minimum number of iterations. Assigns iterations to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests another chunk, until no chunks remain to be assigned.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-par-schedule-guided-analytical or /Qpar-schedule-guided-analytical</td>
<td>Divides iterations by using exponential distribution or dynamic distribution. The method depends on run-time implementation. Loop bounds are calculated with faster synchronization and chunks are dynamically dispatched at run time by threads in the team.</td>
</tr>
<tr>
<td>-par-schedule-runtime or /Qpar-schedule-runtime</td>
<td>Defers the scheduling decision until run time. The scheduling algorithm and chunk size are then taken from the setting of environment variable OMP_SCHEDULE.</td>
</tr>
</tbody>
</table>

**Alternate Options**

None

**par-threshold, Qpar-threshold**

*Sets a threshold for the auto-parallelization of loops.*

**IDE Equivalent**

Windows: None

Linux: **Optimization > Auto-Parallelization Threshold**

Mac OS X: **Optimization > Auto-Parallelization Threshold**
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-par-threshold[n]

Windows:
/Qpar-threshold[[:]]n

Arguments

\( n \)

Is an integer whose value is the threshold for the auto-parallelization of loops. Possible values are 0 through 100.
- If \( n \) is 0, loops get auto-parallelized always, regardless of computation work volume.
- If \( n \) is 100, loops get auto-parallelized when performance gains are predicted based on the compiler analysis data.
- Loops get auto-parallelized only if profitable parallel execution is almost certain.
- The intermediate 1 to 99 values represent the percentage probability for profitable speed-up. For example, \( n=50 \) directs the compiler to parallelize only if there is a 50% probability of the code speeding up if executed in parallel.

Default
-par-threshold100 or /Qpar-threshold100

Loops get auto-parallelized only if profitable parallel execution is almost certain. This is also the default if you do not specify \( n \).

Description

This option sets a threshold for the auto-parallelization of loops based on the probability of profitable execution of the loop in parallel. To use this option, you must also specify -parallel (Linux and Mac OS X) or /Qparallel (Windows).

This option is useful for loops whose computation work volume cannot be determined at compile-time. The threshold is usually relevant when the loop trip count is unknown at compile-time.
The compiler applies a heuristic that tries to balance the overhead of creating multiple threads versus the amount of work available to be shared amongst the threads.

**Alternate Options**

None

**parallel, Qparallel**

*Tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.*

**IDE Equivalent**

Windows: **Optimization > Parallelization**

Linux: **Optimization > Parallelization**

Mac OS X: **Optimization > Parallelization**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `parallel`

**Windows:**

/Qparallel

**Arguments**

None

**Default**

OFF Multithreaded code is not generated for loops that can be safely executed in parallel.

**Description**

This option tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.
To use this option, you must also specify option O2 or O3.

**NOTE.** On Mac OS X systems, when you enable automatic parallelization, you must also set the `DYLD_LIBRARY_PATH` environment variable within Xcode or an error will be displayed.

**Alternate Options**

None

**See Also**

- `par-report, Qpar-report`
- `par-affinity, Qpar-affinity`
- `par-num-threads, Qpar-num-threads`
- `par-runtime-control, Qpar-runtime-control`
- `par-schedule, Qpar-schedule`

**pc, Qpc**

*Enables control of floating-point significand precision.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

*Linux and Mac OS X:*

- `-pcn`

*Windows:*

- `/Qpcn`
Arguments

\(n\)

Is the floating-point significand precision. Possible values are:

- 32: Rounds the significand to 24 bits (single precision).
- 64: Rounds the significand to 53 bits (double precision).
- 80: Rounds the significand to 64 bits (extended precision).

Default

- \(\text{-pc80}\) or \(\text{/Qpc64}\)

On Linux* and Mac OS* X systems, the floating-point significand is rounded to 64 bits. On Windows* systems, the floating-point significand is rounded to 53 bits.

Description

This option enables control of floating-point significand precision.

Some floating-point algorithms are sensitive to the accuracy of the significand, or fractional part of the floating-point value. For example, iterative operations like division and finding the square root can run faster if you lower the precision with this option.

Note that a change of the default precision control or rounding mode, for example, by using the \(\text{-pc32}\) (Linux and Mac OS X) or \(\text{/Qpc32}\) (Windows) option or by user intervention, may affect the results returned by some of the mathematical functions.

Alternate Options

None

Qpch

Enable precompiled header coexistence to reduce build time.

IDE Equivalent

None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/Qpchi
/Qpchi-

Arguments
None

Default
ON The compiler enables precompiled header coexistence.

Description
This option enables precompiled header (PCH) files generated by the Intel® C++ compiler and those generated by the Microsoft Visual C++® compiler to coexist, which reduces build time.

If build time is not an issue and you do not want an additional set of PCH files on your system, specify /Qpchi-.

Alternate Options
None

mp1, Qprec
Improves floating-point precision and consistency.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
-mp1

Windows:
/Qprec

Arguments
None

Default
OFF The compiler provides good accuracy and run-time performance at the expense of less consistent floating-point results.

Description
This option improves floating-point consistency. It ensures the out-of-range check of operands of transcendental functions and improves the accuracy of floating-point compares.

This option prevents the compiler from performing optimizations that change NaN comparison semantics and causes all values to be truncated to declared precision before they are used in comparisons. It also causes the compiler to use library routines that give better precision results compared to the X87 transcendental instructions.

This option disables fewer optimizations and has less impact on performance than option mp or Op.

Alternate Options
None

See Also
•
•
• mp
prec-div, Qprec-div

Improves precision of floating-point divides.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
-prec-div
-no-prec-div

**Windows:**
/Qprec-div
/Qprec-div-

**Arguments**
None

**Default**
-prec-div or/Qprec-div

The compiler uses this method for floating-point divides.

**Description**
This option improves precision of floating-point divides. It has a slight impact on speed.

With some optimizations, such as -xSSE2 (Linux) or /QxSSE2 (Windows), the compiler may change floating-point division computations into multiplication by the reciprocal of the denominator. For example, A/B is computed as A * (1/B) to improve the speed of the computation.
However, sometimes the value produced by this transformation is not as accurate as full IEEE division. When it is important to have fully precise IEEE division, use this option to disable the floating-point division-to-multiplication optimization. The result is more accurate, with some loss of performance.

If you specify `-no-prec-div` (Linux and Mac OS X) or `/Qprec-div-` (Windows), it enables optimizations that give slightly less precise results than full IEEE division.

**Alternate Options**

None

**prec-sqrt, /Qprec-sqrt**

*Improves precision of square root implementations.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-prec-sqrt`
- `-no-prec-sqrt`

**Windows:**

/`Qprec-sqrt`

/`Qprec-sqrt-`

**Arguments**

None

**Default**

- `-no-prec-sqrt`
- `/Qprec-sqrt-`  

The compiler uses a faster but less precise implementation of square root.
However, the default is -prec-sqrt or /Qprec-sqrt if any of the following options are specified: /Od, /Op, or /Qprec on Windows systems; -O0, -mp, or -mp1 on Linux and Mac OS X systems.

**Description**

This option improves precision of square root implementations. It has a slight impact on speed.

This option inhibits any optimizations that can adversely affect the precision of a square root computation. The result is fully precise square root implementations, with some loss of performance.

**Alternate Options**

None

**prof-data-order, Qprof-data-order**

Enables or disables data ordering if profiling information is enabled.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**

- prof-data-order
- no-prof-data-order

**Mac OS X:**

None

**Windows:**

/Qprof-data-order
/Qprof-data-order-
Arguments
None

Default
-no-prof-data-order Data ordering is disabled.
or/Qprof-data-order-

Description
This option enables or disables data ordering if profiling information is enabled. It controls the use of profiling information to order static program data items.

For this option to be effective, you must do the following:

• For instrumentation compilation, you must specify -prof-gen=globdata (Linux) or /Qprof-gen:globdata (Windows).

• For feedback compilation, you must specify -prof-use (Linux) or /Qprof-use (Windows). You must not use multi-file optimization by specifying options such as option -ipo (Linux) or /Qipo (Windows), or option -ipo-c (Linux) or /Qipo-c (Windows).

Alternate Options
None

See Also
•
  •
  • prof-gen, Qprof-gen
  • prof-use, Qprof-use
  • prof-func-order, Qprof-func-order

prof-dir, Qprof-dir
Specifies a directory for profiling information output files.

IDE Equivalent
Windows: General > Profile Directory
Linux: Compiler > Profile Directory
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- `-prof-dir dir`

Windows:
- `/Qprof-dir dir`

**Arguments**

`dir` is the name of the directory.

**Default**

OFF

Profiling output files are placed in the directory where the program is compiled.

**Description**

This option specifies a directory for profiling information output files (*.dyn and *.dpi). The specified directory must already exist.

You should specify this option using the same directory name for both instrumentation and feedback compilations. If you move the .dyn files, you need to specify the new path.

**Alternate Options**

None

`prof-file`, `Qprof-file`

Specifies an alternate file name for the profiling summary files.

**IDE Equivalent**

None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-`-prof-file file`

Windows:
`/Qprof-file file`

Arguments
`file` Is the name of the profiling summary file.

Default
OFF The profiling summary files have the file name pgopti.*

Description
This option specifies an alternate file name for the profiling summary files. The `file` is used as the base name for files created by different profiling passes.

If you add this option to proffmerge, the `.dpi` file will be named `file.dpi` instead of pgopti.dpi.

If you specify `-prof-genx` (Linux and Mac OS X) or `/Qprof-genx` (Windows) with this option, the `.spi` and `.spl` files will be named `file.spi` and `file.spl` instead of pgopti.spi and pgopti.spl.

If you specify `-prof-use` (Linux and Mac OS X) or `/Qprof-use` (Windows) with this option, the `.dpi` file will be named `file.dpi` instead of pgopti.dpi.

Alternate Options
None

See Also

•
•
• `prof-gen`, `Qprof-gen`
• `prof-use`, `Qprof-use`
prof-func-order, Qprof-func-order

Enables or disables function ordering if profiling information is enabled.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**
- `prof-func-order`
- `no-prof-func-order`

**Mac OS X:**
None

**Windows:**
- `/Qprof-func-order`
- `/Qprof-func-order-`

**Arguments**
None

**Default**
- `no-prof-func-order` Function ordering is disabled.
- `/Qprof-func-order-`

**Description**
This option enables or disables function ordering if profiling information is enabled.

For this option to be effective, you must do the following:

- For instrumentation compilation, you must specify `-prof-gen=srcpos` (Linux) or `/Qprof-gen:srcpos` (Windows).
For feedback compilation, you must specify `–prof-use` (Linux) or `/Qprof-use` (Windows). You must not use multi-file optimization by specifying options such as option `–ipo` (Linux) or `/Qipo` (Windows), or option `–ipo-c` (Linux) or `/Qipo-c` (Windows).

If you enable profiling information by specifying option `–prof-use` (Linux) or `/Qprof-use` (Windows), `–prof-func-groups` (Linux) and `/Qprof-func-groups` (Windows) are set and function grouping is enabled. However, if you explicitly enable `–prof-func-order` (Linux) or `/Qprof-func-order` (Windows), function ordering is performed instead of function grouping.

On Linux* systems, this option is only available for Linux linker 2.15.94.0.1, or later.

To set the hotness threshold for function grouping and function ordering, use option `–prof-hotness-threshold` (Linux) or `/Qprof-hotness-threshold` (Windows).

**Alternate Options**

None

The following example shows how to use this option on a Windows system:

```bash
icl /Qprof-gen:globdata file1.c file2.c /Fe instrumented.exe
  ./instrumented.exe
icl /Qprof-use /Qprof-func-order file1.c file2.c /Fe feedback.exe
```

The following example shows how to use this option on a Linux system:

```bash
icl –prof-gen:globdata file1.c file2.c –o instrumented
  ./instrumented.exe
icl –prof-use –prof-func-order file1.c file2.c –o feedback
```

**See Also**

- `prof-hotness-threshold`, `Qprof-hotness-threshold`
- `prof-gen`, `Qprof-gen`
- `prof-use`, `Qprof-use`
- `prof-data-order`, `Qprof-data-order`
- `prof-func-groups`
prof-gen, Qprof-gen

Produces an instrumented object file that can be used in profile-guided optimization.

**IDE Equivalent**

Windows: General > Profile Guided Optimization

**Optimization > Profile Guided Optimization**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-prof-gen[=keyword]`
- `-no-prof-gen`

Windows:

- `/Qprof-gen[:keyword]`
- `/Qprof-gen-`

**Arguments**

*keyword*

Specifies details for the instrumented file. Possible values are:

*default* Produces an instrumented object file. This is the same as specifying `-prof-gen` (Linux* and Mac OS* X) or `/Qprof-gen` (Windows*) with no keyword.
srcpos

Produces an instrumented object file that includes extra source position information. This option is the same as option `-prof-genx` (Linux® and Mac OS® X) or `/Qprof-genx` (Windows®), which are deprecated.

globdata

Produces an instrumented object file that includes information for global data layout.

**Default**

`-no-prof-gen` or `/Qprof-gen-` Profile generation is disabled.

**Description**

This option produces an instrumented object file that can be used in profile-guided optimization. It gets the execution count of each basic block.

If you specify keyword `srcpos` or `globdata`, a static profile information file (.spi) is created. These settings may increase the time needed to do a parallel build using `-prof-gen`, because of contention writing the .spi file.

These options are used in phase 1 of the Profile Guided Optimizer (PGO) to instruct the compiler to produce instrumented code in your object files in preparation for instrumented execution.

**Alternate Options**

None

`prof-hotness-threshold`, `Qprof-hotness-threshold`

Let you set the hotness threshold for function grouping and function ordering.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux:
-prof-hotness-threshold=n

Mac OS X:
None

Windows:
/Qprof-hotness-threshold:n

Arguments

n

Is the hotness threshold. n is a percentage having a value between 0 and 100 inclusive. If you specify 0, there will be no hotness threshold setting in effect for function grouping and function ordering.

Default

OFF

The compiler's default hotness threshold setting of 10 percent is in effect for function grouping and function ordering.

Description

This option lets you set the hotness threshold for function grouping and function ordering.

The "hotness threshold" is the percentage of functions in the application that should be placed in the application's hot region. The hot region is the most frequently executed part of the application. By grouping these functions together into one hot region, they have a greater probability of remaining resident in the instruction cache. This can enhance the application's performance.

For this option to take effect, you must specify option -prof-use (Linux) or /Qprof-use (Windows) and one of the following:

- On Linux systems: -prof-func-groups or -prof-func-order
- On Windows systems: /Qprof-func-order

Alternate Options

None
See Also

- prof-use, Qprof-use
- prof-func-groups
- prof-func-order, Qprof-func-order

prof-src-dir, Qprof-src-dir

Determines whether directory information of the source file under compilation is considered when looking up profile data records.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
- -prof-src-dir
- -no-prof-src-dir

**Windows:**
/ Qprof-src-dir
/Qprof-src-dir-

Arguments

None

Default

-prof-src-dir or/Qprof-src-dir

Directory information is used when looking up profile data records in the .dpi file.
Description

This option determines whether directory information of the source file under compilation is considered when looking up profile data records in the .dpi file. To use this option, you must also specify option -prof-use (Linux and Mac OS X) or /Qprof-use (Windows).

If the option is enabled, directory information is considered when looking up the profile data records within the .dpi file. You can specify directory information by using one of the following options:

- Linux and Mac OS X: -prof-src-root or -prof-src-root-cwd
- Windows: /Qprof-src-root or /Qprof-src-root-cwd

If the option is disabled, directory information is ignored and only the name of the file is used to find the profile data record.

Note that options -prof-src-dir (Linux and Mac OS X) and /Qprof-src-dir (Windows) control how the names of the user's source files get represented within the .dyn or .dpi files. Options -prof-dir (Linux and Mac OS X) and /Qprof-dir (Windows) specify the location of the .dyn or the .dpi files.

Alternate Options

None

See Also

- prof-use, Qprof-use
- prof-src-root, Qprof-src-root
- prof-src-root-cwd, Qprof-src-root-cwd

prof-src-root, Qprof-src-root

Lets you use relative directory paths when looking up profile data and specifies a directory as the base.

IDE Equivalent

None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-prof-src-root=dir

Windows:
/Qprof-src-root:dir

Arguments
dir Is the base for the relative paths.

Default
OFF The setting of relevant options determines the path used when looking up profile data records.

Description
This option lets you use relative directory paths when looking up profile data in .dpi files. It lets you specify a directory as the base. The paths are relative to a base directory specified during the -prof-gen (Linux and Mac OS X) or /Qprof-gen (Windows) compilation phase.

This option is available during the following phases of compilation:
• Linux and Mac OS X: -prof-gen and -prof-use phases
• Windows: /Qprof-gen and /Qprof-use phases

When this option is specified during the -prof-gen or /Qprof-gen phase, it stores information into the .dyn or .dpi file. Then, when .dyn files are merged together or the .dpi file is loaded, only the directory information below the root directory is used for forming the lookup key.

When this option is specified during the -prof-use or /Qprof-use phase, it specifies a root directory that replaces the root directory specified at the -prof-gen or /Qprof-gen phase for forming the lookup keys.

To be effective, this option or option -prof-src-root-cwd (Linux and Mac OS X) or /Qprof-src-root-cwd (Windows) must be specified during the -prof-gen or /Qprof-gen phase. In addition, if one of these options is not specified, absolute paths are used in the .dpi file.
Alternate Options

None

Consider the initial -prof-gen compilation of the source file
\texttt{c:\user1\feature\foo\myproject\common\glob.c}:
\texttt{icc -prof-gen -prof-src-root=c:\user1\feature\foo\myproject -c common\glob.c}

For the -prof-use phase, the file \texttt{glob.c} could be moved into the directory
\texttt{c:\user2\feature\bar\myproject\common\glob.c} and profile information would be found from the
\texttt{.dpi} when using the following:
\texttt{icc -prof-use -prof-src-root=c:\user2\feature\bar\myproject -c common\glob.c}

If you do not use option \texttt{-prof-src-root} during the -prof-gen phase, by default, the -prof-use
compilation can only find the profile data if the file is compiled in the
\texttt{c:\user1\feature\foo\my\project\common} directory.

See Also

- prof-gen, Qprof-gen
- prof-use, Qprof-use
- prof-src-dir, Qprof-src-dir
- prof-src-root-cwd, Qprof-src-root-cwd

\textbf{prof-src-root-cwd, Qprof-src-root-cwd}

\textit{Lets you use relative directory paths when looking
up profile data and specifies the current working
directory as the base.}

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

**Linux and Mac OS X:**
- `prof-src-root-cwd`

**Windows:**
/`Qprof-src-root-cwd`

**Arguments**

None

**Default**

OFF  
The setting of relevant options determines the path used when looking up profile data records.

**Description**

This option lets you use relative directory paths when looking up profile data in `.dpi` files. It specifies the current working directory as the base. To use this option, you must also specify option `-prof-use` (Linux and Mac OS) or `/Qprof-use` (Windows).

This option is available during the following phases of compilation:

- Linux and Mac OS X: `-prof-gen` and `-prof-use` phases
- Windows: `/Qprof-gen` and `/Qprof-use` phases

When this option is specified during the `-prof-gen` or `/Qprof-gen` phase, it stores information into the `.dyn` or `.dpi` file. Then, when `.dyn` files are merged together or the `.dpi` file is loaded, only the directory information below the root directory is used for forming the lookup key.

When this option is specified during the `-prof-use` or `/Qprof-use` phase, it specifies a root directory that replaces the root directory specified at the `-prof-gen` or `/Qprof-gen` phase for forming the lookup keys.

To be effective, this option or option `-prof-src-root` (Linux and Mac OS X) or `/Qprof-src-root` (Windows) must be specified during the `-prof-gen` or `/Qprof-gen` phase. In addition, if one of these options is not specified, absolute paths are used in the `.dpi` file.

**Alternate Options**

None
See Also

- prof-gen, Qprof-gen
- prof-use, Qprof-use
- prof-src-dir, Qprof-src-dir
- prof-src-root, Qprof-src-root

prof-use, Qprof-use
Enables the use of profiling information during optimization.

IDE Equivalent
Windows: General > Profile Guided Optimization
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-prof-use[=arg]
-no-prof-use

Windows:
/Qprof-use[:arg]
/Qprof-use-

Arguments
arg
Specifies additional instructions. Possible values are:

weighted
Tells the profmerge utility to apply a weighting to the .dyn file values when creating the .dpi file to normalize the data
counts when the training runs have
different execution durations. This
argument only has an effect when the
compiler invokes the profmerge utility to
create the .dpi file. This argument does
not have an effect if the .dpi file was
previously created without weighting.

```
[no]merge
```

Enables or disables automatic invocation
of the profmerge utility. The default is
merge. Note that you cannot specify both
weighted and nomerge. If you try to
specify both values, a warning will be
displayed and nomerge takes precedence.

```
default
```

Enables the use of profiling information
during optimization. The profmerge utility
is invoked by default. This value is the
same as specifying `--prof-use` (Linux and
Mac OS X) or `/Qprof-use` (Windows)
with no argument.

**Default**

```
-no-prof-use or /Qprof-use-
```

Profiling information is not used during optimization.

**Description**

This option enables the use of profiling information (including function splitting and function
grouping) during optimization. It enables option `-fnsplit` (Linux) or `/Qfnsplit` (Windows).

This option instructs the compiler to produce a profile-optimized executable and it merges
available profiling output files into a pgopti.dpi file.

Note that there is no way to turn off function grouping if you enable it using this option.

To set the hotness threshold for function grouping and function ordering, use option `--prof-
hotness-threshold` (Linux) or `/Qprof-hotness-threshold` (Windows).

**Alternate Options**

None
See Also

- 
- 
- prof-hotness-threshold, Qprof-hotness-threshold

rcd, Qrcd

*Enables fast float-to-integer conversions.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X:

```bash
-rcd
```

Windows:

```bash
/Qrcd
```

**Arguments**

None

**Default**

**OFF**

Floating-point values are truncated when a conversion to an integer is involved. On Windows, this is the same as specifying `/QIfist-`.

**Description**

This option enables fast float-to-integer conversions. It can improve the performance of code that requires floating-point-to-integer conversions.

The system default floating-point rounding mode is round-to-nearest. However, the C language requires floating-point values to be truncated when a conversion to an integer is involved. To do this, the compiler must change the rounding mode to truncation before each floating-point-to-integer conversion and change it back afterwards.
This option disables the change to truncation of the rounding mode for all floating-point calculations, including floating point-to-integer conversions. This option can improve performance, but floating-point conversions to integer will not conform to C semantics.

**Alternate Options**

Linux and Mac OS X: None

Windows: /QIfist

**rct, Qrct**

*Sets the internal FPU rounding control to Truncate.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X:

- rct

Windows:

/Qrct

**Arguments**

None

**Default**

OFF

The compiler uses the default setting for the FPU rounding control.

**Description**

This option sets the internal FPU rounding control to Truncate.

**Alternate Options**

None
**restrict, Qrestrict**

Determines whether pointer disambiguation is enabled with the restrict qualifier.

**IDE Equivalent**

Windows: Language > Recognize Restrict Keyword

Linux: Language > Recognize Restrict Keyword

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- -restrict
- -no-restrict

Windows:
/ Qrestrict
/ Qrestrict-

**Arguments**

None

**Default**

- -no-restrict or /Q restrict- Pointers are not qualified with the restrict keyword.

**Description**

This option determines whether pointer disambiguation is enabled with the restrict qualifier. Options -restrict (Linux and Mac OS X) and /Q restrict (Windows OS) enable the recognition of the restrict keyword as defined by the ANSI standard.
By qualifying a pointer with the restrict keyword, you assert that an object accessed by the pointer is only accessed by that pointer in the given scope. You should use the restrict keyword only when this is true. When the assertion is true, the restrict option will have no effect on program correctness, but may allow better optimization.

Alternate Options
None

See Also

•
•
• Qc99

Qsafeseh
Registers exception handlers for safe exception handling.

IDE Equivalent
None

Architectures
IA-32 architecture

Syntax

Linux and Mac OS X:
None

Windows:
/Qsafeseh[-]

Arguments
None

Default
ON (if /Qvc7.1 or higher is specified)
**Description**

Registers exception handlers for safe exception handling. It also marks objects as "compatible with the Registered Exception Handling feature" whether they contain handlers or not. This is important because the Windows linker will only generate the "special registered EH table" if ALL objects that it is building into an image are marked as compatible. If any objects are not marked as compatible, the EH table is not generated.

Digital signatures certify security and are required for components that are shipped with Windows, such as device drivers.

**Alternate Options**

None

**See Also**

• /EH

**save-temps, Qsave-temps**

_Tells the compiler to save intermediate files created during compilation._

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

  - `save-temps`
  - `no-save-temps`

Windows:

  /Qsave-temps
  /Qsave-temps-
Arguments
None

Default
Linux and Mac OS X: -no-save-temps
Windows: .obj files are saved

On Linux and Mac OS X systems, the compiler deletes intermediate files after compilation is completed. On Windows systems, the compiler saves only intermediate object files after compilation is completed.

Description
This option tells the compiler to save intermediate files created during compilation. The names of the files saved are based on the name of the source file; the files are saved in the current working directory.

If -save-temps or /Qsave-temps is specified, the following occurs:
• The object .o file (Linux and Mac OS X) or .obj file (Windows) is saved.
• The assembler .s file (Linux and Mac OS X) or .asm file (Windows) is saved if you specified -use-asm (Linux or Mac OS X) or /Quse-asm (Windows).

If -no-save-temps is specified on Linux or Mac OS X systems, the following occurs:
• The .o file is put into /tmp and deleted after calling ld.
• The preprocessed file is not saved after it has been used by the compiler.

If /Qsave-temps is specified on Windows systems, the following occurs:
• The .obj file is not saved after the linker step.
• The preprocessed file is not saved after it has been used by the compiler.

NOTE. This option only saves intermediate files that are normally created during compilation.

Alternate Options
None
Example

If you compile program `my_foo.c` on a Linux or Mac OS X system and you specify option `-save-temps` and option `-use-asm`, the compilation will produce files `my_foo.o` and `my_foo.s`.

If you compile program `my_foo.c` on a Windows system and you specify option `/Qsave-temps` and option `/Quse-asm`, the compilation will produce files `my_foo.o` and `my_foo.asm`.

**scalar-rep, Qscalar-rep**

*Enables scalar replacement performed during loop transformation.*

**IDE Equivalent**

None

**Architectures**

IA-32 architecture

**Syntax**

*Linux and Mac OS X:*

- `-scalar-rep`
- `-no-scalar-rep`

*Windows:*

/`Qscalar-rep`
/`Qscalar-rep-`

**Arguments**

None

**Default**

- `-no-scalar-rep`
  
  Scalar replacement is not performed during loop transformation.
  
  
  or `/Qscalar-rep-`
**Description**

This option enables scalar replacement performed during loop transformation. To use this option, you must also specify O3.

**Alternate Options**

None

**See Also**

* None
  
  mserialize-volatile, Qserialize-volatile

* Determines whether strict memory access ordering is imposed for volatile data object references.

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

**Linux:**

-mserialize-volatile
-mno-serialize-volatile

**Mac OS X:**

None

**Windows:**

/Qserialize-volatile
/Qserialize-volatile-
Arguments
None

Default

\[-mno-serialize-volatile\]

The compiler uses default memory access ordering.

-volatile or /Qserialize-volatile

Description
This option determines whether strict memory access ordering is imposed for volatile data object references.

If you specify \(-mno-serialize-volatile\), the compiler may suppress both run-time and compile-time memory access ordering for volatile data object references. Specifically, the \(.rel/.acq\) completers will not be issued on referencing loads and stores.

Alternate Options
None

Qsfalign
Specify stack alignment for functions.

IDE Equivalent
None

Architectures
IA-32 architecture

Syntax

Linux and Mac OS X:
None

Windows:
/Qsfalign[n]
Arguments

\( n \)  
Is the byte size of aligned variables. Possible values are:

8  
Specifies that alignment should occur for functions with 8-byte aligned variables. At this setting the compiler aligns the stack to 16 bytes if there is any 16-byte or 8-byte data on the stack. For 8-byte data, the compiler only aligns the stack if the alignment will produce a performance advantage.

16  
Specifies that alignment should occur for functions with 16-byte aligned variables. At this setting, the compiler only aligns the stack for 16-byte data. No attempt is made to align for 8-byte data.

Default

/Qsfalign8  
Alignment occurs for functions with 8-byte aligned variables.

Description

This option specifies stack alignment for functions. It lets you disable the normal optimization that aligns a stack for 8-byte data.

If you do not specify \( n \), stack alignment occurs for all functions. If you specify /Qsfalign-, no stack alignment occurs for any function.

Alternate Options

None

std, Qstd

_Tells the compiler to conform to a specific language standard._

IDE Equivalent

Windows: Language > Enable C++0x Support  
Language > Enable C99 Support
Linux: **Language > ANSI Conformance**

Mac OS X: **Language > C ANSI Conformance**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
-std=val

**Windows:**
/Qstd=val

**Arguments**

<table>
<thead>
<tr>
<th>val</th>
<th>Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>gnu89</td>
<td>Conforms to ISO C90 plus GNU* extensions.</td>
</tr>
<tr>
<td>gnu++98</td>
<td>Conforms to the 1998 ISO C++ standard plus GNU extensions.</td>
</tr>
<tr>
<td>c++0x</td>
<td>Enable support for the following C++0x features:</td>
</tr>
<tr>
<td></td>
<td>• Empty macro arguments</td>
</tr>
<tr>
<td></td>
<td>• Variadic macros</td>
</tr>
<tr>
<td></td>
<td>• Type long long</td>
</tr>
<tr>
<td></td>
<td>• Trailing comma in enum definition</td>
</tr>
<tr>
<td></td>
<td>• Concatenation of mixed-width string literals</td>
</tr>
<tr>
<td></td>
<td>• Extended friend declarations</td>
</tr>
</tbody>
</table>
• Use of ">>" to close two template argument lists
• Relaxed rules for use of "typename"
• Relaxed rules for disambiguation using the "template" keyword
• Copy constructor does not need to be callable on direct reference
• Binding to class rvalue
• "extern template" to suppress instantiation of an entity
• "auto" type specifier
• decltype operator
• static_assert
• compliant __func__
• lambda expressions

Default

-std=gnu89 (default for C) Conforms to ISO C90 plus GNU extensions.
-std=gnu++98 (default for C++) Conforms to the 1998 ISO C++ standard plus GNU* extensions.

Description
Tells the compiler to conform to a specific language standard.

Alternate Options
None
sox, Qsox
Tells the compiler to save the compilation options and version number in the Linux* OS executable or the Windows* OS object file.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:
- sox
- no-sox

Mac OS X:
None

Windows:
/Qsox
/Qsox-

Arguments
None

Default
-no-sox
or/Qsox-
The compiler does not save the compiler options and version number in the executable.

Description
Tells the compiler to save the compilation options and version number in the Linux* OS executable or the Windows* OS object file.

On Linux systems, the size of the executable on disk is increased slightly by the inclusion of these infoimation strings.
This option forces the compiler to embed in each object file or assembly output a string that contains information about the compiler version and compilation options for each source file that has been compiled.

On Windows systems, the information stays in the object file. On Linux systems, when you link the object files into an executable file, the linker places each of the information strings into the header of the executable. It is then possible to use a tool, such as a strings utility, to determine what options were used to build the executable file.

If `-no-sox` or `/Qsox-` is specified, this extra information is not put into the object or assembly output generated by the compiler.

**Alternate Options**

None

**tbb, Qtbb**

*Tells the compiler to link to the Intel® Threading Building Blocks (Intel® TBB) libraries.*

**IDE Equivalent**

Windows: None

Linux: Performance Library Build Components > Use Intel(R) Threading Building Blocks Library

Mac OS X: Libraries > Use Intel(R) Threading Building Blocks Library

**Architectures**

IA-32, Intel® 64 architecture, IA-64 architecture

**Syntax**

Linux and Mac OS X:

- `tbb`

Windows:

- `/Qtbb`

**Arguments**

None
**Default**
OFF
The compiler does not link to the Intel® TBB libraries.

**Description**
This option tells the compiler to link to the Intel® Threading Building Blocks (Intel® TBB) libraries and include the Intel® TBB headers.

On IA-64 architecture, this option is not available on Windows systems.

**Alternate Options**
None

**tcheck, Qtcheck**
Enables analysis of threaded applications.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux:
-tcheck

Mac OS X:
None

Windows:
/Qtcheck

**Arguments**
None
Default

OFF  Threaded applications are not instrumented by the compiler for analysis by Intel® Thread Checker.

Description

This option enables analysis of threaded applications.

To use this option, you must have Intel® Thread Checker installed, which is one of the Intel® Threading Analysis Tools. If you do not have this tool installed, the compilation will fail. Remove the -tcheck (Linux) or /Qtcheck (Windows) option from the command line and recompile.

For more information about Intel® Thread Checker (including an evaluation copy), open the page associated with threading tools at Intel® Software Development Products.

Alternate Options

None

tcollect, Qtcollect

*Inserts instrumentation probes calling the Intel® Trace Collector API.*

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux:

-tcollect[lib]

Mac OS X:

None

Windows:

/Qtcollect[:lib]
Arguments

lib

Is one of the Intel® Trace Collector libraries; for example, VT, VTcs, VTmc, or VTfs. If you do not specify lib, the default library is VT.

Default

OFF

Instrumentation probes are not inserted into compiled applications.

Description

This option inserts instrumentation probes calling the Intel® Trace Collector API. To use this option, you must have the Intel® Trace Collector installed and set up through one of its set-up scripts. This tool is a component of the Intel® Trace Analyzer and Collector.

This option provides a flexible and convenient way of instrumenting functions of a compiled application. For every function, the entry and exit points are instrumented at compile time to let the Intel® Trace Collector record functions beyond the default MPI calls. For non-MPI applications (for example, threaded or serial), you must ensure that the Intel® Trace Collector is properly initialized (VT_initialize/VT_init).

CAUTION. Be careful with full instrumentation because this feature can produce very large trace files.

For more details, see the Intel® Trace Collector User Guide.

Alternate Options

None

See Also

•

•

• tcollect-filter, Qtcollect-filter
tcollect-filter, Qtcollect-filter

Lets you enable or disable the instrumentation of specified functions.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux:
-tcollect-filter file

Mac OS X:

None

Windows:

/Qtcollect-filter:file

Arguments

file

Is a configuration file that lists filters, one per line. Each filter consists of a regular expression string and a switch. Strings with leading or trailing white spaces must be quoted. Other strings do not have to be quoted. The switch value can be ON, on, OFF, or off.

Default

OFF

Functions are not instrumented. However, if option -tcollect (Linux) or /Qtcollect (Windows) is specified, the filter setting is ".* ON" and all functions get instrumented.

Description

This option lets you enable or disable the instrumentation of specified functions.
During instrumentation, the regular expressions in the file are matched against the function names. The switch specifies whether matching functions are to be instrumented or not. Multiple filters are evaluated from top to bottom with increasing precedence.

The names of the functions to match against are formatted as follows:

- C++ function names are demangled and the C++ class hierarchy is used. Function parameters are stripped to keep the function names shorter.
- The source file name is followed by a colon-separated function name. Source file names should contain the full path, if available. For example:
  /home/joe/src/foo.c:FOO_bar
- Classes and function names are separated by double colons. For example:
  /home/joe/src/foo.cpp:app::foo::bar

You can use option `-opt-report` (Linux) or `/Qopt-report` (Windows) to get a full list of file and function names that the compiler recognizes from the compilation unit. This list can be used as the basis for filtering in the configuration file.

To use this option, you must have the Intel® Trace Collector installed and set up through one of its set-up scripts. This tool is a component of the Intel® Trace Analyzer and Collector.

For more details, see the Intel® Trace Collector User Guide.

**Alternate Options**

None

Consider the following filters in a configuration file:

`.**` OFF `.*vector.*` ON

The above will cause instrumentation of only those functions having the string 'vector' in their names. No other function will be instrumented. Note that reversing the order of the two lines will prevent instrumentation of all functions.

To get a list of the file or routine strings that can be matched by the regular expression filters, generate an optimization report with tcollect information. For example:

Windows OS: icl /Qtcollect /Qopt-report /Qopt-report-phase tcollect

Linux OS: icc -tcollect -opt-report -opt-report-phase tcollect

**See Also**

- 
- 

• tcollect, Qtcollect

\textbf{ftemplate-depth, Qtemplate-depth}

\textit{Control the depth in which recursive templates are expanded.}

\textbf{IDE Equivalent}

None

\textbf{Architectures}

IA-32, Intel® 64, IA-64 architectures

\textbf{Syntax}

\textbf{Linux and MacOS X:}

\texttt{-ftemplate-depth-n}

\textbf{Windows:}

\texttt{/Qtemplate-depth-n}

\textbf{Arguments}

\textit{n} \hfill The number of recursive templates that are expanded.

\textbf{Default}

OFF

\textbf{Description}

Control the depth in which recursive templates are expanded. On Linux*, this option is supported only by invoking the compiler with icpc.

\textbf{Alternate Options}

None
ftrapuv, Qtrapuv
Initialization stack local variables to an unusual value to aid error detection.

IDE Equivalent
Windows: Code Generation > Initialize Local Variables to NaN
Linux: Code Generation > Initialize Local Variables to NaN
Mac OS X: Code Generation > Initialize Local Variables to NaN

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-ftrapuv
Windows:
/Qtrapuv

Arguments
None

Default
OFF The compiler does not initialize local variables.

Description
This option initializes stack local variables to an unusual value to aid error detection. Normally, these local variables should be initialized in the application.

The option sets any uninitialized local variables that are allocated on the stack to a value that is typically interpreted as a very large integer or an invalid address. References to these variables are then likely to cause run-time errors that can help you detect coding errors.

This option sets option -g (Linux and Mac OS X) and /Zi or /Z7 (Windows).
Alternate Options
None

See Also
•
•
• g, zi, Z7

unroll-aggressive, Qunroll-aggressive
Determines whether the compiler uses more aggressive unrolling for certain loops.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- unroll-aggressive
- no-unroll-aggressive

Windows:
/Qunroll-aggressive
/Qunroll-aggressive-

Arguments
None

Default
-no-unroll-aggressive or /Qunroll-aggressive-

The compiler uses default heuristics when unrolling loops.
**Description**

This option determines whether the compiler uses more aggressive unrolling for certain loops. The positive form of the option may improve performance.

On IA-32 architecture and Intel® 64 architecture, this option enables aggressive, complete unrolling for loops with small constant trip counts.

On IA-64 architecture, this option enables additional complete unrolling for loops that have multiple exits or outer loops that have a small constant trip count.

**Alternate Options**

None

**unroll, Qunroll**

*Tells the compiler the maximum number of times to unroll loops.*

**IDE Equivalent**

Windows: **Optimization > Loop Unrolling**

Linux: **Optimization > Loop Unroll Count**

Mac OS X: **Optimization > Loop Unrolling**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```bash
-unroll[n]
```

Windows:

```cmd
/Qunroll[:n]
```

**Arguments**

\(n\)  
Is the maximum number of times a loop can be unrolled. To disable loop unrolling, specify 0. On systems using IA-64 architecture, you can only specify a value of 0.
Default
- unroll
  or /Qunroll
  The compiler uses default heuristics when unrolling loops.

Description
This option tells the compiler the maximum number of times to unroll loops.
If you do not specify \( n \), the optimizer determines how many times loops can be unrolled.

Alternate Options
Linux and Mac OS X: -funroll-loops
Windows: None

use-asm, /Quse-asm
Tells the compiler to produce objects through the assembler.

IDEEquivalent
None

Architectures
- use-asm: IA-32 architecture, Intel® 64 architecture, IA-64 architecture
  /Quse-asm: IA-64 architecture

Syntax
Linux and Mac OS X:
- use-asm
  - no-use-asm

Windows:
  /Quse-asm
  /Quse-asm-

Arguments
None
**Default**

- **no-use-asm**  
  The compiler produces objects directly.

  or **/Quse-asm**

**Description**

This option tells the compiler to produce objects through the assembler.

**Alternate Options**

None

**use-intel-optimized-headers, Quse-intel-optimized-headers**

*Determines whether the performance headers directory is added to the include path search list.*

**IDE Equivalent**

Windows: **Optimization > Use Intel Optimized Headers**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- **use-intel-optimized-headers**

  **Windows:**
  
  **/Quse-intel-optimized-headers**

**Arguments**

None
**Default**

-no-use-intel-optimized-headers or /Quse-intel-optimized-headers

The performance headers directory is not added to the include path search list.

**Description**

This option determines whether the performance headers directory is added to the include path search list.

The performance headers directory is added if you specify -use-intel-optimized-headers (Linux and Mac OS X) or /Quse-intel-optimized-headers (Windows OS). Appropriate libraries are also linked in, as needed, for proper functionality.

**Alternate Options**

None

**Quse-msasm-symbols**

*Tells the compiler to use a dollar sign ("$") when producing symbol names.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Quse-msasm-symbols

**Arguments**

None
**Default**  
**OFF**  The compiler uses a period ("." ) when producing symbol names

**Description**  
This option tells the compiler to use a dollar sign ("\$") when producing symbol names.  
Use this option if you require symbols in your .asm files to contain characters that are accepted by the MS assembler.

**Alternate Options**  
None

**V (Linux* and Mac OS* X)**  
*Displays the compiler version information.*

**IDE Equivalent**  
Windows: None  
Linux: General > Show Startup Banner  
Mac OS X: General > Show Startup Banner

**Architectures**  
IA-32, Intel® 64, IA-64 architectures

**Syntax**  
Linux and Mac OS X:  
\[-V\]  
Windows:  
\[/QV\]

**Arguments**  
None

**Default**  
**OFF**  The compiler version information is not displayed.
**Description**

This option displays the startup banner, which contains the following compiler version information:

- **ID**: unique identification number for the compiler
- **x.y.z**: version of the compiler
- **years**: years for which the software is copyrighted

This option can be placed anywhere on the command line.

**Alternate Options**

None

**Qvc**

Specifies compatibility with Microsoft* Visual C++ or Microsoft* Visual Studio.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

- **Linux and Mac OS X**: None
- **Windows**:
  - `/Qvc7.1`
  - `/Qvc8`
  - `/Qvc9`

**Arguments**

None
Default
varies

When the compiler is installed, it detects which version of Visual Studio is on your system. Qvc defaults to the form of the option that is compatible with that version. When multiple versions of Visual Studio are installed, the compiler installation lets you select which version you want to use. In this case, Qvc defaults to the version you choose.

Description
This option specifies compatibility with Visual C++ or Visual Studio.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Qvc7.1</td>
<td>Specifies compatibility with Microsoft* Visual Studio .NET 2003.</td>
</tr>
<tr>
<td>/Qvc8</td>
<td>Specifies compatibility with Microsoft* Visual Studio 2005.</td>
</tr>
<tr>
<td>/Qvc9</td>
<td>Specifies compatibility with Microsoft* Visual Studio 2008.</td>
</tr>
</tbody>
</table>

Alternate Options
None

vec, Qvec

Enables or disables vectorization and transformations enabled for vectorization.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures
Syntax

Linux and Mac OS X:

-vec
-no-vec

Windows:

/Qvec
/Qvec-

Arguments

None

Default

-vec or /Qvec

Vectorization is enabled.

Description

This option enables or disables vectorization and transformations enabled for vectorization.

To disable vectorization and transformations enabled for vectorization, specify -no-vec (Linux and Mac OS X) or /Qvec- (Windows).

Alternate Options

None

See Also

•

• ax, Qax

• x, Qx

• vec-report, Qvec-report

• vec-guard-write, Qvec-guard-write

• vec-threshold, Qvec-threshold
vec-guard-write, Qvec-guard-write

*Tells the compiler to perform a conditional check in a vectorized loop.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

*Linux and Mac OS X:*

- -vec-guard-write
- -no-vec-guard-write

*Windows:*

/Qvec-guard-write
/Qvec-guard-write-

**Arguments**

None

**Default**

- -no-vec-guard-write
- /Qvec-guard-write-

The compiler uses default heuristics when checking vectorized loops.

**Description**

This option tells the compiler to perform a conditional check in a vectorized loop. This checking avoids unnecessary stores and may improve performance.

**Alternate Options**

None
**vec-report, Qvec-report**  
*Controls the diagnostic information reported by the vectorizer.*

**IDE Equivalent**  
Windows: None  
Linux: **Compilation Diagnostics > Vectorizer Report**  
Mac OS X: **Diagnostics > Vectorizer Diagnostic Report**

**Architectures**  
IA-32, Intel® 64 architectures

**Syntax**  
**Linux and Mac OS X:**  
-vec-report[\(n\)]  

**Windows:**  
/Qvec-report[\(n\)]

**Arguments**

\(n\)  
Is a value denoting which diagnostic messages to report. Possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tells the vectorizer to report no diagnostic information.</td>
</tr>
<tr>
<td>1</td>
<td>Tells the vectorizer to report on vectorized loops.</td>
</tr>
<tr>
<td>2</td>
<td>Tells the vectorizer to report on vectorized and non-vectorized loops.</td>
</tr>
<tr>
<td>3</td>
<td>Tells the vectorizer to report on vectorized and non-vectorized loops and any proven or assumed data dependences.</td>
</tr>
<tr>
<td>4</td>
<td>Tells the vectorizer to report on non-vectorized loops.</td>
</tr>
</tbody>
</table>
Tells the vectorizer to report on non-vectorized loops and the reason why they were not vectorized.

**Default**

-vec-report1
or/Qvec-report1

If the vectorizer has been enabled and you do not specify n, the compiler reports diagnostics on vectorized loops. If you do not specify the option on the command line, the default is to display no messages.

**Description**

This option controls the diagnostic information reported by the vectorizer. The vectorizer report is sent to stdout.

If you do not specify n, it is the same as specifying -vec-report1 (Linux and Mac OS X) or /Qvec-report1 (Windows).

The vectorizer is enabled when certain compiler options are specified, such as option -ax or -x (Linux and Mac OS X), option /Qax or /Qx (Windows), option /arch:SSE or /arch:SSE2 (Windows), and option fast.

**Alternate Options**

None

**vec-threshold, Qvec-threshold**

_Sets a threshold for the vectorization of loops._

**IDE Equivalent**

Windows: **Optimization > Threshold For Vectorization**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64 architectures
Syntax

Linux and Mac OS X:

-vec-threshold\([n]\)

Windows:

/Qvec-threshold\([::]n]\)

Arguments

\(n\)

Is an integer whose value is the threshold for the vectorization of loops. Possible values are 0 through 100. If \(n\) is 0, loops get vectorized always, regardless of computation work volume. If \(n\) is 100, loops get vectorized when performance gains are predicted based on the compiler analysis data. Loops get vectorized only if profitable vector-level parallel execution is almost certain. The intermediate 1 to 99 values represent the percentage probability for profitable speed-up. For example, \(n=50\) directs the compiler to vectorize only if there is a 50% probability of the code speeding up if executed in vector form.

Default

-vec-threshold\(100\)

or /Qvec-threshold\(100\)

Loops get vectorized only if profitable vector-level parallel execution is almost certain. This is also the default if you do not specify \(n\).

Description

This option sets a threshold for the vectorization of loops based on the probability of profitable execution of the vectorized loop in parallel.

This option is useful for loops whose computation work volume cannot be determined at compile-time. The threshold is usually relevant when the loop trip count is unknown at compile-time.

The compiler applies a heuristic that tries to balance the overhead of creating multiple threads versus the amount of work available to be shared amongst the threads.
**Alternate Options**

None

**wd, Qwd**

*Disables a soft diagnostic. This is a deprecated option.*

**IDE Equivalent**

Windows: **Advanced > Disable Specific Warnings**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

-wdLn[,Ln,...]

Windows:

/QwdLn[,Ln,...]

**Arguments**

*Ln* is the number of the diagnostic to disable.

**Default**

OFF The compiler returns soft diagnostics as usual.

**Description**

This option disables the soft diagnostic that corresponds to the specified number.

If you specify more than one *Ln*, each *Ln* must be separated by a comma.

**Alternate Options**

None
we, Qwe

Changes a soft diagnostic to an error. This is a deprecated option.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-weLn[,Ln,...]

Windows:
/QweLn[,Ln,...]

Arguments
Ln
Is the number of the diagnostic to be changed.

Default
OFF
The compiler returns soft diagnostics as usual.

Description
This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to an error.
If you specify more than one Ln, each Ln must be separated by a comma.

Alternate Options
None
**wn, Qwn**

*Controls the number of errors displayed before compilation stops. This is a deprecated option.*

**IDE Equivalent**

Windows: Diagnostics > Error Limit

Linux: Compilation Diagnostics > Set Error Limit

Mac OS X: Diagnostics > Error Limit

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-wn n
```

Windows:

```
/Qwn n
```

**Arguments**

```
n
```

Is the number of errors to display.

**Default**

```
100
```

The compiler displays a maximum of 100 errors before aborting compilation.

**Description**

This option controls the number of errors displayed before compilation stops.

**Alternate Options**

None
wo, Qwo
 Tells the compiler to issue one or more diagnostic messages only once. This is a deprecated option.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-woLn[,Ln,...]

Windows:
/QwoLn[,Ln,...]

Arguments
Ln Is the number of the diagnostic.

Default
OFF

Description
Specifies the ID number of one or more messages. If you specify more than one Ln, each Ln must be separated by a comma.

Alternate Options
None
**wr, Qwr**

Changes a soft diagnostic to a remark. This is a deprecated option.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `C:\wrLn[,Ln,...]`

**Windows:**

`/QwrLn[,Ln,...]`

**Arguments**

`Ln` is the number of the diagnostic to be changed.

**Default**

OFF The compiler returns soft diagnostics as usual.

**Description**

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to a remark.

If you specify more than one `Ln`, each `Ln` must be separated by a comma.

**Alternate Options**

None
**ww, Qww**

Changes a soft diagnostic to a warning. This is a deprecated option.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-wwLn[,Ln,...]

**Windows:**

/QwwLn[,Ln,...]

**Arguments**

Ln Is the number of the diagnostic to be changed.

**Default**

OFF The compiler returns soft diagnostics as usual.

**Description**

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to a warning.

If you specify more than one Ln, each Ln must be separated by a comma.

**Alternate Options**

None
**x, Qx**

*Tells the compiler to generate optimized code specialized for the Intel processor that executes your program.*

**IDE Equivalent**

Windows: **Code Generation > Intel Processor-Specific Optimization**

Linux: **Code Generation > Intel Processor-Specific Optimization**

Mac OS X: **Code Generation > Intel Processor-Specific Optimization**

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X:

`-xprocessor`

Windows:

`/Qxprocessor`

**Arguments**

*processor*

Indicates the processor for which code is generated. Many of the following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE). Possible values are:

**Host**

Can generate instructions for the highest instruction set available on the compilation host processor. On Intel processors, this may correspond to the most suitable `-x` (Linux* and Mac OS® X) or `/Qx` (Windows*) option. On non-Intel processors, this may correspond to the most suitable `-m` (Linux and Mac OS X) or `/arch` (Windows) option.
The resulting executable may not run on a processor different from the host in the following cases:

- If the processor does not support all of the instructions supported by the host processor.
- If the host is an Intel processor and the other processor is a non-Intel processor.

**AVX**

Optimizes for Intel processors that support Intel® Advanced Vector Extensions (Intel® AVX).

**SSE4.2**

Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.

**SSE4.1**

Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture. This replaces value S, which is deprecated.

**SSE3_ATOM**

Optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology. Can generate MOVBE instructions, depending on the setting of option -minstruction (Linux and Mac OS) or /Qinstruction (Windows).
SSSE3 Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™ 2 Duo processor family. For Mac OS* X systems, this value is only supported on Intel® 64 architecture. This replaces value T, which is deprecated.

SSE3 Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel® NetBurst® microarchitecture. For Mac OS* X systems, this value is only supported on IA-32 architecture. This replaces value P, which is deprecated.

SSE2 Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2. This value is not available on Mac OS* X systems. This replaces value N, which is deprecated.

**Default**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows* systems</td>
<td>/arch:SSE2</td>
</tr>
<tr>
<td>Linux* systems</td>
<td>/msse2</td>
</tr>
<tr>
<td>Mac OS* X systems using</td>
<td></td>
</tr>
<tr>
<td>IA-32 architecture: SSE3</td>
<td></td>
</tr>
<tr>
<td>Mac OS* X systems using Intel®</td>
<td></td>
</tr>
<tr>
<td>64 architecture: SSSE3</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

This option tells the compiler to generate optimized code specialized for the Intel processor that executes your program. It also enables optimizations in addition to Intel processor-specific optimizations. The specialized code generated by this option may run only on a subset of Intel processors.
This option can enable optimizations depending on the argument specified. For example, it may enable Intel® Streaming SIMD Extensions 4 (Intel® SSE4), Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Intel® Streaming SIMD Extensions 2 (Intel® SSE2), or Intel® Streaming SIMD Extensions (Intel® SSE) instructions.

The binaries produced by these values will run on Intel processors that support all of the features for the targeted processor. For example, binaries produced with SSE3 will run on an Intel® Core™ 2 Duo processor, because that processor completely supports all of the capabilities of the Intel® Pentium® 4 processor, which the SSE3 value targets. Specifying the SSSE3 value has the potential of using more features and optimizations available to the Intel® Core™ 2 Duo processor.

Do not use processor values to create binaries that will execute on a processor that is not compatible with the targeted processor. The resulting program may fail with an illegal instruction exception or display other unexpected behavior. For example, binaries produced with SSE3 may produce code that will not run on Intel® Pentium® III processors or earlier processors that do not support SSE3 instructions.

Compiling the function main() with any of the processor values produces binaries that display a fatal run-time error if they are executed on unsupported processors. For more information, see Optimizing Applications.

If you specify more than one processor value, code is generated for only the highest-performing processor specified. The highest-performing to lowest-performing processor values are: SSE4.2, SSE4.1, SSSE3, SSE3, SSE2. Note that processor values AVX and SSE3_ATOM do not fit within this group.

Compiler options m and arch produce binaries that should run on processors not made by Intel that implement the same capabilities as the corresponding Intel processors.

Previous value O is deprecated and has been replaced by option -msse3 (Linux and Mac OS X) and option /arch:SSE3 (Windows).

Previous values W and K are deprecated. The details on replacements are as follows:

- Mac OS X systems: On these systems, there is no exact replacement for W or K. You can upgrade to the default option -msse3 (IA-32 architecture) or option -mssse3 (Intel® 64 architecture).
- Windows and Linux systems: The replacement for W is -msse2 (Linux) or /arch:SSE2 (Windows). There is no exact replacement for K. However, on Windows systems, /QxK is interpreted as /arch:IA32; on Linux systems, -xK is interpreted as -m1a32. You can also do one of the following:
- Upgrade to option `-msse2` (Linux) or option `arch:SSE2` (Windows). This will produce one code path that is specialized for Intel® SSE2. It will not run on earlier processors
- Specify the two option combination `-mia32` `-axSSE2` (Linux) or `/arch:IA32 /QaxSSE2` (Windows). This combination will produce an executable that runs on any processor with IA-32 architecture but with an additional specialized Intel® SSE2 code path.

The `-x` and `/Qx` options enable additional optimizations not enabled with option `-m` or option `/arch`.

On Windows® systems, options `/Qx` and `/arch` are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning. Similarly, on Linux® and Mac OS® X systems, options `-x` and `-m` are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning.

**Alternate Options**

None

**See Also**

- `-m`
- `/arch`
- `-ax`, `/Qax`
- `/Qinstruction`
- `/arch`
- `/Qarch`
- `rcd`, `/Qrcd`

*Enables fast float-to-integer conversions.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures
Syntax

Linux and Mac OS X:
-rcd

Windows:
/Qrcd

Arguments
None

Default
OFF
Floating-point values are truncated when a conversion to an integer
is involved. On Windows, this is the same as specifying /QIfist-.

Description
This option enables fast float-to-integer conversions. It can improve the performance of code
that requires floating-point-to-integer conversions.

The system default floating-point rounding mode is round-to-nearest. However, the C language
requires floating-point values to be truncated when a conversion to an integer is involved. To
do this, the compiler must change the rounding mode to truncation before each
floating-point-to-integer conversion and change it back afterwards.

This option disables the change to truncation of the rounding mode for all floating-point
calculations, including floating point-to-integer conversions. This option can improve
performance, but floating-point conversions to integer will not conform to C semantics.

Alternate Options
Linux and Mac OS X: None
Windows: /QIfist

rct, Qrct
Sets the internal FPU rounding control to Truncate.

IDE Equivalent
None
Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-rct
Windows:
/Qrct

Arguments
None

Default
OFF The compiler uses the default setting for the FPU rounding control.

Description
This option sets the internal FPU rounding control to Truncate.

Alternate Options
None

reserve-kernel-regs
Reserves registers f12-f15 and f32-f127 for use by the kernel.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux:
-reserve-kernel-reg
Mac OS X:
None

Windows:
None

Arguments
None

Default
OFF The compiler can use registers f12-f15 and f32-f127.

Description
This option reserves registers f12-f15 and f32-f127 for use by the kernel.

Alternate Options
None

restrict, Qrestrict
Determines whether pointer disambiguation is enabled with the restrict qualifier.

IDE Equivalent
Windows: Language > Recognize Restrict Keyword
Linux: Language > Recognize Restrict Keyword
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-restrict
-no-restrict
Windows:
/Qrestrict
/Qrestrict-

**Arguments**
None

**Default**
-no-restrict or /Qrestrict-  Pointers are not qualified with the restrict keyword.

**Description**
This option determines whether pointer disambiguation is enabled with the restrict qualifier. Options -restrict (Linux and Mac OS X) and /Qrestrict (Windows OS) enable the recognition of the restrict keyword as defined by the ANSI standard.

By qualifying a pointer with the restrict keyword, you assert that an object accessed by the pointer is only accessed by that pointer in the given scope. You should use the restrict keyword only when this is true. When the assertion is true, the restrict option will have no effect on program correctness, but may allow better optimization.

**Alternate Options**
None

**See Also**

- Qc99

**RTC**
*Enables checking for certain run-time conditions.*

**IDE Equivalent**
Windows: Code Generation > Basic Runtime Checks / Smaller Type Check
Linux: None
Mac OS X: None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/RTC option

Arguments

| option | Specifies the condition to check. Possible values are 1, s, u, or c. |

Default
OFF No checking is performed for these run-time conditions.

Description
This option enables checking for certain run-time conditions. Using the /RTC option sets __MSVC_RUNTIME_CHECKS = 1.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/RTC1</td>
<td>This is the same as specifying /RTC su.</td>
</tr>
<tr>
<td>/RTC s</td>
<td>Enables run-time checks of the stack frame.</td>
</tr>
<tr>
<td>/RTC u</td>
<td>Enables run-time checks for uninitialized variables.</td>
</tr>
<tr>
<td>/RTC c</td>
<td>Enables checks for converting to smaller types.</td>
</tr>
</tbody>
</table>

Alternate Options
None
S
Causes the compiler to compile to an assembly file only and not link.

**IDE Equivalent**
Windows: None
Linux: **Output Files > Generate Assembler Source File**
Mac OS X: **Output Files > Generate Assembler Source File**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
-S
Windows:
/s

**Arguments**
None

**Default**
OFF Normal compilation and linking occur.

**Description**
This option causes the compiler to compile to an assembly file only and not link.
On Linux and Mac OS X systems, the assembly file name has a .s suffix. On Windows systems, the assembly file name has an .asm suffix.

**Alternate Options**
Linux and Mac OS X: None
Windows: `/Fa`

See Also
- Fa

**save-temps, Qsave-temps**
* Tells the compiler to save intermediate files created during compilation.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
- `save-temps`
- `no-save-temps`

**Windows:**
- `/Qsave-temps`
- `/Qsave-temps-`

Arguments
None

Default

**Linux and Mac OS X:** `no-save-temps`
**Windows:** `.obj` files are saved

On Linux and Mac OS X systems, the compiler deletes intermediate files after compilation is completed. On Windows systems, the compiler saves only intermediate object files after compilation is completed.
**Description**

This option tells the compiler to save intermediate files created during compilation. The names of the files saved are based on the name of the source file; the files are saved in the current working directory.

If `-save-temps` or `/Qsave-temps` is specified, the following occurs:

- The object `.o` file (Linux and Mac OS X) or `.obj` file (Windows) is saved.
- The assembler `.s` file (Linux and Mac OS X) or `.asm` file (Windows) is saved if you specified `-use-asm` (Linux or Mac OS X) or `/Quse-asm` (Windows).

If `-no-save-temps` is specified on Linux or Mac OS X systems, the following occurs:

- The `.o` file is put into `/tmp` and deleted after calling `ld`.
- The preprocessed file is not saved after it has been used by the compiler.

If `/Qsave-temps` is specified on Windows systems, the following occurs:

- The `.obj` file is not saved after the linker step.
- The preprocessed file is not saved after it has been used by the compiler.

**NOTE.** This option only saves intermediate files that are normally created during compilation.

**Alternate Options**

None

**Example**

If you compile program `my_foo.c` on a Linux or Mac OS X system and you specify option `-save-temps` and option `-use-asm`, the compilation will produce files `my_foo.o` and `my_foo.s`.

If you compile program `my_foo.c` on a Windows system and you specify option `/Qsave-temps` and option `/Quse-asm`, the compilation will produce files `my_foo.o` and `my_foo.asm`.
**scalar-rep, Qscalar-rep**

Enables scalar replacement performed during loop transformation.

**IDE Equivalent**

None

**Architectures**

IA-32 architecture

**Syntax**

**Linux and Mac OS X:**
- `-scalar-rep`
- `-no-scalar-rep`

**Windows:**
- `/Qscalar-rep`
- `/Qscalar-rep`-

**Arguments**

None

**Default**

- `-no-scalar-rep` or `-Qscalar-rep`

Scalar replacement is not performed during loop transformation.

**Description**

This option enables scalar replacement performed during loop transformation. To use this option, you must also specify `O3`.

**Alternate Options**

None
See Also
•
•
•  

shared
*Tells the compiler to produce a dynamic shared object instead of an executable.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:
-shared

Mac OS X:
None

Windows:
None

Arguments
None

Default
OFF  
The compiler produces an executable.

Description
This option tells the compiler to produce a dynamic shared object (DSO) instead of an executable. This includes linking in all libraries dynamically and passing -shared to the linker.

On systems using IA-32 architecture and Intel® 64 architecture, you must specify option *fpic* for the compilation of each object file you want to include in the shared library.
Alternate Options
None

See Also
•
  • dynamiclib
  • fpic
  • Xlinker

shared-intel
Causes Intel-provided libraries to be linked in dynamically.

IDE Equivalent
Windows: None
Linux: None
Mac OS X: Libraries > Intel Runtime Libraries

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-shared-intel

Windows:
None

Arguments
None

Default
OFF

Intel libraries are linked in statically, with the exception of libguide on Linux* and Mac OS* X systems, where it is linked in dynamically.
**Description**

This option causes Intel-provided libraries to be linked in dynamically. It is the opposite of `-static-intel`.

**NOTE.** On Mac OS X systems, when you set "Intel Runtime Libraries" to "Dynamic", you must also set the DYLD_LIBRARY_PATH environment variable within Xcode or an error will be displayed.

**Alternate Options**

Linux and Mac OS X: `-i-dynamic` (this is a deprecated option)

Windows: None

**See Also**

- `static-intel`

**shared-libgcc**

*Links the GNU libgcc library dynamically.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**

- `-shared-libgcc`

**Mac OS X:**

None

**Windows:**

None
**Arguments**

None

**Default**

- `-shared-libgcc`

The compiler links the `libgcc` library dynamically.

**Description**

This option links the GNU `libgcc` library dynamically. It is the opposite of option `static-libgcc`.

This option is useful when you want to override the default behavior of the `static` option, which causes all libraries to be linked statically.

**Alternate Options**

None

**See Also**

- `% static-libgcc`

**showIncludes**

*Tells the compiler to display a list of the include files.*

**IDE Equivalent**

Windows: **Advanced > Show Includes**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None
Windows:
/showIncludes

Arguments
None

Default
OFF

Description
The compiler does not display a list of the include files.

This option tells the compiler to display a list of the include files. Nested include files (files that are included from the files that you include) are also displayed.

Alternate Options
None

sox, Qsox
 Tells the compiler to save the compilation options and version number in the Linux* OS executable or the Windows* OS object file.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:
-sox
-no-sox

Mac OS X:
None
Windows:
/Qsox
/Qsox-

Arguments
None

Default
-no-sox
or/Qsox-
The compiler does not save the compiler options and version number in the executable.

Description
Tells the compiler to save the compilation options and version number in the Linux* OS executable or the Windows* OS object file.

On Linux systems, the size of the executable on disk is increased slightly by the inclusion of these infotmation strings.

This option forces the compiler to embed in each object file or assembly output a string that contains information about the compiler version and compilation options for each source file that has been compiled.

On Windows systems, the information stays in the object file. On Linux systems, when you link the object files into an executable file, the linker places each of the information strings into the header of the executable. It is then possible to use a tool, such as a strings utility, to determine what options were used to build the executable file.

If -no-sox or /Qsox- is specified, this extra information is not put into the object or assembly output generated by the compiler.

Alternate Options
None

static
Prevents linking with shared libraries.

IDE Equivalent
Windows: None
Linux: **Libraries > Link with static libraries**
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
**Linux:**
- `static`

**Mac OS X:**
None

**Windows:**
None

**Arguments**
None

**Default**
OFF

The compiler links with shared libraries.

**Description**
This option prevents linking with shared libraries. It causes the executable to link all libraries statically.

**Alternate Options**
None

**staticlib**

*Invokes the libtool command to generate static libraries.*

**IDE Equivalent**
None
Architectures
IA-32, Intel® 64 architectures

Syntax
Linux:
None

Mac OS X:
-staticlib

Windows:
None

Arguments
None

Default
OFF  The compiler produces an executable.

Description
This option invokes the libtool command to generate static libraries.
When passed this option, the compiler uses the libtool command to produce a static library instead of an executable when linking.
To build dynamic libraries, you should specify option -dynamiclib or libtool -dynamic <objects>.

Alternate Options
None

See Also
•
  • dynamiclib
static-intel

*Causes Intel-provided libraries to be linked in statically.*

**IDE Equivalent**

Windows: None

Linux: None

Mac OS X: **Libraries > Intel Runtime Libraries**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

- `static-intel`

*Windows:*

None

**Arguments**

None

**Default**

OFF

Intel libraries are linked in statically, with the exception of libguide, which is linked in dynamically.

**Description**

This option causes Intel-provided libraries to be linked in statically. It is the opposite of `-shared-intel`.

**Alternate Options**

*Linux and Mac OS X: i-static (this is a deprecated option)*

Windows: None
**See Also**

- shared-intel

**static-libgcc**

*Links the GNU libgcc library statically.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux:

- `static-libgcc`

Mac OS X:

None

Windows:

None

**Arguments**

None

**Default**

OFF DEFAULT_DESC

**Description**

This option links the GNU libgcc library statically. It is the opposite of option `libgcc`. This option is useful when you want to override the default behavior of the `libgcc` option, which causes all libraries to be linked statically.
Alternate Options
None

See Also
•
  • shared-libgcc

std, Qstd
Tells the compiler to conform to a specific language standard.

IDE Equivalent
Windows: Language > Enable C++0x Support
Language > Enable C99 Support
Linux: Language > ANSI Conformance
Mac OS X: Language > C ANSI Conformance

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-std=val
Windows:
/Qstd=val

Arguments
val Possible values are:
gnu89 Conforms to ISO C90 plus GNU* extensions.
gnu++98 Conforms to the 1998 ISO C++ standard plus GNU extensions.
c++0x Enable support for the following C++0x features:

- Empty macro arguments
- Variadic macros
- Type long long
- Trailing comma in enum definition
- Concatenation of mixed-width string literals
- Extended friend declarations
- Use of ">>" to close two template argument lists
- Relaxed rules for use of "typename"
- Relaxed rules for disambiguation using the "template" keyword
- Copy constructor does not need to be callable on direct reference
- Binding to class rvalue
- "extern template" to suppress instantiation of an entity
- "auto" type specifier
- decltype operator
- static_assert
- compliant __func__
- lambda expressions

Default

-std=gnu89 (default for C) Conforms to ISO C90 plus GNU extensions.
Conforms to the 1998 ISO C++ standard plus GNU* extensions. 
\texttt{-std=gnu++98} (default for C++)

\texttt{/Qstd=c89}

**Description**

Tells the compiler to conform to a specific language standard.

**Alternate Options**

None

**strict-ansi**

*Tells the compiler to implement strict ANSI conformance dialect.*

**IDE Equivalent**

Windows: None

Linux: \textbf{Language > ANSI Conformance}

Mac OS X: \textbf{Language > C ANSI Conformance}

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

\texttt{-strict-ansi}

Windows:

None

**Arguments**

None

**Default**

OFF  The compiler conforms to default standards.
Description
This option tells the compiler to implement strict ANSI conformance dialect. If you need to be compatible with gcc, use the `-ansi` option.

This option sets option `fmath-errno`.

Alternate Options
None

T
*Tells the linker to read link commands from a file.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:
-T<file>

Mac OS X:
None

Windows:
None

Arguments
<file> Is the name of the file.

Default
OFF The linker does not read link commands from a file.
**Description**
This option tells the linker to read link commands from a file.

**Alternate Options**
None

**tbb, Qtbb**
*Tells the compiler to link to the Intel® Threading Building Blocks (Intel® TBB) libraries.*

**IDE Equivalent**
Windows: None
Linux: **Performance Library Build Components > Use Intel(R) Threading Building Blocks Library**
Mac OS X: **Libraries > Use Intel(R) Threading Building Blocks Library**

**Architectures**
IA-32, Intel® 64 architecture, IA-64 architecture

**Syntax**

Linux and Mac OS X:
-
tbb

Windows:
/Qtbb

**Arguments**
None

**Default**
OFF
The compiler does not link to the Intel® TBB libraries.

**Description**
This option tells the compiler to link to the Intel® Threading Building Blocks (Intel® TBB) libraries and include the Intel® TBB headers.
On IA-64 architecture, this option is not available on Windows systems.

**Alternate Options**

None

**Tc**

*Tells the compiler to process a file as a C source file.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Tcfile

**Arguments**

*file*  

Is the file name to be processed as a C source file.

**Default**

OFF  

The compiler uses default rules for determining whether a file is a C source file.

**Description**

This option tells the compiler to process a file as a C source file.

**Alternate Options**

None
See Also

- TC
- Tp

**TC**
*Tells the compiler to process all source or unrecognized file types as C source files.*

**IDE Equivalent**

Windows: Advanced > Compile As

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

None

Windows:

/Tc

**Arguments**

None

**Default**

OFF The compiler uses default rules for determining whether a file is a C source file.

**Description**

This option tells the compiler to process all source or unrecognized file types as C source files.
Alternate Options
None

See Also
•
  • TP
  • TC

tcheck, Qtcheck
Enables analysis of threaded applications.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:
- tcheck

Mac OS X:
None

Windows:
/Qtcheck

Arguments
None

Default
OFF
Threaded applications are not instrumented by the compiler for analysis by Intel® Thread Checker.
**Description**

This option enables analysis of threaded applications.

To use this option, you must have Intel® Thread Checker installed, which is one of the Intel® Threading Analysis Tools. If you do not have this tool installed, the compilation will fail. Remove the `-tcheck` (Linux) or `/Qtcheck` (Windows) option from the command line and recompile.

For more information about Intel® Thread Checker (including an evaluation copy), open the page associated with threading tools at Intel® Software Development Products.

**Alternate Options**

None

**tcollect, Qtcollect**

*Inserts instrumentation probes calling the Intel® Trace Collector API.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**

```
-tcollect[lib]
```

**Mac OS X:**

None

**Windows:**

```
/Qtcollect[=lib]
```

**Arguments**

`lib`  
Is one of the Intel® Trace Collector libraries; for example, VT, VTcs, VTmc, or VTfs. If you do not specify `lib`, the default library is VT.
Default

OFF Instrumentation probes are not inserted into compiled applications.

Description

This option inserts instrumentation probes calling the Intel® Trace Collector API. To use this option, you must have the Intel® Trace Collector installed and set up through one of its set-up scripts. This tool is a component of the Intel® Trace Analyzer and Collector.

This option provides a flexible and convenient way of instrumenting functions of a compiled application. For every function, the entry and exit points are instrumented at compile time to let the Intel® Trace Collector record functions beyond the default MPI calls. For non-MPI applications (for example, threaded or serial), you must ensure that the Intel® Trace Collector is properly initialized (VT_initialize/VT_init).

CAUTION. Be careful with full instrumentation because this feature can produce very large trace files.

For more details, see the Intel® Trace Collector User Guide.

Alternate Options

None

See Also

•

•

• tcollect-filter, Qtcollect-filter

tcollect-filter, Qtcollect-filter

Lets you enable or disable the instrumentation of specified functions.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux:
-tcollect-filter file

Mac OS X:
None

Windows:
/Qtcollect-filter:file

Arguments

file

Arguments are configuration files that list filters, one per line. Each filter consists of a regular expression string and a switch. Strings with leading or trailing white spaces must be quoted. Other strings do not have to be quoted. The switch value can be ON, on, OFF, or off.

Default

Default OFF

Functions are not instrumented. However, if option -tcollect (Linux) or /Qtcollect (Windows) is specified, the filter setting is ".* ON" and all functions get instrumented.

Description

This option lets you enable or disable the instrumentation of specified functions.

During instrumentation, the regular expressions in the file are matched against the function names. The switch specifies whether matching functions are to be instrumented or not. Multiple filters are evaluated from top to bottom with increasing precedence.

The names of the functions to match against are formatted as follows:

- C++ function names are demangled and the C++ class hierarchy is used. Function parameters are stripped to keep the function names shorter.
- The source file name is followed by a colon-separated function name. Source file names should contain the full path, if available. For example:

/home/joe/src/foo.c:FOO_bar
Classes and function names are separated by double colons. For example:
/home/joe/src/foo.cpp:app::foo::bar

You can use option -opt-report (Linux) or /Qopt-report (Windows) to get a full list of file and function names that the compiler recognizes from the compilation unit. This list can be used as the basis for filtering in the configuration file.

To use this option, you must have the Intel® Trace Collector installed and set up through one of its set-up scripts. This tool is a component of the Intel® Trace Analyzer and Collector.

For more details, see the Intel® Trace Collector User Guide.

**Alternate Options**

None

Consider the following filters in a configuration file:
'.*' OFF '.vector.*' ON

The above will cause instrumentation of only those functions having the string 'vector' in their names. No other function will be instrumented. Note that reversing the order of the two lines will prevent instrumentation of all functions.

To get a list of the file or routine strings that can be matched by the regular expression filters, generate an optimization report with tcollect information. For example:

Windows OS: icl /Qtcollect /Qopt-report /Qopt-report-phase tcollect

Linux OS: icc -tcollect -opt-report -opt-report-phase tcollect

**See Also**

- tcollect, Qtcollect

**Tp**

*Tells the compiler to process a file as a C++ source file.*

**IDE Equivalent**

None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/Tp file

Arguments
file
Is the file name to be processed as a C++ source file.

Default
OFF
The compiler uses default rules for determining whether a file is a C++ source file.

Description
This option tells the compiler to process a file as a C++ source file.

Alternate Options
None

See Also
• Tp
• Tc

Kc++, TP
Tells the compiler to process all source or unrecognized file types as C++ source files.
This option is deprecated.

IDE Equivalent
Windows: Advanced > Compile As
Linux: None
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-c++`

Windows:

- `/TP`

**Arguments**

None

**Default**

OFF The compiler uses default rules for determining whether a file is a C++ source file.

**Description**

This option tells the compiler to process all source or unrecognized file types as C++ source files.

**Alternate Options**

None

**tprofile, Qtprofile**

*Generates instrumentation to analyze multi-threading performance.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux:
-tprofile

Mac OS X:
None

Windows:
/Qtprofile

Arguments
None

Default
OFF Instrumentation is not generated by the compiler for analysis by Intel® Thread Profiler.

Description
This option generates instrumentation to analyze multi-threading performance.

To use this option, you must have Intel® Thread Profiler installed, which is one of the Intel® Threading Analysis Tools. If you do not have this tool installed, the compilation will fail. Remove the -tprofile (Linux) or /Qtprofile (Windows) option from the command line and recompile.

For more information about Intel® Thread Profiler (including an evaluation copy), open the page associated with threading tools at Intel® Software Development Products.

Alternate Options
None

traceback
Tells the compiler to generate extra information in the object file to provide source file traceback information when a severe error occurs at run time.

IDE Equivalent
Windows: None
Linux: **Runtime > Generate Traceback Information**

Mac OS X: **Runtime > Generate Traceback Information**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
- `-traceback`
- `-notraceback`

Windows:
- `/traceback`
- `/notraceback`

**Arguments**

None

**Default**

`notraceback`  
No extra information is generated in the object file to produce traceback information.

**Description**

This option tells the compiler to generate extra information in the object file to provide source file traceback information when a severe error occurs at run time. This is intended for use with C code that is to be linked into a Fortran program.

When the severe error occurs, source file, routine name, and line number correlation information is displayed along with call stack hexadecimal addresses (program counter trace).

Note that when a severe error occurs, advanced users can also locate the cause of the error using a map file and the hexadecimal addresses of the stack displayed when the error occurs.

This option increases the size of the executable program, but has no impact on run-time execution speeds.

It functions independently of the debug option.
On Windows systems, traceback sets the /Oy- option, which forces the compiler to use EBP as the stack frame pointer.

On Windows systems, the linker places the traceback information in the executable image, in a section named ".trace". To see which sections are in an image, use the command:

```shell
link -dump -summary your_app_name.exe
```

To see more detailed information, use the command:

```shell
link -dump -headers your_app_name.exe
```

On Linux systems, to display the section headers in the image (including the header for the .trace section, if any), use the command:

```shell
objdump -h your_app_name.exe
```

On Mac OS X systems, to display the section headers in the image, use the command:

```shell
otool -l your_app_name.exe
```

**Alternate Options**

None

**u (Linux*)**

* Tells the compiler the specified symbol is undefined.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-u symbol
```

Windows:

None
**Arguments**
None

**Default**
OFF Standard rules are in effect for variables.

**Description**
This option tells the compiler the specified symbol is undefined.

**Alternate Options**
None

u (Windows*)
_Disables all predefined macros and assertions._

**IDE Equivalent**
Windows: Advanced > Undefine All Preprocessor Definitions

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

- **Linux and Mac OS X:**
  None

- **Windows:**
  /u

**Arguments**
None

**Default**
OFF Defined preprocessor values are in effect until they are undefined.
**Description**
This option disables all predefined macros and assertions.

**Alternate Options**
/QA

**U**
*Undefines any definition currently in effect for the specified macro.*

**IDE Equivalent**
Windows: **Advanced > Undefine Preprocessor Definitions**
Linux: **Preprocessor > Undefine Preprocessor Definitions**
Mac OS X: **Preprocessor > Undefine Preprocessor Definitions**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
```bash
-U name
```
Windows:
```bash
/U name
```

**Arguments**
*name* Is the name of the macro to be undefined.

**Default**
OFF Macro definitions are in effect until they are undefined.

**Description**
This option undefines any definition currently in effect for the specified macro. It is equivalent to a **#undef** preprocessing directive.
On Windows systems, use the `/u` option to undefine all previously defined preprocessor values.

**Alternate Options**

None

**See Also**

- Building Applications: About Preprocessor Options

**unroll, Qunroll**

*Tells the compiler the maximum number of times to unroll loops.*

**IDE Equivalent**

Windows: *Optimization > Loop Unrolling*

Linux: *Optimization > Loop Unroll Count*

Mac OS X: *Optimization > Loop Unrolling*

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

```
-unroll[=n]
```

Windows:

```
/Qunroll[:n]
```

**Arguments**

`n`

Is the maximum number of times a loop can be unrolled. To disable loop unrolling, specify 0. On systems using IA-64 architecture, you can only specify a value of 0.
**Default**

- `unroll`
  The compiler uses default heuristics when unrolling loops.

  or `Qunroll`

**Description**

This option tells the compiler the maximum number of times to unroll loops.

If you do not specify $n$, the optimizer determines how many times loops can be unrolled.

**Alternate Options**

Linux and Mac OS X: `-funroll-loops`

Windows: None

**unroll-aggressive, Qunroll-aggressive**

Determines whether the compiler uses more aggressive unrolling for certain loops.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-unroll-aggressive`
- `-no-unroll-aggressive`

Windows:

/`Qunroll-aggressive`
/`Qunroll-aggressive-`

**Arguments**

None
The compiler uses default heuristics when unrolling loops.

-no-unroll-aggressive  or /Qunroll-aggressive-

Description
This option determines whether the compiler uses more aggressive unrolling for certain loops. The positive form of the option may improve performance.

On IA-32 architecture and Intel® 64 architecture, this option enables aggressive, complete unrolling for loops with small constant trip counts.

On IA-64 architecture, this option enables additional complete unrolling for loops that have multiple exits or outer loops that have a small constant trip count.

Alternate Options
None

use-asm, Quse-asm
Tells the compiler to produce objects through the assembler.

IDE Equivalent
None

Architectures
-use-asm: IA-32 architecture, Intel® 64 architecture, IA-64 architecture
/Quse-asm: IA-64 architecture

Syntax

Linux and Mac OS X:
-use-asm
-no-use-asm

Windows:
/Quse-asm
/Quse-asm-
Arguments
None

Default
-no-use-asm or /Quse-asm-

The compiler produces objects directly.

Description
This option tells the compiler to produce objects through the assembler.

Alternate Options
None

use-intel-optimized-headers, Quse-intel-optimized-headers

Determines whether the performance headers directory is added to the include path search list.

IDE Equivalent
Windows: Optimization > Use Intel Optimized Headers
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-use-intel-optimized-headers

Windows:
/Quse-intel-optimized-headers

Arguments
None
The performance headers directory is not added to the include path search list.

Default
-no-use-intel-optimized-headers or /Quse-intel-optimized-headers-

Description
This option determines whether the performance headers directory is added to the include path search list.

The performance headers directory is added if you specify-use-intel-optimized-headers (Linux and Mac OS X) or/Quse-intel-optimized-headers (Windows OS). Appropriate libraries are also linked in, as needed, for proper functionality.

Alternate Options
None

use-msasm
Tells the compiler to accept the Microsoft* MASM-style inlined assembly format.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:
-use-msasm

Windows:
None

Arguments
None
Default
OFF The compiler accepts the GNU-style inlined assembly format.

Description
This option tells the compiler to accept the Microsoft MASM-style inlined assembly format instead of the GNU-style format.

Alternate Options
None

v
Specifies that driver tool commands should be displayed and executed.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-v [file]

Windows:
None

Arguments
file Is the name of a file.

Default
OFF No tool commands are shown.
**Description**
This option specifies that driver tool commands should be displayed and executed. If you use this option without specifying a file name, the compiler displays only the version of the compiler.

**Alternate Options**
None

**See Also**
- dryrun

**V (Linux* and Mac OS* X)**
*Displays the compiler version information.*

**IDE Equivalent**
Windows: None
Linux: **General > Show Startup Banner**
Mac OS X: **General > Show Startup Banner**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
```
-V
```

**Windows:**
```
/QV
```

**Arguments**
None
**Default**

OFF

The compiler version information is not displayed.

**Description**

This option displays the startup banner, which contains the following compiler version information:

- ID: unique identification number for the compiler
- x.y.z: version of the compiler
- years: years for which the software is copyrighted

This option can be placed anywhere on the command line.

**Alternate Options**

None

**V (Windows*)**

Places the text string specified into the object file being generated by the compiler.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/V:string

**Arguments**

string

Is the text string to go into the object file.
Default
OFF

No text string is placed in the object file.

Description
Places the text string specified into the object file (.obj) being generated by the compiler.
This option places the text string specified into the object file (.obj) being generated by the compiler. The string also gets propagated into the executable file.
For example, this option is useful if you want to place the version number or copyright information into the object and executable.
If the string contains a space or tab, the string must be enclosed by double quotation marks (".). A backslash (\) must precede any double quotation marks contained within the string.

Alternate Options
None

vd
Enable or disable hidden vtordisp field in C++ objects.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None
Windows:
/vdval

Arguments
val Possible values are:
0  disables hidden vtordisp field in C++ objects.
1  enables hidden vtordisp field in C++ objects.

Default
/vd1  The compiler enables hidden vtordisp field in C++ objects.

Description
This option disables or enables hidden vtordisp field in C++ objects.

Alternate Options
None

vec, Qvec
Enables or disables vectorization and transformations enabled for vectorization.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X:
-vec
-no-vec

Windows:
/Qvec
/Qvec-
Arguments
None

Default
-vec  Vectorization is enabled.
/or/Qvec

Description
This option enables or disables vectorization and transformations enabled for vectorization.
To disable vectorization and transformations enabled for vectorization, specify -no-vec (Linux and Mac OS X) or /Qvec- (Windows).

Alternate Options
None

See Also

vec-guard-write, Qvec-guard-write
*Tells the compiler to perform a conditional check in a vectorized loop.*

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures
Syntax

Linux and Mac OS X:
-vec-guard-write
-no-vec-guard-write

Windows:
/Qvec-guard-write
/Qvec-guard-write-

Arguments
None

Default
-no-vec-guard-write or /Qvec-guard-write-
The compiler uses default heuristics when checking vectorized loops.

Description
This option tells the compiler to perform a conditional check in a vectorized loop. This checking avoids unnecessary stores and may improve performance.

Alternate Options
None

vec-report, Qvec-report
Controls the diagnostic information reported by the vectorizer.

IDE Equivalent
Windows: None
Linux: Compilation Diagnostics > Vectorizer Report
Mac OS X: Diagnostics > Vectorizer Diagnostic Report

Architectures
IA-32, Intel® 64 architectures
Syntax

Linux and Mac OS X:
-vec-report\[n\]

Windows:
/Qvec-report\[n\]

Arguments

Is a value denoting which diagnostic messages to report. Possible values are:

0  Tells the vectorizer to report no diagnostic information.
1  Tells the vectorizer to report on vectorized loops.
2  Tells the vectorizer to report on vectorized and non-vectorized loops.
3  Tells the vectorizer to report on vectorized and non-vectorized loops and any proven or assumed data dependences.
4  Tells the vectorizer to report on non-vectorized loops.
5  Tells the vectorizer to report on non-vectorized loops and the reason why they were not vectorized.

Default

If the vectorizer has been enabled and you do not specify \(n\), the compiler reports diagnostics on vectorized loops. If you do not specify the option on the command line, the default is to display no messages.

Description

This option controls the diagnostic information reported by the vectorizer. The vectorizer report is sent to stdout.
If you do not specify \( n \), it is the same as specifying `-vec-report1` (Linux and Mac OS X) or `/Qvec-report1` (Windows).

The vectorizer is enabled when certain compiler options are specified, such as option `-ax` or `-x` (Linux and Mac OS X), option `/Qax` or `/Qx` (Windows), option `/arch:SSE` or `/arch:SSE2` (Windows), and option `fast`.

Alternate Options
None

**vec-threshold, Qvec-threshold**
Sets a threshold for the vectorization of loops.

IDE Equivalent
Windows: **Optimization > Threshold For Vectorization**
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:
`-vec-threshold[n]`

Windows:
`/Qvec-threshold[[:|]n]`

Arguments

\( n \)
Is an integer whose value is the threshold for the vectorization of loops. Possible values are 0 through 100. If \( n \) is 0, loops get vectorized always, regardless of computation work volume.
If \( n \) is 100, loops get vectorized when performance gains are predicted based on the compiler analysis data. Loops get vectorized only if profitable vector-level parallel execution is almost certain. The intermediate 1 to 99 values represent the percentage probability for profitable speed-up. For example, \( n = 50 \) directs the compiler to vectorize only if there is a 50% probability of the code speeding up if executed in vector form.

**Default**

-vec-threshold100 or /Qvec-threshold100

Loops get vectorized only if profitable vector-level parallel execution is almost certain. This is also the default if you do not specify \( n \).

**Description**

This option sets a threshold for the vectorization of loops based on the probability of profitable execution of the vectorized loop in parallel.

This option is useful for loops whose computation work volume cannot be determined at compile-time. The threshold is usually relevant when the loop trip count is unknown at compile-time.

The compiler applies a heuristic that tries to balance the overhead of creating multiple threads versus the amount of work available to be shared amongst the threads.

**Alternate Options**

None

**version**

Display GCC-style version information.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
**Syntax**

*Linux and Mac OS X:*  
--version

*Windows:*  
None

**Arguments**

None

**Default**

OFF

**Description**

Display GCC-style version information.

**Alternate Options**

None

**vmb**

*Selects the smallest representation that the compiler uses for pointers to members.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*  
None

*Windows:*  
/vmb
Arguments
None

Default
OFF The compiler uses default rules to represent pointers to members.

Description
This option selects the smallest representation that the compiler uses for pointers to members. Use this option if you define each class before you declare a pointer to a member of the class.

Alternate Options
None

`vmg`
Selects the general representation that the compiler uses for pointers to members.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
None

**Windows:**
`/vmg`

Arguments
None

Default
OFF The compiler uses default rules to represent pointers to members.
**Description**
This option selects the general representation that the compiler uses for pointers to members. Use this option if you declare a pointer to a member before you define the corresponding class.

**Alternate Options**
None

**vmm**

*Enables pointers to class members with single or multiple inheritance.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

None

*Windows:*

/vmm

**Arguments**
None

**Default**

OFF

The compiler uses default rules to represent pointers to members.

**Description**
This option enables pointers to class members with single or multiple inheritance. To use this option, you must also specify option /vmg.
Alternate Options
None

/vms
Enables pointers to members of single-inheritance classes.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
None

Windows:
/vms

Arguments
None

Default
OFF The compiler uses default rules to represent pointers to members.

Description
This option enables pointers to members of single-inheritance classes. To use this option, you must also specify option /vmg.

Alternate Options
None
**vmv**

*Enables pointers to members of any inheritance type.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/\vmv

**Arguments**

None

**Default**

OFF

The compiler uses default rules to represent pointers to members.

**Description**

This option enables pointers to members of any inheritance type. To use this option, you must also specify option /\vmg.

**Alternate Options**

None
w
Disables all warning messages.

**IDE Equivalent**
Windows: None
Linux: **General > Warning Level**
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
-w
Windows:
/w

**Arguments**
None

**Default**
OFF Default warning messages are enabled.

**Description**
This option disables all warning messages.

**Alternate Options**
Linux and Mac OS X: -W0
Windows: /W0
**w, W**

Specifies the level of diagnostic messages to be generated by the compiler.

**IDE Equivalent**

Windows: **General > Warning Level**

Linux: **General > Warning Level**

Mac OS X: **General > Warning Level**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

\[-w \text{n}\]

Windows:

\[/W\text{n}\]

**Arguments**

\text{n}

Is the level of diagnostic messages to be generated. Possible values are

0  Disables warnings; displays errors.
1  Displays warnings and errors.
2  Displays warnings and errors. This setting is equivalent to level 1 (\text{n}=1).

**Default**

\text{n}=1  The compiler displays warnings and errors.

**Description**

This option specifies the level of diagnostic messages to be generated by the compiler.
Alternate Options

None

\texttt{w, W}

Specifies the level of diagnostic messages to be generated by the compiler.

IDE Equivalent

Windows: General > Warning Level
Linux: General > Warning Level
Mac OS X: General > Warning Level

Architectures

IA-32, Intel\textsuperscript{®} 64, IA-64 architectures

Syntax

Linux and Mac OS X:
\texttt{\neg w n}

Windows:
\texttt{/W n}

Arguments

\texttt{n}

Is the level of diagnostic messages to be generated. Possible values are

- \texttt{0}: Disables warnings; displays errors.
- \texttt{1}: Displays warnings and errors. This setting is equivalent to level 1 (\texttt{n=1}).

Default

\texttt{n=1}

The compiler displays warnings and errors.
Description
This option specifies the level of diagnostic messages to be generated by the compiler.

Alternate Options
None

Wa
Passes options to the assembler for processing.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-Wa,option1[,option2,...]

Windows:
None

Arguments

option Is an assembler option. This option is not processed by the driver and is directly passed to the assembler.

Default
OFF No options are passed to the assembler.

Description
This option passes one or more options to the assembler for processing. If the assembler is not invoked, these options are ignored.

Alternate Options
None
**Wabi**

*Determine whether a warning is issued if generated code is not C++ ABI compliant.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `Wabi`
- `Wno-abi`

**Windows:**

None

**Arguments**

None

**Default**

`Wno-abi`  
No warning is issued when generated code is not C++ ABI compliant.

**Description**

This option determines whether a warning is issued if generated code is not C++ ABI compliant.

**Alternate Options**

None
Wall
Tell the compiler to display errors, warnings, and remarks.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- `Wall`

Windows:
/`Wall`

Arguments
None

Default
OFF Default warning messages are enabled.

Description
This option tells the compiler to display errors, warnings, and remarks.
On Windows, this is the same as specifying the `/W4` option.

Alternate Options
None
**Wbrief**
*Tells the compiler to display a shorter form of diagnostic output.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
*Linux and Mac OS X:*
- `-Wbrief`
*Windows:*
- `/WL`

**Arguments**
None

**Default**
The compiler displays its normal diagnostic output.

**Description**
This option tells the compiler to display a shorter form of diagnostic output. In this form, the original source line is not displayed and the error message text is not wrapped when too long to fit on a single line.

**Alternate Options**
*Linux: None*
*Windows: `/WL`*
**Wcheck**

*Tells the compiler to perform compile-time code checking for certain code.*

**IDE Equivalent**

Windows: None

Linux: *Compilation Diagnostics > Allow Usage Messages*

Mac OS X: *Diagnostics > Allow Usage Messages*

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

-Wcheck

*Windows:*

/Wcheck

**Arguments**

None

**Default**

OFF No compile-time code checking is performed.

**Description**

This option tells the compiler to perform compile-time code checking for certain code. It specifies to check for code that exhibits non-portable behavior, represents a possible unintended code sequence, or possibly affects operation of the program because of a quiet change in the ANSI C Standard.

**Alternate Options**

None
**Wcomment**

Determines whether a warning is issued when /* appears in the middle of a /* */ comment.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-Wcomment

-Wno-comment

**Windows:**

None

**Arguments**

None

**Default**

-Wno-comment

No warning is issued when /* appears in the middle of a /* */ comment.

**Description**

This option determines whether a warning is issued when /* appears in the middle of a /* */ comment.

**Alternate Options**

None
**Wcontext-limit, Qcontext-limit**

*Set the maximum number of template instantiation contexts shown in diagnostic.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-Wcontext-limit=n`

**Windows:**

`/Qcontext-limit:n`

**Arguments**

$n$  
Number of template instantiation contexts.

**Default**

OFF

**Description**

Set maximum number of template instantiation contexts shown in diagnostic.

**Alternate Options**

None

**wd, Qwd**

*Disables a soft diagnostic. This is a deprecated option.*

**IDE Equivalent**

Windows: Advanced > Disable Specific Warnings
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**
- `wdLn[,Ln,...]`

**Windows:**
- `/QwdLn[,Ln,...]`

Arguments

`Ln` Is the number of the diagnostic to disable.

Default

OFF The compiler returns soft diagnostics as usual.

Description

This option disables the soft diagnostic that corresponds to the specified number. If you specify more than one `Ln`, each `Ln` must be separated by a comma.

Alternate Options

None

Wdeprecated

Determines whether warnings are issued for deprecated features.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
**Syntax**

**Linux and Mac OS X:**
- `-Wdeprecated`
- `-Wno-deprecated`

**Windows:**
None

**Arguments**
None

**Default**
- `-Wno-deprecated` No warnings are issued for deprecated features.

**Description**
This option determines whether warnings are issued for deprecated features.

**Alternate Options**
None

**we, Qwe**
Changes a soft diagnostic to an error. *This is a deprecated option.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `weLn[,Ln,...]`
Windows:
/QweLn[,Ln,...]

Arguments
Ln Is the number of the diagnostic to be changed.

Default
OFF The compiler returns soft diagnostics as usual.

Description
This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to an error.
If you specify more than one Ln, each Ln must be separated by a comma.

Alternate Options
None

Weffc++, Qeffc++
This option enables warnings based on certain C++ programming guidelines.

IDE Equivalent
Windows: None
Linux: Compilation Diagnostics > Enable Warnings for Style Guideline Violations
Mac OS X: Diagnostics > Report Effective C++ Violations

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-Weffc++
Windows:
/Qeffc++

Arguments
None

Default
OFF
Diagnoses are not enabled.

Description
This option enables warnings based on certain programming guidelines developed by Scott Meyers in his books on effective C++ programming. With this option, the compiler emits warnings for these guidelines:

- Use `const` and `inline` rather than `#define`. Note that you will only get this in user code, not system header code.
- Use `<iostream>` rather than `<stdio.h>`.
- Use `new` and `delete` rather than `malloc` and `free`.
- Use C++ style comments in preference to C style comments. C comments in system headers are not diagnosed.
- Use `delete` on pointer members in destructors. The compiler diagnoses any pointer that does not have a `delete`.
- Make sure you have a user copy constructor and assignment operator in classes containing pointers.
- Use initialization rather than assignment to members in constructors.
- Make sure the initialization list ordering matches the declaration list ordering in constructors.
- Make sure base classes have virtual destructors.
- Make sure `operator=` returns `*this`.
- Make sure prefix forms of increment and decrement return a `const` object.
- Never overload operators `&&`, `||`, and `.`.
NOTE. The warnings generated by this compiler option are based on the following books from Scott Meyers:

- Effective C++ Second Edition - 50 Specific Ways to Improve Your Programs and Designs
- More Effective C++ - 35 New Ways to Improve Your Programs and Designs

Alternate Options
None

Werror, WX
Changes all warnings to errors.

IDE Equivalent
Windows: General > Treat Warnings As Errors
Linux: Compilation Diagnostics > Treat Warnings As Errors
Mac OS X: Diagnostics > Treat Warnings As Errors

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- Werror

Windows:
/WX

Arguments
None

Default
OFF  
The compiler returns diagnostics as usual.
Description
This option changes all warnings to errors.

Alternate Options
Linux and Mac OS X: -diag-error warn
Windows: /Qdiag-error:warn

Werror-all
Changes all warnings and remarks to errors.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-Werror-all

Windows:
/Werror-all

Arguments
None

Default
OFF

The compiler returns diagnostics as usual.

Description
This option changes all warnings and remarks to errors.

Alternate Options
Linux and Mac OS X: -diag-error warn, remark
**Wextra-tokens**

Determines whether warnings are issued about extra tokens at the end of preprocessor directives.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-Wextra-tokens  
-Wno-extra-tokens

**Windows:**

None

**Arguments**

None

**Default**

-Wno-extra-tokens  The compiler does not warn about extra tokens at the end of preprocessor directives.

**Description**

This option determines whether warnings are issued about extra tokens at the end of preprocessor directives.

**Alternate Options**

None
**Wformat**

Determines whether argument checking is enabled for calls to `printf`, `scanf`, and so forth.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `-Wformat`
- `-Wno-format`

Windows:

None

**Arguments**

None

**Default**

`-Wno-format` Argument checking is not enabled for calls to `printf`, `scanf`, and so forth.

**Description**

This option determines whether argument checking is enabled for calls to `printf`, `scanf`, and so forth.

**Alternate Options**

None
**Wformat-security**

*Determines whether the compiler issues a warning when the use of format functions may cause security problems.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

- `-Wformat-security`
- `-Wno-format-security`

*Windows:*

None

**Arguments**

None

**Default**

- `-Wno-format-security`  
  No warning is issued when the use of format functions may cause security problems.

**Description**

This option determines whether the compiler issues a warning when the use of format functions may cause security problems.

When `-Wformat-security` is specified, it warns about uses of format functions where the format string is not a string literal and there are no format arguments.

**Alternate Options**

None
**Winline**

*Enables diagnostics about what is inlined and what is not inlined.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-Winline

**Windows:**

None

**Arguments**

None

**Default**

OFF

No diagnostics are produced about what is inlined and what is not inlined.

**Description**

This option enables diagnostics about what is inlined and what is not inlined. The diagnostics depend on what interprocedural functionality is available.

**Alternate Options**

None
Wl

*Passes options to the linker for processing.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

-Wl,option1[,option2,...]

*Windows:*

None

**Arguments**

*option*  
Is a linker option. This option is not processed by the driver and is directly passed to the linker.

**Default**

OFF  
No options are passed to the linker.

**Description**

This option passes one or more options to the linker for processing. If the linker is not invoked, these options are ignored.

This option is equivalent to specifying option -Qoption,link,options.

**Alternate Options**

None

**See Also**

- Qoption
WL
Tells the compiler to display a shorter form of diagnostic output.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
See Wbrief.

Windows:
/WL

Arguments
None

Default
OFF The compiler displays its normal diagnostic output.

Description
This option tells the compiler to display a shorter form of diagnostic output. In this form, the original source line is not displayed and the error message text is not wrapped when too long to fit on a single line.

Alternate Options
Linux: -Wbrief
Windows: None
**Wmain**

*Determines whether a warning is issued if the return type of* main *is not expected.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-Wmain`
- `-Wno-main`

**Windows:**

None

**Arguments**

None

**Default**

- `-Wno-main`  
  No warning is issued if the return type of `main` is not expected.

**Description**

This option determines whether a warning is issued if the return type of `main` is not expected.

**Alternate Options**

None
**Wmissing-declarations**

*Determines whether warnings are issued for global functions and variables without prior declaration.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-Wmissing-declarations`
- `-Wno-missing-declarations`

**Windows:**

None

**Arguments**

None

**Default**

- `-Wno-missing-declarations`

No warnings are issued for global functions and variables without prior declaration.

**Description**

This option determines whether warnings are issued for global functions and variables without prior declaration.

**Alternate Options**

None
Wmissing-prototypes
Determines whether warnings are issued for missing prototypes.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-Wmissing-prototypes
-Wno-missing-prototypes

Windows:
None

Arguments
None

Default
-Wno-missing-prototypes No warnings are issued for missing prototypes.

Description
Determines whether warnings are issued for missing prototypes.

Alternate Options
None
Wnon-virtual-dtor

Issue a warning when a class appears to be polymorphic, yet it declares a non-virtual one.

IDE Equivalent

Mac OS X: Diagnostics > Report Non-Virtual Destructor

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-Wnon-virtual-dtor

Windows:

None

Arguments

None

Default

OFF The compiler does not issue a warning.

Description

Issue a warning when a class appears to be polymorphic, yet it declares a non-virtual one. This option is supported in C++ only.

Alternate Options

None
wn, Qwn
Controls the number of errors displayed before compilation stops. This is a deprecated option.

**IDE Equivalent**
Windows: Diagnostics > Error Limit
Linux: Compilation Diagnostics > Set Error Limit
Mac OS X: Diagnostics > Error Limit

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
\(~wn\)

Windows:
/\Qwn\n
**Arguments**

\(n\) Is the number of errors to display.

**Default**

100 The compiler displays a maximum of 100 errors before aborting compilation.

**Description**
This option controls the number of errors displayed before compilation stops.

**Alternate Options**
None
wo, Qwo
Tells the compiler to issue one or more diagnostic messages only once. This is a deprecated option.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-woLn[,Ln,...]

Windows:
/QwoLn[,Ln,...]

Arguments
Ln Is the number of the diagnostic.

Default
OFF

Description
Specifies the ID number of one or more messages. If you specify more than one Ln, each Ln must be separated by a comma.

Alternate Options
None

Wp
Passes options to the preprocessor.

IDE Equivalent
None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:

```
-Wp,option1[,option2,...]
```

Windows:
None

Arguments

| option | Is a preprocessor option. This option is not processed by the driver and is directly passed to the preprocessor. |

Default

OFF No options are passed to the preprocessor.

Description

This option passes one or more options to the preprocessor. If the preprocessor is not invoked, these options are ignored.

This option is equivalent to specifying option `-Qoption, cpp, options`.

Alternate Options
None

See Also

- `Qoption`
- `Wp64`

Wp64

_Tells the compiler to display diagnostics for 64-bit porting._

IDE Equivalent

Windows: **General > Detect 64-bit Portability Issues**
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-Wp64

Windows:
/Wp64

Arguments
None

Default
OFF

Description
The compiler does not display diagnostics for 64-bit porting.

Alternate Options
None

Wpointer-arith
Determines whether warnings are issued for questionable pointer arithmetic.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:

- -Wpointer-arith
- -Wno-pointer-arith

Windows:

None

Arguments

None

Default

- -Wno-pointer-arith  No warnings are issued for questionable pointer arithmetic.

Description

Determines whether warnings are issued for questionable pointer arithmetic.

Alternate Options

None

Wport

*Tells the compiler to issue portability diagnostics.*

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

None
Windows:
/Wport

Arguments
None

Default
OFF The compiler issues default diagnostics.

Description
This option tells the compiler to issue portability diagnostics.

Alternate Options
None

Wpragma-once
Determine whether a warning is issued about the use of #pragma once.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
- Wpragma-once
- WHO-pragma-once

Windows:
None

Arguments
None
**Default**

- `-Wno-pragmaconce`  
  No warning is issued about the use of `#pragma once`.

**Description**

This option determines whether a warning is issued about the use of `#pragma once`.

**Alternate Options**

None

**wr, Qwr**

*Changes a soft diagnostic to an remark. This is a deprecated option.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-wrLn[,Ln,...]`

**Windows:**

- `/QwrLn[,Ln,...]`

**Arguments**

*Ln*  
Is the number of the diagnostic to be changed.

**Default**

OFF  
The compiler returns soft diagnostics as usual.

**Description**

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to a remark.
If you specify more than one \( L_n \), each \( L_n \) must be separated by a comma.

**Alternate Options**
None

**Wreorder**
*Issue a warning when the order of member initializers does not match the order in which they must be executed.*

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
-Wreorder

**Windows:**
None

**Arguments**
None

**Default**
OFF The compiler does not issue a warning.

**Description**
Issue a warning when the order of member initializers does not match the order in which they must be executed. This option is supported with C++ only.

**Alternate Options**
None
**Wreturn-type**

Determines whether warnings are issued when a function uses the default `int` return type or when a return statement is used in a void function.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-Wreturn-type`
- `-Wno-return-type`

**Windows:**

None

**Arguments**

None

**Default**

- `-Wno-return-type`

No warnings are issued when a function uses the default `int` return type or when a return statement is used in a void function.

**Description**

This option determines whether warnings are issued when a function uses the default `int` return type or when a return statement is used in a void function.

**Alternate Options**

None
**Wshadow**

*Determines whether a warning is issued when a variable declaration hides a previous declaration.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:*

- `-Wshadow`
- `-Wno-shadow`

*Windows:*

None

**Arguments**

None

**Default**

- `-Wno-shadow`  
  No warning is issued when a variable declaration hides a previous declaration.

**Description**

This option determines whether a warning is issued when a variable declaration hides a previous declaration. Same as `-Ww1599`.

**Alternate Options**

None
**Wstrict-prototypes**

Determines whether warnings are issued for functions declared or defined without specified argument types.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

`-Wstrict-prototypes`

`-Wno-strict-prototypes`

**Windows:**

None

**Arguments**

None

**Default**

`-Wno-strict-prototypes` No warnings are issued for functions declared or defined without specified argument types.

**Description**

This option determines whether warnings are issued for functions declared or defined without specified argument types.

**Alternate Options**

None
**Wtrigraphs**

Determines whether warnings are issued if any trigraphs are encountered that might change the meaning of the program.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**
- `-Wtrigraphs`
- `-Wno-trigraphs`

**Windows:**

None

**Arguments**

None

**Default**

- `-Wno-trigraphs`

No warnings are issued if any trigraphs are encountered that might change the meaning of the program.

**Description**

This option determines whether warnings are issued if any trigraphs are encountered that might change the meaning of the program.

**Alternate Options**

None
**Wuninitialized**

Determines whether a warning is issued if a variable is used before being initialized.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-Wuninitialized

-Wno-uninitialized

**Windows:**

None

**Arguments**

None

**Default**

-Wno-uninitialized  No warning is issued if a variable is used before being initialized.

**Description**

This option determines whether a warning is issued if a variable is used before being initialized. Equivalent to -w592 and -wd592.

**Alternate Options**

-ww592 and -wd592
**Wunknown-pragmas**

Determines whether a warning is issued if an unknown `#pragma` directive is used.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `-Wunknown-pragmas`
- `-Wno-unknown-pragmas`

**Windows:**

None

**Arguments**

None

**Default**

- `-Wunknown-pragmas`  
  No warning is issued if an unknown `#pragma` directive is used.

**Description**

This option determines whether a warning is issued if an unknown `#pragma` directive is used.

**Alternate Options**

None
**Wunused-function**

Determines whether a warning is issued if a declared function is not used.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

- `Wunused-function`
- `Wno-unused-function`

Windows:

None

**Arguments**

None

**Default**

- `Wno-unused-function`  No warning is issued if a declared function is not used.

**Description**

This option determines whether a warning is issued if a declared function is not used.

**Alternate Options**

None
**Wunused-variable**

Determines whether a warning is issued if a local or non-constant static variable is unused after being declared.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- `Wunused-variable`
- `Wno-unused-variable`

**Windows:**

None

**Arguments**

None

**Default**

`Wno-unused-variable`  No warning is issued if a local or non-constant static variable is unused after being declared.

**Description**

This option determines whether a warning is issued if a local or non-constant static variable is unused after being declared.

**Alternate Options**

None
**ww, Qww**

Changes a soft diagnostic to a warning. This is a deprecated option.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

`-ww Ln[,Ln,...]`

Windows:

`/Qww Ln[,Ln,...]`

**Arguments**

Ln  
Is the number of the diagnostic to be changed.

**Default**

OFF  
The compiler returns soft diagnostics as usual.

**Description**

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to a warning.

If you specify more than one Ln, each Ln must be separated by a comma.

**Alternate Options**

None
**Wwrite-strings**

*Issues a diagnostic message if const char * is converted to (non-const) char*.  

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

-Wwrite-strings

**Windows:**

None

**Arguments**

None

**Default**

OFF

No diagnostic message is issued if const char * is converted to (non-const) char*.

**Description**

This option issues a diagnostic message if const char* is converted to (non-const) char *.

**Alternate Options**

None

**Werror, WX**

*Changes all warnings to errors.*

**IDE Equivalent**

Windows: **General > Treat Warnings As Errors**
Linux: **Compilation Diagnostics > Treat Warnings As Errors**
Mac OS X: **Diagnostics > Treat Warnings As Errors**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:
- `--error`

Windows:
- `/WX`

**Arguments**
None

**Default**
OFF  The compiler returns diagnostics as usual.

**Description**
This option changes all warnings to errors.

**Alternate Options**
Linux and Mac OS X: `--diag-error warn`
Windows: `/Qdiag-error:warn`

**x, Qx**
*Tells the compiler to generate optimized code specialized for the Intel processor that executes your program.*

**IDE Equivalent**
Windows: **Code Generation > Intel Processor-Specific Optimization**
Linux: **Code Generation > Intel Processor-Specific Optimization**
Mac OS X: **Code Generation > Intel Processor-Specific Optimization**

**Architectures**
IA-32, Intel® 64 architectures

**Syntax**
Linux and Mac OS X:
-xprocessor

Windows:
/Qxprocessor

**Arguments**
processor

Indicates the processor for which code is generated. Many of the following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE). Possible values are:

- **Host**
  
  Can generate instructions for the highest instruction set available on the compilation host processor.
  
  On Intel processors, this may correspond to the most suitable -x (Linux* and Mac OS* X) or /Qx (Windows*) option. On non-Intel processors, this may correspond to the most suitable -m (Linux and Mac OS X) or /arch (Windows) option.
  
  The resulting executable may not run on a processor different from the host in the following cases:

  - If the processor does not support all of the instructions supported by the host processor.
  - If the host is an Intel processor and the other processor is a non-Intel processor.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVX</td>
<td>Optimizes for Intel processors that support Intel® Advanced Vector Extensions (Intel® AVX).</td>
</tr>
<tr>
<td>SSE4.2</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.</td>
</tr>
<tr>
<td>SSE4.1</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture. This replaces value S, which is deprecated.</td>
</tr>
<tr>
<td>SSE3_ATOM</td>
<td>Optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology. Can generate MOVBE instructions, depending on the setting of option -minstruction (Linux and Mac OS) or /Qinstruction (Windows).</td>
</tr>
<tr>
<td>SSSE3</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. For Mac OS* X systems, this value is only supported on Intel® 64 architecture. This replaces value T, which is deprecated.</td>
</tr>
<tr>
<td>SSE3</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. For Mac OS*</td>
</tr>
</tbody>
</table>
X systems, this value is only supported on IA-32 architecture. This replaces value P, which is deprecated.

SSE2

Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2. This value is not available on Mac OS® X systems. This replaces value N, which is deprecated.

Default

Windows* systems: None
Linux* systems: None
Mac OS® X systems using IA-32 architecture: SSE3
Mac OS® X systems using Intel® 64 architecture: SSSE3

On Windows systems, if neither /Qx nor /arch is specified, the default is /arch:SSE2.
On Linux systems, if neither -x nor -m is specified, the default is -msse2.

Description

This option tells the compiler to generate optimized code specialized for the Intel processor that executes your program. It also enables optimizations in addition to Intel processor-specific optimizations. The specialized code generated by this option may run only on a subset of Intel processors.

This option can enable optimizations depending on the argument specified. For example, it may enable Intel® Streaming SIMD Extensions 4 (Intel® SSE4), Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Intel® Streaming SIMD Extensions 2 (Intel® SSE2), or Intel® Streaming SIMD Extensions (Intel® SSE) instructions.

The binaries produced by these values will run on Intel processors that support all of the features for the targeted processor. For example, binaries produced with SSE3 will run on an Intel® Core™ 2 Duo processor, because that processor completely supports all of the capabilities of the Intel® Pentium® 4 processor, which the SSE3 value targets. Specifying the SSSE3 value has the potential of using more features and optimizations available to the Intel® Core™ 2 Duo processor.
Do not use processor values to create binaries that will execute on a processor that is not compatible with the targeted processor. The resulting program may fail with an illegal instruction exception or display other unexpected behavior. For example, binaries produced with SSE3 may produce code that will not run on Intel® Pentium® III processors or earlier processors that do not support SSE3 instructions.

Compiling the function main() with any of the processor values produces binaries that display a fatal run-time error if they are executed on unsupported processors. For more information, see Optimizing Applications.

If you specify more than one processor value, code is generated for only the highest-performing processor specified. The highest-performing to lowest-performing processor values are: SSE4.2, SSE4.1, SSSE3, SSE3, SSE2. Note that processor values AVX and SSE3_ATOM do not fit within this group.

Compiler options m and arch produce binaries that should run on processors not made by Intel that implement the same capabilities as the corresponding Intel processors.

Previous value O is deprecated and has been replaced by option -msse3 (Linux and Mac OS X) and option /arch:SSE3 (Windows).

Previous values W and K are deprecated. The details on replacements are as follows:

- Mac OS X systems: On these systems, there is no exact replacement for W or K. You can upgrade to the default option -msse3 (IA-32 architecture) or option -mssse3 (Intel® 64 architecture).

- Windows and Linux systems: The replacement for W is -msse2 (Linux) or /arch:SSE2 (Windows). There is no exact replacement for K. However, on Windows systems, /QxK is interpreted as /arch:IA32; on Linux systems, -xK is interpreted as -mia32. You can also do one of the following:
  - Upgrade to option -msse2 (Linux) or option /arch:SSE2 (Windows). This will produce one code path that is specialized for Intel® SSE2. It will not run on earlier processors
  - Specify the two option combination -mia32 -axSSE2 (Linux) or /arch:IA32 /QaxSSE2 (Windows). This combination will produce an executable that runs on any processor with IA-32 architecture but with an additional specialized Intel® SSE2 code path.

The -x and /Qx options enable additional optimizations not enabled with option -m or option /arch.
On Windows* systems, options /Qx and /arch are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning. Similarly, on Linux* and Mac OS* X systems, options -x and -m are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning.

**Alternate Options**

None

**See Also**

•

•

• ax, Qax

• m

• arch

• minstruction, Qinstruction

• Targeting IA-32 and Intel 64 Architecture Processors Manually

**x (type option)**

*All source files found subsequent to -x type will be recognized as a particular type.*

**IDE Equivalent**

Windows: None

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

- x type

**Windows:**

None
Arguments

`type` is the type of source file. Possible values are:

- `c` C source file
- `c++` C++ source file
- `c-header` C header file
- `cpp-output` C pre-processed file
- `c++-cpp-output` C++ pre-processed file
- `assembler` Assembly file
- `assembler-with-cpp` Assembly file that needs to be preprocessed
- `none` Disable recognition, and revert to file extension

Default

`none` Disable recognition and revert to file extension.

Description

All source files found subsequent to `-x` will be recognized as a particular type.

Alternate Options

None

Example

Suppose you want to compile the following C and C++ source files whose extensions are not recognized by the compiler:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>file1.c99</td>
<td>C</td>
</tr>
<tr>
<td>file2.cplusplus</td>
<td>C++</td>
</tr>
</tbody>
</table>

We will also include these files whose extensions are recognized:
The command-line invocation using the \(-x\) option follows:
```
icpc -x c file1.c99 -x c++ file2.cplusplus -x none file3.c file4.cpp
```

**X**

*Removes standard directories from the include file search path.*

**IDE Equivalent**

Windows: *Preprocessor > Ignore Standard Include Path*

Linux: *Preprocessor > Ignore Standard Include Path*

Mac OS X: *Preprocessor > Ignore Standard Include Path*

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

\(-x\)

Windows:

\(/x\)

**Arguments**

None

**Default**

OFF  

*Standard directories are in the include file search path.*
Description

This option removes standard directories from the include file search path. It prevents the compiler from searching the default path specified by the INCLUDE environment variable.

On Linux and Mac OS X systems, specifying -X (or -noinclude) prevents the compiler from searching in /usr/include for files specified in an INCLUDE statement.

You can use this option with the I option to prevent the compiler from searching the default path for include files and direct it to use an alternate path.

Alternate Options

Linux and Mac OS X: -nostdinc

Windows: None

See Also

•
  • I

Xlinker

Passes a linker option directly to the linker.

IDE Equivalent

Windows: None

Linux: Linker > Miscellaneous > Other Options

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:

-Xlinker option

Windows:

None
**Arguments**

*option*  
Is a linker option.

**Default**

OFF  
No options are passed directly to the linker.

**Description**

This option passes a linker option directly to the linker.  
If `-Xlinker -shared` is specified, only `-shared` is passed to the linker and no special work is done to ensure proper linkage for generating a shared object. `-Xlinker` just takes whatever arguments are supplied and passes them directly to the linker.

If you want to pass compound options to the linker, for example "-L $HOME/lib", you must use the following method:

- `-Xlinker -L -Xlinker $HOME/lib`

**Alternate Options**

None

**See Also**

- shared
- link

**Y-**

*Tells the compiler to ignore all other precompiled header files.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:
None

Windows:
/Y-

Arguments
None

Default
OFF

The compiler recognizes precompiled header files when certain compiler options are specified.

Description

This option tells the compiler to ignore all other precompiled header files.

Alternate Options

None

See Also

•
  • Yc
  • Yu
  • YX

pch-create, Yc

Lets you create and specify a name for a precompiled header file.

IDE Equivalent

Windows: Precompiled Headers > Create-Use Precompiled Header / Create-Use PCH

Through File

Linux: None
Mac OS X: None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
-pch-create file

Windows:
/Yc file

Arguments

file Is the name for the precompiled header file.

Default

OFF The compiler does not create or use precompiled headers unless you tell it to do so.

Description

This option lets you specify a name for a precompiled header (PCH) file. It is supported only for single source file compilations.

The .pchi extension is not automatically appended to the file name.

This option cannot be used in the same compilation as the -pch-use option.

Depending on how you organize the header files listed in your sources, this option may increase compile times.

To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

Alternate Options
None

Example

Consider the following command line:

icpc -pch-create /pch/source32.pchi source.cpp
It produces the following output:
"source.cpp": creating precompiled header file "/pch/source32.pch"

**See Also**

- Using Precompiled Header Files

**Yd**

*Tells the compiler to add complete debugging information in all object files created from a precompiled header (.pch) file when option /Zi or /Z7 is specified.*

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Yd

**Arguments**

None

**Default**

OFF

*If /Zi or /Z7 is specified when you are compiling with a precompiled header file using /Yc or /Yu, only one .obj file contains the common debugging information.*
Description
This option tells the compiler that complete debugging information should be added to all object files created from a precompiled header file when option /zi or /Z7 is specified. It affects precompiled header (.pch) files that were created by specifying the /Yc option.

Option /Yd has no effect if option /zi or /Z7 is not specified.

When option /zi or /Z7 is specified and option /Yd is omitted, the compiler stores common debugging information in only the first object (.obj) file created from the .pch file. This information is not inserted into any .obj files subsequently created from the .pch file, only cross-references to the information are inserted.

Alternate Options
None

Yu
*Tells the compiler to use a precompiled header file.*

IDE Equivalent
Windows: **Language > Create/Use Precompiled Header**
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
**Linux and Mac OS X:**
None

**Windows:**
/Yu file

Arguments
*file* Is the name of the precompiled header file to use.
The compiler does not use precompiled header files unless it is

told to do so.

This option tells the compiler to use a precompiled header file.

When this option is specified, the Microsoft Visual C++* compiler ignores all text, including
declarations preceding the #include statement of the specified file.

None

Tells the compiler to use a precompiled header file
or to create one if none exists.

Windows: Language > Create/Use Precompiled Header

Linux: None

Mac OS X: None

IA-32, Intel® 64, IA-64 architectures

None

None
Windows:
/YX[file]

Arguments
file
Is the name of the precompiled header file to use.

Default
OFF
The compiler does not use or create precompiled header files unless it is told to do so.

Description
This option tells the compiler to use a precompiled header file or to create one if none exists.

Alternate Options
None

See Also
• Y-
• Yc
• Yu
• Fp

\texttt{g, Zi, Z7}
\textit{Tells the compiler to generate full debugging information in the object file.}

IDE Equivalent
Windows: \texttt{General > Debug Information Format}
Linux: \texttt{General > Include Debug Information}
Mac OS X: \texttt{General > Generate Debug Information}
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-g
Windows:
/zi
/z7

Arguments
None

Default
OFF No debugging information is produced in the object file.

Description
This option tells the compiler to generate symbolic debugging information in the object file for use by debuggers.

The compiler does not support the generation of debugging information in assemblable files. If you specify this option, the resulting object file will contain debugging information, but the assemblable file will not.

This option turns off O2 and makes O0 (Linux and Mac OS X) or Od (Windows) the default unless O2 (or another O option) is explicitly specified in the same command line.

On Linux systems using Intel® 64 architecture and Linux and Mac OS X systems using IA-32 architecture, specifying the -g or -O0 option sets the -fno-omit-frame-pointer option.

Alternate Options
Linux: None
Windows: /ZI, /debug
**Za**

*Disable Microsoft Visual C++ compiler language extensions.*

**IDE Equivalent**

Windows: **Language > Disable Language Extensions**
Linux: None
Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
None

Windows:
/za

**Arguments**

None

**Default**

OFF                     The compiler provides support for extended ANSI C.

**Description**

Disable Microsoft Visual C++ compiler language extensions.

**Alternate Options**

None

**See Also**

* Za
  * Ze
• **Zc**

**Zc**

*Let you specify ANSI C standard conformance for certain language features.*

**IDE Equivalent**

Windows: **Language > Treat wchar_t as Built-in Type / Force Conformance In For Loop Scope**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:

None

Windows:

```
/Zc:arg[,arg]
```

**Arguments**

**arg**

Is the language feature for which you want standard conformance. Possible values are:

- **forScope**
  - Enforce standard behavior for initializers of for loops.
- **wchar_t**
  - Specify that wchar_t is a native data type.

**Default**

**OFF**

```
/Zc:forScope, wchar_t is disabled if /Qvc8 is not specified.
```

**ON**

```
/Zc:forScope, wchar_t is enabled when /Qvc8 is specified.
```
**Description**
This option lets you specify ANSI C standard conformance for certain language features when you also specify `/ze`.

**Alternate Options**
None

**See Also**
- `Ze`

**Zd**
This option has been deprecated. Use keyword `minimal` in `debug (Windows*)`.

**Ze**
Enables Microsoft Visual C++* compiler language extensions. This option has been deprecated.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:
None

Windows:
`/Ze`

**Arguments**
None
Default

ON The compiler provides support for extended ANSI C.

Description
This option enables Microsoft Visual C++* compiler language extensions.

Alternate Options
None

See Also
• Zc
• Za

Zg
Tells the compiler to generate function prototypes.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:
None

Windows:
/Zg

Arguments
None

Default

OFF The compiler does not create function prototypes.
Description
This option tells the compiler to generate function prototypes.

Alternate Options
None

\texttt{g, Zi, Z7}
\textit{Tells the compiler to generate full debugging information in the object file.}

IDE Equivalent
Windows: General > Debug Information Format
Linux: General > Include Debug Information
Mac OS X: General > Generate Debug Information

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
\texttt{-g}

Windows:
\texttt{/zi}
\texttt{/z7}

Arguments
None

Default
OFF No debugging information is produced in the object file.
Description
This option tells the compiler to generate symbolic debugging information in the object file for use by debuggers.

The compiler does not support the generation of debugging information in assemblable files. If you specify this option, the resulting object file will contain debugging information, but the assemblable file will not.

This option turns off \texttt{O2} and makes \texttt{O0} (Linux and Mac OS X) or \texttt{Od} (Windows) the default unless \texttt{O2} (or another \texttt{O} option) is explicitly specified in the same command line.

On Linux systems using Intel\textsuperscript{\textregistered} 64 architecture and Linux and Mac OS X systems using IA-32 architecture, specifying the \texttt{-g} or \texttt{-O0} option sets the \texttt{-fno-omit-frame-pointer} option.

Alternate Options
Linux: None
Windows: \texttt{/ZI}, \texttt{/debug}

\texttt{ZI}
\textit{Tells the compiler to generate full debugging information in the object file.}

IDE Equivalent
Windows: General > Debug Information Format
Linux: None
Mac OS X: None

Architectures
IA-32, Intel\textsuperscript{\textregistered} 64, IA-64 architectures

Syntax
Linux and Mac OS X:
See \texttt{g}.

Windows:
\texttt{/ZI}
**Arguments**

None

**Default**

OFF

No debugging information is produced in the object file.

**Description**

For details, see `/zi`.

**Alternate Options**

- **Linux**: `-g`
- **Windows**: `/Zi, /Zl`

**Zi**

_Causes library names to be omitted from the object file._

**IDE Equivalent**

**Windows**: Advanced > Omit Default Library Names

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:**

None

**Windows:**

/Zi

**Arguments**

None
Default
OFF
Default or specified library names are included in the object file.

Description
This option causes library names to be omitted from the object file.

Alternate Options
None

Zp
Specifies alignment for structures on byte boundaries.

IDEEquivalent
Windows: Code Generation > Struct Member Alignment
Linux: Data > Structure Member Alignment
Mac OS X: Data > Structure Member Alignment

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:
-Zp\[n\]
Windows:
/Zp\[n\]

Arguments
n
Is the byte size boundary. Possible values are 1, 2, 4, 8, or 16.
Default

\texttt{Zp16} Structures are aligned on either size boundary 16 or the boundary that will naturally align them.

Description
This option specifies alignment for structures on byte boundaries.
If you do not specify \( n \), you get \texttt{Zp16}.

Alternate Options
None

\texttt{Zs}
\textit{Tells the compiler to check only for correct syntax.}

IDE Equivalent
None

Architectures
IA-32, Intel\textsuperscript{®} 64, IA-64 architectures

Syntax

\textbf{Linux and Mac OS X:}
None

\textbf{Windows:}
\texttt{/Zs}

Arguments
None

Default
OFF Normal compilation is performed.

Description
This option tells the compiler to check only for correct syntax.
Alternate Options

Linux: -syntax, -fsyntax-only

Windows: None

Zx

Disables certain optimizations that make it difficult to debug optimized code.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux and Mac OS X:

None

Windows:

/Zx

Arguments

None

Default

OFF Optimizations are not disabled.

Description

Disables certain optimizations, such as software pipelining and global scheduling, that make it difficult to debug resultant code because of speculation.

Alternate Options

None
The table in this section summarizes Intel® C++ compiler options used on Linux* OS and Mac OS* X systems. Each summary also shows the equivalent compiler options on Windows* operating systems, if any.

Some compiler options are only available on systems using certain architectures, as indicated by these labels:

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td>The option is available on systems using IA-32 architecture.</td>
</tr>
<tr>
<td>i64em</td>
<td>The option is available on systems using Intel® 64 architecture.</td>
</tr>
<tr>
<td>i64</td>
<td>The option is available on systems using IA-64 architecture.</td>
</tr>
</tbody>
</table>

If "only" appears in the label, the option is only available on the identified system or architecture.

If no label appears, the option is available on all supported systems and architectures.

For more details on the options, refer to the Alphabetical Compiler Options section.

The Intel® C++ Compiler includes the Intel® Compiler Option Mapping tool. This tool lets you find equivalent options by specifying compiler option -map-opts (Linux OS and Mac OS X) or /Qmap-opts (Windows OS).

For information on conventions used in this table, see Conventions.

Quick Reference of Linux OS and Mac OS X Options

The following table summarizes all supported Linux OS and Mac OS X options. It also shows equivalent Windows* OS options, if any.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A&lt;name&gt;[(&lt;value&gt;)]</td>
<td>Specifies an identifier for an assertion.</td>
<td>OFF</td>
<td>/QA&lt;name&gt;[(&lt;value&gt;)]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><img src="equiv" alt="equiv" /></td>
<td>Determines whether function arguments can alias each other. Deprecated; use -fargument- [no]alias.</td>
<td><img src="default" alt="default" /></td>
<td><img src="equiv" alt="equiv" /></td>
</tr>
<tr>
<td><img src="equiv" alt="equiv" /></td>
<td>Determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const.</td>
<td><img src="default" alt="default" /></td>
<td><img src="equiv" alt="equiv" /></td>
</tr>
<tr>
<td><img src="equiv" alt="equiv" /></td>
<td>Determines whether variables and arrays are naturally aligned.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><img src="equiv" alt="equiv" /></td>
<td>Enables language compatibility with the gcc option -ansi.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><img src="equiv" alt="equiv" /></td>
<td>Enables or disables use of ANSI aliasing rules in optimizations.</td>
<td><img src="default" alt="default" /></td>
<td><img src="equiv" alt="equiv" /></td>
</tr>
<tr>
<td><img src="equiv" alt="equiv" /></td>
<td>Instructs the compiler to analyze the program to determine if there are 64-bit pointers which can be safely shrunk into 32-bit pointers.</td>
<td>OFF</td>
<td><img src="equiv" alt="equiv" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-ax&lt;processor&gt;</code></td>
<td>Tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit.</td>
<td>OFF</td>
<td><code>/Qax&lt;processor&gt;</code> (i32, i64em)</td>
</tr>
<tr>
<td><code>-B&lt;dir&gt;</code></td>
<td>Specifies a directory that can be used to find include files, libraries, and executables.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-Bdynamic</code></td>
<td>Enables dynamic linking of libraries at run time.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-Bstatic</code></td>
<td>Enables static linking of a user's library.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-c</code></td>
<td>Prevents linking.</td>
<td>OFF</td>
<td><code>/c</code></td>
</tr>
<tr>
<td><code>-C</code></td>
<td>Place comments in preprocessed source output.</td>
<td>OFF</td>
<td><code>/C</code></td>
</tr>
<tr>
<td><code>-[no-]c99</code></td>
<td>Determines whether C99 support is enabled for C programs.</td>
<td><code>-no-c99</code></td>
<td><code>/Qc99[-]</code></td>
</tr>
<tr>
<td><code>-[no-]check-uninit</code></td>
<td>Determines whether checking occurs for uninitialized variables.</td>
<td><code>-no-check-uninit</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-[no-]complex-limited-range</code></td>
<td>Determines whether the use of basic algebraic expansions</td>
<td><code>-no-complex-limited-range</code></td>
<td><code>/Qcomplex-limited-range[-]</code></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-cxxlib[=&lt;dir&gt;]</td>
<td>Determines whether the compile links using the C++ run-time libraries and header files provided by gcc.</td>
<td>C++: -cxxlib</td>
<td>None</td>
</tr>
<tr>
<td>-cxxlib-nostd</td>
<td></td>
<td>C: -no-cxxlib</td>
<td></td>
</tr>
<tr>
<td>-no-cxxlib</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-D&lt;name&gt;[=&lt;value&gt;]</td>
<td>Defines a macro name that can be associated with an optional value.</td>
<td>OFF</td>
<td>/D&lt;name&gt;[=&lt;value&gt;]</td>
</tr>
<tr>
<td>-dD</td>
<td>Same as -dN, but outputs #define directives in preprocessed source.</td>
<td>OFF</td>
<td>/QdD</td>
</tr>
<tr>
<td>-debug [keyword]</td>
<td>Enables or disables generation of debugging information.</td>
<td>-debug none</td>
<td>/debug [keyword]</td>
</tr>
<tr>
<td>-diag-&lt;type&gt; &lt;diag-list&gt;</td>
<td>Controls the display of diagnostic information.</td>
<td>OFF</td>
<td>/Qdiag-&lt;type&gt;:&lt;diag-list&gt;</td>
</tr>
<tr>
<td>-diag-dump</td>
<td>Tells the compiler to print all enabled diagnostic messages and stop compilation</td>
<td>OFF</td>
<td>/Qdiag-dump</td>
</tr>
<tr>
<td>-diag-enable sc-include</td>
<td>Tells the Source Checker to analyze include files and source files when issuing diagnostic</td>
<td>OFF</td>
<td>/Qdiag-enable:sc-include</td>
</tr>
</tbody>
</table>

Equivalent Windows OS Option

None

OFF

/D<name>[=<value>]
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-diag-enable sc-parallel (i32, i64em)</td>
<td>Enables analysis of parallelization in source code (parallel lint diagnostics).</td>
<td>OFF</td>
<td>/Qdiag-enable:sc-parallel (i32, i64em)</td>
</tr>
<tr>
<td>-diag-error-limit &lt;n&gt;</td>
<td>Specifies the maximum number of errors allowed before compilation stops.</td>
<td>&lt;n&gt;=30</td>
<td>/Qdiag-error-limit:&lt;n&gt;</td>
</tr>
<tr>
<td>-diag-file=[file]</td>
<td>Causes the results of diagnostic analysis to be output to a file.</td>
<td>OFF</td>
<td>/Qdiag-file[:file]</td>
</tr>
<tr>
<td>-diag-file-append=[file]</td>
<td>Causes the results of diagnostic analysis to be appended to a file.</td>
<td>OFF</td>
<td>/Qdiag-file-append[:file]</td>
</tr>
<tr>
<td>-[no-]diag-id-numbers</td>
<td>Tells the compiler to display diagnostic messages by using their ID number values.</td>
<td>-diag-id-numbers</td>
<td>/Qdiag-id-numbers[-]</td>
</tr>
<tr>
<td>-diag-once &lt;id&gt;[,&lt;id&gt;,...]</td>
<td>Tells the compiler to issue one or more diagnostic messages only once.</td>
<td>OFF</td>
<td>/Qdiag-once:&lt;id&gt;[,&lt;id&gt;,...]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-dM</td>
<td>Output macro definitions in effect after preprocessing (use with -E).</td>
<td>OFF</td>
<td>/QdM</td>
</tr>
<tr>
<td>-dN</td>
<td>Same as -dD, but output #define directives contain only macro names.</td>
<td>OFF</td>
<td>/QdN</td>
</tr>
<tr>
<td>-dryrun</td>
<td>Specifies that driver tool commands should be shown but not executed.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-dumpmachine</td>
<td>Displays the target machine and operating system configuration.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-dumpversion</td>
<td>Displays the version number of the compiler.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-dynamiclib</td>
<td>Invokes the libtool command to generate dynamic libraries.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>(Mac OS* X only; i32, i64em)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-dynamic-linker</td>
<td>Specifies a dynamic linker other than the default.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>&lt;file&gt;</td>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-E</td>
<td>Preprocess to stdout.</td>
<td>OFF</td>
<td>/E</td>
</tr>
<tr>
<td>-[no-]early-tem-plate-check</td>
<td>Lets you semantically check template function template prototypes before instantiation.</td>
<td>-no-early-template-check</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-EP</td>
<td>Preprocess to stdout omitting #line directives.</td>
<td>OFF</td>
<td>/EP</td>
</tr>
<tr>
<td>-export</td>
<td>Enables support for the C++ export template feature.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-export-dir &lt;dir&gt;</td>
<td>Specifies a directory name for the exported template search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>F&lt;dir&gt; (Mac OS* X only)</td>
<td>Adds framework directory to head of include file search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fabi-version=&lt;n&gt; (i32 only)</td>
<td>Instructs the compiler to select a specific ABI implementation.</td>
<td>Varies</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]alias</td>
<td>Tells the compiler to assume aliasing in a program.</td>
<td>-falias</td>
<td>/Oa</td>
</tr>
<tr>
<td>-f[no-]align-functions[=&lt;n&gt;] (i32, i64em)</td>
<td>Aligns functions on optimal byte boundary.</td>
<td>-fno-align-functions</td>
<td>/Qfnalign[:&lt;n&gt;][-] (i32, i64em)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>falign-stack=&lt;mode&gt; (i32 only)</td>
<td>Tells the compiler the stack alignment to use on entry to routines.</td>
<td>OFF</td>
<td>-falign-stack=default</td>
</tr>
<tr>
<td>-fargument-[no]alias</td>
<td>Determines whether function arguments can alias each other.</td>
<td>-fargument-alias</td>
<td>/Qalias-args[-]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>-fargument-noalias-global</code></td>
<td>Tells the compiler that arguments cannot alias each other and cannot alias global storage.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-fasm-blocks</code> (Mac OS X only; i32, i64em)</td>
<td>Enables the use of blocks and entire functions of assembly code within a C or C++ file.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-fast</code></td>
<td>Maximizes speed across the entire program.</td>
<td>OFF</td>
<td><code>/fast</code></td>
</tr>
<tr>
<td><code>-f[no-]fast-transcendentals</code></td>
<td>Enables the compiler to replace calls to transcendental functions with faster but less precise implementation.</td>
<td>OFF</td>
<td><code>/Qfast-transcendentals[-]</code></td>
</tr>
<tr>
<td><code>-f[no-]builtin[func]</code></td>
<td>Enables or disables inline expansion of intrinsic functions.</td>
<td>OFF</td>
<td><code>/Oi[-]</code></td>
</tr>
<tr>
<td><code>-fcode-asm</code></td>
<td>Produces an assembly listing with machine code annotations.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-f[no-]common</code></td>
<td>Determines whether the compiler treats common symbols as global definitions.</td>
<td><code>-fcommon</code></td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-fno-exceptions</td>
<td>Enables exception handling table generation.</td>
<td>C++: -fexceprecptions C: -fno-exceptions</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]fnalias</td>
<td>Assume aliasing within functions.</td>
<td>-ffnalias</td>
<td>/Ow[-]</td>
</tr>
<tr>
<td>-ffreestanding</td>
<td>Ensures that compilation takes place in a freestanding environment.</td>
<td>OFF</td>
<td>/Qfreestanding</td>
</tr>
<tr>
<td>-ffunction-sections</td>
<td>Places each function in its own COMDAT section.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]inline</td>
<td>Tells the compiler to inline functions declared with __inline and perform C++ inlining.</td>
<td>-fno-inline</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]inline-functions</td>
<td>Enables function inlining for single file compilation.</td>
<td>-finline-functions</td>
<td>None</td>
</tr>
<tr>
<td>-finline-limit=&lt;n&gt;</td>
<td>Lets you specify the maximum size of a function to be inlined.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]instrument-functions</td>
<td>Determines whether function entry and exit points are instrumented.</td>
<td>-fno-instrument-functions</td>
<td>/Qinstrument-functions[-]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-f[no-]jump-tables</td>
<td>Determines whether jump tables are generated for switch statements.</td>
<td>-fjump-tables</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]keep-static-consts</td>
<td>Tells the compiler to preserve allocation of variables that are not referenced in the source.</td>
<td>-fno-keep-static-consts</td>
<td>/Qkeep-static-consts[-]</td>
</tr>
<tr>
<td>-f[no-]fma</td>
<td>enable/disable the combining of floating point multiplies and add/subtract operations</td>
<td>-fma</td>
<td>/Qfma[-] (i64 only)</td>
</tr>
<tr>
<td>(i64 only, Linux* OS only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-f[no-]math-errno</td>
<td>Tells the compiler that errno can be reliably tested after calls to standard math library functions.</td>
<td>-fno-math-errno</td>
<td>None</td>
</tr>
<tr>
<td>-fminshared</td>
<td>Specifies that a compilation unit is a component of a main program and should not be linked as part of a shareable object.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fmudflap</td>
<td>The compiler instruments risky pointer operations to prevent buffer overflows and invalid heap use.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-fno-gnu-keywords</td>
<td>Does not recognize typeof as keyword.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fno-implicit-in-line-templates</td>
<td>Tells the compiler to not emit code for implicit instantiations of inline templates.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fno-implicit-templates</td>
<td>Tells the compiler to not emit code for non-inline templates that are instantiated implicitly.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fno-operator-names</td>
<td>Disables support for the operator names specified in the standard.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fno-rtti (i32, i64em)</td>
<td>Disables support for run-time type information.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]non-call-exceptions</td>
<td>Allows trapping instructions to throw C++ exceptions.</td>
<td>-fno-non-call-exceptions</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]non-lvalue-assign</td>
<td>Determines whether casts and conditional expressions can be used as lvalues.</td>
<td>-fnon-lvalue-assign</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]fnsplit (i32 only)</td>
<td>Enables or disables function splitting (enabled with -prof-use).</td>
<td>-no-fnsplit</td>
<td>/Qfnsplit[-] (i32, i64)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>-f[no-]omit-frame-pointer</td>
<td>Enables or disables using EBP as a general-purpose register in optimizations.</td>
<td>-fomt-frame-pointer</td>
<td>/Oy[-]</td>
</tr>
<tr>
<td>-fp-model keyword</td>
<td>Controls the semantics of floating-point calculations.</td>
<td>-fp-model fast=1</td>
<td>/fp keyword:</td>
</tr>
<tr>
<td>-[no-]fp-port (i32, i64em)</td>
<td>Round floating-point results at assignments and casts (some speed impact).</td>
<td>-no-fp-port</td>
<td>/Qfp-port[-] (i32, i64em)</td>
</tr>
<tr>
<td>-[no-]fp-relaxed (i64 only; Linux* OS only)</td>
<td>Enables or disables use of faster but slightly less accurate code sequences for math functions.</td>
<td>-no-fp-relaxed</td>
<td>/Qfp-relaxed[-] (i64 only)</td>
</tr>
<tr>
<td>-fp-speculation=&lt;mode&gt;=mode</td>
<td>Tells the compiler the mode in which to speculate on floating-point operations.</td>
<td>-fp-speculation=fast</td>
<td>/Qfp-speculation:&lt;mode&gt;</td>
</tr>
<tr>
<td>-fp-stack-check (i32, i64em)</td>
<td>Enables FP stack checking after every function/procedure call.</td>
<td>OFF</td>
<td>/Qfp-stack-check (i32, i64em)</td>
</tr>
<tr>
<td>-fpascal-strings</td>
<td>Allow for Pascal-style string literals.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fpack-struct</td>
<td>Specifies that structure members should be packed together.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-fpermissive (i32 only)</td>
<td>Allow for non-conformant code.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]pic</td>
<td>Determines whether the compiler generates position-independent code.</td>
<td>varies</td>
<td>None</td>
</tr>
<tr>
<td>-fpie (Linux* OS only)</td>
<td>Tells the compiler to generate position-independent code.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fr32 (Linux* OS only; i64 only)</td>
<td>Disables the use of the high floating-point registers.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-freg-struct-return</td>
<td>Return struct and union values in registers when possible.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fshort-enums</td>
<td>Tells the compiler to allocate as many bytes as needed for enumerated types.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fs source-asm</td>
<td>Produce assembly file with optional source annotations.</td>
<td>OFF</td>
<td>/FAs</td>
</tr>
<tr>
<td>-f[no-]stack-protector</td>
<td>Same as -f[no-]stack-security-check.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------</td>
<td>------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>-f[no-]stack-security-check</td>
<td>Determines whether the compiler generates code that detects some buffer overruns.</td>
<td>-fno-stack-security-check</td>
<td>/GS[-]</td>
</tr>
<tr>
<td>-fsyntax-only</td>
<td>Performs syntax and semantic checking only (no object file produced).</td>
<td>OFF</td>
<td>/Zs</td>
</tr>
<tr>
<td>-ftemplate-depth-&lt;n&gt;</td>
<td>Control the depth in which recursive templates are expanded.</td>
<td>OFF</td>
<td>/Qtemplate-depth-&lt;n&gt;</td>
</tr>
<tr>
<td>-ftls-model=model</td>
<td>Change thread local storage mode.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-ftrapuv</td>
<td>Trap uninitialized variables.</td>
<td>OFF</td>
<td>/Qtrapuv</td>
</tr>
<tr>
<td>-[no-]ftz</td>
<td>Enables or disables flush denormal results to zero.</td>
<td>i64: -no-ftz</td>
<td>/Qftz[-]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(i32, i64em: -ftz</td>
<td></td>
</tr>
<tr>
<td>-funroll-loops</td>
<td>Same as -unroll.</td>
<td>ON</td>
<td>/Qunroll</td>
</tr>
<tr>
<td>-funroll-all-loops</td>
<td>Unroll all loops even if the number of iterations is uncertain when the loop is entered.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]unsigned-bitfields</td>
<td>Changes default bitfield type to unsigned.</td>
<td>-fno-unsigned-bitfields</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-funsigned-char</td>
<td>Changes default char type to unsigned.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]verbose-asm</td>
<td>Produces an assembly listing file with compiler comments.</td>
<td>-fno-verbose-asm</td>
<td>None</td>
</tr>
<tr>
<td>-fvisibility=keyword</td>
<td>Specifies the default visibility for global symbols.</td>
<td>-fvisibility=default</td>
<td>None</td>
</tr>
<tr>
<td>-fvisibility-keyword=&lt;file&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-fvisibility-inlines-hidden</td>
<td>Causes inline member functions to be marked hidden.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-g</td>
<td>Generates full debugging information in the object file.</td>
<td>OFF</td>
<td>/Zi, Z7</td>
</tr>
<tr>
<td>-g0</td>
<td>Disables generation of symbolic debug information.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]gcc</td>
<td>Defines or undefines GNU macros.</td>
<td>-gcc</td>
<td>None</td>
</tr>
<tr>
<td>-gcc-sys</td>
<td>Defines GNU macros only during compilation of system headers.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-gcc-name=&lt;dir&gt;</td>
<td>Specifies the location of the gcc compiler.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-gcc-version</td>
<td>Provides compatible behavior with gcc.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>-g</code>dwarf-2</td>
<td>Enables generation of debug information using the DWARF2 format.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-g</code>lobal-hoist</td>
<td>Enables or disables external globals to be load safe.</td>
<td><code>-global-hoist</code></td>
<td><code>/Qglobal-hoist[-]</code></td>
</tr>
<tr>
<td><code>-g</code>x<code>x-name=&lt;dir&gt;</code></td>
<td>Uses the g++ compiler as environment for C++ compilation.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-H</code></td>
<td>Prints include file order.</td>
<td>OFF</td>
<td><code>/QH</code></td>
</tr>
<tr>
<td><code>-help [category]</code></td>
<td>Displays all available compiler options or a category of compiler options.</td>
<td>OFF</td>
<td><code>/help [category]</code></td>
</tr>
<tr>
<td><code>-help-pragmas (i32, i64em)</code></td>
<td>Displays all supported pragmas.</td>
<td>OFF</td>
<td><code>/Qhelp-pragmas</code></td>
</tr>
<tr>
<td><code>-I&lt;dir&gt;</code></td>
<td>Adds directory to include file search path.</td>
<td>OFF</td>
<td><code>/I&lt;dir&gt;</code></td>
</tr>
<tr>
<td><code>-i</code>cc</td>
<td>Defines certain Intel compiler macros.</td>
<td><code>-icc</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-idirafter&lt;dir&gt;</code></td>
<td>Adds a directory to the second include file search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-imacros &lt;file&gt;</code></td>
<td>Specifies a lead header in a translation unit.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-[no-]inline-calloc (i32, i64em)</td>
<td>tells the compiler to inline calls to calloc() as calls to malloc() and memset().</td>
<td>-no-inline-calloc (i32, i64em)</td>
<td>/Qinline-calloc [-] (i32, i64em)</td>
</tr>
<tr>
<td>-inline-debug-info (Linux* OS only)</td>
<td>preserve the source position of inlined code instead of assigning the call-site source position to inlined code</td>
<td>OFF</td>
<td>/Qinline-debug-info</td>
</tr>
<tr>
<td>-[no-]inline-factor [=n]</td>
<td>Specify percentage multiplier that should be applied to all inlining options that define upper limits.</td>
<td>-no-inline-factor</td>
<td>/Qinline-factor [=n] [-]</td>
</tr>
<tr>
<td>-inline-forceinline</td>
<td>Inline routine whenever the compiler can do.</td>
<td>OFF</td>
<td>/Qinline-forceinline</td>
</tr>
<tr>
<td>-inline-level=&lt;n&gt;</td>
<td>control inline expansion</td>
<td>-inline-level=2 (if -O2 or -O3 is in effect) -inline-level=0 (if -O0 is in effect)</td>
<td>/O&lt;n&gt;</td>
</tr>
<tr>
<td>-[no-]inline-max-per-compile [=n]</td>
<td>Specify maximum number of times inlining may be applied to an entire compilation unit.</td>
<td>-no-inline-max-per-compile</td>
<td>/Qinline-max-per-compile [=n] [-]</td>
</tr>
<tr>
<td>-[no-]inline-max-per-routine [=n]</td>
<td>Specify maximum number of times the inliner may inline into a particular routine.</td>
<td>-no-inline-max-per-routine</td>
<td>/Qinline-max-per-routine [=n] [-]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-[no-]inline-max-size[=n]</td>
<td>Specify lower limit for the size of what the inliner considers to be a large routine.</td>
<td>-no-inline-max-size</td>
<td>/Qinline-max-size[=n][-]</td>
</tr>
<tr>
<td>-[no-]inline-max-total-size[=n]</td>
<td>Specify how much larger a routine can normally grow when inline expansion is performed.</td>
<td>-no-inline-max-total-size</td>
<td>/Qinline-max-total-size[=n][-]</td>
</tr>
<tr>
<td>-[no-]inline-min-size[=n]</td>
<td>Specify upper limit for the size of what the inliner considers to be a small routine.</td>
<td>-no-inline-min-size</td>
<td>/Qinline-min-size[=n][-]</td>
</tr>
<tr>
<td>-ip</td>
<td>Enables single-file IP optimizations (within files).</td>
<td>OFF</td>
<td>/Qip</td>
</tr>
<tr>
<td>-ip-no-inlining</td>
<td>Disables full and partial inlining (requires -ip or -ipo).</td>
<td>OFF</td>
<td>/Qip-no-inlining</td>
</tr>
<tr>
<td>-ip-no-pinlining (i32, i64em)</td>
<td>Disables partial inlining (requires -ip or -ipo).</td>
<td>OFF</td>
<td>/Qip-no-pinlining (i32, i64em)</td>
</tr>
<tr>
<td>-IPF-flt-eval-method0 (i64 only; Linux* OS only)</td>
<td>Evaluates floating-point operands evaluated to the precision indicated by program; deprecated.</td>
<td>OFF</td>
<td>/QIPF-flt-eval-method0 (i64 only)</td>
</tr>
<tr>
<td>-[no-]IPF-fltacc (i64 only; Linux* OS only)</td>
<td>Enables or disables optimizations that affect floating point accuracy; deprecated.</td>
<td>-no-IPF-fltacc</td>
<td>/QIPF-fltacc[-] (i64 only)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><img src="equivalentwindows.png" alt="Alt Text" /> <img src="osoption.png" alt="Alt Text" /> <img src="defaultdescriptionoption.png" alt="Alt Text" /> <img src="fma.png" alt="Alt Text" /> <img src="i64only.png" alt="Alt Text" /></td>
<td>Enables or disable the combining of floating point multiplies and add/subtract operations; deprecated, use <code>-fma</code>.</td>
<td><code>-IPF-fma</code></td>
<td><code>/QIPF-fma[ ]</code> (i64 only)</td>
</tr>
<tr>
<td><img src="no.png" alt="Alt Text" /> <img src="fp-reaxed.png" alt="Alt Text" /> <img src="i64only.png" alt="Alt Text" /> <img src="laxed.png" alt="Alt Text" /> <img src="no.png" alt="Alt Text" /> <img src="fp-relaxed.png" alt="Alt Text" /> <img src="i64only.png" alt="Alt Text" /> <img src="x.png" alt="Alt Text" /></td>
<td>Enables or disables use of faster but slightly less accurate code sequences for math functions; deprecated, use <code>-fp-relaxed</code>.</td>
<td><code>-no-IPF-fp-relaxed</code></td>
<td><code>/QIPF-fp-relaxed[ ]</code> (i64 only)</td>
</tr>
<tr>
<td><img src="ipo.png" alt="Alt Text" /> <img src="n.png" alt="Alt Text" /> <img src="x.png" alt="Alt Text" /></td>
<td>Enables multi-file IP optimizations (between files).</td>
<td>OFF</td>
<td><code>/Qipo[n]</code></td>
</tr>
<tr>
<td><img src="ipo.png" alt="Alt Text" /> <img src="c.png" alt="Alt Text" /></td>
<td>Generates a multi-file object file (ipo_out.o).</td>
<td>OFF</td>
<td><code>/Qipo-c</code></td>
</tr>
<tr>
<td><img src="ipo.png" alt="Alt Text" /> <img src="jobs.png" alt="Alt Text" /> <img src="n.png" alt="Alt Text" /></td>
<td>Specifies the number of commands to be executed simultaneously during the link phase of Interprocedural Optimization (IPO).</td>
<td>OFF</td>
<td><code>/Qipo-jobs:&lt;n&gt;</code></td>
</tr>
<tr>
<td><img src="ipo.png" alt="Alt Text" /> <img src="S.png" alt="Alt Text" /></td>
<td>Generates a multi-file assembly file (ipo_out.s).</td>
<td>OFF</td>
<td><code>/Qipo-S</code></td>
</tr>
<tr>
<td><img src="ipo.png" alt="Alt Text" /> <img src="separate.png" alt="Alt Text" /> <img src="LinuxOS.png" alt="Alt Text" /> <img src="only.png" alt="Alt Text" /></td>
<td>Creates one object file for every source file.</td>
<td>OFF</td>
<td><code>/Qipo-separate</code></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-iprefix &lt;prefix&gt;</td>
<td>Indicates the prefix for referencing directories containing header files.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-iquote &lt;dir&gt;</td>
<td>Adds directory for files included with quotes to front of include files search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-isystem&lt;dir&gt;</td>
<td>Specifies a directory to add to the system include path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-ivdep-parallel</td>
<td>Makes ivdep directives mean no loop carried dependencies.</td>
<td>OFF</td>
<td>/Qivdep-parallel (i64 only)</td>
</tr>
<tr>
<td></td>
<td>(i64 only; Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-iwithprefix&lt;dir&gt;</td>
<td>Appends &lt;dir&gt; to prefix passed in by -iprefix and puts it at the end of the include search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-iwithprefixbefore &lt;dir&gt;</td>
<td>Similar to -iwithprefix except include directory is placed in the same place as -I directories.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-Kc++</td>
<td>Compiles all source or unrecognized file types as C++ source files.</td>
<td>OFF</td>
<td>/TP</td>
</tr>
<tr>
<td>-kernel</td>
<td>Generates code for inclusion in the kernel.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>(i64 only; Linux* OS only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><code>-l&lt;string&gt;</code></td>
<td>Tells the linker to search for a specified library.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-L&lt;dir&gt;</code></td>
<td>Tells the linker to search for libraries in a specified directory before searching for them in the standard directories.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-m[&lt;processor&gt;][i32, i64em]</code></td>
<td>Tells the compiler to generate optimized code specialized for the processor that executes your program.</td>
<td>OFF</td>
<td><code>/arch:&lt;processor&gt;</code></td>
</tr>
<tr>
<td><code>-M</code></td>
<td>Generates makefile dependency information.</td>
<td>OFF</td>
<td><code>/QM</code></td>
</tr>
<tr>
<td><code>-m32[i32, i64em]</code></td>
<td>Tells the compiler to generate code for IA-32 architecture.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-m64[i32, i64em]</code></td>
<td>Tells the compiler to generate code for Intel® 64 architecture.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-malign-double[i32, i64em]</code></td>
<td>Aligns double, long double, and long long types for systems based on IA-32 architecture.</td>
<td>OFF</td>
<td>None</td>
</tr>
</tbody>
</table>
## Equivalent Windows OS Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-malign-mac68k</code> (i32, i64em; Mac OS* X only)</td>
<td>Aligns structure fields on 2-byte boundaries (m68k compatible).</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-malign-natural</code> (i32, i64em; Mac OS* X only)</td>
<td>Aligns larger types on natural size-based boundaries (overrides ABI).</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-malign-power</code> (i32, i64em; Mac OS* X only)</td>
<td>Aligns based on ABI-specified alignment rules.</td>
<td>ON</td>
<td>None</td>
</tr>
<tr>
<td><code>-map-opts</code> (Linux* OS only)</td>
<td>Enables option mapping tool.</td>
<td>OFF</td>
<td>/Qmap-opts</td>
</tr>
<tr>
<td><code>-march=&lt;processor&gt;</code> (i32, i64em; Linux* OS only)</td>
<td>Generates code for a specified processor.</td>
<td>i32: OFF</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i64em: <code>-march=pentium4</code></td>
<td>i64em: <code>-march=pentium4</code></td>
</tr>
<tr>
<td><code>-mcmodel=&lt;mem_model&gt;</code> (i64em; Linux* OS only)</td>
<td>Uses a specific memory model to generate code and store data.</td>
<td><code> -mcmodel=small</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-MD</code></td>
<td>Preprocesses and compiles, generating output file containing dependency information ending with extension .d.</td>
<td>OFF</td>
<td>/QMD</td>
</tr>
<tr>
<td><code>-mdynamic-no-pic</code> (i32; Mac OS* X only)</td>
<td>Generates code that is not position-independent</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-MF&lt;file&gt;</td>
<td>Generates makefile dependency information in file (must specify -M or -MM).</td>
<td>OFF</td>
<td>/QMF&lt;file&gt;</td>
</tr>
<tr>
<td>-mfixed-range (i64; Mac OS* X only)</td>
<td>Reserves certain register for use by the Linux* OS kernel.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-MG</td>
<td>Similar to -M, but treat missing header files as generated files.</td>
<td>OFF</td>
<td>/QMG</td>
</tr>
<tr>
<td>-mkl[=lib]</td>
<td>Tells the compiler to link to certain parts of the Intel® Math Kernel Library.</td>
<td>OFF</td>
<td>/Qmkl[=lib]</td>
</tr>
<tr>
<td>-minstruction=[no]movbe (i32, i64em)</td>
<td>Determines whether MOVBE instructions are generated for Intel processors.</td>
<td>OFF</td>
<td>/Qinstruction:[no]movbe (i32, i64em)</td>
</tr>
<tr>
<td>-MM</td>
<td>Similar to -M, but do not include system header files.</td>
<td>OFF</td>
<td>/QMM</td>
</tr>
<tr>
<td>-MMD</td>
<td>Similar to -MD, but do not include system header files.</td>
<td>OFF</td>
<td>/QMMD</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-mp</td>
<td>Maintains floating-point precision while disabling some optimization.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-MP</td>
<td>Adds a phony target for each dependency.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-mp1</td>
<td>Improves precision (speed impact is less than -mp).</td>
<td>OFF</td>
<td>/Qprec</td>
</tr>
<tr>
<td>-MQ&lt;target&gt;</td>
<td>Changes the default target rule for dependency generation.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-mregparm=&lt;value&gt;</td>
<td>Controls the number registers used to pass integer argument.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-m[no-]relax (i64 only)</td>
<td>Passes linker option -relax to the linker.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-m[no-]serialize-volatile (Linux* OS only)</td>
<td>Imposes strict memory access ordering for volatile data object references.</td>
<td>OFF</td>
<td>/Qserialize-volatile[-]</td>
</tr>
<tr>
<td>-MT&lt;target&gt;</td>
<td>Changes the default target rule for dependency generation.</td>
<td>OFF</td>
<td>/QMT&lt;target&gt;</td>
</tr>
<tr>
<td>-mtune=&lt;processor&gt;</td>
<td>Performs optimizations for specific processors.</td>
<td>i32, i64em: -mtune=generic i64: -mtune=itanium2-p9000</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-[no-]multibyte-chars</td>
<td>Provides support for multi-byte characters</td>
<td>ON</td>
<td>/Qmultibyte-chars[~]</td>
</tr>
<tr>
<td>-multiple-processes[=&lt;n&gt;]</td>
<td>Creates multiple processes.</td>
<td>OFF</td>
<td>/MP[&lt;:n&gt;]</td>
</tr>
<tr>
<td>-no-bss-init</td>
<td>Disable placement of zero-initialized variables in BSS (use DATA).</td>
<td>OFF</td>
<td>/Qnobss-init</td>
</tr>
<tr>
<td>-nodefaultlibs</td>
<td>Prevents using standard libraries when linking.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-nolib-inline</td>
<td>Disables inline expansion of standard library or intrinsic functions.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-nostartfiles</td>
<td>Prevents the compiler from using standard startup files when linking.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-nostdinc++</td>
<td>Do not search for header files in the standard directories for C++.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-nostdlib</td>
<td>Prevents the compiler from using standard libraries and startup files when linking.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-o&lt;file&gt;</td>
<td>Specifies name of output file.</td>
<td>OFF</td>
<td>/Fe&lt;file&gt;</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>-0</code></td>
<td>Enables optimizations.</td>
<td><code>-O2</code></td>
<td><code>/O</code></td>
</tr>
<tr>
<td><code>-00</code></td>
<td>Disables optimizations.</td>
<td><code>-O2</code></td>
<td><code>/Od</code></td>
</tr>
<tr>
<td><code>-openmp</code></td>
<td>Generates multi-threaded code based on the OpenMP* directives</td>
<td>OFF</td>
<td><code>/Qopenmp</code></td>
</tr>
<tr>
<td><code>-openmp-lib &lt;type&gt;</code></td>
<td>(Linux* OS only) Lets you specify an OpenMP* run-time library to use for linking.</td>
<td><code>-openmp-lib legacy</code></td>
<td><code>/Qopenmp-lib:&lt;type&gt;</code></td>
</tr>
<tr>
<td><code>-openmp-link &lt;library&gt;</code></td>
<td>Links to static or dynamic OpenMP run-time libraries.</td>
<td><code>-openmp-link dynamic</code></td>
<td><code>/Qopenmp-link:&lt;library&gt;</code></td>
</tr>
<tr>
<td><code>-openmp-profile</code></td>
<td>(Linux* OS only) Links with instrumented OpenMP run-time library to generate OpenMP profiling information.</td>
<td>OFF</td>
<td><code>/Qopenmp-profile</code></td>
</tr>
<tr>
<td><code>-openmp-report&lt;n&gt;</code></td>
<td>Controls the OpenMP parallelizer diagnostic level.</td>
<td><code>-openmp-report1</code></td>
<td><code>/Qopenmp-report&lt;n&gt;</code></td>
</tr>
<tr>
<td><code>-openmp-stubs</code></td>
<td>Enables the user to compile OpenMP programs in sequential mode.</td>
<td>OFF</td>
<td><code>/Qopenmp-stubs</code></td>
</tr>
<tr>
<td><code>-openmp-task &lt;model&gt;</code></td>
<td>Lets you choose an OpenMP* tasking model.</td>
<td><code>-openmp-task omp</code></td>
<td><code>/Qopenmp-task:&lt;model&gt;</code></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-openmp-threadprivate &lt;type&gt;</td>
<td>Lets you specify an OpenMP* threadprivate implementation.</td>
<td>- openmp-threadprivate legacy</td>
<td>/Openmp-threadprivate:&lt;type&gt;</td>
</tr>
<tr>
<td>-opt-block-factor=&lt;n&gt;</td>
<td>Lets you specify a loop blocking factor.</td>
<td>OFF</td>
<td>/Opt-block-factor:&lt;n&gt;</td>
</tr>
<tr>
<td>-[no]opt-calloc (i32, i64em; Linux* OS only)</td>
<td>Tells the compiler to substitute a call to _intel_fast_calloc() for a call to calloc().</td>
<td>-no-opt-calloc</td>
<td>/Openmp-stubs</td>
</tr>
<tr>
<td>-[no]-opt-class-analysis</td>
<td>Tells the compiler to use C++ class hierarchy information to analyze and resolve C++ virtual function calls at compile time.</td>
<td>-no-opt-class-analysis</td>
<td>/Opt-class-analysis[-]</td>
</tr>
<tr>
<td>-[no]-opt-jump-tables=keyword</td>
<td>Enables or disables generation of jump tables for switch statements.</td>
<td>- opt-jump-tables=default</td>
<td>/Opt-jump-tables[-]</td>
</tr>
<tr>
<td>-[no]-opt-loadpair (i64 only; Linux* OS only)</td>
<td>Enables or disables loadpair optimization.</td>
<td>-no-opt-loadpair</td>
<td>/Opt-loadpair[-] (i64 only)</td>
</tr>
<tr>
<td>-[no]-opt-malloc-options=&lt;n&gt;</td>
<td>Lets you specify an alternate algorithm for malloc().</td>
<td>-opt-malloc-options=0</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>-opt-mem-band-width&lt;n&gt;</td>
<td>Enables performance tuning and heuristics to control memory bandwidth use among processors.</td>
<td>-opt-mem-band-width0</td>
<td>/Qopt-mem-band-width&lt;n&gt; (i64 only)</td>
</tr>
<tr>
<td>(i64 only; Linux* OS only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-mod-versioning</td>
<td>Enables or disables versioning of modulo operations for certain types of operands.</td>
<td>- no-opt-mod-versioning</td>
<td>/Qopt-mod-versioning[-] (i64 only)</td>
</tr>
<tr>
<td>(i64 only; Linux* OS only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-multi-version-aggressive</td>
<td>tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement</td>
<td>-no-opt-multi-version-aggressive</td>
<td>/Qopt-multi-version-aggressive[-] (i32, i64em)</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-prefetch[=&lt;n&gt;]</td>
<td>Enables or disables prefetch insertion optimization.</td>
<td>i32, i64em: -opt-prefetch</td>
<td>/Qopt-prefetch[:&lt;n&gt;]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i64: -opt-prefetch</td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-prefetch-initial-values</td>
<td>Enables or disables prefetches that are issued before a loop is entered.</td>
<td>-no-opt-prefetch-initial-values</td>
<td>/Qopt-prefetch-initial-values [-] (i64 only)</td>
</tr>
<tr>
<td>(i64 only; Linux* OS only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-prefetch-issue-excl-hint</td>
<td>Enables or disables prefetches for stores with exclusive hint</td>
<td>-no-opt-prefetch-issue-excl-hint</td>
<td>/Qopt-prefetch-issue-excl-hint [-] (i64 only)</td>
</tr>
<tr>
<td>(i64 only; Linux* OS only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>-[no-]opt-prefetch-next-iteration</td>
<td>Enables or disables prefetches for a memory access in the next iteration of a loop.</td>
<td>-no-opt-prefetch-next-iteration</td>
<td>/Qopt-prefetch-next-iteration [-] (i64 only)</td>
</tr>
<tr>
<td>(i64 only; Linux* OS only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-ra-region-strategy[:keyword]</td>
<td>Selects the method that the register allocator uses to partition each routine into regions.</td>
<td>-opt-ra-region-strategy=default</td>
<td>/Qopt-ra-region-strategy[:keyword] (i32, i64em)</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-opt-report-file=&lt;file&gt;</td>
<td>Specifies the file name for the generated report.</td>
<td>OFF</td>
<td>/Qopt-report-file:&lt;file&gt;</td>
</tr>
<tr>
<td>-opt-report-help</td>
<td>Displays the optimization phases available for reporting.</td>
<td>OFF</td>
<td>/Qopt-report-help</td>
</tr>
<tr>
<td>-opt-report-phase=&lt;phase&gt;</td>
<td>Specifies the phase that reports are generated against.</td>
<td>OFF</td>
<td>/Qopt-report-phase:&lt;phase&gt;</td>
</tr>
<tr>
<td>-opt-report-routine=&lt;string&gt;</td>
<td>Reports on routines containing the given name.</td>
<td>OFF</td>
<td>/Qopt-report-routine:&lt;string&gt;</td>
</tr>
<tr>
<td>-opt-streaming-stores keyword</td>
<td>Enables generation of streaming stores for optimization.</td>
<td>-opt-streaming-stores auto</td>
<td>/Qopt-streaming-stores:keyword (i32, i64em)</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-[no-]opt-subscript-in-range (i32, i64em)</td>
<td>Enables or disables overflows in the intermediate computation of subscript expressions in loops.</td>
<td>-[no-]opt-subscript-in-range</td>
<td>/Qopt-subscript-in-range[-] (i32, i64em)</td>
</tr>
<tr>
<td>-Os</td>
<td>Enables speed optimizations, but disable some optimizations which increase code size for small speed benefit.</td>
<td>OFF</td>
<td>/Os</td>
</tr>
<tr>
<td>-P</td>
<td>Compiles and links for function profiling with gprof(1).</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-P (or -F)</td>
<td>Preprocesses to file omitting #line directives.</td>
<td>OFF</td>
<td>/P</td>
</tr>
<tr>
<td>-par-affinity=[modifier,...]type[,permute][,offset] (Linux* OS only)</td>
<td>Specifies thread affinity.</td>
<td>OFF</td>
<td>/Qpar-affinity:modifier,...]type[,permute][,offset]</td>
</tr>
<tr>
<td>-par-num-threads=n</td>
<td>Specifies the number of threads to use in a parallel region.</td>
<td>OFF</td>
<td>/Qpar-num-threads:n</td>
</tr>
<tr>
<td>-par-report[n]</td>
<td>control the auto-parallelizer diagnostic level</td>
<td>-par-report1</td>
<td>/Qpar-report[n]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>-{no-}par-runtime-control</td>
<td>Generates code to perform run-time checks for loops that have symbolic loop bounds.</td>
<td>-no-par-runtime-control</td>
<td>/Qpar-runtime-control[-]</td>
</tr>
<tr>
<td>-par-schedule-&lt;keyword&gt;[=n]</td>
<td>Specifies a scheduling algorithm for DO loop iterations.</td>
<td>OFF</td>
<td>/Qpar-schedule-&lt;keyword&gt;[: n ]</td>
</tr>
<tr>
<td>-parallel</td>
<td>Enables the auto-parallelizer to generate multi-threaded code for loops that can be safely executed in parallel.</td>
<td>OFF</td>
<td>/Qparallel</td>
</tr>
<tr>
<td>-pc&lt;n&gt;</td>
<td>Enables control of floating-point significand precision.</td>
<td>-pc80</td>
<td>/Qpc&lt;n&gt;</td>
</tr>
<tr>
<td>-pch</td>
<td>Enable sautomatic precompiled header file creation/usage.</td>
<td>OFF</td>
<td>/YX</td>
</tr>
<tr>
<td>-pch-create &lt;file&gt;</td>
<td>Creates precompiled header files.</td>
<td>OFF</td>
<td>/Yc[file]</td>
</tr>
<tr>
<td>-pch-dir &lt;dir&gt;</td>
<td>Tells the compiler where to find or create a file for precompiled headers.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-pch-use {&lt;file&gt;</td>
<td>&lt;dir&gt;}</td>
<td>Lets you use a specific precompiled header file.</td>
<td>OFF</td>
</tr>
<tr>
<td>-pie</td>
<td>Produces a position-independent executable on processors that support it.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-pragma-optimization-level=&lt;interpretation&gt;</td>
<td>Specifies which interpretation of the optimization_level pragma should be used if no prefix is specified.</td>
<td>-pragma-optimization-level=Intel</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]prec-div</td>
<td>Improve precision of floating-point divides (some speed impact).</td>
<td>-prec-div</td>
<td>/Qprec-div[-]</td>
</tr>
<tr>
<td>-[no-]prec-sqrt (i32, i64em)</td>
<td>Determine if certain square root optimizations are enabled.</td>
<td>-no-prec-sqrt</td>
<td>/Qprec-sqrt[-] (i32, i64em)</td>
</tr>
<tr>
<td>-print-multi-lib</td>
<td>Prints information about where system libraries should be found.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]prof-data-order (Linux* OS only)</td>
<td>Enables or disables data ordering if profiling information is enabled.</td>
<td>-no-prof-data-order</td>
<td>/Qprof-data-order[-]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-prof-dir &lt;dir&gt;</td>
<td>Specifies directory for profiling output files (*.dyn and *.dpi).</td>
<td>OFF</td>
<td>/Qprof-dir &lt;dir&gt;</td>
</tr>
<tr>
<td>-prof-file &lt;file&gt;</td>
<td>Specifies file name for profiling summary file.</td>
<td>OFF</td>
<td>/Qprof-file &lt;file&gt;</td>
</tr>
<tr>
<td>-[no-]prof-func-groups</td>
<td>Enables or disables function grouping if profiling information is enabled.</td>
<td>-no-prof-func-groups</td>
<td>None</td>
</tr>
<tr>
<td>(i32, i64em; Linux* OS only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]prof-func-order</td>
<td>Enables or disables function ordering if profiling information is enabled.</td>
<td>-no-prof-func-order</td>
<td>/Qprof-func-order [-]</td>
</tr>
<tr>
<td>-prof-gen[x]</td>
<td>Instruments program for profiling.</td>
<td>OFF</td>
<td>/Qprof-gen[x]</td>
</tr>
<tr>
<td>-prof-hotness-threshold=&lt;n&gt;</td>
<td>Lets you set the hotness threshold for function grouping and function ordering.</td>
<td>OFF</td>
<td>/Qprof-hotness-threshold:&lt;n&gt;</td>
</tr>
<tr>
<td>(Linux* OS only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]prof-src-dir</td>
<td>Determines whether directory information of the source file under compilation is considered when looking up profile data records.</td>
<td>-prof-src-dir</td>
<td>/Qprof-src-dir [-]</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-prof-src-root=&lt;dir&gt;</td>
<td>Lets you use relative directory paths when looking up profile data and specifies a directory as the base.</td>
<td>OFF</td>
<td>/Qprof-src-root=&lt;dir&gt;</td>
</tr>
<tr>
<td>-prof-src-root-cwd</td>
<td>Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.</td>
<td>OFF</td>
<td>/Qprof-src-root-cwd</td>
</tr>
<tr>
<td>-prof-use</td>
<td>Enables use of profiling information during optimization.</td>
<td>OFF</td>
<td>/Qprof-use</td>
</tr>
<tr>
<td>-pthread</td>
<td>Uses pthreads library for multithreading support.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-Qlocation,&lt;string&gt;,&lt;dir&gt;</td>
<td>Sets &lt;dir&gt; as the location of tool specified by &lt;string&gt;; supported tools depend on the operating system.</td>
<td>OFF</td>
<td>/Qlocation,&lt;string&gt;,&lt;dir&gt;</td>
</tr>
<tr>
<td>-Qoption,&lt;string&gt;,&lt;options&gt;</td>
<td>Passes options &lt;options&gt; to tool specified by &lt;string&gt;; supported tools depend on the operating system.</td>
<td>OFF</td>
<td>/Qoption,&lt;string&gt;,&lt;options&gt;</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><code>-rcd</code> (i32, i64em)</td>
<td>Sets rounding mode to enable fast float-to-int conversions.</td>
<td>OFF</td>
<td><code>/Qrcd</code> (i32, i64em)</td>
</tr>
<tr>
<td><code>-rct</code> (i32, i64em)</td>
<td>Sets the internal FPU rounding control to Truncate.</td>
<td>OFF</td>
<td><code>/Qrct</code> (i32, i64em)</td>
</tr>
<tr>
<td><code>-reserve-kernel-reg</code> (i64 only; Linux* OS only)</td>
<td>Reserves registers f12-f15 and f32-f127 for use by the kernel.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-[no]restrict</code></td>
<td>Enables or disables the 'restrict' keyword for disambiguating pointers.</td>
<td>OFF</td>
<td><code>/Qrestrict[-]</code></td>
</tr>
<tr>
<td><code>-S</code></td>
<td>Compiles to assembly (.s) only, do not link (*I).</td>
<td>OFF</td>
<td><code>/S</code></td>
</tr>
<tr>
<td><code>-[no-]save-temps</code></td>
<td>Tells the compiler to save intermediate files created during compilation.</td>
<td><code>-no-save-temps</code></td>
<td><code>/Qsave-temps[-]</code></td>
</tr>
<tr>
<td><code>-[no-]scalar-rep</code></td>
<td>Enables or disables scalar replacement performed during loop transformation.</td>
<td><code>-no-scalar-rep</code></td>
<td><code>/Qscalar-rep[-]</code></td>
</tr>
<tr>
<td><code>-shared</code> (Linux* OS only)</td>
<td>Produces a dynamic shared object instead of an executable.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-shared-intel</td>
<td>Causes Intel-provided libraries to be linked in dynamically.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-shared-libgcc</td>
<td>Links the GNU libgcc library dynamically.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>![Linux* OS only]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]sox</td>
<td>Enables or disables saving of compilation options and version number in the Linux OS executable.</td>
<td>-no-sox</td>
<td>/Qsox[-]</td>
</tr>
<tr>
<td>![Linux* OS only]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-static</td>
<td>Prevents linking with shared libraries.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>![Linux* OS only]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-staticlib</td>
<td>Invokes the libtool command to generate static libraries.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>![Mac OS* X only]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-static-intel</td>
<td>Causes Intel-provided libraries to be linked in statically.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>![Linux* OS only]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-static-libgcc</td>
<td>Links the GNU libgcc library statically.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>![Linux* OS only]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-std=&lt;val&gt;</td>
<td>Conforms to a specific language standard.</td>
<td>OFF</td>
<td>/Qstd:&lt;val&gt;</td>
</tr>
<tr>
<td>-strict-ansi</td>
<td>Strict ANSI conformance dialect.</td>
<td>OFF</td>
<td>/Za</td>
</tr>
<tr>
<td>-T &lt;file&gt;</td>
<td>Tells the linker to read link commands from a file.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>![Linux* OS only]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-tcheck</td>
<td>Generates instrumentation to detect multi-threading bugs.</td>
<td>OFF</td>
<td>/Qtcheck</td>
</tr>
<tr>
<td>-tcollect [&lt;lib&gt;]</td>
<td>Inserts instrumentation probes calling the Intel® Trace Collector API.</td>
<td>OFF</td>
<td>/Qtcollect[:&lt;lib&gt;]</td>
</tr>
<tr>
<td>-tcollect-filter &lt;file&gt;</td>
<td>Enables or disables the instrumentation of specified functions.</td>
<td>OFF</td>
<td>/Qtcollect-filter[::&lt;file&gt;]</td>
</tr>
<tr>
<td>-tprofile</td>
<td>Generates instrumentation to analyze multi-threading performance.</td>
<td>OFF</td>
<td>/Qtprofile</td>
</tr>
<tr>
<td>-[no]traceback</td>
<td>Specifies whether the compiler generates data to allow for source file traceback information at runtime.</td>
<td>-notraceback</td>
<td>/[no]traceback</td>
</tr>
<tr>
<td>-U&lt;name&gt;</td>
<td>remove predefined macro</td>
<td>OFF</td>
<td>/U&lt;name&gt;</td>
</tr>
<tr>
<td>-unroll[n]</td>
<td>Set maximum number of times to unroll loops.</td>
<td>-unroll</td>
<td>/Qunroll[:n]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>-{no-}unroll-aggressive (i32, i64em)</td>
<td>Tells the compiler to use aggressive, complete unrolling for loops with small constant trip counts.</td>
<td>OFF</td>
<td>/Qunroll-aggressive[-] (i32, i64em)</td>
</tr>
<tr>
<td>-{no-}use-asm</td>
<td>Produces objects through assembler.</td>
<td>-no-use-asm</td>
<td>/Quse-asm[-] (i32 only)</td>
</tr>
<tr>
<td>-use-msasm</td>
<td>Accepts the Microsoft* MASM-style inlined assembly format.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-use-intel-optimized-headers (i32, i64em)</td>
<td>Adds performance headers directory to the include path search list.</td>
<td>-no-use-intel-optimized-headers</td>
<td>/Quse-intel-optimized-headers [-] (i32, i64em)</td>
</tr>
<tr>
<td>-V</td>
<td>Displays compiler version information.</td>
<td>OFF</td>
<td>/QV</td>
</tr>
<tr>
<td>-{no-}vec</td>
<td>Enables or disables vectorization.</td>
<td>-vec</td>
<td>/Qvec[-]</td>
</tr>
<tr>
<td>-{no-}vec-guard-write (i32, i64em)</td>
<td>Tells the compiler to perform a conditional check in a vectorized loop.</td>
<td>-no-vec-guard-write</td>
<td>/Qvec-guard-write[-] (i32, i64em)</td>
</tr>
<tr>
<td>-vec-report[&lt;n&gt;] (i32, i64em)</td>
<td>Controls amount of vectorizer diagnostic information-opt-report generate an optimization report to stderr.</td>
<td>-vec-report1</td>
<td>/Qvec-report[&lt;n&gt;] (i32, i64em)</td>
</tr>
<tr>
<td>--version</td>
<td>Displays GCC-style version information.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><code>-vec-threshold-old[&lt;n&gt;]</code> (i32, i64em)</td>
<td>Sets threshold for the vectorization of loops.</td>
<td><code>-vec-threshold100</code></td>
<td><code>/Qvec-threshold-old[[:]n]</code></td>
</tr>
<tr>
<td><code>-w</code></td>
<td>Disables all warning messages.</td>
<td>OFF</td>
<td><code>/w</code></td>
</tr>
<tr>
<td><code>-w&lt;n&gt;</code></td>
<td>Determines which diagnostic message level is set.</td>
<td>OFF</td>
<td><code>/W&lt;n&gt;</code></td>
</tr>
<tr>
<td><code>-Wa,&lt;option1&gt;,&lt;option2&gt;,...</code></td>
<td>Passes options to the assembler for processing.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-W[no-]abi</code></td>
<td>Determines whether a warning is issued if generated code is not C++ ABI compliant.</td>
<td><code>-Wno-abi</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-Wall</code></td>
<td>Enables all warnings.</td>
<td>OFF</td>
<td><code>/Wall</code></td>
</tr>
<tr>
<td><code>-Wbrief</code></td>
<td>Prints brief one-line diagnostics.</td>
<td>OFF</td>
<td><code>/WL</code></td>
</tr>
<tr>
<td><code>-Wcheck</code></td>
<td>Enables compile-time code checking for certain code.</td>
<td>OFF</td>
<td><code>/Wcheck</code></td>
</tr>
<tr>
<td><code>-W[no-]comment</code></td>
<td>Determines whether a warning is issued when /* appears in the middle of a /* */ comment.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-Wcontext-limit=&lt;n&gt;</td>
<td>Sets maximum number of template instantiation contexts shown in diagnostic.</td>
<td>OFF</td>
<td>/Qcontext-limit:&lt;n&gt;</td>
</tr>
<tr>
<td>-wd&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
<td>Disables diagnostics L1 through Ln.</td>
<td>OFF</td>
<td>/Qwd&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
</tr>
<tr>
<td>-W[no-]deprecated</td>
<td>Determines whether warnings are issued for deprecated features.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-we&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
<td>Changes severity of soft diagnostics L1 through Ln to error.</td>
<td>OFF</td>
<td>/Qwe&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
</tr>
<tr>
<td>-Weffc++</td>
<td>Enables warnings based on certain C++ programming guidelines.</td>
<td>OFF</td>
<td>/Qeffc++</td>
</tr>
<tr>
<td>-Werror</td>
<td>Forces warnings to be reported as errors.</td>
<td>OFF</td>
<td>/WX</td>
</tr>
<tr>
<td>-Werror-all</td>
<td>Changes all warnings and remarks to errors.</td>
<td>OFF</td>
<td>/Werror-all</td>
</tr>
<tr>
<td>-W[no-]extra-tokens</td>
<td>Determines whether warnings are issued about extra tokens at the end of preprocessor directives.</td>
<td>OFF</td>
<td>-Wno-extra-tokens</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><code>-W[no-]format</code></td>
<td>Determines whether argument checking is enabled for calls to printf, scanf, and so forth.</td>
<td><code>-Wno-format</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-W[no-]format-security</code></td>
<td>Determines whether the compiler issues a warning when the use of format functions may cause security problems.</td>
<td><code>-Wno-format-security</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-Winline</code></td>
<td>Enables diagnostics about what is inlined and what is not inlined.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-Wl,&lt;option1&gt;,&lt;option2&gt;,...</code></td>
<td>Passes options to the linker for processing.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-W[no-]main</code></td>
<td>Determines whether a warning is issued if the return type of main is not expected.</td>
<td><code>-Wno-main</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-W[no-]missing-declaration</code></td>
<td>Determines whether warnings are issued for global functions and variables without prior declaration.</td>
<td><code>-Wno-missing-declaration</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-W[no-]missing-prototype</code></td>
<td>Determines whether warnings are issued for missing prototypes.</td>
<td><code>-Wno-missing-prototype</code></td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-Wnon-virtual-dtor</td>
<td>Issues a warning when a class appears to be polymorphic, yet it declares a non-virtual one. (C++ only.)</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-wn&lt;n&gt;</td>
<td>Prints a maximum number of errors.</td>
<td>-wn100</td>
<td>/Qwn&lt;n&gt;</td>
</tr>
<tr>
<td>-wo&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
<td>Issues one or more diagnostic messages only once.</td>
<td>OFF</td>
<td>/Qwo&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
</tr>
<tr>
<td>-Wp,&lt;option1&gt;[,&lt;option2&gt;,...]</td>
<td>Passes options to the preprocessor.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-Wp64</td>
<td>Prints diagnostics for 64-bit porting.</td>
<td>OFF</td>
<td>/Wp64</td>
</tr>
<tr>
<td>-W[no-]pointer-arith</td>
<td>Determines whether warnings are issued for questionable pointer arithmetic.</td>
<td>-Wno-pointer-arith</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]pragma-once</td>
<td>Determines whether a warning is issued about the use of #pragma once.</td>
<td>-Wno-pragmasonce</td>
<td>None</td>
</tr>
<tr>
<td>-wr&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
<td>Changes severity of soft diagnostics L1 through Ln to remark.</td>
<td>OFF</td>
<td>/Qwr&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
</tr>
<tr>
<td>-Wreorder</td>
<td>Issues a warning when the order of member initializers does not</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>match the order in which they must be executed.</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-W[no-]return-type</code></td>
<td>Determines whether warnings are issued when a function uses the default int return type.</td>
<td><code>-Wno-return-type</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-W[no-]shadow</code></td>
<td>Determines whether a warning is issued when a variable declaration hides a previous declaration.</td>
<td><code>-Wno-shadow</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-W[no-]strict-prototypes</code></td>
<td>Determines whether warnings are issued for functions declared or defined without specified argument types.</td>
<td><code>-Wno-strict-prototypes</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-W[no-]trigraphs</code></td>
<td>Determines whether warnings are issued if any trigraphs are encountered.</td>
<td><code>-Wno-trigraphs</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-W[no-]uninitialized</code></td>
<td>Determines whether a warning is issued if a variable is used before being initialized.</td>
<td><code>-Wno-uninitialized</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-W[no-]unknown-pragmas</code></td>
<td>Determines whether a warning is issued if an unknown #pragma directive is used.</td>
<td><code>-Wno-unknown-pragmas</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td><code>-W[no-]unused-function</code></td>
<td>Determines whether a warning is issued if a declared function is not used.</td>
<td><code>-Wno-unused-function</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-W[no-]unused-variable</code></td>
<td>Determines whether a warning is issued if a local or non-constant static variable is unused after being declared.</td>
<td><code>-Wno-unused-variable</code></td>
<td>None</td>
</tr>
<tr>
<td><code>-ww&lt;L1&gt;[,&lt;Ln&gt;,...]</code></td>
<td>Changes soft diagnostics L1 through Ln to warning.</td>
<td>OFF</td>
<td>/Qww&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
</tr>
<tr>
<td><code>-Wwrite-strings</code></td>
<td>Issues a diagnostic message if const char <code>*</code> is converted to (non-const) char <code>*</code>.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-x&lt;processor&gt; (i32, i64em)</code></td>
<td>Generates optimized code specialized for the Intel processor that executes your program.</td>
<td>varies; see option description</td>
<td>/Qx&lt;processor&gt; (i32, i64em)</td>
</tr>
<tr>
<td>`-X</td>
<td>Removes standard directories from include file search path.</td>
<td>OFF</td>
<td>/X</td>
</tr>
<tr>
<td><code>-Xlinker &lt;option&gt;</code></td>
<td>Passes a linker option directly to the linker.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-Zp[n]</code></td>
<td>Specifies alignment for structures on byte boundaries.</td>
<td><code>-Zp16</code></td>
<td>/Zp[n]</td>
</tr>
</tbody>
</table>
Related Options

This topic lists related options that can be used under certain conditions.

Linking Tools and Options

This topic describes how to use the Intel® linking tools, xild (Linux® OS and Mac OS® X) or xilink (Windows® OS).

The Intel linking tools behave differently on different platforms. The following sections summarize the primary differences between the linking behaviors.

<table>
<thead>
<tr>
<th>Linux OS and Mac OS X Linking Behavior Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>The linking tool invokes the compiler to perform IPO if objects containing IR (intermediate representation) are found. (These are mock objects.) It invokes GNU ld to link the application.</td>
</tr>
<tr>
<td>The command-line syntax for xild is the same as that of the GNU linker:</td>
</tr>
<tr>
<td>xild [options] normal command-line</td>
</tr>
<tr>
<td>where:</td>
</tr>
<tr>
<td>• [options]: (optional) one or more options supported only by xild.</td>
</tr>
<tr>
<td>• normal command-line: linker command line containing a set of valid arguments for ld.</td>
</tr>
<tr>
<td>To create app using IPO, use the option -ofile as shown in the following example:</td>
</tr>
<tr>
<td>xild -qipo_fas -oapp a.o b.o c.o</td>
</tr>
<tr>
<td>The linking tool calls the compiler to perform IPO for objects containing IR and creates a new list of object(s) to be linked. The linker then calls ld to link the object files that are specified in the new list and produce the application with the name specified by the -o option.</td>
</tr>
<tr>
<td>The linker supports the -ipo, -ipoN, and -ipo-separate options.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Windows OS Linking Behavior Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>The linking tool invokes the Intel compiler to perform multi-file IPO if objects containing IR (intermediate representation) is found. These are mock objects. It invokes Microsoft* link.exe to link the application.</td>
</tr>
</tbody>
</table>
Windows OS Linking Behavior Summary

The command-line syntax for the Intel® linker is the same as that of the Microsoft linker:

```
xilink [<options>] <normal command-line>
```

where:

- `[<options>]`: (optional) one or more options supported only by xilink.
- `<normal command-line>`: linker command line containing a set of valid arguments for the Microsoft linker.

To place the multifile IPO executable in `ipo_file.exe`, use the linker option `/out:file`; for example:

```
xilink -qipo_fas /out:ipo_file.exe a.obj b.obj c.obj
```

The linker calls the compiler to perform IPO for objects containing IR and creates a new list of object(s) to be linked. The linker calls Microsoft `link.exe` to link the object files that are specified in the new list and produce the application with the name specified by the `/out:file` linker option.

Using the Linking Tools

You must use the Intel linking tools to link your application if the following conditions apply:

- Your source files were compiled with multifile IPO enabled. Multi-file IPO is enabled by specifying the `-ipo` (Linux OS and Mac OS X) or `/Qipo` (Windows OS) command-line option.
- You normally would invoke either the GNU linker (`ld`) or the Microsoft linker (`link.exe`) to link your application.

The following table lists the available, case-insensitive options supported by the Intel linking tools and briefly describes the behavior of each option:

<table>
<thead>
<tr>
<th>Linking Tools Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-qhelp</td>
<td>Lists the available linking tool options. Same as passing no option.</td>
</tr>
<tr>
<td>-qnoipo</td>
<td>Disables multi-file IPO compilation.</td>
</tr>
<tr>
<td>Linking Tools Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| `-qipo_fa[/file|dir/]` | Produces assembly listing for the multi-file IPO compilation. You may specify an optional name for the listing file, or a directory (with the backslash) in which to place the file. The default listing name is depends on the platform:  
  - Linux OS and Mac OS X: `ipo_out.s`  
  - Windows OS: `ipo_out.asm`  
  If the Intel linking tool invocation results in multi-object compilation, either because the application is big or because the user explicitly instructed the compiler to generate multiple objects, the first `.s` (Linux OS and Mac OS X) or `.asm` (Windows OS) file takes its name from the `-qipo_fa` option. The compiler derives the names of subsequent `.s` (Linux OS and Mac OS X) or `.asm` (Windows OS) files by appending an incrementing number to the name, for example, `foo.asm` and `fool.asm` for `ipo_fafoo.asm`. The same is true for the `-qipo_fo` option (listed below). |
| `-qipo_fo[/file|dir/]` | Produces object file for the multi-file IPO compilation. You may specify an optional name for the object file, or a directory (with the backslash) in which to place the file. The default object file name is depends on the platform:  
  - Linux OS and Mac OS X: `ipo_out.o`  
  - Windows OS: `ipo_out.obj` |
<p>| <code>-qipo_fas</code> | Add source lines to assembly listing. |</p>
<table>
<thead>
<tr>
<th>Linking Tools Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-qipo_fac</td>
<td>Adds code bytes to the assembly listing.</td>
</tr>
<tr>
<td>-qipo_facs</td>
<td>Add code bytes and source lines to assembly listing.</td>
</tr>
<tr>
<td>-quseenv</td>
<td>Disables override of existing PATH, LIB, and INCLUDE variables.</td>
</tr>
<tr>
<td>-lib</td>
<td>Invokes librarian instead of linker.</td>
</tr>
<tr>
<td>-libtool</td>
<td>Mac OS X: Invokes libtool to create a library instead of ld.</td>
</tr>
<tr>
<td>-qv</td>
<td>Displays version information.</td>
</tr>
</tbody>
</table>

See Also
- Related Options
- Using IPO

Portability Options
A challenge in porting applications from one compiler to another is making sure there is support for the compiler options you use to build your application. The Intel® C++ Compiler supports many of the options that are valid on other compilers you may be using.

The following sections list compiler options that are supported by the Intel® C++ Compiler and the following:
- Microsoft* C++ compiler
- gcc* Compiler

Options that are unique to either compiler are not listed in these sections.

Options Equivalent to Microsoft* C++ Options (Windows* OS)
The following table lists compiler options that are supported by both the Intel® C++ Compiler and the Microsoft* C++ compiler.

/arch:<SSE|SSE2>
/G2
/GA
/Gd
/Ge
/GF
/Gh
/GH
/Gr
/GR[-]
/GS[-]
/Gs[num]
/GT
/GX[-]
/Gy[-]
/Gz
/GZ
/H<num>
/help
/I<dir>
/J
/LD
/RTCc
/RTCs
/RTCu
/showIncludes
/TC
/Tc<source file>
/TP
/Tp<source file>
/u
/U<string>
/V<string>
/vd
/vmb
/vmg
/vmm
/vms
/vmv
/w
/W<n>
/Wall
/wd<n>
Options Equivalent to gcc* Options (Linux* OS)

The following table lists compiler options that are supported by both the Intel® C++ Compiler and the gcc* compiler.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/we&lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>/WL</td>
<td></td>
</tr>
<tr>
<td>/Wp64</td>
<td></td>
</tr>
<tr>
<td>/WX</td>
<td></td>
</tr>
<tr>
<td>/X</td>
<td></td>
</tr>
<tr>
<td>/Y-</td>
<td></td>
</tr>
<tr>
<td>/Yc[file]</td>
<td></td>
</tr>
<tr>
<td>/Yu[file]</td>
<td></td>
</tr>
<tr>
<td>/Z7</td>
<td></td>
</tr>
<tr>
<td>/Za</td>
<td></td>
</tr>
<tr>
<td>/zc:arg1[,arg2]</td>
<td></td>
</tr>
<tr>
<td>/Ze</td>
<td></td>
</tr>
<tr>
<td>/Zg</td>
<td></td>
</tr>
<tr>
<td>/Zi</td>
<td></td>
</tr>
<tr>
<td>/ZI</td>
<td></td>
</tr>
<tr>
<td>/Zl</td>
<td></td>
</tr>
<tr>
<td>/Zp[n]</td>
<td></td>
</tr>
<tr>
<td>/zs</td>
<td></td>
</tr>
</tbody>
</table>
-f[no-]inline
-ff[no-]inline-functions
-ff[no-]math-errno
-ff[no-]operator-names
-ff[no-]stack-protector
-ff[no-]unsigned-bitfields
-ffpack-struct
-ffpermissive
-ffPIC
-ffpic
-ffreg-struct-return
-ffshort-enums
-ffsyntax-only
-fftemplate-depth
-fftls-model=global-dynamic
-fftls-model=initial-exec
-fftls-model=local-dynamic
-fftls-model=local-exec
-ffunroll-loops
-ffunsigned-char
-ffverbose-asm
-fvisibility=default
-fvisibility=hidden
-fvisibility=internal
-fvisibility=protected
-H
-help
-I
-ldirafter
-imacros
-iprefix
-ivwithprefix
-ivwithprefixbefore
-l
-L
-M
-malign-double
-march
-mcpu
-MD
-MF
-MG
-MM
-MM
-m[no-]ieee-fp
-MP
-mp
-MQ
-msse
-msse2
-msse3
-NT
-ntune
-nodefaultlibs
-nostartfiles
-nostdinc
-nostdinc++
-nostdlib
-o
-O
-O0
-O1
-O2
-O3
-Ob
-p
-s
-shared
-static
-std
-trigraphs
-U
-u
-v
-V
-w
-Wall
-Werror
-Winline
-W[no-]cast-qual
-W[no-]comment
-W[no-]comments
-W[no-]deprecated
-W[no-]fatal-errors
- W[no-] format-security
- W[no-] main
- W[no-] missing-declarations
- W[no-] missing-prototypes
- W[no-] overflow
- W[no-] overloaded-virtual
- W[no-] pointer-arith
- W[no-] return-type
- W[no-] strict-prototypes
- W[no-] trigraphs
- W[no-] uninitialized
- W[no-] unknown-pragmas
- W[no-] unused-function
- W[no-] unused-variable
- X
- X assembler-with-cpp
- X c
- X c++
- X linker
Unsupported Microsoft Visual Studio* Compiler Options

The Intel® C++ Compiler supports most of the same options as the Microsoft Visual Studio* compiler. However, a small subset of Microsoft Visual Studio compiler options are not supported by the Intel C++ Compiler. Most of the unsupported options, while useful for development purposes, are not required to build a working application. The following table lists some of these unsupported options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/AI&lt;dir&gt;</td>
<td>Add to assembly search path.</td>
</tr>
<tr>
<td>/clr</td>
<td>Compile for the common language runtime (managed C++).</td>
</tr>
<tr>
<td>/fa-v:blend</td>
<td>Use /QaxSSE3, /arch:SSE2, and /QxSSE3 instead.</td>
</tr>
<tr>
<td>/Fd</td>
<td>Name the PDB file used for debug information for specified source files.</td>
</tr>
<tr>
<td>/FU&lt;file&gt;</td>
<td>Force using assembly/module.</td>
</tr>
<tr>
<td>/Fx</td>
<td>Merge injected code to file.</td>
</tr>
<tr>
<td>/Gi</td>
<td>Enable incremental compilation.</td>
</tr>
<tr>
<td>/openmp</td>
<td>Use /Qopenmp instead.</td>
</tr>
<tr>
<td>/w&lt;l&gt;&lt;n&gt;</td>
<td>Set warning level 1-4 for $n$</td>
</tr>
<tr>
<td>/Yd</td>
<td>Put debug information in every object (Microsoft PCH-specific option)</td>
</tr>
<tr>
<td>/Zm&lt;n&gt;</td>
<td>Control of maximum memory allocated by the compiler.</td>
</tr>
</tbody>
</table>

Unsupported options are not limited to this list.

The Intel C++ Compiler issues a remark stating lack of support for many of these options, but it silently ignores the options.
Floating-Point Environment

The floating-point (FP) environment is a collection of registers that control the behavior of FP machine instructions and indicate the current FP status. The floating-point environment may include rounding mode controls, exception masks, flush-to-zero controls, exception status flags, and other floating-point related features.
Part

III

Optimizing Applications

Topics:

- Overview: Optimizing Applications
- Evaluating Performance
- Using Compiler Optimizations
- Using Parallelism: OpenMP* Support
- Using Parallelism: Automatic Parallelization
- Using Parallelism: Automatic Vectorization
- Using Interprocedural Optimization (IPO)
- Using Profile-Guided Optimization (PGO)
- Using High-Level Optimization (HLO)
- Optimization Support Features
- Programming Guidelines
Overview: Optimizing Applications

Optimizing Applications explains how to use the Intel® C++ Compiler to help improve application performance.

How you use the information presented in this document depends on what you are trying to accomplish. You can start with the following topics:

- Optimizing with the Intel Compiler
- Optimizing for Performance
- Overview of Parallelism Methods
- Quick Reference Lists
- Other Resources
- Performance Analysis

Where applicable this document explains how compiler options and optimization methods differ on IA-32, Intel® 64, and IA-64 architectures on Linux* operating systems (OS), Intel®-based systems running Mac OS* X, and Windows* operating systems.

While the compiler supports Integrated Development Environments (IDE) on several different operating systems, the concepts and examples included here illustrate using the compiler from the command line.

In most cases, the compiler features and options supported for IA-32 or Intel® 64 architectures on Linux OS are also supported on Intel-based systems running Mac OS X. For more detailed information about support for specific operating systems, refer to the appropriate option in Compiler Options.

Optimizing with the Intel® Compiler

The Intel compiler supports a variety of options and features that allow optimization opportunities; however, in most cases you will benefit by applying optimization strategies in the order listed below.

Use Automatic Optimizations

Use the automatic optimization options, like -O1, -O2, -O3, or -fast (Linux* and Mac OS* X) and /O1, /O2, /O3, or /fast (Windows*) to determine what works best for your application. Use these options and measure the resulting performance after each compilation.
NOTE. The optimizer that integrates parallelization (IA-32, Intel® 64, and IA-64 architectures) and vectorization (IA-32 and Intel® 64 architectures) has been redesigned to provide performance improvements when specifying the -02 or -03 (Linux and Mac OS X) and /02 or /03 (Windows) options.

You might find specific options to work better on a particular architecture:

- IA-32 and Intel® 64 architectures: start with -02 (Linux and Mac OS X) or /02 (Windows).
- IA-64 architecture: start with -03 (Linux and Mac OS X) or /03 (Windows).

If you plan to run your application on specific architectures, experiment with combining the automatic optimizations with compiler options that specifically target processors.

You can combine the -x and -ax (Linux* and Mac OS* X) or /Qx and /Qax (Windows*) options to generate both code that is optimized for specific Intel processors and generic code that will run on most processors based on IA-32 and Intel® 64 architectures. See the following topics for more information about targeting processors:

- Targeting IA-32 and Intel® 64 Architecture Processors Automatically
- Targeting Multiple IA-32 and Intel 64 Architecture Processors Automatically for Run-time Performance
- Targeting Itanium® Processors Automatically

Attempt to combine the automatic optimizations with the processor-specific options before applying other optimizations techniques.

Use IPO and PGO

Experiment with Interprocedural Optimization (IPO) and Profile-guided Optimization (PGO). Measure performance after applying the optimizations to determine whether the application performance improved.

Use a top-down, iterative method for identifying and resolving performance-hindering code using performance monitoring tools, like the compiler reports.

Use Parallelism

If you are planning to run the application on multi-core or multi-processor systems, start the parallelism process by using the parallelism options or OpenMP* options.
Compiling for Older Processors

Use automatic optimization options and other processor-independent compiler options to generate optimized code that do not take advantage of advances in processor design or extension support. Use the \texttt{-x} (Linux* and Mac OS X) or \texttt{/Qx} (Windows*) option to generate processor dispatch for older processors.

Optimizing for Performance

The following table lists possible starting points for your optimization efforts.

<table>
<thead>
<tr>
<th>If you are trying to...</th>
<th>Start with these topics or sections.</th>
</tr>
</thead>
<tbody>
<tr>
<td>use performance analysis to begin the optimization process</td>
<td>• Optimizing with Intel® Compilers  \newline • Using a Performance Enhancement Methodology  \newline • Intel® Performance Analysis Tools and Libraries  \newline • Performance Enhancement Strategies</td>
</tr>
<tr>
<td>optimize for speed or a specific architecture</td>
<td>• Enabling Automatic Optimizations  \newline • Targeting IA-32 and Intel® 64 Architecture Processors Automatically  \newline • Targeting Multiple IA-32 and Intel® 64 Architecture Processors for Run-time Performance  \newline • Targeting IA-64 Architecture Processors Automatically</td>
</tr>
<tr>
<td>create parallel programs or parallelize existing programs</td>
<td>• Using Parallelism  \newline • Automatic Vectorization Overview  \newline • OpenMP* Support Overview  \newline • Auto-parallelization Overview</td>
</tr>
<tr>
<td>use Interprocedural Optimization</td>
<td>• Using IPO  \newline • IPO for Large Programs</td>
</tr>
</tbody>
</table>
Overview of Parallelism Method

The three major features of parallel programming supported by the Intel® compiler include:

- OpenMP*
- Auto-parallelization
- Auto-vectorization

Each of these features contributes to application performance depending on the number of processors, target architecture (IA-32, Intel® 64, and IA-64 architectures), and the nature of the application. These features of parallel programming can be combined to contribute to application performance.

Parallelism defined with the OpenMP* API is based on thread-level and task-level parallelism. Parallelism defined with auto-parallelization techniques is based on thread-level parallelism (TLP). Parallelism defined with auto-vectorization techniques is based on instruction-level parallelism (ILP).

Parallel programming can be explicit, that is, defined by a programmer using the OpenMP* API and associate options. Parallel programming can also be implicit, that is, detected automatically by the compiler. Implicit parallelism implements auto-parallelization of outer-most loops and auto-vectorization of innermost loops (or both).
To enhance the compilation of the code with auto-vectorization, users can also add to their program.

**NOTE.** Software pipelining (SWP), a technique closely related to auto-vectorization, is available on systems based on IA-64 architecture.

The following table summarizes the different ways in which parallelism can be exploited with the Intel® Compiler.

Intel provides performance libraries that contain highly optimized, extensively threaded routines, including the Intel® Math Kernel Library (Intel® MKL) and the Intel® Integrated Performance Primitives (Intel® IPP).

In addition to these major features supported by the Intel compiler, certain operating systems support application program interface (API) function calls that provide explicit threading controls. For example, Windows* operating systems support API calls such as CreateThread, and multiple operating systems support POSIX* threading APIs. Intel also provides the Intel® Threading Building Blocks (Intel® TBB), a C++ run-time library that helps simplify threading for scalable, multi-core performance.

<table>
<thead>
<tr>
<th>Parallelism Method</th>
<th>Supported On</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Implicit</strong> (parallelism generated by the compiler and by user-supplied hints)</td>
<td></td>
</tr>
<tr>
<td>Auto-parallelization</td>
<td>- IA-32 architecture, Intel® 64 architecture, IA-64 architecture based multi-processor systems, and multi-core processors</td>
</tr>
<tr>
<td>(Thread-Level Parallelism)</td>
<td>- Hyper-Threading Technology-enabled systems</td>
</tr>
<tr>
<td>Auto-vectorization</td>
<td>- Pentium®, Pentium with MMX™ Technology, Pentium II, Pentium III, Pentium 4 processors, Intel® Core™ processor, and Intel® Core™ 2 processor.</td>
</tr>
<tr>
<td>(Instruction-Level Parallelism)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Explicit</strong> (parallelism programmed by the user)</td>
<td></td>
</tr>
<tr>
<td>OpenMP* (Thread-Level and Task-Level Parallelism)</td>
<td>- IA-32 architecture, Intel® 64 architecture, IA-64 architecture-based multiprocessor systems, and multi-core processors</td>
</tr>
<tr>
<td></td>
<td>- Hyper-Threading Technology-enabled systems</td>
</tr>
</tbody>
</table>
Threading Resources

For general information about threading an existing serial application or design considerations for creating new threaded applications, see Other Resources and the web site http://go-parallel.com.

To display diagnostic messages about the use of global variables, use the Intel C++ Compiler option `-diag-enable thread` (Linux* and Mac OS* X) or `/Qdiag-enable thread` (Windows*). For example, when threading an existing serial application, the diagnostic messages can help you identify places where you need to protect access to global variables. For more information about this option, see `-diag` in Compiler Options.

Quick Reference Lists

There are several quick reference guides in this document. Use these quick reference guides to quickly familiarize yourself with the compiler options or features available for specific optimizations.

- Enabling Automatic Optimizations
- Interprocedural Optimization (IPO) Quick Reference
- Profile-guided Optimization (PGO) Quick Reference
- Auto-Vectorization Options Quick Reference
- OpenMP* Options Quick Reference
- Auto-Parallelization Options Quick Reference
- Compiler Reports Quick Reference
- PGO Environment Variables
- OpenMP* Environment Variables

Other Resources

Understanding the capabilities of the specific processors and the underlying architecture on which your application will run is key to optimization. Intel distributes many hardware and software development resources that can help better understand how to optimize application source code for specific architectures.
**Processor Information**

You can find detailed information about processor numbers, capabilities, and technical specifications, along with documentation, from the following web sites:

- Intel® Processor Spec Finder (http://processorfinder.intel.com/)
- Intel® Processor Numbers (http://www.intel.com/products/processor_number/)
- Intel® Processor Identification Utility (http://www.intel.com/support/processors/tools/piu/)

**Architecture Information**

The architecture manuals provide specific details about the basic architecture, supported instruction sets, programming guidelines for specific operating systems, and performance monitoring.

The optimization manuals provide insight for developing high-performance applications for Intel® architectures.

- IA-64 Architecture: http://www.intel.com/design/itanium/documentation.htm

**Optimization Strategy Resources**

For more information on advanced or specialized optimization strategies, refer to the Resource Centers for Software Developers, which can be accessed from www.intel.com. Refer to the articles, community forums, and links to additional resources in the listed areas of the following Developer Centers:

**Tools and Technologies:**

- Threading/Multi-core
- Intel® Software Products

**Intel® Processors:**

- Intel® 64 and IA-32 Architectures Software Developer’s Manuals
- Itanium® Processor Family
- Pentium® 4 Processor
- Intel® Xeon® Processor
Environments:

- High Performance Computing
Evaluating Performance

Performance Analysis

The high-level information presented in this section discusses methods, tools, compiler options, and pragmas used for analyzing runtime performance-related problems and increasing application performance.

The topics in this section discuss performance issues and methods from a general point of view, focusing primarily on general performance enhancements. The information in this section is separated in the following topics:

- Using a Performance Enhancement Methodology
- Performance Enhancement Strategies
- Using Intel Performance Analysis Tools

In most cases, other sections and topics in this document contain detailed information about the options, concepts, and strategies mentioned in this section.

This section also contains information about using the compiler-supporting reports and diagnostics.

Using a Performance Enhancement Methodology

The recommended performance enhancement method for optimizing applications consists of several phases. When attempting to identify performance issues, move through the following general phases in the order presented:

- Gather performance data
- Analyze the data
- Generate alternatives
- Implement enhancements
- Test the results

The following figure shows the methodology phases and their relationships, along with some recommended tools to use in each appropriate phase.
The methodology can be summarized by the following statements:

- Make small changes and measure often.
- If you approach a point of diminishing return and can find no other performance issues, stop optimizing.

Gather performance data

Use tools to measure where performance bottlenecks occur; do not waste time guessing. Using the right tools for analysis provides an objective data set and baseline criteria to measure implementation changes and improvements introduced in the other stages.
See Using Intel Performance Analysis Tool and Libraries for more information about some tools you can use to gather performance data.

**Analyze the data**

Determine if the data meet your expectations about the application performance. If not, choose one performance problem at a time for special interest. Limiting the scope of the corrections is critical in effective optimization.

In most cases, you will get the best results by resolving hotspots first. Since hotspots are often responsible for excessive activity or delay, concentrating on these areas tends to resolve or uncover other performance problems that would otherwise be undetectable.

**Generate alternatives**

As in the analysis phase, limit the focus of the work. Concentrate on generating alternatives for the one problem area you are addressing. Identify and use tools and strategies to help resolve the issues. For example, you can use compiler optimizations, use Intel® Performance Library routines, or use some other optimization (like improved memory access patterns, reducing or eliminating division or other floating-point operations, rewriting the code to include intrinsics or assembly code, or other strategies).

See Performance Enhancement Strategies for suggestions.

While optimizing for the compiler and source levels, consider using the following strategies in the order presented:

1. Use available supported compiler options. This is the most portable, least intrusive optimization strategy.
2. Use compiler directives embedded in the source. This strategy is not overly intrusive since the method involves including a single line in code, which can be ignored (optionally) by the compiler.

The preferred strategy within optimization is to use available compiler intrinsics. Intrinsics are usually small single-purpose built-in library routines whose function names usually start with an underscore (_), such as the \_mm_prefetch intrinsic.

If intrinsics are not available, try to manually apply the optimization. Manual optimizations, both high-level language and assembly, are more labor intensive, more prone to error, less portable, and more likely to interfere with future compiler optimizations that become available.

See C++ Intrinsics Reference for more information about the available intrinsics.
Implement enhancements

As with the previous phases, limit the focus of the implementation. Make small, incremental changes. Trying to address too many issues at once can defeat the purpose and reduce your ability to test the effectiveness of your enhancements.

The easiest enhancements will probably involve enabling common compiler optimizations for easy gains. For applications that can benefit from the libraries, consider implementing Intel® Performance Library routines that may require some interface coding.

Test the results

If you have limited the scope of the analysis and implementation, you should see measurable differences in performance in this phase. Have a target performance level in mind so you know when you have reached an acceptable gain in performance.

Use a consistent, reliable test that reports a quantifiable item, like seconds elapsed, frames per second, and so forth, to determine if the implementation changes have actually helped performance.

If you think you can make significant improvement gains or you still have other performance issues to address, repeat the phases beginning with the first one: gather performance data.

Intel® Performance Analysis Tools and Libraries

Intel Corporation offers a variety of performance analysis tools and libraries that can help you optimize your application performance.

Performance Analysis Tools

These performance tools can help you analyze your application, find problem areas, and develop efficient programs. In some cases, these tools are critical to the optimization process.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Operating System</th>
<th>Description</th>
</tr>
</thead>
</table>
The Intel® Thread Checker can help identify shared and private variable conflicts, and can isolate threading bugs to the source code line where the bug occurs.

The Intel® Thread Profiler can show the critical path of an application as it moves from thread to thread, and identify synchronization issues and excessive blocking time that cause delays for Win32*, POSIX* threaded, and OpenMP* code.

Performance Libraries

These performance libraries can decrease development time and help to increase application performance.

<table>
<thead>
<tr>
<th>Library</th>
<th>Operating System</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Threading Building Blocks</td>
<td>Linux, Mac OS X, Windows</td>
<td>Intel® Threading Building Blocks is a C++ runtime library that simplifies threading for performance.</td>
</tr>
<tr>
<td>Intel® Integrated Performance Primitives</td>
<td>Linux, Mac OS X, Windows</td>
<td>Intel® Integrated Performance Primitives is a software library of performance-optimized functions for digital media and data processing applications.</td>
</tr>
</tbody>
</table>
**Intel® Math Kernel Library**

*Operating System*: Linux, Mac OS X, Windows

**Description**: Intel® Math Kernel Library offers highly optimized, thread-safe math routines for science, engineering, and financial applications that require maximum performance.

---

**Performance Enhancement Strategies**

Improving performance starts with identifying the characteristics of the application you are attempting to optimize. The following table lists some common application characteristics, indicates the overall potential performance impact you can expect, and provides suggested solutions to try. These strategies have been found to be helpful in many cases; experimentation is key with these strategies.

In the context of this discussion, view the potential impact categories as an indication of the possible performance increases that might be achieved when using the suggested strategy. It is possible that application or code design issues will prohibit achieving the indicated increases; however, the listed impacts are generally true. The impact categories are defined in terms of the following performance increases, when compared to the initially tested performance:

- **Significant**: more than 50%
- **High**: up to 50%
- **Medium**: up to 25%
- **Low**: up to 10%

The following table is ordered by application characteristics and then by strategy with the most significant potential impact.

<table>
<thead>
<tr>
<th>Application Characteristics</th>
<th>Impact</th>
<th>Suggested Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical applications with loopy code</td>
<td>High</td>
<td>Technical applications are those programs that have some subset of functions that consume a majority of total CPU cycles in loop nests.</td>
</tr>
<tr>
<td>Application Characteristics</td>
<td>Impact</td>
<td>Suggested Strategies</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>----------------------</td>
</tr>
<tr>
<td>(same as above) IA-64 architecture only</td>
<td>High</td>
<td>Target loop nests using <code>-03</code> (Linux* and Mac OS* X) or <code>/03</code> (Windows*) to enable more aggressive loop transformations and prefetching. Use High-Level Optimization (HLO) reporting to determine which HLO optimizations the compiler elected to apply. See High-Level Optimization Report. For <code>-02</code> and <code>-03</code> (Linux) or <code>/02</code> and <code>/03</code> (Windows), use the <code>swp report</code> to determine if Software Pipelining occurred on key loops, and if not, why not. You might be able to change the code to allow software pipelining under the following conditions: • If recurrences are listed in the report that you suspect do not exist, eliminate aliasing problems (for example, using the restrict keyword), or use <code>ivdep</code> pragma on the loop. • If the loop is too large or runs out of registers, you might be able to distribute the loop into smaller segments; distribute the loop manually or by using the <code>distribute</code> pragma.</td>
</tr>
<tr>
<td>Application Characteristics</td>
<td>Impact</td>
<td>Suggested Strategies</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If the compiler determines the Global Acyclic Scheduler can produce better results but you think the loop should still be pipelined, use the \texttt{SWP} pragma on the loop.</td>
</tr>
<tr>
<td>(same as above) IA-32 and Intel® 64 architectures only</td>
<td>High</td>
<td>See Vectorization Overview and the remaining topics in the Auto-Vectorization section for applicable options. See Vectorization Report for specific details about when you can change code.</td>
</tr>
<tr>
<td>(same as above)</td>
<td>Medium</td>
<td>Use PGO profile to guide other optimizations. See Profile-guided Optimizations Overview.</td>
</tr>
<tr>
<td>Applications with many denormalized floating-point value operations</td>
<td>Significant</td>
<td>Experiment with \texttt{-fp-model fast=2} (Linux and Mac OS X) or \texttt{/fp:fast=2} or \texttt{-ftz} (Linux and Mac OS X) or \texttt{/Qftz} (Windows). The resulting performance increases can adversely affect floating-point calculation precision and reproducibility. See Floating-point Operations for more information about using the floating point options supported in the compiler.</td>
</tr>
<tr>
<td>Sparse matrix applications</td>
<td>Medium</td>
<td>See the suggested strategy for memory pointer disambiguation (below).</td>
</tr>
<tr>
<td>Application Characteristics</td>
<td>Impact</td>
<td>Suggested Strategies</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Flat profile applications</td>
<td>Medium</td>
<td>Flat profile applications are those applications where no single module seems to consume CPU cycles inordinately. Use PGO to communicate typical hot paths and functions to the compiler, so the Intel® compiler can arrange code in the optimal manner. Use PGO on as much of the application as is feasible. See Profile-guided Optimizations Overview.</td>
</tr>
<tr>
<td>Very large server applications (similar to above)</td>
<td>Low</td>
<td>Use -O1 (Linux and Mac OS X) or /O1 (Windows) to optimize for this type of application. Streamlines code in the most generic manner available. This strategy reduces the amount of code being generated, disables inlining, disables speculation, and enables caching of as much of the instruction code as possible.</td>
</tr>
<tr>
<td>Application Characteristics</td>
<td>Impact</td>
<td>Suggested Strategies</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Database engines</td>
<td>Medium</td>
<td>Use <code>-O1</code> (Linux and Mac OS X) or <code>/O1</code> (Windows) and PGO to optimize the application code.</td>
</tr>
<tr>
<td>(same as above)</td>
<td>Medium</td>
<td>Use <code>-ipo</code> (Linux and Mac OS X) or <code>/Qipo</code> (Windows) on entire application. See Interprocedural Optimizations Overview.</td>
</tr>
<tr>
<td>Other Application Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications with many small functions that are called from multiple locations</td>
<td>Low</td>
<td>Use <code>-ip</code> (Linux and Mac OS X) or <code>/Qip</code> (Windows) to enable inter-procedural inlining within a single source module. Streamlines code execution for simple functions by duplicating the code within the code block that originally called the function. This will increase application size. As a general rule, do not inline large, complicated functions. See Interprocedural Optimizations Overview.</td>
</tr>
<tr>
<td>(same as above)</td>
<td>Low</td>
<td>Use <code>-ipo</code> (Linux and Mac OS X) or <code>/Qipo</code> (Windows) to enable inter-procedural inlining both within and between multiple source modules. You might experience an additional</td>
</tr>
</tbody>
</table>
Suggested Strategies

Impact Application

increase over using `-ip` (Linux and Mac OS X) or `/Qip` (Windows).

Using this option will increase link time due to the extended program flow analysis that occurs.

Use Interprocedural Optimization (IPO) to attempt to perform whole program analysis, which can help memory pointer disambiguation.

Apart from application-specific suggestions listed above, there are many application-, OS/Library-, and hardware-specific recommendations that can improve performance as suggested in the following tables:

### Application-specific Recommendations

<table>
<thead>
<tr>
<th>Application Area</th>
<th>Impact</th>
<th>Suggested Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Blocking</td>
<td>High</td>
<td>Use <code>-O3</code> (Linux and Mac OS X) or <code>/O3</code> (Windows) to enable automatic cache blocking; use the HLO report to determine if the compiler enabled cache blocking automatically. If not consider manual cache blocking. See Cache Blocking.</td>
</tr>
<tr>
<td>Compiler pragmas for better alias analysis</td>
<td>Medium</td>
<td>Ignore vector dependencies. Use <code>ivdep</code> and other pragmas to increase application speed. See .</td>
</tr>
<tr>
<td>Memory pointer disambiguation compiler keywords and options</td>
<td>Medium</td>
<td>Use restrict keyword and the <code>-restrict</code> (Linux and Mac OS X) or <code>/Qrestrict</code></td>
</tr>
<tr>
<td>Application Area</td>
<td>Impact</td>
<td>Suggested Strategies</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Windows) option to disambiguate memory pointers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you use the restrict keyword in your source code, you must use <code>restrict</code> (Linux and Mac OS X) or <code>/Qrestrict</code> (Windows) option during compilation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Instead of using the restrict keyword and option, you can use the following compiler options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>fno-fnalias</code> (Linux and Mac OS X)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>ansi-alias</code> (Linux and Mac OS X) or <code>/Qansi-alias</code> (Windows)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>/Oa</code> (Windows)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>/Ow</code> (Windows)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>fargument-alias</code> (Linux and Mac OS X) or <code>/Qalias-args</code> (Windows)</td>
</tr>
<tr>
<td>Light-weight volatile</td>
<td>Low</td>
<td>Some application use volatile to ensure memory operations occur, the application does not need strong memory ordering in the hardware.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IA-64 architecture:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use <code>mno-serialize-volatile</code>.</td>
</tr>
</tbody>
</table>
### Suggested Strategies

<table>
<thead>
<tr>
<th>Application Area</th>
<th>Impact</th>
<th>Suggested Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math functions</td>
<td>Low</td>
<td>Use float intrinsics for single precision data type, for example, <code>sqrtf()</code> not <code>sqrt()</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call Math Kernel Library (MKL) instead of user code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call F90 intrinsics instead of user code (to enable optimizations).</td>
</tr>
<tr>
<td>Use intrinsics instead of calling a function in assembly code</td>
<td>Low</td>
<td>The Intel® compiler includes intrinsics; use these intrinsics to optimize your code. Using compiler intrinsics can help to increase application performance while helping to your code to be more portable.</td>
</tr>
</tbody>
</table>

### Library/OS Recommendations

<table>
<thead>
<tr>
<th>Area</th>
<th>Impact</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Low</td>
<td>Systems on IA-64 architecture only. If you have been using the <code>setjump</code> function, you might consider using the <code>light (_setjmp)</code> version instead of the heavy (<code>setjmp</code>) version of the function to reduce the amount of floating-point state saved in the <code>setjmp</code> buffer.</td>
</tr>
<tr>
<td>Symbol preemption</td>
<td>Low</td>
<td>Linux has a less performance-friendly symbol preemption model than Windows. Linux uses full preemption, and Windows</td>
</tr>
</tbody>
</table>
### Memory allocation
- **Impact**: Low
- **Description**: Using third-party memory management libraries can help improve performance for applications that require extensive memory allocation.

### Hardware/System Recommendations

<table>
<thead>
<tr>
<th>Component</th>
<th>Impact</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>Medium</td>
<td>Consider using more advanced hard drive storage strategies. For example, consider using SCSI instead of IDE. Consider using the appropriate RAID level. Consider increasing the number hard drives in your system.</td>
</tr>
<tr>
<td>Memory</td>
<td>Low</td>
<td>You can experience performance gains by distributing memory in a system. For example, if you have four open memory slots and only two slots are populated, populating the other two slots with memory will increase performance.</td>
</tr>
<tr>
<td>Processor</td>
<td></td>
<td>For many applications, performance scales is directly affected by processor speed,</td>
</tr>
</tbody>
</table>
Using Compiler Reports

Compiler Reports Overview
The Intel® compiler provides several reports that can help identify performance issues. Some of these compiler reports are architecture-specific, most are more general. Start with the general reports that are common to all platforms (operating system/architecture), then use the reports unique to an architecture.

- Generating reports
- Interprocedural Optimizations (IPO) report
- Profile-Guided Optimizations (PGO) report
- High-level Optimizations (HLO) report
- Software-pipelining (SWP) report (IA-64 architecture only)
- Vectorization report
- Parallelism report
- OpenMP® report

Compiler Reports Quick Reference
The Intel® compiler provides the following options to generate and manage optimization reports:

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-opt-report</td>
<td>/Qopt-report</td>
<td>Generates optimization report, with different levels of detail, directed to stderr. Valid values for N are 0 through 3. By default, when you specify this option without passing a value the</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>-opt-report-phase</td>
<td>/Qopt-report-phase</td>
<td>compiler will generate a report with a medium level of detail. Specifies the optimization phase to use when generating reports. If you do not specify a phase the compiler defaults to all, which can adversely affect compile times. See Generating Reports for more information about the supported phases.</td>
</tr>
<tr>
<td>-opt-report-file</td>
<td>/Qopt-report-file</td>
<td>Generates an optimization report and directs the report output to the specified file name. If the file is not in the local directory, supply the full path to the output file. This option overrides the opt-report option. Generates reports from all routines with names containing a string as part of their name; pass the string as an argument to this option. If not specified, the compiler will generate reports on all routines. This option is not a compiler reports option; however, it controls the diagnostic message levels generated by the compiler, and it offers a means to provide useful information without needing</td>
</tr>
</tbody>
</table>
Generating Reports

Use the options listed in this topic to generate reports on the following optimizers.

- Interprocedural Optimization (IPO)
- Profile-guided Optimization (PGO)
- High Performance Optimizer (HPO)
- High-level Optimization (HLO)
- Intermediate Language Scalar Optimization (ILO)
- Software Pipelining (SWP)

Specify an optimizer phase by passing the `phase` argument to the `-opt-report-phase` (Linux* and Mac OS* X) or `/Qopt-report-phase` (Windows*) option.

<table>
<thead>
<tr>
<th>Optimizer Phase</th>
<th>Supported Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgo</td>
<td>Profile-guided Optimizer</td>
</tr>
<tr>
<td>ipo</td>
<td>Interprocedural Optimizer</td>
</tr>
<tr>
<td>ilo</td>
<td>Intermediate Language Scalar Optimizer</td>
</tr>
<tr>
<td>hpo</td>
<td>High Performance Optimizer</td>
</tr>
<tr>
<td>hlo</td>
<td>High-level Optimizer</td>
</tr>
<tr>
<td>ecg</td>
<td>Itanium® Compiler Code Generator</td>
</tr>
<tr>
<td></td>
<td>Mac OS X: This phase is not supported.</td>
</tr>
</tbody>
</table>

This quick reference does not list the options for the vectorization, parallelism, or OpenMP* reports.

Refer to Quick Reference Lists for a complete listing of the quick reference topics.

If you use interprocedural optimization (IPO) options, you can request compiler reports when using the xi* tools, as described in Requesting Compiler Reports with the xi* Tools.
The software pipelining component of the 
Code Generator phase (Windows and Linux 
systems using IA-64 architecture only)

All optimizers supported on the architecture. 
This is not recommended; the resulting 
output can be too extensive to be useful. 
Experiment with targeted phase reports first.

### Reports Available by Architecture and Option

**IA-32, Intel® 64, and IA-64 architectures:**

- `ilo` and `pgo`
- `hpo` and `hlo`: For IA-32 architecture, supported with `-x` (Linux and Mac OS X) or `/Qx` (Windows) option.
  
  For Intel® 64 architecture, supported with `-O2` (Linux and Mac OS X) or `/O2` (Windows) option. For IA-32 and Intel® 64 architectures, a subset of these optimizations are enabled at default optimization level (`-O2`). For IA-64 architecture, supported with `-O3` (Linux and Mac OS X) or `/O3` (Windows) option.
- `ipo`: Interprocedural optimization is enabled for `-O2` (Linux and Mac OS X) or `/O2` (Windows) option or above.
- `all`: All of the above.

**IA-64 architecture only:**

- `ecg`

### Running the Reports

Use syntax similar to the following to run the compiler reports.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Sample Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -c -opt-report 2 -opt-report-phase=all sample.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /c/Qopt-report:2 /Qopt-report-phase:all sample.cpp</td>
</tr>
</tbody>
</table>
The sample command instructs the compiler to generate a report and send the results to `stderr` and specifies the reports should include information about all available optimizers. In most cases, specifying `all` as the phase will generate too much information to be useful.

If you want to capture the report in an output file instead of sending it to `stderr`, specify `-opt-report-file` (Linux and Mac OS X) or `/Qopt-report-file` (Windows) and indicate an output file name. If you do not want the compiler to invoke the linker, specify `-c` (Linux and Mac OS X) or `/c` (Windows) as shown in the sample syntax above; by specifying the option you instruct the compiler to stop after generating object code and reporting the results.

See Compiler Reports Quick Reference for information about how to use the report related options.

When you specify a phase name, as shown above, the compiler generates all reports from that optimizer. The option can be used multiple times on the same command line to generate reports for multiple optimizers. For example, for if you specified `-opt-report-phase ipo -opt-report-phase hlo` (Linux and Mac OS X) or `/Qopt-report-phase ipo /Qopt-report-phase hlo` (Windows) the compiler generates reports from the interprocedural optimizer and the high-level optimizer code generator.

You do not need to fully specify an optimizer name in the command; in many cases, the first few characters should suffice to generate reports; however, all optimization reports that have a matching prefix are generated.

Each of the optimizer logical names supports many specific, targeted optimizations within them. Each of the targeted optimizations have the prefix of the optimizer name. Enter `-opt-report-help` (Linux and Mac OS X) or `/Qopt-report-help` (Windows) to list the names of optimizers that are supported. The following table lists some examples:

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipo_inl</td>
<td>Interprocedural Optimizer, inline expansion of functions</td>
</tr>
<tr>
<td>ipo_cp</td>
<td>Interprocedural Optimizer, constant propagation</td>
</tr>
<tr>
<td>hlo_unroll</td>
<td>High-level Optimizer, loop unrolling</td>
</tr>
<tr>
<td>hlo_prefetch</td>
<td>High-level Optimizer, prefetching</td>
</tr>
</tbody>
</table>

Viewing the Optimization Reports Graphically (Linux)

To generate the graphical report display, you must compile the application using the following optimization reports options, at a minimum: `-opt-report-phase` and `-opt-report-file`. 

---

1059
As with the text-based reports, the graphical report information can be generated on all architectures.

**Interprocedural Optimizations (IPO) Report**

The IPO report provides information on the functions that have been inlined and can help to identify the problem loops. The report can help to identify how and where the compiler applied IPO to the source files.

The following command examples demonstrate how to run the IPO reports with the minimum output.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux* and Mac OS* X</td>
<td>icc -opt-report 1</td>
</tr>
<tr>
<td></td>
<td>-opt-report-phase=ipo a.cpp b.cpp</td>
</tr>
<tr>
<td>Windows*</td>
<td>icl /Qopt-report:1</td>
</tr>
<tr>
<td></td>
<td>/Qopt-report-phase:ipo a.cpp b.cpp</td>
</tr>
</tbody>
</table>

where `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows) invokes the report generator, and `-opt-report-phase=ipo` (Linux and Mac OS X) or `/Qopt-report-phase:ipo` (Windows) indicates the phase (ipo) to report.

You can use `-opt-report-file` (Linux and Mac OS X) or `/Qopt-report-file` (Windows) to specify an output file to capture the report results. Specifying a file to capture the results can help to reduce the time you spend analyzing the results and can provide a baseline for future testing.
**Reading the Output**

The IPO report details information in two general sections: whole program analysis and inlining. By default, the report generates a medium level of detail. You can specify an output file to capture the report results. Running maximum IPO report results can be very extensive and technical; specifying a file to capture the results can help to reduce analysis time. The following sample report illustrates the general layout.

---

**Sample IPO Report**

<table>
<thead>
<tr>
<th>IP OPTIMIZATION REPORT:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>WHOLE PROGRAM (SAFE) [EITHER METHOD]: TRUE</td>
<td></td>
</tr>
<tr>
<td>WHOLE PROGRAM (SEEN) [TABLE METHOD]: TRUE</td>
<td></td>
</tr>
<tr>
<td>WHOLE PROGRAM (READ) [OBJECT READER METHOD]: TRUE</td>
<td></td>
</tr>
<tr>
<td>INLINING OPTION VALUES:</td>
<td></td>
</tr>
<tr>
<td>- inline-factor: 100</td>
<td></td>
</tr>
<tr>
<td>- inline-min-size: 7</td>
<td></td>
</tr>
<tr>
<td>- inline-max-size: 230</td>
<td></td>
</tr>
<tr>
<td>- inline-max-total-size: 2000</td>
<td></td>
</tr>
<tr>
<td>- inline-max-per-routine: disabled</td>
<td></td>
</tr>
<tr>
<td>- inline-max-per-compile: disabled</td>
<td></td>
</tr>
<tr>
<td>INLINING REPORT: (main) [1/5=20.0%]</td>
<td></td>
</tr>
<tr>
<td>-&gt; INLINE: _Z3bari(6) (isz = 12) (sz = 17 (5+12))</td>
<td></td>
</tr>
<tr>
<td>-&gt; _ZN10sEPFPRoSRoES_E(EXTERN)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

The following table summarizes the common report elements and provides a general description to help interpret the results.

<table>
<thead>
<tr>
<th>Report Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHOLE PROGRAM (SAFE) [EITHER METHOD]:</td>
<td>TRUE or FALSE.</td>
</tr>
<tr>
<td>Report Element</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TRUE</strong></td>
<td>The compiler determined, using one or both of the whole program analysis models, that the whole program was present during compilation.</td>
</tr>
<tr>
<td><strong>FALSE</strong></td>
<td>The compiler determined the whole program was not present during compilation. The compiler could not apply whole program IPO.</td>
</tr>
<tr>
<td><strong>TRUE</strong></td>
<td>The compiler resolved all references, either within the application code or in the standard table for the functions within the compiler.</td>
</tr>
<tr>
<td><strong>FALSE</strong></td>
<td>The compiler could not resolve all references. One or more functions references could not be found either in the user code or standard functions table.</td>
</tr>
<tr>
<td><strong>TRUE</strong></td>
<td>The compiler determined that all conditions were met for linking at a level equivalent to the (-O0) (Linux and Mac OS X) or (/O) (Windows) option.</td>
</tr>
<tr>
<td><strong>FALSE</strong></td>
<td>The compiler could not resolve one or more references. The linking step failed.</td>
</tr>
</tbody>
</table>
### INLINING OPTION VALUES:

Displays the compilation values used for the following developer-directed inline expansion options:

- inline-factor
- inline-min-size
- inline-max-size
- inline-max-total-size
- inline-max-per-routine
- inline-max-per-compile

If you specify the one or more of the appropriate options, the report lists the values you specified; if you do not specify an option and value the compiler uses the defaults values for the listed options, and the compiler will list the default values.

The values indicate the same intermediate language units listed in Compiler Options for each of these options. See Developer Directed Expansion of User Functions for more information about using these options.

Includes a string in the format of the following

\[(<name>) \ [<current number>/<total number>=<percent complete>]\]

where

- `<name>`: the name of the function being reported on.
- `<current number>` is the number of the function being reported on. Not every function can be inlined; gaps in the current number are common.
<table>
<thead>
<tr>
<th>Report Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;total number&gt;</td>
<td>is the total number of functions being evaluated.</td>
</tr>
<tr>
<td>&lt;percent complete&gt;</td>
<td>is a simple percentage of the functions being inlined.</td>
</tr>
</tbody>
</table>

If a function is inlined, the function line has the prefix "-> INLINE: ".

The option reports displays the mangled names of the functions.

The report uses the following general syntax format:

```
-> INLINE: _<name>(#) (isz) (sz)
```

where

- `<name>`: Indicates the mangled name of the function for C++; for C functions the names are not mangled. Static functions display the mangled function name with numbered suffix.
- `#`: Indicates the unique integer specifying the function number.
- `sz`: Indicates the function size before optimization. This is a rough estimate loosely representative of the number of original instructions before optimization.
- `isz`: Indicates the function size after optimization. This value (isz) will always be less than or equal to the unoptimized size (sz).
- `exec_cnt`: Indicates that Profile-guided Optimization was specified during compilation. Indicates the number of times the function was called from this site.
<table>
<thead>
<tr>
<th>Report Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTE.</strong> You can make the mangled names readable by entering the following:</td>
<td></td>
</tr>
<tr>
<td><strong>Linux:</strong> `echo &lt;mangled_name&gt;</td>
<td>c++filt`</td>
</tr>
<tr>
<td><strong>Windows:</strong> <code>undname &lt;mangled_name&gt;</code></td>
<td></td>
</tr>
<tr>
<td>Function calls that could not be inlined lack the <strong>INLINE</strong> prefix. Additionally, the compiler marks non-inlined functions using the following conventions:</td>
<td></td>
</tr>
<tr>
<td>• <strong>EXTERN</strong> indicates the function contained an external function call for which the compiler was not supplied the code.</td>
<td></td>
</tr>
<tr>
<td>• <strong>ARGS_IN_REGS</strong> indicates inlining did not occur.</td>
<td></td>
</tr>
<tr>
<td>For IA-32 and Intel® 64 architectures, an alternative for functions that were not inlined is to allow the compiler to pass the function arguments in registers rather than using standard calling conventions; however, for IA-64 architecture this is the default behavior.</td>
<td></td>
</tr>
<tr>
<td>Indicates the reported function is a dead static. Code does not need to be created for these functions. This behavior allows the compiler to reduce the overall code size.</td>
<td></td>
</tr>
</tbody>
</table>
Profile-guided Optimization (PGO) Report

The PGO report can help to identify where and how the compiler used profile information to optimize the source code. The PGO report is most useful when combined with the PGO compilation steps outlined in Profile an Application. Without the profiling data generated during the application profiling process the report will generally not provide useful information.

Combine the final PGO step with the reporting options by including `-prof-use` (Linux* and Mac OS* X) or `/Qprof-use` (Windows*). The following syntax examples demonstrate how to run the report using the combined options.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -prof-use -opt-report</td>
</tr>
<tr>
<td></td>
<td>-opt-report-phase=pgo</td>
</tr>
<tr>
<td></td>
<td>pgotools_sample.c</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Qprof-use /Qopt-report</td>
</tr>
<tr>
<td></td>
<td>/Qopt-report-phase=pgo</td>
</tr>
<tr>
<td></td>
<td>pgotools_sample.c</td>
</tr>
</tbody>
</table>

By default the PGO report generates a medium level of detail. You can use `-opt-report-file` (Linux and Mac OS X) or `/Qopt-report-file` (Windows) to specify an output file to capture the report results. Specifying a file to capture the results can help to reduce the time you spend analyzing the results and can provide a baseline for future testing.
Reading the Output

Running maximum PGO report results can produce long and detailed results. Depending on the sources being profiled, analyzing the report could be very time consuming. The following sample report illustrates typical results and element formatting for the default output.

Sample PGO Report

<table>
<thead>
<tr>
<th>Description Report Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String listing information about the function being reported on. The string uses the following format.</td>
<td>The compact string contains the following information:</td>
</tr>
<tr>
<td>• <code>&lt;source name&gt;</code>: Name of the source file being examined.</td>
<td></td>
</tr>
<tr>
<td>Report Element</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>• &lt;source name&gt;;&lt;start line&gt;;&lt;end line&gt;;&lt;optimization&gt;; &lt;function name&gt;;&lt;element type&gt;</td>
<td>Indicates that valid profiling data was generated for the function indicated; the source file containing the function is also listed.</td>
</tr>
<tr>
<td>DYN-VAL</td>
<td>Indicates that no profiling data was generated for the function indicated; the source file containing the function is also listed.</td>
</tr>
<tr>
<td>NO-DYN</td>
<td>Indicates the number of functions that had valid profile information.</td>
</tr>
<tr>
<td>FUNCTIONS HAD VALID DYNAMIC PROFILES</td>
<td>Indicated the number of functions that did not have valid profile information. This element could indicate that the function was not executed during the instrumented executable runs.</td>
</tr>
<tr>
<td>FUNCTIONS HAD NO DYNAMIC PROFILES</td>
<td>Indicates the number of functions for which static profiles were generated.</td>
</tr>
</tbody>
</table>
The most likely cause for having a non-zero number is that dynamic profiling did not happen and static profiles were generated for all of the functions.

Indicates the general quality, represented as a percentage value between 50\% and 100\%. A value of 50\% means no functions had dynamic profiles, and a value of 100\% means that all functions have dynamic profiles. The larger the number the greater the percentage of functions that had dynamic profiles.

Indicates the number of possible dynamic profiles. This number represents the best possible value, as a percentage, for Current Quality. This number is the highest possible value and represents the ideal quality for the given data set and the instrumented executable.

Indicates the ratio of Possible Quality to Current Quality. A value of 100\% indicates that all dynamic profiles were accepted. Any value less than 100\% indicates rejected profiles.

<table>
<thead>
<tr>
<th>Report Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPO CURRENT QUALITY METRIC</td>
<td>The most likely cause for having a non-zero number is that dynamic profiling did not happen and static profiles were generated for all of the functions. Indicates the general quality, represented as a percentage value between 50% and 100%. A value of 50% means no functions had dynamic profiles, and a value of 100% means that all functions have dynamic profiles. The larger the number the greater the percentage of functions that had dynamic profiles.</td>
</tr>
<tr>
<td>IPO POSSIBLE QUALITY METRIC</td>
<td>Indicates the number of possible dynamic profiles. This number represents the best possible value, as a percentage, for Current Quality. This number is the highest possible value and represents the ideal quality for the given data set and the instrumented executable.</td>
</tr>
<tr>
<td>IPO QUALITY METRIC RATIO</td>
<td>Indicates the ratio of Possible Quality to Current Quality. A value of 100% indicates that all dynamic profiles were accepted. Any value less than 100% indicates rejected profiles.</td>
</tr>
</tbody>
</table>

**High-level Optimization (HLO) Report**

High-level Optimization (HLO) performs specific optimizations based on the usefulness and applicability of each optimization. The HLO report can provide information on all relevant areas plus structure splitting and loop-carried scalar replacement, and it can provide information about interchanges not performed for the following reasons:

- Function call are inside the loop
- Imperfect loop nesting
- Reliance on data dependencies; dependencies preventing interchange are also reported.
Original order was proper but it might have been considered inefficient to perform the interchange.

For example, the report can provide clues to why the compiler was unable to apply loop interchange to a loop nest that might have been considered a candidate for optimization. If the reported problems (bottlenecks) can be removed by changing the source code, the report suggests the possible loop interchanges.

Depending on the operating system, you must specify the following options to enable HLO and generate the reports:

- **Linux* and Mac OS* X**: `-x`, `-O2` or `-O3`, `-opt-report 3`, `-opt-report-phase=hlo`
- **Windows***: `/Qx`, `/O2` or `/O3`, `/Qopt-report:3`, `/Qopt-report-phase:hlo`

See [High-level Optimization Overview](#) for information about enabling HLO.

The following command examples illustrate the general command needed to create HLO report with combined options.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -c -xSSE3 -O3 -opt-report 3 -opt-report-phase=hlo sample.cpp</td>
</tr>
</tbody>
</table>

You can use `-opt-report-file` (Linux and Mac OS X) or `/Qopt-report-file` (Windows) to specify an output file to capture the report results. Specifying a file to capture the results can help to reduce the time you spend analyzing the results and can provide a baseline for future testing.

**Reading the report results**

The report provides information using a specific format. The report format for Windows* is different from the format on Linux* and Mac OS* X. While there are some common elements in the report output, the best way to understand what kinds of advice the report can provide is to show example code and the corresponding report output.
Example 1: This example illustrates the condition where a function call is inside a loop.

```c
Example 1

void bar (int *A, int **B);
int foo (int *A, int **B, int N)
{
    int i, j;
    for (j=0; j<N; j++) {
        for (i=0; i<N; i++) {
            B[i][j] += A[j];
            bar(A, B);
        }
    }
    return 1;
}
```

Regardless of the operating system, the reports list optimization results on specific functions by presenting a line above there reported action. The line format and description are included below.

The following table summarizes the common report elements and provides a general description to help interpret the results.

<table>
<thead>
<tr>
<th>Report Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String listing information about the function being reported on. The string uses the following format.</td>
<td>The compact string contains the following information:</td>
</tr>
<tr>
<td><code>&lt;source name&gt;;&lt;start line&gt;;&lt;end line&gt;;&lt;optimization&gt;; &lt;function name&gt;;&lt;element type&gt;</code></td>
<td>- <code>&lt;source name&gt;</code>: Name of the source file being examined.</td>
</tr>
<tr>
<td>For example, the reports listed below report the following information:</td>
<td>- <code>&lt;start line&gt;</code>: Indicates the starting line number for the function being examined. A value of -1 means that the report applies to the entire function.</td>
</tr>
<tr>
<td>Report Element</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;end line&gt;</code>: Indicates the ending line number for the function being examined.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;optimization&gt;</code>: Indicates the optimization phase; for this report the indicated phase should be hlo.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;function name&gt;</code>: Name of the function being examined.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;element type&gt;</code>: Indicates the type of the report element; 0 indicates the element is a comment.</td>
</tr>
</tbody>
</table>

Linux and Mac OS X:
<sample1.c;-1:-1;hlo;foo;0>

Windows:
<sample1.c;-1:-1;hlo;_foo;0>

Windows only: This section of the report lists the following information:

- QLOOPS: Indicates the number of well-formed loops found out of the loops discovered.
- ENODE LOOPS: Indicates number of preferred forms (canonical) of the loops generated by HLO. This indicates the number of loops generated by HLO.
- unknown: Indicates the number of loops that could not be counted.
- multi_exit_do: Indicates the countable loops containing multiple exits.
- do: Indicates the total number of loops with trip counts that can be counted.
- linear_do: Indicates the number of loops with bounds that can be represented in a linear form.

Several report elements grouped together.

QLOOPS 2/2   ENODE LOOPS 2
unknown 0 multi_exit_do 0 do 2
linear_do 2
LINEAR HLO EXPRESSIONS:  17 / 18
<table>
<thead>
<tr>
<th>Report Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• LINEAR HLO EXPRESSIONS:</td>
<td>Indicates the number of expressions (first number) in all of the intermediate forms (ENODE) of the expression (second number) that can be represented in a linear form.</td>
</tr>
</tbody>
</table>

The code sample list above will result in a report output similar to the following.
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example 1 Report Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>&lt;samplel.c;-1:-1;hlo;foo;0&gt; High Level Optimizer Report (foo)</td>
</tr>
<tr>
<td></td>
<td>Block, Unroll, Jam Report: (loop line numbers, unroll factors and type of transformation)</td>
</tr>
<tr>
<td></td>
<td>&lt;samplel.c;7:7;hlo_unroll;foo;0&gt;</td>
</tr>
<tr>
<td></td>
<td>Loop at line 7 unrolled with remainder by 2</td>
</tr>
<tr>
<td>Windows</td>
<td>&lt;samplel.c;-1:-1;hlo;_foo;0&gt; High Level Optimizer Report (_foo)</td>
</tr>
<tr>
<td></td>
<td>QLOOPS 2/2 ENODE LOOPS 2 unknown 0 multi_exit_do 0 do 2 linear_do 2</td>
</tr>
<tr>
<td></td>
<td>LINEAR HLO EXPRESSIONS: 17 / 18</td>
</tr>
<tr>
<td></td>
<td>&lt;samplel.c;6:6;hlo_linear_trans;_foo;0&gt;</td>
</tr>
<tr>
<td></td>
<td>Loop Interchange not done due to: User Function Inside Loop Nest</td>
</tr>
<tr>
<td></td>
<td>Advice: Loop Interchange, if possible, might help Loopnest at lines: 6 7</td>
</tr>
<tr>
<td></td>
<td>: Suggested Permutation: (1 2 ) --&gt; ( 2 1 )</td>
</tr>
</tbody>
</table>

Example 2: This example illustrates the condition where the loop nesting prohibits interchange.

```cpp
def foo (int *A, int **B, int N)
{
    int i, j;
    for (j=0; j<N; j++) {
        A[j] = i + B[i][j];
    }
```
Example 2

```c
for (i=0; i<N; i++) {
    B[i][j] += A[j];
}
return 1;
```

The code sample listed above will result in a report output similar to the following.
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example 2 Report Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td><code>&lt;sample2.c;1:-1;hlo;foo;0&gt;</code></td>
</tr>
<tr>
<td></td>
<td>High Level Optimizer Report (foo)</td>
</tr>
<tr>
<td></td>
<td><code>&lt;sample2.c;7:7;hlo_scalar_replacement;in foo;0&gt;</code></td>
</tr>
<tr>
<td></td>
<td>#of Array Refs Scalar Replaced in foo at line 7=2</td>
</tr>
<tr>
<td></td>
<td>#of Array Refs Scalar Replaced in foo at line 7=1</td>
</tr>
<tr>
<td></td>
<td>Block, Unroll, Jam Report:</td>
</tr>
<tr>
<td></td>
<td>(loop line numbers, unroll factors and type of transformation)</td>
</tr>
<tr>
<td></td>
<td><code>&lt;sample2.c;7:7;hlo_unroll;foo;0&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Loop at line 7 unrolled with remainder by 2</td>
</tr>
<tr>
<td></td>
<td><code>&lt;sample2.c;1:-1;hlo;_foo;0&gt;</code></td>
</tr>
<tr>
<td></td>
<td>High Level Optimizer Report (_foo)</td>
</tr>
<tr>
<td></td>
<td>QLOOPS 2/2 ENODE LOOPS 2 unknown 0 multi_exit_do 0 do 2 linear_do 2</td>
</tr>
<tr>
<td></td>
<td>LINEAR HLO EXPRESSIONS: 22 / 21</td>
</tr>
<tr>
<td></td>
<td><code>&lt;sample2.c;7:7;hlo_scalar_replacement;in _foo;0&gt;</code></td>
</tr>
<tr>
<td></td>
<td>#of Array Refs Scalar Replaced in _foo at line 7=1</td>
</tr>
<tr>
<td></td>
<td><code>&lt;sample2.c;5:5;hlo_linear_trans;_foo;0&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Loop Interchange not done due to: Imperfect Loop Nest (Either at Source or due to other Compiler Transformations)</td>
</tr>
<tr>
<td></td>
<td>Advice: Loop Interchange, if possible, might help Loopnest at lines: 5 7</td>
</tr>
<tr>
<td></td>
<td>: Suggested Permutation: (1 2 ) --&gt;</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 3: This example illustrates the condition where data dependence prohibits loop interchange.

### Example 3

```c
int foo (int **A, int **B, int **C, int N)
{
    int i, j;
    for (j=0; j<N; j++) {
        for (i=0; i<N; i++) {
            A[i][j] = C[i][j] * 2;
            B[i][j] += A[i][j] * C[i][j];
        }
    }
    return 1;
}
```

The code sample listed above will result in a report output similar to the following.
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example 3 Report Output</th>
</tr>
</thead>
</table>
| Linux and Mac OS X | <sample3.c; -1:-1; hlo; foo; 0>  
High Level Optimizer Report (foo)  
Block, Unroll, Jam Report:  
(loop line numbers, unroll factors and type of transformation)  
<samp...at line 6 unrolled with remainder by 2  
<sample3.c; -1:-1; hlo; _foo; 0>  
High Level Optimizer Report (_foo)  
QLOOPS 2/2 ENODE LOOPS 2 unknown 0 multi_exit_do 0 do 2 linear_do 2  
LINEAR HLO EXPRESSIONS: 36 / 36  
------------------------------------------------------------------------------  
<samp...not done due to: Data Dependencies  
    Dependencies found between following statements:  
    [From_Line# -> (Dependency Type)  
    To_Line#]  
    [7 ->(Anti) 8] [7 ->(Flow) 8] [7 ->(Output) 8]  
    [7 ->(Flow) 7] [7 ->(Anti) 7] [7 ->(Output) 7]  
Advice: Loop Interchange, if possible, might help Loopnest at lines: 5 6  
    : Suggested Permutation: (1 2 ) -->( 2 1 ) |
| Windows | |

Example 4: This example illustrates the condition where the loop order was determined to be proper, but loop interchange might offer only marginal relative improvement. To compile this
code add the `-restrict` (Linux and Mac OS X) or `/Qrestrict` (Windows) option to the other options when generating the report.

**Example 4**

```c
int foo (int ** restrict A, int ** restrict B, int N)
{
    int i, j, value;
    for (j=0; j<N; j++) {
        for (i=0; i<N; i++) {
            A[j][i] += B[i][j];
        }
    }
    value = A[1][1];
    return value;
}
```

The code sample listed above will result in a report output similar to the following.
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example 4 Report Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>&lt;sample4.c;1:-1;hlo;foo;0&gt; High Level Optimizer Report (foo)</td>
</tr>
<tr>
<td></td>
<td>Block, Unroll, Jam Report: (loop line numbers, unroll factors and type of transformation)</td>
</tr>
<tr>
<td>Windows</td>
<td>&lt;sample4.c;6:6;hlo_unroll;foo;0&gt; Loop at line 6 unrolled with remainder by 2</td>
</tr>
<tr>
<td></td>
<td>&lt;sample4.c;-1:-1;hlo;_foo;0&gt; High Level Optimizer Report (_foo)</td>
</tr>
<tr>
<td></td>
<td>QLOOPS 2/2 ENODE LOOPS 2 unknown 0 multi_exit_do 0 do 2 linear_do 2</td>
</tr>
<tr>
<td></td>
<td>LINEAR HLO EXPRESSIONS: 18 / 18</td>
</tr>
</tbody>
</table>

Example 5: This example illustrates the conditions where the loop nesting was imperfect and the loop order was good, but loop interchange would offer marginal relative improvements. To compile this code add the `-restrict` (Linux and Mac OS X) or `/Qrestrict` (Windows) option to the other options when generating the report.

**Example**

```c
int foo (int ** restrict A, int ** restrict B, int ** restrict C, int N)
{
    int i, j, sum;
    for (j=0; j<N; j++) {
        sum += A[1][1];
        for (i=0; i<N; i++) {
            sum = B[j][i] + C[i][j];
        }
    }
    return sum;
}
```
The code sample listed above will result in a report output similar to the following.
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example 5 Report Output</th>
</tr>
</thead>
</table>
| Linux and Mac OS X       | `<sample5.c;-1:-1;hlo;foo;0>`  
High Level Optimizer Report (foo)  
`<sample5.c;-1:-1;hlo;_foo;0>`  
High Level Optimizer Report (_foo)  
QLOOPS 2/2  
ENODE LOOPS 2 unknown 0  
multi_exit_do 0 do 2 linear_do 2  
LINEAR HLO EXPRESSIONS: 16 / 19  
|  
| Windows                  | `<sample5.c;5:5;hlo_linear_trans;_foo;0>`  
Loop Interchange not done due to: Imperfect  
Loop Nest (Either at Source or due to other Compiler Transformations)  
Advice: Loop Interchange, if possible, might help Loopnest at lines: 5 7  
: Suggested Permutation: (1 2 ) --> ( 2 1 )  
|

Example 6: This example illustrates the condition where perfect and imperfect loop nesting exists; however, the correctly nested loop contains data dependency.

```c
int foo (int ***A, int ***B, int **C, int N)
{
    int q, i, j, k;
    q = 0;
    while ( A[q][0][0] != 0) {
        for (j=0; j<N; j++) {
            A[j][0][0] = j + B[j][0][0];
        }
        for (i=0; i<N; i++) {
            ...
        }
    }
}
```
The code sample listed above will result in a report output similar to the following.
### Operating System | Example Report Output
---|---
Linux and Mac OS X | `<sample6.c;-1:-1;hlo;foo;0>`
High Level Optimizer Report (foo)  
Block, Unroll, Jam Report:  
(loop line numbers, unroll factors and type of transformation)  
`<sample6.c;9:9;hlo_unroll;foo;0>`  
Loop at line 9 unrolled with remainder by 2

[root@infodev-test hlo_samples_cpp]#

Windows | `<sample6.c;-1:-1;hlo;_foo;0>`
High Level Optimizer Report (_foo)  
QLOOPS 2/4  ENODE LOOPS 2 unknown 0  
multi_exit_do 0 do 2 linear_do 2  
LINEAR HLO EXPRESSIONS: 34 / 34

`<sample6.c;8:8;hlo_linear_trans;_foo;0>`
Loop Interchange not done due to: Data Dependencies  
  Dependencies found between following statements:
  
<table>
<thead>
<tr>
<th>From Line#</th>
<th>(Dependency Type)</th>
<th>To Line#</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10 --&gt; (Flow) 10]</td>
<td>[10 --&gt; (Anti) 10]</td>
<td>[10 --&gt; (Output) 10]</td>
</tr>
</tbody>
</table>

Advice: Loop Interchange, if possible, might help Loopnest at lines: 8 9  
  : Suggested Permutation: (1 2 ) --> ( 2 1 )

### Changing Code Based on the Report Results
While the HLO report tells you what loop transformations the compiler performed and provides some advice, the omission of a given loop transformation might imply that there are transformations the compiler might attempt. The following list suggests some transformations you might want to apply. (Manual optimization techniques, like manual cache blocking, should be avoided or used only as a last resort.)

- Loop Interchanging - Swap the execution order of two nested loops to gain a cache locality or unit-stride access performance advantage.
- Distributing - Distribute or split up one large loop into two smaller loops. This strategy might provide an advantage when too many registers are being consumed in a large loop.
- Fusing - Fuse two smaller loops with the same trip count together to improve data locality.
- Loop Blocking - Use cache blocking to arrange a loop so it will perform as many computations as possible on data already residing in cache. (The next block of data is not read into cache until computations using the first block are finished.)
- Unrolling - Unrolling is a way of partially disassembling a loop structure so that fewer numbers of iterations of the loop are required; however, each resulting loop iteration is larger. Unrolling can be used to hide instruction and data latencies, to take advantage of floating point loadpair instructions, and to increase the ratio of real work done per memory operation.
- Prefetching - Request the compiler to bring data in from relatively slow memory to a faster cache several loop iterations ahead of when the data is actually needed.
- Load Pairing - Use an instruction to bring two floating point data elements in from memory in a single step.

**High Performance Optimizer (HPO) Report**

The following command examples illustrate the general command needed to create HPO report with combined options.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux* and Mac OS* X</td>
<td>icpc -opt-report:0 -opt-report-phase:hpo sample.cpp</td>
</tr>
<tr>
<td>Windows*</td>
<td>icl /Qopt-report:0 /Qopt-report-phase:hpo sample.cpp</td>
</tr>
</tbody>
</table>

Use `-opt-report-help` (Linux and Mac OS X) or `/Qopt-report-help` (Windows) to list the names of HPO report categories.

You must specify different compiler options to use the specific HPO report categories.
For OpenMP*, add -openmp (Linux and Mac OS X) or /Qopenmp (Windows) to the command line.

For parallelism, add -parallel (Linux and Mac OS X) or /Qparallel (Windows) to the command line.

For vectorization, add -x (Linux and Mac OS X) or /Qx (Windows) to the command line; valid processor values are SSE4.1, SSSE3, SSE3, and SSE2.

Parallelism Report

The -par-report (Linux* and Mac OS* X) or /Qpar-report (Windows*) option controls the diagnostic levels 0, 1, 2, or 3 of the auto-parallelizer. Specify a value of 3 to generate the maximum diagnostic details.

Run the diagnostics report by entering commands similar to the following:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -c -parallel -par-report 3 sample.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /c /Qparallel /Qpar-report:3 sample.cpp</td>
</tr>
</tbody>
</table>

where -c (Linux and Mac OS X) or /c (Windows) instructs the compiler to compile the example without generating an executable.

NOTE. Linux and Mac OS X: The space between the option and the phase is optional. Windows: The colon between the option and phase is optional.
For example, assume you want a full diagnostic report on the following example code:

```c
Example

void no_par(void)
{
    int i;
    int a[1000];
    for (i=1; i<1000; i++) {
        a[i] = (i * 2) % i + 1 + sqrt(i);
        a[i] = a[i-1] + i;
    }
}
```

The following example output illustrates the diagnostic report generated by the compiler for the example code shown above. In most cases, the comment listed next to the line is self-explanatory.

```plaintext
Example Report Output

procedure: no_par
sample.c(13):(3) remark #15048: DISTRIBUTED LOOP WAS AUTO-PARALLELIZED
sample.c(13):(3) remark #15050: loop was not parallelized: existence of parallel dependence
sample.c(19):(5) remark #15051: parallel dependence: proven FLOW dependence between a line 19, and a line 19
```

**Responding to the results**

The `-par-threshold{n}` (Linux* and Mac OS* X) or `/Qpar-threshold[:n]` (Windows*) option sets a threshold for auto-parallelization of loops based on the probability of profitable execution of the loop in parallel. The value of \( n \) can be from 0 to 100. You can use `-par-threshold0` (Linux and Mac OS X) or `/Qpar-threshold:0` (Windows) to auto-parallelize loops regardless of computational work.

Use `-ipo[value]` (Linux and Mac OS X) or `/Qipo` (Windows) to eliminate assumed side-effects done to function calls.
Software Pipelining (SWP) Report (Linux* and Windows*)

The SWP report can provide details information about loops currently taking advantage of software pipelining available on IA-64 architecture based systems. The report can suggest reasons why the loops are not being pipelined.

The following command syntax examples demonstrates how to generate a SWP report for the Itanium® Compiler Code Generator (ECG) Software Pipeliner (SWP).

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*</td>
<td>icpc -c -opt-report</td>
</tr>
<tr>
<td></td>
<td>-opt-report-phase=ecg_swip sample.cpp</td>
</tr>
<tr>
<td>Windows*</td>
<td>icl /c /Qopt-report</td>
</tr>
<tr>
<td></td>
<td>/Qopt-report-phase:ecg_swip sample.cpp</td>
</tr>
</tbody>
</table>

where -c (Linux) or /c (Windows) tells the compiler to stop at generating the object code (no linking occurs), -opt-report (Linux) or /Qopt-report (Windows) invokes the report generator, and -opt-report-phase=ecg_swip (Linux) or /Qopt-report-phase:ecg_swip (Windows) indicates the phase (ecg) for which to generate the report.

You can use -opt-report-file (Linux) or /Qopt-report-file (Windows) to specify an output file to capture the report results. Specifying a file to capture the results can help to reduce the time you spend analyzing the results and can provide a baseline for future testing.

Typically, loops that software pipeline will have a line that indicates the compiler has scheduled the loop for SWP in the report. If the -O3 (Linux) or /O3 (Windows) option is specified, the SWP report merges the loop transformation summary performed by the loop optimizer.

Some loops will not software pipeline (SWP) and others will not vectorize if function calls are embedded inside your loops. One way to get these loops to SWP or to vectorize is to inline the functions using IPO.
You can compile this example code to generate a sample SWP report, but you must use compile the example using a combination of `-c -restrict` (Linux) and `/c /Qrestrict` (Windows). The sample reports is also shown below.

```
#define NUM 1024

void multiply_d(double a[][NUM], double b[][NUM], double c[restrict][NUM]){
    int i,j,k;
    double temp;
    for(i=0;i<NUM;i++) {
        for(j=0;j<NUM;j++) {
            for(k=0;k<NUM;k++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
}
```

The following sample report shows the report phase that results from compiling the example code shown above (when using the `ecg_swp` phase).

```
Sample SWP Report

Swp report for loop at line 8 in _Z10multiply_dPA1024_dS0_S0_ in file SWP report.cpp
Resource II = 2
Recurrence II = 2
Minimum II = 2
Scheduled II = 2
Estimated GCS II = 7
```
**Sample SWP Report**

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percent of Resource II needed by arithmetic ops = 100%</td>
<td></td>
</tr>
<tr>
<td>Percent of Resource II needed by memory ops = 50%</td>
<td></td>
</tr>
<tr>
<td>Percent of Resource II needed by floating point ops = 50%</td>
<td></td>
</tr>
<tr>
<td>Number of stages in the software pipeline = 6</td>
<td></td>
</tr>
</tbody>
</table>

**Reading the Reports**

To understand the SWP report results, you must know something about the terminology used and the related concepts. The following table describes some of the terminology used in the SWP report.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>II</td>
<td>Initiation Interval (II). The number of cycles between the start of one iteration and the next in the SWP. The presence of the term II in any SWP report indicates that SWP succeeded for the loop in question. II can be used in a quick calculation to determine how many cycles your loop will take, if you also know the number of iterations. Total cycle time of the loop is approximately $N \times \text{Scheduled II} + \text{Number of stages}$ (Where N is the number of iterations of the loop). This is an approximation because it does not take into account the ramp-up and ramp-down of the prolog and epilog of the SWP, and only considers the kernel of the SWP loop. As you modify your code, it is generally better to see scheduled II go down, though it is really $N \times \text{Scheduled II} + \text{Number of stages}$ in the software pipeline that is ultimately the figure of merit.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Resource II</td>
<td>Resource II implies what the Initiation Interval should be when considering the number of functional units available.</td>
</tr>
<tr>
<td>Recurrence II</td>
<td>Recurrence II indicates what the Initiation Interval should be when there is a recurrence relationship in the loop. A recurrence relationship is a particular kind of a data dependency called a flow dependency like $a[i] = a[i-1]$ where $a[i]$ cannot be computed until $a[i-1]$ is known. If Recurrence II is non-zero and there is no flow dependency in the code, then this indicates either Non-Unit Stride Access or memory aliasing. See <a href="#">Helping the Compiler</a> for more information.</td>
</tr>
<tr>
<td>Minimum II</td>
<td>Minimum II is the theoretical minimum Initiation Interval that could be achieved.</td>
</tr>
<tr>
<td>Scheduled II</td>
<td>Scheduled II is what the compiler actually scheduled for the SWP.</td>
</tr>
<tr>
<td>number of stages</td>
<td>Indicates the number of stages. For example, in the report results below, the line &quot;Number of stages in the software pipeline = 3&quot; indicates there were three stages of work, which will show, in assembly, to be a load, an FMA instruction and a store.</td>
</tr>
<tr>
<td>loop-carried memory dependence edges</td>
<td>The loop-carried memory dependence edges means the compiler avoided WAR (Write After Read) dependency.</td>
</tr>
<tr>
<td></td>
<td>Loop-carried memory dependence edges can indicate problems with memory aliasing. See <a href="#">Helping the Compiler</a>.</td>
</tr>
</tbody>
</table>
Using the Report to Resolve Issues

One fast way to determine if specific loops have been software pipelined is to look for "r;Number of stages in the software pipeline" in the report; the phrase indicates that software pipelining for the associated loop was successfully applied.

Analyze the loops that did not SWP in order to determine how to enable SWP. If the compiler reports the "Loop was not SWP because...", see the following table for suggestions about how to correct possible problems:

<table>
<thead>
<tr>
<th>Message in Report</th>
<th>Suggested Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>acyclic global scheduler can achieve a better schedule: =&gt; loop not pipelined</td>
<td>Indicates that the most likely cause is memory aliasing issues. For memory aliasing problems see memory aliasing (restrict, #pragma ivdep). Might indicate the application is accessing memory in a non-Unit Stride fashion. Non-Unit Stride issues may be indicated by an artificially high recurrence II; If you know there is no recurrence relationship ( a[i] = a[i-1] + b[i] ) in the loop, then a high recurrence II (greater than 0) is a sign that you are accessing memory non-Unit Stride. Rearranging code, perhaps a loop interchange, might help mitigate this problem.</td>
</tr>
<tr>
<td>Loop body has a function call</td>
<td>Indicates inlining the function might help solve the problem.</td>
</tr>
<tr>
<td>Not enough static registers</td>
<td>Indicates you should distribute the loop by separating it into two or more loops. On IA-64 architecture based systems you may use #pragma distribute point.</td>
</tr>
<tr>
<td>Not enough rotating registers</td>
<td>Indicates the loop carried values use the rotating registers. Distribute the loop. On IA-64 architecture based systems you may use #pragma distribute point.</td>
</tr>
</tbody>
</table>
#### Message in Report | Suggested Action
---|---
Loop too large | Indicates you should distribute the loop. On IA-64 architecture based systems you may use the `#pragma distribute point`.
Loop has a constant trip count < 4 | Indicates unrolling was insufficient. Attempt to fully unroll the loop. However, with small loops fully unrolling the loop is not likely to affect performance significantly.
Too much flow control | Indicates complex loop structure. Attempt to simplify the loop.

Index variable type used can greatly impact performance. In some cases, using loop index variables of type short or unsigned int can prevent software pipelining. If the report indicates performance problems in loops where the index variable is not int and if there are no other obvious causes, try changing the loop index variable to type int.

**Vectorization Report**

The vectorization report can provide information about loops that could take advantage of Intel® Streaming SIMD Extensions (Intel® SSE3, SSE2, and SSE) vectorization, and it is available on systems based on IA-32 and Intel® 64 architectures.

See Using Parallelism for information on other vectorization options.

The `-vec-report` (Linux* and Mac OS* X) or `/Qvec-report` (Windows*) option directs the compiler to generate the vectorization reports with different levels of information. Specify a value of 3 to generate the maximum diagnostic details.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td><code>icpc -c -xSSSE3 -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>icl /c /QxSSSE3 /Qvec-report:3 sample.cpp</code></td>
</tr>
</tbody>
</table>

where `-c` (Linux and Mac OS X) or `/c` (Windows) instructs the compiler to compile the example without generating an executable.
NOTE. Linux and Mac OS X: The space between the option and the phase is optional. Windows: The colon between the option and phase is optional.

The following example results illustrate the type of information generated by the vectorization report:

<table>
<thead>
<tr>
<th>Example results</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample.cpp(10) : (col. 2) remark: loop was not vectorized: not inner loop.</td>
</tr>
<tr>
<td>sample.cpp(11) : (col. 6) remark: loop was not vectorized: not inner loop.</td>
</tr>
<tr>
<td>sample.cpp(12) : (col. 5) remark: vector dependence: assumed FLOW dependence between c line 13 and b line 13.</td>
</tr>
<tr>
<td>sample.cpp(12) : (col. 5) remark: vector dependence: assumed FLOW dependence between c line 13 and a line 13.</td>
</tr>
<tr>
<td>sample.cpp(12) : (col. 5) remark: vector dependence: assumed FLOW dependence between c line 13 and c line 13.</td>
</tr>
<tr>
<td>sample.cpp(12) : (col. 5) remark: loop was not vectorized: existence of vector dependence.</td>
</tr>
</tbody>
</table>

If the compiler reports "r;Loop was not vectorized" because of the existence of vector dependence, then you should analyze the loop for vector dependence. If you determine there is no legitimate vector dependence, then the message indicates that the compiler was assuming the pointers or arrays in the loop were dependent, which implies the pointers or arrays were aliased. Use memory disambiguation techniques to resolve these cases.

There are three major types of vector dependence: FLOW, ANTI, and OUTPUT.

See Loop Independence to determine if there is a valid vector dependence. Many times the compiler report will assert a vector dependence where none exists - this is because the compiler assumes memory aliasing. The action to take in these cases is to check code for dependencies; if there are none, inform the compiler using methods described in memory aliasing including restrict or ivdep.

There are a number of situations where the vectorization report may indicate vector dependencies. The following situations will sometimes be reported as vector dependencies, non-unit stride, low trip count, and complex subscript expression.

**Non-Unit Stride**

The report might indicate that a loop could not be vectorized when the memory is accessed in a non-Unit Stride manner. This means that nonconsecutive memory locations are being accessed in the loop. In such cases, see if loop interchange can help or if it is practical. If not then you can force vectorization sometimes through vector always pragma; however, you should verify improvement.
See *Understanding Runtime Performance* for more information about non-unit stride conditions.

**Usage with Other Options**

The vectorization reports are generated during the final compilation phase, which is when the executable is generated; therefore, there are certain option combinations you cannot use if you are attempting to generate a report. If you use the following option combinations, the compiler issues a warning and does not generate a report:

- `-c` or `-ipo` or `-x` with `-vec-report` (Linux* and Mac OS* X) and `/c` or `/Qipo` or `/Qx` with `/Qvec-report` (Windows*)
- `-c` or `-ax` with `-vec-report` (Linux and Mac OS X) and `/c` or `/Qax` with `/Qvec-report` (Windows)

The following example commands can generate vectorization reports:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Command Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>The following commands generate a vectorization report:</td>
</tr>
<tr>
<td></td>
<td><code>icpc -xSSE3 -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td></td>
<td><code>icpc -xSSE3 -ipo -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td></td>
<td><code>icpc -c -xSSE3 -ipo -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td></td>
<td>The following commands will not generate a vectorization report:</td>
</tr>
<tr>
<td></td>
<td><code>icpc -c -xSSE3 -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td></td>
<td><code>icpc -xSSE3 -ipo -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td></td>
<td><code>icpc -c -xSSE3 -ipo -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td>Windows</td>
<td>The following commands generate a vectorization report:</td>
</tr>
<tr>
<td></td>
<td><code>icpc -xSSE3 -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td></td>
<td><code>icpc -xSSE3 -ipo -vec-report3 sample.cpp</code></td>
</tr>
<tr>
<td></td>
<td><code>icpc -c -xSSE3 -ipo -vec-report3 sample.cpp</code></td>
</tr>
</tbody>
</table>
### Operating System

<table>
<thead>
<tr>
<th>Command Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>icl /QxSSSE3 /Qvec-report:3 sample.cpp</td>
</tr>
<tr>
<td>icl /QxSSSE3 /Qipo /Qvec-report:3 sample.cpp</td>
</tr>
<tr>
<td>icl /c /QxSSSE3 /Qipo /Qvec-report:3 sample.cpp</td>
</tr>
</tbody>
</table>

The following commands will not generate a vectorization report:

<table>
<thead>
<tr>
<th>Command Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>icl /c /QxSSSE3 /Qvec-report:3 sample.cpp</td>
</tr>
<tr>
<td>icl /QxSSSE3 /Qipo /Qvec-report:3 sample.cpp</td>
</tr>
<tr>
<td>icl /c /QxSSSE3 /Qipo /Qvec-report:3 sample.cpp</td>
</tr>
</tbody>
</table>

### Responding to the Results

You might consider changing existing code to allow vectorization under the following conditions:

- The vectorization report indicates that the program "contains unvectorizable statement at line XXX"; eliminate conditions such as, a printf() or user defined foo() the loop.

- The vectorization report states there is a "vector dependence: proven FLOW dependence between 'r;variable' line XXX, and 'r;variable' line XXX" or "loop was not vectorized: existence of vector dependence." Generally, these conditions indicate true loop dependencies are stopping vectorization. In such cases, consider changing the loop algorithm.
For example, consider the two equivalent algorithms producing identical output below. "Foo" will not vectorize due to the FLOW dependence but "bar" does vectorize.

**Example**

```c
void foo(double *y)
{
    for(int i=1;i<10;i++) {
        // a loop that puts sequential numbers into array y
        y[i] = y[i-1]+1;
    }
}

void bar(double *y)
{
    for(int i=1;i<10;i++) {
        // a loop that puts sequential numbers into array y
        y[i] = y[0]+i;
    }
}
```
Unsupported loop structures may prevent vectorization. An example of an unsupported loop structure is a loop index variable that requires complex computation. Change the structure to remove function calls to loop limits and other excessive computation for loop limits.

**Example**

```c
int function(int n)
{
    return (n*n-1);
}

void unsupported_loop_structure(double *y, int n)
{
    for (int i=0; i<function(n); i++) {
        *y = *y * 2.0;
    }
}
```

Non-unit stride access might cause the report to state that "vectorization possible but seems inefficient". Try to restructure the loop to access the data in a unit-stride manner (for example, apply loop interchange), or try `#pragma vector always`.

Using mixed data types in the body of a loop might prevent vectorization. In the case of mixed data types, the vectorization report might state something similar to "loop was not vectorized: condition too complex".
The following example code demonstrates a loop that cannot vectorize due to mixed data types within the loop. For example, `withinborder` is an integer while all other data types in loop are not. Simply changing the `withinborder` data type will allow this loop to vectorize.

**Example**

```c
int howmany_close(double *x, double *y)
{
    int withinborder=0;
    double dist;
    for(int i=0;i<100;i++) {
        dist=sqrtf(x[i]*x[i] + y[i]*y[i]);
        if (dist<5) withinborder++;
    }
    return 0;
}
```

**OpenMP* Report**

The `-openmp-report` (Linux* and Mac OS* X) or `/Qopenmp-report` (Windows*) option controls the diagnostic levels for OpenMP* reporting. The OpenMP* report information is not generated unless you specify the option with a value of either 1 or 2. Specifying 2 provides the most useful information.

You must specify the `-openmp` (Linux and Mac OS X) or `/Qopenmp` (Windows) along with this option.

<table>
<thead>
<tr>
<th>Linux OS and Mac OS X</th>
<th>Windows OS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-openmp-report 2</code></td>
<td><code>/Qopenmp-report:2</code></td>
<td>Report results are same as when specifying 1 except the results also include diagnostics indicating constructs, like directives, that were handled successfully. This is the recommend level.</td>
</tr>
</tbody>
</table>
Reports on loops, regions, and sections that were parallelized successfully. This is the default level.

No diagnostics report generated.

The following example commands demonstrate how to run the report using the combined commands.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -openmp -openmp-report 2 sample1.cpp sample2.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Qopenmp /Qopenmp-report:2 sample1.cpp sample2.cpp</td>
</tr>
</tbody>
</table>

**NOTE.** Linux and Mac OS X: The space between the option and the level is optional. Windows: The colon between the option and level is optional.

The following example results illustrate the typical format of the generated information:

**Example results**

openmp_sample.c(96): (col. 5) remark: OpenMP multithreaded code generation for SINGLE was successful.

openmp_sample.c(106): (col. 5) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.

openmp_sample.c(113): (col. 5) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.

openmp_sample.c(93): (col. 3) remark: OpenMP DEFINED REGION WAS PARALLELIZED.

See also:

- [OpenMP* Options Quick Reference](#) for information about these options.
- [OpenMP* Support Overview](#) for information on using OpenMP* in Intel® compilers.
Automatic Optimizations Overview

Intel® compilers allow you to compile applications for processors based on IA-32 architectures (32-bit applications), Intel® 64 architectures (32-bit and 64-bit applications), or IA-64 architectures (64-bit applications).

By default the compiler chooses a set of optimizations that balances compile-time and run-time performance for your application. Also, you can manually select optimizations based on the specific needs of the application.

The following table summarizes the common optimization options you can use for quick, effective results.

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O3</td>
<td>/O3</td>
<td>Enables aggressive optimization for code speed. Recommended for code with loops that perform substantial calculations or process large data sets.</td>
</tr>
<tr>
<td>-O2 (or -O)</td>
<td>/O2</td>
<td>Affects code speed. This is the default option; the compiler uses this optimization level if you do not specify anything.</td>
</tr>
<tr>
<td>-O1</td>
<td>/O1</td>
<td>Affects code size and locality. Disables specific optimizations.</td>
</tr>
<tr>
<td>-Od</td>
<td>/Od</td>
<td>Disables optimization. Use this for rapid compilation while debugging an application.</td>
</tr>
</tbody>
</table>
The variety of automatic optimizations enable you to quickly enhance your application performance. In addition to automatic optimizations, the compiler invokes other optimization enabled with source code directives, pragmas, intrinsics, optimized library routines, and performance-enhancing utilities.

The remaining topics in this section provide more details on the automatic optimizations supported in the Intel compilers.

**See Also**
- Using Compiler Optimizations
- Enabling Automatic Optimizations
- Restricting Optimizations

### Enabling Automatic Optimizations

This topic lists the most common code optimization options, describes the characteristics shared by IA-32, Intel® 64, and IA-64 architectures, and describes the general behavior for each architecture.

The architectural differences and compiler options enabled or disabled by these options are also listed in more specific detail in the associated Compiler Options topics; therefore, each option discussion listed below includes a link to the appropriate reference topic.

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
</table>
| -O1 | /O1 | Optimizes to favor smaller code size and code locality. In most cases, -O2 (Linux* OS and Mac OS* X) or /O2 (Windows* OS) is recommended over this option.

This optimization disables some optimizations that normally increase code size. This level might improve performance for applications with very large code size, many branches, and execution time not dominated by code within loops. In general, this optimization level does the following:
<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/O2</td>
<td>-O2 or -0</td>
</tr>
</tbody>
</table>

- Enables global optimization.
- Disables intrinsic recognition and inlining of intrinsics.

IA-64 architecture:
- The option disables software pipelining, loop unrolling, and global code scheduling.

Optimizes for code speed. Since this is the default optimization, if you do not specify an optimization level the compiler will use this optimization level automatically. This is the generally recommended optimization level; however, specifying other compiler options can affect the optimization normally gained using this level.

In general, the resulting code size will be larger than the code size generated using -O1 (Linux and Mac OS X) or /O1 (Windows).

This option enables the following capabilities for performance gain: inlining intrinsic functions, constant propagation, copy propagation, dead-code elimination, global register allocation, global instruction scheduling and control speculation, loop unrolling,
<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>optimized code selection, partial redundancy elimination, strength reduction/induction variable simplification, variable renaming, exception handling optimizations, tail recursions, peephole optimizations, structure assignment lowering optimizations, and dead store elimination. For IA-32 and Intel 64 architectures:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enables certain optimizations for speed, such as vectorization.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IA-64 architecture:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enables optimizations for speed, including global code scheduling, software pipelining, predication, speculation, and data prefetch.</td>
</tr>
<tr>
<td>-O3</td>
<td>/O3</td>
<td>Enables -O2 (Linux and Mac OS X) or /O2 (Windows) optimizations, as well as more aggressive optimizations, including prefetching, scalar replacement, cache blocking, and loop and memory access transformations. As compared to -O2 (Linux) or /O2 (Windows), the optimizations enabled by this option often result in faster program execution, but can</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>-fast</td>
<td>/fast</td>
<td>slow down code execution in some cases. Using this option may result in longer compilation times. This option is recommended for loop-intensive applications that perform substantial floating-point calculations or process large data sets. Provides a single, simple optimization that enables a collection of optimizations that favor run-time performance. This is a good, general option for increasing performance in many programs. For IA-32 and Intel 64 architectures, the -xSSSE3 (Linux and Mac OS X) or /QxSSSE3 (Windows) option that is set by this option cannot be overridden by other command line options. If you specify this option along with a different processor-specific option, such as -xSSE2 (Linux) or /QxSSE2 (Windows), the compiler will issue a warning stating the -xSSSE3 or /QxSSSE3 option cannot be overridden; the best strategy for dealing with this restriction is to explicitly specify the options you want to set from the command line.</td>
</tr>
</tbody>
</table>
Programs compiled with the \texttt{-xSSSE3} (Linux and Mac OS X) or \texttt{/QxSSSE3} (Windows) option will detect non-compatible processors and generate an error message during execution.

While this option enables other options quickly, the specific options enabled by this option might change from one compiler release to the next. Be aware of this possible behavior change in the case where you use makefiles.

The following syntax examples demonstrate using the default option to compile an application:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>\texttt{icpc -02 prog.cpp}</td>
</tr>
<tr>
<td>Windows</td>
<td>\texttt{icl /02 prog.cpp}</td>
</tr>
</tbody>
</table>

Refer to \textit{Quick Reference Lists} for a complete listing of the quick reference topics.

### Targeting IA-32 and Intel(R) 64 Architecture Processors Automatically

The \texttt{-x} (Linux* and Mac OS* X) or \texttt{/Qx} (Windows*) option can automatically optimize your application for specific Intel® processors based on IA-32 and Intel® 64 architectures.

The automatic optimizations allow you to take advantage of the architectural differences, new instruction sets, or advances in processor design; however, the resulting, optimized code might contain unconditional use of features that are not supported on other, earlier processors. Therefore, using these options effectively sets a minimum hardware requirement for your application.
The optimizations can include generating Intel® Streaming SIMD Extensions 4 (SSE4), Supplemental Streaming SIMD Extensions 3 (SSSE3), Streaming SIMD Extensions 3 (SSE3), Streaming SIMD Extensions 2 (SSE2), or Streaming SIMD Extensions (SSE) instructions.

If you intend to run your programs on multiple processors based on IA-32 or Intel® 64 architectures, do not use this option; instead, consider using the –ax (Linux and Mac OS X) or /Qax (Windows) option to achieve both processor-specific performance gains and portability among different processors.

<table>
<thead>
<tr>
<th>Linux OS and Mac OS X</th>
<th>Windows OS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-xHost</td>
<td>/QxHost</td>
<td>Can generate instructions for the highest instruction set and processor available on the compilation host.</td>
</tr>
<tr>
<td>-xAVX</td>
<td>/QxAVX</td>
<td>Optimizes for Intel processors that support Intel® Advanced Vector Extensions (Intel® AVX).</td>
</tr>
<tr>
<td>-xSSE4.1</td>
<td>/QxSSE4.1</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture. This replaces value S, which is deprecated.</td>
</tr>
<tr>
<td>-xSSE4.2</td>
<td>/QxSSE4.2</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions.</td>
</tr>
<tr>
<td>Linux OS and Mac OS X</td>
<td>Windows OS</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-xSSSE3</td>
<td>/QxSSSE3</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™ processor family. This replaces value T, which is deprecated.</td>
</tr>
<tr>
<td>-xSSE3_ATOM</td>
<td>/QxSSE3_ATOM</td>
<td>Optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology. Can generate MOVBE instructions, depending on the setting of option -minstruction (Linux and Mac OS) or /Qin-struction (Windows). Mac OS X: Supported on IA-32 architectures.</td>
</tr>
<tr>
<td>-xSSE3</td>
<td>/QxSSE3</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. This replaces value P, which is deprecated. Mac OS X: Supported on IA-32 architectures.</td>
</tr>
<tr>
<td>-xSSE2</td>
<td>/QxSSE2</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® processors.</td>
</tr>
<tr>
<td>Linux OS and Mac OS X</td>
<td>Windows OS</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 processors, Intel(^\text{\textregistered}) Pentium(^\text{\textregistered}) M processors, and Intel(^\text{\textregistered}) Xeon(^\text{\textregistered}) processors with Intel(^\text{\textregistered}) SSE2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mac OS X: Not supported.</td>
</tr>
</tbody>
</table>

Certain keywords for compiler options -m and /arch produce binaries that should run on processors not made by Intel that implement the same capabilities as the corresponding Intel processors. For details, see Compiler Options.

To prevent illegal instruction and similar unexpected run-time errors during program execution, the compiler inserts code in the main routine of the program to check for proper processor usage. Using this option limits you to a minimum processor level. For example, if you target an application to run on Intel\(^\text{\textregistered}\) Xeon\(^\text{\textregistered}\) processors based on the Intel\(^\text{\textregistered}\) Core™ microarchitecture, it is unlikely the resulting application will operate correctly on earlier Intel processors.

If you target more than one processor value, the resulting code will be generated for the highest-performing processor specified if the compiler determines there is an advantage in doing so. The highest- to lowest-performing processor values are as follows:

1. SSE4.1
2. SSSE3
3. SSE3
4. SSE2

Executing programs compiled with processor values of SSE4.1, SSSE3, SSE3, or SSE2 on unsupported processors will display a run-time error. For example, if you specify the SSSE3 processor value to compile an application but execute the application on an Intel\(^\text{\textregistered}\) Pentium\(^\text{\textregistered}\) 4 processor, the application generates an error similar to the following:

**Run-time Error**

Fatal Error: This program was not built to run on the processor in your system.

The allowed processors are: Intel(R) Core(TM) Duo processors and compatible Intel processors with supplemental Streaming SIMD Extensions 3 (SSSE3) instruction support.

The following examples demonstrate compiling an application for Intel\(^\text{\textregistered}\) Core™2 Duo processor and compatible processors. The resulting binary might not execute correctly on earlier processors or on IA-32 architecture processors not made by Intel Corporation.
Targeting Multiple IA-32 and Intel(R) 64 Architecture Processors for Run-time Performance

The `-ax` (Linux* and Mac OS* X) or `/Qax` (Windows*) option instructs the compiler to determine if opportunities exist to generate multiple, specialized code paths to take advantage of performance gains and features available on newer Intel® processors based on IA-32 and Intel® 64 architectures. This option also instructs the compiler to generate a more generic (baseline) code path that should allow the same application to run on a larger number of processors; however, the baseline code path is usually slower than the specialized code.

The compiler inserts run-time checking code to help determine which version of the code to execute. The size of the compiled binary increases because it contains both a processor-specific version of some of the code and a generic baseline version of all code. Application performance is affected slightly due to the run-time checks needed to determine which code to use. The code path executed depends strictly on the processor detected at run time.

Processor support for the baseline code path is determined by the processor family or instruction set specified in the `-m` or `-x` (Linux and Mac OS X) or `/arch` or `/Qx` (Windows) option, which has default values for each architecture.

This allows you to impose a more strict processor or instruction set requirement for the baseline code path; however, such generic baseline code will not operate correctly on processors that are not compatible with the minimum processor or instruction set requirement. For the IA-32 architecture, you can specify a baseline code path that will work on all IA-32 compatible processors using the `-mia32` (Linux) or `/arch:IA32` (Windows) options. You should always specify the processor or instruction set requirements explicitly for the baseline code path, rather than depend on the defaults for the architecture.

Optimizations in the specialized code paths can include generating and using Intel® Streaming SIMD Extensions 4 (SSE4), Supplemental Streaming SIMD Extensions 3 (SSSE3), Streaming SIMD Extensions 3 (SSE3), or Streaming SIMD Extensions 2 (SSE2) instructions for supported Intel processors; however, such specialized code paths are executed only after checking verifies that the code is supported by the run-time host processor.

If not indicated otherwise, the following processor values are valid for IA-32 and Intel® 64 architectures.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -xSSSE3 sample.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /QxSSSE3 sample.cpp</td>
</tr>
<tr>
<td>Linux OS and Mac OS X</td>
<td>Windows OS</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
</tr>
<tr>
<td>-axSSE4.2</td>
<td>/QaxSSE4.2</td>
</tr>
<tr>
<td>-axSSE4.1</td>
<td>/QaxSSE4.1</td>
</tr>
<tr>
<td>-axSSSE3</td>
<td>/QaxSSSE3</td>
</tr>
</tbody>
</table>

Mac OS X: IA-32 and Intel® 64 architectures.
<table>
<thead>
<tr>
<th>Linux OS and Mac OS X</th>
<th>Windows OS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-axSSE3_ATOM</td>
<td>/QaxSSE3_ATOM</td>
<td>Optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology. Can generate MOVBE instructions, depending on the setting of option -minstruction (Linux and Mac OS) or /Qinstruction (Windows). Mac OS X: Supported on IA-32 architectures.</td>
</tr>
<tr>
<td>-axSSE3</td>
<td>/QaxSSE3</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. This replaces value P, which is deprecated. Mac OS X: IA-32 architecture.</td>
</tr>
<tr>
<td>-axSSE2</td>
<td>/QaxSSE2</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2. Linux and Windows: IA-32 architecture.</td>
</tr>
</tbody>
</table>

**NOTE.** You can specify `-diag-disable cpu-dispatch` (Linux and Mac OS X) or `/Qdiag-disable:cpu-dispatch` (Windows) to disable the display of remarks about multiple code paths for CPU dispatch.
If your application for IA-32 or Intel® 64 architectures does not need to run on multiple processors, consider using the \texttt{-x} (Linux and Mac OS X) or \texttt{/Qx} (Windows) option instead of this option.

The following compilation examples demonstrate how to generate an IA-32 architecture executable that includes an optimized version for Intel® Core™2 Duo processors, as long as there is a performance gain, an optimized version for Intel® Core™ Duo processors, as long as there is a performance gain, and a generic baseline version that runs on any IA-32 architecture processor.

\textbf{NOTE.} If you combine the arguments, you must add a comma (","), separator between the individual arguments.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icpc -axSSSE3,SSE3 -mia32 sample.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /QaxSSSE3,SSE3 /arch:IA32 sample.cpp</td>
</tr>
</tbody>
</table>

\section*{Targeting IA-32 and Intel 64 Architecture Processors Manually}

Using manual processor dispatch, your code can detect the IA-32 or Intel® 64 architecture processor at run time through the \texttt{cpu\_specific} and \texttt{cpu\_dispatch} keywords, enabling you to write one code path that runs only on the targeted processor, and other code path(s) that are taken on other processors. Manual processor dispatch will not recognize processors based on IA-64 architecture.

Use the \texttt{__declspec(cpu\_specific)} and \texttt{__declspec(cpu\_dispatch)} syntax in your code to create code specific to a targeted Intel® processor and allow the other code paths to execute correctly on other IA-32 or Intel 64 architecture processors.

Refer to the \textit{Programming with Mixed Languages} section in Building Applications for information on using these C++ keywords.

The general syntax for these keywords changes a function declaration by using the following arguments:

\begin{itemize}
  \item \texttt{cpu\_specific(cpuid)}
  \item \texttt{cpu\_dispatch(cpuid-list)}
\end{itemize}

The following table lists the values for \texttt{cpuid}:
The following table lists the syntax for `cpuid-list`:

<table>
<thead>
<tr>
<th>Syntax for cpuid-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpuid</td>
</tr>
<tr>
<td>cpuid-list, cpuid</td>
</tr>
</tbody>
</table>

The attributes are not case sensitive. The body of a function declared with `__declspec(cpu_dispatch)` must be empty, and is referred to as a stub (an empty-bodied function).

Manual processor dispatch can disable some types of inlining, almost always results in larger code and executable sizes, and can introduce additional performance overhead because of the additional function calls. Test your application on all of the targeted platforms before release. Before using manual dispatch, consider whether the benefits outweigh the additional effort and possible performance issues.

Use the following guidelines to implement processor dispatch support:
• A stub for `cpu_dispatch` must have a `cpuid` defined in `cpu_specific` elsewhere if the `cpu_dispatch` stub for a function `f` contains the `cpuid p`, then a `cpu_specific` definition of `f` with `cpuid p` must appear somewhere in the program; otherwise, an unresolved external error is reported.

• A `cpu_specific` function definition need not appear in the same translation unit as the corresponding `cpu_dispatch` stub, unless the `cpu_specific` function is declared `static`. The inline attribute is disabled for all `cpu_specific` and `cpu_dispatch` functions.

• Have a stub for `cpu_specific` function if a function `f` is defined as `__declspec(cpu_specific(p))`, then a `cpu_dispatch` stub must also appear for `f` within the program, and `p` must be in the `cpuid-list` of that stub; otherwise, that `cpu_specific` definition cannot be called nor generate an error condition. (This overrides command line settings when a `cpu_dispatch` stub is compiled, its body is replaced with code that determines the processor on which the program is running, then dispatches the best `cpu_specific` implementation available as defined by the `cpuid-list`.)

• A `cpu_specific` function optimizes to the specified Intel processor regardless of command-line option settings.
The following example demonstrates using manual dispatch with both `cpu_specific` and `cpu_dispatch`.

### Example

```c
#include <stdio.h>
#include <mmintrin.h>

/* Pentium processor function does not use intrinsics
to add two arrays. */
__declspec(cpu_specific(pentium))
void array_sum1(int *result, int *a, int *b, size_t len)
{
    for (; len > 0; len--)
        *result++ = *a++ + *b++;
}

/* Implementation for a Pentium processor with MMX technology uses
an MMX instruction intrinsic to add four elements simultaneously. */
__declspec(cpu_specific(pentium_MMX))
void array_sum2(int *result, int const *a, int *b, size_t len)
{
    __m64 *mmx_result = (__m64 *)result;
    __m64 const *mmx_a = (__m64 const *)a;
    __m64 const *mmx_b = (__m64 const *)b;
    for (; len > 3; len -= 4)
        *mmx_result++ = _mm_add_pi16(*mmx_a++, *mmx_b++);
    /* The following code, which takes care of excess elements, is not
     * needed if the array sizes passed are known to be multiples of four. */
    result = (unsigned short *)mmx_result;
    a = (unsigned short const *)mmx_a;
    b = (unsigned short const *)mmx_b;
```
Example

```c
for (; len > 0; len--)
    *result++ = *a++ + *b++;

__declspec(cpu_dispatch(pentium, pentium_MMX))
void array_sum3(int *result, int const *a, int *b, size_t len)
{
    /* Empty function body informs the compiler to generate the
     CPU-dispatch function listed in the cpu_dispatch clause. */
}
```

Targeting IA-64 Architecture Processors Automatically

The Intel compiler supports options that optimize application performance for Intel® Itanium® processors based on the IA-64 architecture.

<table>
<thead>
<tr>
<th>Linux* OS</th>
<th>Windows* OS</th>
<th>Optimizes applications for...</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mtune=itanium2-p9000</td>
<td>/G2-p9000</td>
<td>Default. Dual-Core Intel® Itanium® 2 processor (9000 series)</td>
</tr>
<tr>
<td>-mtune=itanium2</td>
<td>/G2</td>
<td>Intel® Itanium® 2 processors</td>
</tr>
</tbody>
</table>

**NOTE.** Mac OS* X: These options are not supported.

While the resulting executable is backward compatible, generated code is optimized for specific processors; therefore, code generated with `-mtune=itanium2-p9000` (Linux) or `/G2-p9000` (Windows) will run correctly on Itanium® 2 processors.

The following examples demonstrate using the default options to target an Itanium® 2 processor (9000 series). The same binary will also run on Intel® Itanium® 2 processors.
Restricting Optimizations

The following table lists options that restrict the ability of the Intel® compiler to optimize programs.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icpc -mtune=itanium2-p9000 prog.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /G2-p9000 prog.cpp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>-00</td>
<td>/Od</td>
<td>Disables all optimizations. Use this during development stages where fast compile times are desired.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux* and Mac OS* X:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Sets option -fomit-frame-pointer and option -fmath-errno.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Windows*:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use /Od to disable all optimizations while specifying particular optimizations, such as: /Od /Ob1 (disables all optimizations, but only enables inlining)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see the following topic:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• /Od compiler option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -00 compiler option</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Effect</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
</tbody>
</table>
| -g                  | /Zi, /Z7 | Generates symbolic debugging information in object files for use by debuggers. This option enables or disables other compiler options depending on architecture and operating system; for more information about the behavior, see the following topic:  
  - /Zi compiler option  
  - -g compiler option |
| -fno-builtin        | /Oi-     | Disables inline expansion of intrinsic functions. For more information, see the following topic:  
  - -fbuiltin compiler option |
| -fmath-errno,       | No equivalent | Instructs the compiler to assume that the program tests errno after calls to math library functions. For more information, see the following topic:  
  - -fmath-errno compiler option |
## Diagnostic Options

<table>
<thead>
<tr>
<th>Linux and Mac OS X</th>
<th>Windows</th>
<th>Effect</th>
</tr>
</thead>
</table>
| `-sox`             | `/Qsox`   | Instructs the compiler to save the compiler options and version number in the executable. During the linking process, the linker places information strings into the resulting executable. Slightly increases file size, but using this option can make identifying versions for regression issues much easier. For more information, see the following topic:  
  - `/Qsox` compiler option  
  - `-sox` compiler option |
Using Parallelism: OpenMP* Support

OpenMP* Support Overview

The Intel® compiler supports the OpenMP* Version 3.0 API specification. For complete C++ language support for OpenMP, see the OpenMP Application Program Interface Version 3.0 specification, which is available from the OpenMP web site (http://www.openmp.org/, click the Specifications link).

This version of the Intel compiler also introduces OpenMP API Version 3.0 API specification support, as described in the OpenMP web site (http://www.openmp.org/, click Specifications).

OpenMP provides symmetric multiprocessing (SMP) with the following major features:

- Relieves the user from having to deal with the low-level details of iteration space partitioning, data sharing, and thread creation, scheduling, and synchronization.
- Provides the benefit of the performance available from shared memory multiprocessor and multi-core processor systems on IA-32, Intel® 64, and IA-64 architectures, including those processors with Hyper-Threading Technology.

The compiler performs transformations to generate multithreaded code based on a developer's placement of OpenMP directives in the source program making it easy to add threading to existing software. The Intel compiler supports all of the current industry-standard OpenMP directives and compiles parallel programs annotated with OpenMP directives.

The compiler provides Intel-specific extensions to the OpenMP Version 3.0 specification including run-time library routines and environment variables. However, these extensions are only supported by the Intel compilers. A summary of the compiler options that apply to OpenMP* appears in OpenMP* Options Quick Reference.

Parallel Processing with OpenMP

To compile with OpenMP, you need to prepare your program by annotating the code with OpenMP directives. The Intel compiler processes the application and internally produces a multithreaded version of the code which is then compiled. The output is an executable with the parallelism implemented by threads that execute parallel regions or constructs.
Using Other Compilers

The OpenMP specification does not define interoperability of multiple implementations; therefore, the OpenMP implementation supported by other compilers and OpenMP support in Intel compilers might not be interoperable. Even if you compile and build the entire application with one compiler, be aware that different compilers might not provide OpenMP source compatibility that would allow you to compile and link the same set of application sources with a different compiler and get the expected parallel execution results.

Intel compilers include two sets of OpenMP libraries, as described in OpenMP Source Compatibility and Interoperability with Other Compilers.

OpenMP* Options Quick Reference

These options are supported on IA-32, Intel® 64, and IA-64 architectures.

<table>
<thead>
<tr>
<th>Linux® OS and Mac OS® X</th>
<th>Windows® OS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-openmp</td>
<td>/Qopenmp</td>
<td>This option enables the parallelizer to generate multi-threaded code based on the OpenMP* directives. The code can be executed in parallel on both uniprocessor and multiprocessor systems.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IA-64 architecture only:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Implies -opt-mem-bandwidth1 (Linux) or /Qopt-mem-bandwidth1 (Windows).</td>
</tr>
<tr>
<td>-openmp-report</td>
<td>/Qopenmp-report</td>
<td>This option controls the OpenMP parallelizer’s level of diagnostic messages. To use this option, you must also specify -openmp (Linux and Mac OS X) or /Qopenmp (Windows).</td>
</tr>
<tr>
<td>Linux* OS and Mac OS* X</td>
<td>Windows* OS</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-openmp-stubs</td>
<td>/Qopenmp-stubs</td>
<td>This option enables compilation of OpenMP programs in sequential mode. The OpenMP directives are ignored and a stub OpenMP library is linked.</td>
</tr>
<tr>
<td>-openmp-profile</td>
<td>/Qopenmp-profile</td>
<td>This option enables analysis of OpenMP* applications. To use this option, you must have previously installed Intel® Thread Profiler, which is one of the Intel® Threading Analysis Tools. This option can adversely affect performance because of the additional profiling and error checking invoked to enable compatibility with the threading tools. Do not use this option unless you plan to use the Intel® Thread Profiler.</td>
</tr>
<tr>
<td>-openmp-lib</td>
<td>/Qopenmp-lib</td>
<td>This option lets you specify an OpenMP* run-time library to use for linking. The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers. The compatibility OpenMP run-time library is compatible with object files created using the Microsoft* OpenMP run-time library (vcomp) and GNU OpenMP run-time library (libgomp).</td>
</tr>
</tbody>
</table>
This option controls whether the compiler links to static or dynamic OpenMP run-time libraries. To link to the static OpenMP run-time library (RTL) and create a purely static executable, you must specify `-openmp-link static` (Linux and Mac OS X) or `/Qopenmp-link static` (Windows). However, we strongly recommend you use the default setting, `-openmp-link dynamic` (Linux and Mac OS X) or `/Qopenmp-link dynamic` (Windows).

The option lets you choose an OpenMP tasking model. To use this option, you must also specify option `-openmp` (Linux and Mac OS X) or `/Qopenmp` (Windows).

This option lets you specify an OpenMP threadprivate implementation. The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers.

When both `-openmp` and `-parallel` (Linux OS and Mac OS X) or `/Qopenmp` and `/Qparallel` (Windows OS) are specified on the command line, the parallel option is only applied in loop nests that do not have OpenMP directives. For loop nests with OpenMP directives, only the openmp option is applied.

Refer to the following topics for information about OpenMP environment variable and run-time routines:

- OpenMP Environment Variables
• OpenMP Run-time Library Routines
• Intel Extension Routines to OpenMP

Refer to Quick Reference Lists for a complete listing of the quick reference topics.

OpenMP* Source Compatibility and Interoperability with Other Compilers

Intel compilers include two sets of OpenMP libraries:

• The Compatibility OpenMP libraries, which provide compatibility with OpenMP support provided by certain versions of the Microsoft Visual C++* compiler on Windows* OS, certain versions of the GNU* compilers on Linux* OS and Mac OS* X, as well as the Intel compiler version 10.× (and later).
• The Legacy OpenMP libraries, which provide compatibility with OpenMP support provided by Intel compilers, including Intel compilers prior to version 10.0.

To select the Compatibility (default) or Legacy OpenMP libraries, use the Intel compiler to link your application and specify the Intel compiler option /Qopenmp-lib (Windows OS) or -openmp-lib (Linux OS and Mac OS X).

The term "object-level interoperability" refers to the ability to link object files and libraries generated by one compiler with object files and libraries generated by the second compiler, such that the resulting executable runs successfully. In contrast, "source compatibility" means that the entire application is compiled and linked by one compiler, and you do not need to modify the sources to get the resulting executable to run successfully.

Different compilers support different versions of the OpenMP specification. Based on the OpenMP features your application uses, determine what version of the OpenMP specification your application requires. If your application uses an OpenMP specification level equal or less than the OpenMP specification level supported by all the compilers, your application should have source compatibility with all compilers, but you need to link all object files and libraries with the same compiler's OpenMP libraries.

OpenMP Compatibility Libraries Provided by Intel Compilers

The Compatibility libraries provide source compatibility and object-level interoperability with the OpenMP support provided by:

• On Windows* OS, certain versions of Microsoft Visual C++* that support OpenMP, starting with Microsoft Visual C++ 2005.
• On Linux* OS and Mac OS* X, certain versions of GNU* gcc* that support OpenMP, starting with GNU* gcc* version 4.2.
• Intel compilers versions 10.0 and later and their supplied OpenMP libraries.
For Fortran applications on Linux systems, it is not possible to link objects compiled by the Intel® Fortran Compiler (ifort) with objects compiled by the GNU* Fortran compiler (gfortran). Thus, for mixed-language C++ and Fortran applications, you can do one of the following:

- Combine objects created by gfortran and Intel® C++ objects, if you specify the Intel OpenMP Compatibility libraries during linking.
- Combine objects created by the Intel C++ compiler and the Intel Fortran Compiler, using Intel OpenMP Compatibility or Legacy libraries.

**OpenMP Legacy Libraries Provided by Intel Compilers**

The set of Legacy OpenMP libraries has been provided by Intel compilers for multiple releases and provide source compatibility and object-level interoperability with the current Legacy libraries and OpenMP libraries provided by previous Intel compiler versions, including those prior to version 10.0. The Legacy libraries are not compatible with OpenMP support from non-Intel compilers, such as Microsoft Visual C++*, GNU gcc*, or GNU Fortran.

You should only use the Legacy libraries if your application requires object-level interoperability with OpenMP library versions provided prior to Intel compilers version 10.0.

**Guidelines for Using Different Intel Compiler Versions**

To avoid possible linking or run-time problems, follow these guidelines:

- If you compile your application using only the Intel compilers, avoid mixing the Compatibility and Legacy OpenMP runtime libraries. That is, you must link the entire application with either the Compatibility or Legacy libraries.
- When using the Legacy libraries, use the most recent Intel compiler to link the entire application. However, be aware that the Legacy libraries are deprecated, so for a future release, you will need to link the entire application with the Compatibility libraries.
- Use dynamic instead of static OpenMP libraries to avoid linking multiple copies of the libraries into a single program. For details, see [OpenMP Support Libraries](#).

**Guidelines for Using Intel and Non-Intel Compilers**

To avoid possible linking or run-time problems, follow these guidelines:

- Always link the entire application using the Intel compiler OpenMP Compatibility libraries. This avoids linking multiple copies of the OpenMP runtime libraries from different compilers. It is easiest if you use the Intel compiler command (driver) to link the application, but it is possible to link with the Intel compiler OpenMP Compatibility libraries when linking the application using the GNU* or Visual C++ compiler (or linker) commands.
- If possible, compile all the OpenMP sources with the same compiler. If you compile (not link) using multiple compilers such as the Microsoft Visual C++* or GNU compilers that provide object-level interoperability with the Compatibility libraries, see the instructions in [Using the OpenMP Compatibility Libraries](#).
• Use dynamic instead of static OpenMP libraries to avoid linking multiple copies of the libraries into a single program. For details, see OpenMP Support Libraries.

**Limitations When Using OpenMP Compatibility Libraries with Other Compilers**

Limitations of threadprivate objects on object-level interoperability:

• On Windows OS systems, the Microsoft Visual C++* compiler uses a different mechanism than the Intel compilers to reference threadprivate data. If you declare a variable as threadprivate in your code and you compile the code with both Intel compilers and Visual C++ compilers, the code compiled by the Intel compiler and the code compiled by the Visual C++* compiler will reference different locations for the variable even when referenced by the same thread. Thus, use the same compiler to compile all source modules that use the same threadprivate objects.

• On Linux OS systems, the GNU* compilers use a different mechanism than the Intel compilers to reference threadprivate data. If you declare a variable as threadprivate in your code and you compile the code with both Intel compilers and GNU compilers, the code compiled by the Intel compiler and the code compiled by the GNU compiler will reference different locations for the variable even when referenced by the same thread. Thus, use the same compiler to compile all source modules that use the same threadprivate objects.

• On Mac OS* X systems, the operating system does not currently support the mechanism used by the GNU* compiler to support threadprivate data. Threadprivate data objects will only be accessible by name from object files compiled by the Intel compilers.

**Using OpenMP**

Using OpenMP* in your application requires several steps. To use OpenMP, you must do the following:

1. Add OpenMP directives to your application source code.

2. Compile the application with `-openmp` (Linux* and Mac OS* X) or `/Qopenmp` (Windows*) option.

3. For applications with large local or temporary arrays, you may need to increase the stack space available at run-time. In addition, you may need to increase the stack allocated to individual threads by using the KMP_STACKSIZE environment variable or by setting the corresponding library routines.

You can set other environment variables for the multi-threaded code execution.
Add OpenMP Support to the Application

Add the OpenMP API routine declarations to your application by adding a statement similar to the following in your code:

**Example**

```c
use omp_lib
```

OpenMP Directive Syntax

OpenMP directives use a specific format and syntax. Intel Extension Routines to OpenMP* describes the OpenMP extensions to the specification that have been added to the Intel® compiler.

The following syntax illustrates using the directives in your source.

**Example**

```c
(prefix) <directive> [clause, ...] <newline>
```

where:

- `<prefix>` - Required for all OpenMP directives. The prefix must be `#pragma omp`.
- `<directive>` - A valid OpenMP directive. Must immediately follow the prefix.
- `[<clause>]` - Optional. Clauses can be in any order and repeated as necessary, unless otherwise restricted.
- `[<newline>]` - A required component of directive syntax. It precedes the structured block which is enclosed by this directive.

The directives are interpreted as comments if you omit the `-openmp` (Linux and Mac OS X) or `/Qopenmp` (Windows*) option.
The following example demonstrates one way of using an OpenMP directive to parallelize a loop.

**Example**

```c
#include <omp.h>
void simple_omp(int *a){
  int i;
  #pragma omp parallel for
  for (i=0; i<1024; i++)
    a[i] = i*2;
}
```

See [OpenMP* Examples](#) for more examples on using directives in specific circumstances.

**Compile the Application**

The `-openmp` (Linux* and Mac OS* X) or `/Qopenmp` (Windows*) option enables the parallelizer to generate multi-threaded code based on the OpenMP directives in the source. The code can be executed in parallel on single processor, multi-processor, or multi-core processor systems.

**NOTE.** IA-64 Architecture: Specifying this option implies `-opt-mem-bandwidth1` (Linux) or `/Qopt-mem-bandwidth1` (Windows).

The `openmp` option works with both `-O0` (Linux and Mac OS X) and `/Od` (Windows) and with any optimization level of `-O1`, `-O2` and `-O3` (Linux and Mac OS X) or `/O1`, `/O2` and `/O3` (Windows).

Specifying `-O0` (Linux and Mac OS X) or `/Od` (Windows) with the OpenMP option helps to debug OpenMP applications.

Compile your application using commands similar to those shown below:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td><code>icc -openmp source_file</code></td>
</tr>
<tr>
<td>Operating System</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Qopenmp source_file</td>
</tr>
</tbody>
</table>

Assume that you compile the sample above, using the commands similar to the following, where
-c (Linux and Mac OS X) or /c (Windows) instructs the compiler to compile the code without
generating an executable:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icc -openmp -c parallel.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Qopenmp /c parallel.cpp</td>
</tr>
</tbody>
</table>

The compiler might return a message similar to the following:
parallel.c(20) : (col. 3) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.

**Configure the OpenMP Environment**

Before you run the multi-threaded code, you can set the number of desired threads using the
OpenMP environment variable, OMP_NUM_THREADS. See the [OpenMP Environment Variables](#).

**Parallel Processing Model**

A program containing OpenMP* API compiler directives begins execution as a single thread,
called the initial thread of execution. The initial thread executes sequentially until the first
parallel construct is encountered.

In the OpenMP API, the #pragma omp parallel directive defines the extent of the parallel
construct. When the initial thread encounters a parallel construct, it creates a team of threads,
with the initial thread becoming the master of the team. All program statements enclosed by
the parallel construct are executed in parallel by each thread in the team, including all routines
called from within the enclosed statements.

The statements enclosed lexically within a construct define the static extent of the construct.
The dynamic extent includes all statements encountered during the execution of a construct
by a thread, including all called routines.
When a thread encounters the end of a structured block enclosed by a parallel construct, the thread waits until all threads in the team have arrived. When that happens the team is dissolved, and only the master thread continues execution of the code following the parallel construct. The other threads in the team enter a wait state until they are needed to form another team. You can specify any number of parallel constructs in a single program. As a result, thread teams can be created and dissolved many times during program execution.

The following example illustrates, from a high level, the execution model for the OpenMP constructs. The comments in the code explain the structure of each construct or section.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| main() { // Begin serial execution. 
... // Only the initial thread executes 
#pragma omp parallel // Begin a parallel construct and form 
{ // a team. 
  #pragma omp sections // Begin a worksharing construct. 
  { 
    #pragma omp section // One unit of work. 
    {...} 
    #pragma omp section // Another unit of work. 
    {...} 
  } // Wait until both units of work complete. 
... // This code is executed by each team member. 
#pragma omp for nowait // Begin a worksharing Construct 
for(...) 
{ // Each iteration chunk is unit of work. 
... // Work is distributed among the team members. } |
**Example**

```cpp
} // End of worksharing construct.
     // nowait was specified so threads proceed.
    #pragma omp critical // Begin a critical section.
{(...) // Only one thread executes at a time.
   ... // This code is executed by each team member.
   #pragma omp barrier // Wait for all team members to arrive.
   ... // This code is executed by each team member.
} // End of Parallel Construct
} // Disband team and continue serial execution.
... // Possibly more parallel constructs.
} // End serial execution.
```

**Using Orphaned Directives**

In routines called from within parallel constructs, you can also use directives. Directives that are not in the static extent of the parallel construct, but are in the dynamic extent, are called orphaned directives. Orphaned directives allow you to execute portions of your program in
parallel with only minimal changes to the sequential version of the program. Using this functionality, you can code parallel constructs at the top levels of your program call tree and use directives to control execution in any of the called routines. For example:

```
Example

```int main(void)
{
    #pragma omp parallel
    {
        phase1();
    }
}

void phase1(void)
{
    #pragma omp for // This is an orphaned pragma.
    for(i=0; i < n; i++)
    {
        some_work(i);
    }
}
```

This is an orphaned for loop directive since the parallel region is not lexically present in routine phase 1.

**Data Environment Controls**

You can control the data environment within parallel and worksharing constructs. Using directives and data environment clauses on directives, you can privatize named global-lifetime objects by using THREADPRIVATE directive, or control data scope attributes by using the data environment clauses for directives that support them.

The data scope attribute clauses are:

- default
- private
The data copying clauses are:

- `copyin`
- `copyprivate`

You can use several directive clauses to control the data scope attributes of variables for the duration of the construct in which you specify them; however, if you do not specify a data scope attribute clause on a directive, the behavior for the variable is determined by the default scoping rules, which are described in the OpenMP API specification, for the variables affected by the directive.

**Determining How Many Threads to Use**

For applications where the workload depends on application input that can vary widely, delay the decision about the number of threads to employ until runtime when the input sizes can be examined. Examples of workload input parameters that affect the thread count include things like matrix size, database size, image/video size and resolution, depth/breadth/bushiness of tree based structures, and size of list based structures. Similarly, for applications designed to run on systems where the processor count can vary widely, defer the number of threads to employ until application run-time when the machine size can be examined.

For applications where the amount of work is unpredictable from the input data, consider using a calibration step to understand the workload and system characteristics to aid in choosing an appropriate number of threads. If the calibration step is expensive, the calibration results can be made persistent by storing the results in a permanent place like the file system.

Avoid simultaneously using more threads than the number of processing units on the system. This situation causes the operating system to multiplex the processors and typically yields sub-optimal performance.

When developing a library as opposed to an entire application, provide a mechanism whereby the user of the library can conveniently select the number of threads used by the library, because it is possible that the user has higher-level parallelism that renders the parallelism in the library unnecessary or even disruptive.
Use the `num_threads` clause on parallel regions to control the number of threads employed and use the `if` clause on parallel regions to decide whether to employ multiple threads at all. The `omp_set_num_threads` routine can also be used, but it also affects parallel regions created by the calling thread. The `num_threads` clause is local in its effect, so it does not impact other parallel regions.

By default, the Intel OpenMP runtime lets you to create a large number of threads and active nested parallel regions. Use `OMP_GET_THREAD_LIMIT()` and `OMP_GET_MAX_ACTIVE_LEVELS()` to determine the limits. Developers should carefully consider their thread usage and nesting of parallelism to avoid overloading the system. The `OMP_THREADS_LIMIT` environment variable limits the number of OpenMP threads to use for the whole OpenMP program. The `OMP_MAX_ACTIVE_LEVELS` environment variable limits the number of active nested parallel regions.

**Worksharing Using OpenMP**

To get the maximum performance benefit from a processor with multi-core and Hyper-Threading Technology, an application needs to be executed in parallel. Parallel execution requires threads, and threading an application is not a simple thing to do; using OpenMP* can make the process a lot easier. Using the OpenMP pragmas, most loops with no loop-carried dependency can be threaded with one simple statement. This topic explains how to start using OpenMP to parallelize loops, which is also called worksharing.

Most loops can be threaded by inserting one pragma immediately prior to the loop. Further, by leaving the details to the compiler and OpenMP, you can spend more time determining which loops should be threaded and how to best restructure the algorithms for maximum performance. The maximum performance of OpenMP is realized when it is used to thread hotspots, the most time-consuming loops in your application.
The power and simplicity of OpenMP is demonstrated by looking at an example. The following loop converts a 32-bit RGB (red, green, blue) pixel to an 8-bit gray-scale pixel. One pragma, which has been inserted immediately before the loop, is all that is needed for parallel execution.

```
#pragma omp parallel for
for (i=0; i < numPixels; i++)
{
    pGrayScaleBitmap[i] = (unsigned BYTE)
        (pRGBBitmap[i].red * 0.299 +
         pRGBBitmap[i].green * 0.587 +
         pRGBBitmap[i].blue * 0.114);
}
```

First, the example uses worksharing, which is the general term used in OpenMP to describe distribution of work across threads. When worksharing is used with the for construct, as shown in the example, the iterations of the loop are distributed among multiple threads so that each loop iteration is executed exactly once and in parallel by one or more threads. OpenMP determines the number of threads to create and how to best create, synchronize, and destroy them. OpenMP places the following five restrictions on which loops can be threaded:

- The loop variable must be of type signed integer or unsigned integers.
- The comparison operation must be in the form loop_variable <, <=, >, or >= loop_invariant_integer.
- The third expression or increment portion of the for loop must be either integer addition or integer subtraction and by a loop invariant value.
- If the comparison operation is < or <=, the loop variable must increment on every iteration; conversely, if the comparison operation is > or >=, the loop variable must decrement on every iteration.
- The loop must be a basic block, meaning no jumps from inside to outside the loop are permitted with the exception of the exit statement, which terminates the whole application. If the statements goto or break are used, the statements must jump within the loop, not outside it. The same goes for exception handling; exceptions must be caught within the loop.

Although these restrictions might sound somewhat limiting, non-conforming loops can easily be rewritten to follow these restrictions.
**Basics of Compilation**

Using the OpenMP pragmas requires an OpenMP-compatible compiler and thread-safe libraries. Adding the `/Qopenmp` option to the compiler instructs the compiler to pay attention to the OpenMP pragmas and to insert threads. If you omit the `/Qopenmp` option, the compiler will ignore OpenMP pragmas, which provides a very simple way to generate a single-threaded version without changing any source code.

For conditional compilation, the compiler defines `_OPENMP`. If needed, the define can be tested as shown in the following example.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| #ifdef __OPENMP
  fn();
#endif |

**A Few Simple Examples**

The following examples illustrate how simple OpenMP is to use. In common practice, additional issues need to be addressed, but these example illustrate a good starting place.

In the first example, the following loop clips an array to the range 0...255.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| // clip an array to 0 <= x <= 255
for (i=0; i < numElements; i++)
{
  if (array[i] < 0)
    array[i] = 0;
  else if (array[i] > 255)
    array[i] = 255;
} |
You can thread it using a single OpenMP pragma; insert the following pragma immediately prior to the loop:

```
#pragma omp parallel for
for (i=0; i < numElements; i++)
{
    if (array[i] < 0)
        array[i] = 0;
    else if (array[i] > 255)
        array[i] = 255;
}
```

In the second example, the loop generates a table of square roots for the numbers 0...100.

```
double value;
double roots[100];
for (value = 0.0; value < 100.0; value++)
{
    roots[(int)value] = sqrt(value);
}
```
Thread the loop by changing the loop variable to a signed integer or unsigned integer and inserting a #pragma omp parallel pragma.

**Example**

```c
int value;
double roots[100];
#pragma omp parallel for
for (value = 0; value < 100; value++)
{
    roots[value] = sqrt((double)value);
}
```

**Avoiding Data Dependencies and Race Conditions**

When a loop meets all five loop restrictions (listed above) and the compiler threads the loop, the loop still might not work correctly due to the existence of data dependencies.

Data dependencies exist when different iterations of a loop-more specifically a loop iteration that is executed on a different thread-reads or writes shared memory. Consider the following example that calculates factorials.

**Example**

```c
// Each loop iteration writes a value that a different iteration reads.
#pragma omp parallel for
for (i=2; i < 10; i++)
{
    factorial[i] = i * factorial[i-1];
}
```

The compiler will thread this loop, but the threading will fail because at least one of the loop iterations is data-dependent upon a different iteration. This situation is referred to as a race condition. Race conditions can only occur when using shared resources (like memory) and parallel execution. To address this problem either rewrite the loop or pick a different algorithm, one that does not contain the race condition.
Race conditions are difficult to detect because, for a given case or system, the variables might win the race in the order that happens to make the program function correctly. Because a program works once does not mean that the program will work under all conditions. Testing your program on various machines, some with Hyper-Threading Technology and some with multiple physical processors, is a good starting point to help identify race conditions.

Traditional debuggers are useless for detecting race conditions because they cause one thread to stop the race while the other threads continue to significantly change the runtime behavior; however, thread checking tools can help.

Managing Shared and Private Data

 Nearly every loop (in real applications) reads from or writes to memory; it’s your responsibility, as the developer, to instruct the compiler what memory should be shared among the threads and what memory should be kept private. When memory is identified as shared, all threads access the exact same memory location. When memory is identified as private, however, a separate copy of the variable is made for each thread to access in private. When the loop ends, the private copies are destroyed. By default, all variables are shared except for the loop variable, which is private. Memory can be declared as private in the following two ways:

- Declare the variable inside the loop-really inside the parallel OpenMP directive-without the static keyword.
- Specify the private clause on an OpenMP directive.

The following loop fails to function correctly because the variable temp is shared. It needs to be private.

```
// Variable temp is shared among all threads, so while one thread
// is reading variable temp another thread might be writing to it
#pragma omp parallel for
for (i=0; i < 100; i++)
{
    temp = array[i];
    array[i] = do_something(temp);
}
```
The following two examples both declare the variable `temp` as private memory, which solves the problem.

**Example**

```c
#pragma omp parallel for
for (i=0; i < 100; i++)
{
    int temp; // variables declared within a parallel construct
    // are, by definition, private
    temp = array[i];
    array[i] = do_something(temp);
}
```

The temporary variable can also be made private in the following way:

**Example**

```c
#pragma omp parallel for private(temp)
for (i=0; i < 100; i++)
{
    temp = array[i];
    array[i] = do_something(temp);
}
```

Every time you use OpenMP to parallelize a loop, you should carefully examine all memory references, including the references made by called functions. Variables declared within a parallel construct are defined as private except when they are declared with the static declarator, because static variables are not allocated on the stack.
Reductions

Loops that accumulate a value are fairly common, and OpenMP has a specific clause to accommodate them. Consider the following loop that calculates the sum of an array of integers.

Example

```c
sum = 0;
for (i=0; i < 100; i++)
{
    sum += array[i]; // this variable needs to be shared to generate
                    // the correct results, but private to avoid
                    // race conditions from parallel execution
}
```

The variable sum in the previous loop must be shared to generate the correct result, but it also must be private to permit access by multiple threads. OpenMP provides the reduction clause that is used to efficiently combine the mathematical reduction of one or more variables in a loop. The following example demonstrates how the loop can use the reduction clause to generate the correct results.

Example

```c
sum = 0;
#pragma omp parallel for reduction(+:sum)
for (i=0; i < 100; i++)
{
    sum += array[i];
}
```

In the case of the example listed above, the reduction provides private copies of the variable sum for each thread, and when the threads exit, it adds the values together and places the result in the one global copy of the variable.

The following table lists the possible reductions, along with the initial variables—which is also the mathematical identify value—for the temporary private variables.
Multiple reductions in a loop are possible by specifying comma-separated variables and reductions on a given parallel construct. Reductions variables must meet the following requirements:

- can be listed in just one reduction
- cannot be declared constant
- cannot be declared private in the parallel construct

## Load Balancing and Loop Scheduling

Load balancing, the equal division of work among threads, is among the most important attributes for parallel application performance. Load balancing is extremely important, because it ensures that the processors are busy most, if not all, of the time. Without a balanced load, some threads may finish significantly before others, leaving processor resources idle and wasting performance opportunities.

Within loop constructs, poor load balancing is usually caused by variations in compute time among loop iterations. It is usually easy to determine the variability of loop iteration compute time by examining the source code. In most cases, you will see that loop iterations consume a uniform amount of time. When that is not true, it may be possible to find a set of iterations that consume similar amounts of time. For example, sometimes the set of all even iterations consumes about as much time as the set of all odd iterations. Similarly, it might be the case that the set of the first half of the loop consumes about as much time as the second half. In contrast, it might be impossible to find sets of loop iterations that have a uniform execution
Regardless of the case, you should provide this extra loop scheduling information to OpenMP so it can better distribute the iterations of the loop across the threads (and therefore processors) for optimum load balancing.

By default OpenMP assumes that all loop iterations consume the same amount of time. This assumption leads OpenMP to distribute the iterations of the loop among the threads in roughly equal amounts and in such a way as to minimize the chances of memory conflicts due to false sharing. This behavior is possible because loops generally touch memory sequentially, so splitting up the loop in large chunks—like the first half and second half when using two threads—will result in the least chance for overlapping memory. While this may be the best choice for memory issues, it may be bad for load balancing. Unfortunately, the reverse is also true; what might be best for load balancing may be bad for memory performance. You must strike a balance between optimal memory usage and optimal load balancing by measuring the performance to see what method produces the best results.

Use the following general on the parallel construct syntax to instruct OpenMP to loop schedule:

```
#pragma omp parallel for schedule(kind [, chunk size])
```

Four different loop scheduling types (kinds) can be provided to OpenMP, as shown in the following table. The optional parameter (chunk), when specified, must be a loop-invariant positive integer.

<table>
<thead>
<tr>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
</table>
| static | Divide the loop into equal-sized chunks or as equal as possible in the case where the number of loop iterations is not evenly divisible by the number of threads multiplied by the chunk size. By default, chunk size is loop count/number of threads.  
Set chunk to 1 to interleave the iterations. |
| dynamic | Use the internal work queue to give a chunk-sized block of loop iterations to each thread. When a thread is finished, it retrieves the next block of loop iterations from the top of the work queue.  
By default, the chunk size is 1. Be careful when using this scheduling hint because of the extra overhead requirement. |
Similar to dynamic scheduling, but the chunk size starts off large and shrinks in an effort to reduce the amount of time threads have to go to the work queue to get more work. The optional chunk parameter specifies them minimum size chunk to use.

By default the chunk size is approximately the loop count/number of threads.

When schedule (auto) is specified, the decision regarding scheduling is delegated to the compiler. The programmer gives the compiler the freedom to choose any possible mapping of iterations to threads in the team.

Uses the `OMP_SCHEDULE` environment variable to specify which one of the three loop-scheduling types should be used.

`OMP_SCHEDULE` is a string formatted exactly the same as would appear on the parallel construct.

Assume that you want to parallelize the following loop.

```c
for (i=0; i < NumElements; i++)
{
    array[i] = StartVal;
    StartVal++;
}
```
As written, the loop contains a data dependency, making it impossible to parallelize without a quick change. The new loop, shown below, fills the array in the same manner, but without data dependencies. The new loop can also be written using the SIMD instructions.

```
Example

#pragma omp parallel for
for (i=0; i < NumElements; i++)
{
    array[i] = StartVal + i;
}
```

Observe that the code is not 100% identical because the value of variable `StartVal` is not incremented. As a result, when the parallel loop is finished, the variable will have a value different from the one produced by the serial version. If the value of `StartVal` is needed after the loop, the additional statement, shown below, is needed.

```
Example

// This works and is identical to the serial version.
#pragma omp parallel for
for (i=0; i < NumElements; i++)
{
    array[i] = StartVal + i;
}
StartVal += NumElements;
```

Tasking Model

The tasking model implemented by the Intel® compiler enables OpenMP® to parallelize a large range of applications. The directives used for tasking are:

```
#pragma omp task [clause[, clause] ...] new-line structured-block
```

where clause is one of the following:
• if(scalar-expression)
• untied
• default(shared | none)
• private(list)
• firstprivate(list)
• shared(list)

#pragma omp taskwait new-line

The #pragma omp task directive defines an explicit task region as follows:

Example

```c
void test1(LIST *head){
    #pragma intel omp parallel shared(head)
    {
        #pragma omp single
        { LIST *p = head;

        while (p != NULL) {
            #pragma omp task firstprivate(p)
            {
                do_work1(p);
            } // do_work1(p)
            p = p->next;
        } // while(p!=NULL)
    } // #pragma omp parallel
}
```

The binding thread set of the task region is the current parallel team. A task region binds to the innermost enclosing PARALLEL region. When a thread encounters a task construct, a task is generated from the structured block enclosed in the construct. The encountering thread may immediately execute the task, or defer its execution. A task construct may be nested inside an outer task, but the task region of the inner task is not a part of the task region of the outer task.

**Clauses Used**

The TASK directive takes an optional comma-separated list of clauses. The data environment of the task is created according to the data-sharing attribute clauses on the task construct and any defaults that apply. These clauses are in previous sections: Below example shows a way to generate N tasks with one thread and execute them with the threads in the parallel team:

```c
#pragma omp parallel shared(data)
{
    #pragma omp single private(i)
    {
        for (i=0, i<N; i++) {
            #pragma omp task firstprivate(i shared(data)
            {
                do_work(data(i));
            }
        }
    }
}
```

**Task Scheduling**

When a thread reaches a task scheduling point, it may perform a task switch, beginning or resuming execution of a different task bound to the current team. Task scheduling points are implied at the following locations:
• the point immediately following the generation of an explicit task
• after the last instruction of a task region
• in taskwait regions
• in implicit and explicit barrier regions

When a thread encounters a task scheduling point it may do one of the following:
• begin execution of a tied task bound to the current team
• resume any suspended task region, bound to the current team, to which it is tied
• begin execution of an untied task bound to the current team
• resume any suspended untied task region bound to the current team

If more than one of the above choices is available, it is unspecified as to which will be chosen.

**Task Scheduling Constraints**

1. An explicit task whose construct contained an if clause whose if clause expression evaluated to false is executed immediately after generation of the task.

2. Other scheduling of new tied tasks is constrained by the set of task regions that are currently tied to the thread, and that are not suspended in a barrier region. If this set is empty, any new tied task may be scheduled. Otherwise, a new tied task may be scheduled only if it is a descendant of every task in the set. A program relying on any other assumption about task scheduling is non-conforming.

**NOTE.** Task scheduling points dynamically divide task regions into parts. Each part is executed uninterruptedly from start to end. Different parts of the same task region are executed in the order in which they are encountered. In the absence of task synchronization constructs, the order in which a thread executes parts of different schedulable tasks is unspecified.

A correct program must behave correctly and consistently with all conceivable scheduling sequences that are compatible with the rules above.
**TASKWAIT pragma**

The TASKWAIT directive specifies a wait on the completion of child tasks generated since the beginning of the current task. A taskwait region binds to the current task region. The binding thread set of the taskwait region is the encountering thread.

The taskwait region includes an implicit task scheduling point in the current task region. The current task region is suspended at the task scheduling point until execution of all its child tasks generated before the taskwait region are completed.

**Example**

```cpp
#pragma omp task
{ ...
  #pragma omp task
  { do_work1(); }
  #pragma omp task
  { ...
    #pragma omp task
    { do_work2(); }
    ...
  }
  ...
}
#pragma omp taskwait
...
```

For more details on these directives, see OpenMP* C++ Compiler Directives.

**Verifying OpenMP* Using Parallel Lint**

To accelerate migration of sequential applications to parallel applications using OpenMP, parallel lint can be very helpful by reducing application development and debugging time. This topic explains how to use parallel lint to optimize your parallel application. Parallel lint performs static
global analysis of a program to diagnose existing and potential issues with parallelization. One of the advantages of parallel lint is that it makes its checks considering the whole stack of parallel regions and worksharing constructs, even when placed in different routines.

Example

```c
#include <stdio.h>
#include <omp.h>

void fff(int ii) {
  printf("We've got i=%d NTR=%d\n",ii, omp_get_thread_num() );
}

void sec2(int i){
  #pragma omp single
  fff(i+2);
}

int main(int n) {
  int i=3;
  omp_set_num_threads(3);
  #pragma omp parallel
  #pragma omp sections
```
Example

```cpp
25    { 
26        #pragma omp sections 
27        sec2(i); 
28    } 
29    return 0; 
30 }
```

as_12_01.cpp(16): error #12200: single pragma is not allowed in
the dynamic extent of sections pragma (file:as_12_01.cpp line:24)
This makes parallel lint a powerful tool for diagnosing OpenMP pragmas in whole program context. Parallel lint also provides checks to debug errors connected with data dependencies and race conditions.

### Example

```c
#include <stdio.h>
#include <omp.h>

int main(void)
{
    int i;
    int factorial[10];

    factorial[0]=1;
    #pragma omp parallel for
    for (i=1; i < 10; i++) {
        factorial[i] = i * factorial[i-1];
    }

    return 0;
}
```

omp.c(13): warning #12246: flow data dependence from (file:omp.c line:13) to (file:omp.c line:13),
due to "factorial" may lead to incorrect program execution in parallel mode

### Basics of Compilation

To enable parallel lint analysis, pass the `/Qdiag-enable:sc-parallel[n]` *(Windows)*, `-diag-enable sc-parallel[n]` *(Linux and Mac OS)* option to the compiler.

Parallel lint is available for IA-32 and Intel® 64 architectures only.
Parallel lint requires the OpenMP option, /Qopenmp (Windows) -openmp (Linux and Mac OS). This option forces the compiler to process OpenMP pragmas to make parallelization specifics available for parallel lint analysis. If parallel lint is used without OpenMP, the compiler issues the following error message:

command line error: parallel lint not called due to lack of OpenMP parallelization option, please add option /Qopenmp when using parallel lint.

If you are using Microsoft Visual Studio*, you should create a separate build configuration devoted to parallel lint, since object and library files produced by parallel lint should not be used to build your product.

**Basic Checks**

Parallel lint provides a broad set of OpenMP checks which are useful both for beginners in parallel programming using OpenMP and for advanced parallel developers. See the **Overview** section of this manual.

The examples below highlight the most useful features of parallel lint.

**Case 1: Nested Regions**

An OpenMP program is much more difficult to debug if it has nested parallel regions. Various restrictions apply to nested parallel constructs. Parallel lint can check nested parallel statements even if they are located in different files.
In the example below, a worksharing construct may not be closely nested inside a worksharing, critical, ordered, or master construct.

Example

```c
#include <stdio.h>
#include <omp.h>

int fff(int ii)
{
  int rez;

  #pragma omp sections
  {
    rez = ii;
    #pragma omp section
    rez = ii+2;
  }
  return rez;
}

int main(int n)
{
  int i;

  omp_set_num_threads(3);
  #pragma omp parallel
```
Case 2: Data-Sharing Attribute Clauses

Parallelization of an existing serial application requires accurate placement of data sharing clauses. Parallel lint can help determine not only improper usage of sharing clauses but also lack of proper data sharing pragmas.
The example below demonstrates the OpenMP standard restriction: "If the `lastprivate` clause is used on a construct to which `nowait` is also applied, then the original list item remains undefined until a barrier synchronization has been performed to ensure that the thread that executed the sequentially last iteration, or the lexically last `SECTION` construct, has stored that list item." [OpenMP standard]

```c
#include <stdio.h>
#include <omp.h>

int main(void) {
    int last, i;
    float a[10], b[10];

    for (i=0; i < 10; i++) {
        b[i] = i*0.5;
    }

    #pragma omp parallel shared(a,b,last)
    {
        #pragma omp for lastprivate(last) nowait
        for (i=0; i < 10; i++) {
            a[i] = b[i] * 2;
            last = a[i];
        }
        #pragma omp single
        printf("%d\n", last);
    }

    return 0;
}
```
Example

24  }

omp.c(20): error #12220: lastprivate variable "last" in
nowait work-sharing construct is used before barrier synchronization

Case 3: Data Dependence

Data dependency issues are very difficult to debug in parallel programs due to non-deterministic behavior. Parallel lint is able to determine data dependency issues in programs without executing them.
To turn on data dependency analysis you should specify severity level 3 parallel lint in diagnostics.

**Example**

```c
#include <stdio.h>
#include <omp.h>

int main(void) {
  int i;
  float a[100];

  #pragma omp parallel for
  for (i=0; i < 100; i++) {
    a[i] = i*0.66;
  }

  #pragma omp parallel for
  for (i=1; i < 100; i++) {
    a[i] = a[i-1]*0.5 + a[i]*0.5;
  }

  return 0;
}
```

`omp.c(16): warning #12246: flow data dependence from (file:omp.c line:16) to (file:omp.c line:16), due to "a" may lead to incorrect program execution in parallel mode`
Case 4: Threadprivate Variables

Example

```
#include <stdio.h>
#include <omp.h>

int a[1000];
#pragma omp threadprivate (a)

int main(int n) {
    int i;
    int sum = 0;

    #pragma omp parallel for
    for (i=0; i < 1000; i++) {
        a[i] = i;
    }

    #pragma omp parallel for reduction (+:sum)
    for (i=10; i < 1000; i++) { // inconsistent init value
        sum = sum + a[i];
    }

    printf("%d\n", sum);
    return 0;
}
```

omp.cpp(17): error #12344: threadprivate variable "a" is used in loops with different initial values. See loops (file:omp.cpp line:12) and (file:omp.cpp line:16).
OpenMP* Directives

**THREADPRIVATE threadprivate Directive**

You can make named global-lifetime objects private to a thread, but global within the thread, by using the `threadprivate` directive.

Each thread gets its own copy of the object with the result that data written to the object by one thread is not directly visible to other threads. During serial portions and master sections of the program, accesses are to the master thread copy of the object.

You cannot use a thread private variable in any clause other than the following:

- `copyin`
- `copyprivate`
- `schedule`
- `num_threads`
- `if`

In the following example the variable, `counter`, is specified as thread private:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| int counter = 0;  
#pragma omp threadprivate(counter); |

OpenMP* Advanced Issues

This topic discusses how to use the OpenMP* library functions and environment variables and discusses some guidelines for enhancing performance with OpenMP*.

OpenMP* provides specific function calls, and environment variables. See the following topics to refresh you memory about the primary functions and environment variable used in this topic:

- OpenMP* Run-time Library Routines
- OpenMP* Environment Variables

To use the function calls, include the `omp.h` and `omp_lib.h` header files, which are installed in the INCLUDE directory during the compiler installation, and compile the application using the `-openmp` (Linux* and Mac OS* X) or `/Qopenmp` (Windows*) option.
The following example, which demonstrates how to use the OpenMP* functions to print the alphabet, also illustrates several important concepts.

First, when using functions instead of pragmas, your code must be rewritten; rewrites can mean extra debugging, testing, and maintenance efforts.

Second, it becomes difficult to compile without OpenMP support.

Third, it is very easy to introduce simple bugs, as in the loop (below) that fails to print all the letters of the alphabet when the number of threads is not a multiple of 26.

Fourth, you lose the ability to adjust loop scheduling without creating your own work-queue algorithm, which is a lot of extra effort. You are limited by your own scheduling, which is mostly likely static scheduling as shown in the example.

---

**Example**

```c
#include <stdio.h>
#include <omp.h>

int main(void)
{

    int i;
    omp_set_num_threads(4);
    #pragma omp parallel private(i)
    {
        // OMP_NUM_THREADS is not a multiple of 26,
        // which can be considered a bug in this code.
        int LettersPerThread = 26 / omp_get_num_threads();
        int ThisThreadNum = omp_get_thread_num();
        int StartLetter = 'a'+ThisThreadNum*LettersPerThread;
        int EndLetter = 'a'+ThisThreadNum*LettersPerThread+LettersPerThread;
        for (i=StartLetter; i<EndLetter; i++)
            printf("%c", i);
    }

    printf("\n");
    return 0;
}
```
Debugging threaded applications is a complex process, because debuggers change the run-time performance, which can mask race conditions. Even print statements can mask issues, because they use synchronization and operating system functions. OpenMP* itself also adds some complications, because it introduces additional structure by distinguishing private variables and shared variables, and inserts additional code. A specialized debugger that supports OpenMP, such as the Intel® Debugger, can help you to examine variables and step through threaded code. You can also use the Intel® Thread Checker to detect many hard-to-find threading errors analytically. Sometimes, a process of elimination can help identify problems without resorting to sophisticated debugging tools.

Remember that most mistakes are race conditions. Most race conditions are caused by shared variables that really should have been declared private. Start by looking at the variables inside the parallel regions and make sure that the variables are declared private when necessary. Next, check functions called within parallel constructs. By default, variables declared on the stack are private, but the C/C++ keyword static will change the variable to be placed on the global heap and therefore shared for OpenMP loops.

The default (none) clause, shown below, can be used to help find those hard-to-spot variables. If you specify default(none), then every variable must be declared with a data-sharing attribute clause.

```
#pragma omp parallel for default(none) private(x,y) shared(a,b)
```

Another common mistake is using uninitialized variables. Remember that private variables do not have initial values upon entering a parallel construct. Use the firstprivate and lastprivate clauses to initialize them only when necessary, because doing so adds extra overhead.

If you still can’t find the bug, then consider the possibility of reducing the scope. Try a binary-hunt. Force parallel sections to be serial again with if(0) on the parallel construct or commenting out the pragma altogether. Another method is to force large chunks of a parallel region to be critical sections. Pick a region of the code that you think contains the bug and place it within a critical section. Try to find the section of code that suddenly works when it is within a critical section and fails when it is not. Now look at the variables, and see if the bug is apparent. If that still doesn’t work, try setting the entire program to run in serial by setting the compiler-specific environment variable KMP_LIBRARY=serial.

If the code is still not working, compile it without the -openmp (Linux and Mac OS X) or /Qopenmp (Windows) option to make sure the serial version works.

Performance

OpenMP threaded application performance is largely dependent upon the following things:
• The underlying performance of the single-threaded code.
• CPU utilization, idle threads, and load balancing.
• The percentage of the application that is executed in parallel by multiple threads.
• The amount of synchronization and communication among the threads.
• The overhead needed to create, manage, destroy, and synchronize the threads, made worse by the number of single-to-parallel or parallel-to-single transitions called fork-join transitions.
• Performance limitations of shared resources such as memory, bus bandwidth, and CPU execution units.
• Memory conflicts caused by shared memory or falsely shared memory.

Performance always begins with a properly constructed parallel algorithm or application. For example, parallelizing a bubble-sort, even one written in hand-optimized assembly language, is not a good place to start. Keep scalability in mind; creating a program that runs well on two CPUs is not as efficient as creating one that runs well on \( n \) CPUs. With OpenMP, the number of threads is chosen by the compiler, so programs that work well regardless of the number of threads are highly desirable. Producer/consumer architectures are rarely efficient, because they are made specifically for two threads.

Once the algorithm is in place, make sure that the code runs efficiently on the targeted Intel® architecture; a single-threaded version can be a big help. Turn off the -openmp (Linux and Mac OS X) or /Qopenmp (Windows) option to generate a single-threaded version, or build with -openmp-stubs (Linux and Mac OS X) or /Qopenmp-stubs (Windows), and run the single-threaded version through the usual set of optimizations. See Worksharing Using OpenMP* for more information.

Once you have gotten the single-threaded performance, it is time to generate the multi-threaded version and start doing some analysis.

Optimizations are really a combination of patience, experimentation, and practice. Make little test programs that mimic the way your application uses the computer resources to get a feel for what things are faster than others. Be sure to try the different scheduling clauses for the parallel sections of code. If the overhead of a parallel region is large compared to the compute time, you may want to use an if clause to execute the section serially.

**OpenMP* Examples**

The following examples show how to use several OpenMP* features.
A Simple Difference Operator

This example shows a simple parallel loop where the amount of work in each iteration is different. Dynamic scheduling is used to improve load balancing.

The `for` has a `nowait` because there is an implicit barrier at the end of the parallel region.

```c
void for1(float a[], float b[], int n)
{
    int i, j;
    #pragma omp parallel shared(a,b,n)
    {
        #pragma omp for schedule(dynamic,1) private (i,j) nowait
        for (i = 1; i < n; i++)
        for (j = 0; j < i; j++)
            b[j + n*i] = (a[j + n*i] + a[j + n*(i-1)]) / 2.0;
    }
}
```
Two Difference Operators: for Loop Version

The example uses two parallel loops fused to reduce fork/join overhead. The first for directive has a nowait clause because all the data used in the second loop is different than all the data used in the first loop.

Example

```cpp
void for2(float a[], float b[], float c[], float d[],
           int n, int m)
{
  int i, j;
  #pragma omp parallel shared(a,b,c,d,n,m) private(i,j)
  {
    #pragma omp for schedule(dynamic,1) nowait
    for (i = 1; i < n; i++)
      for (j = 0; j < i; j++)
        b[j + n*i] = ( a[j + n*i] + a[j + n*(i-1)] )/2.0;
    #pragma omp for schedule(dynamic,1) nowait
    for (i = 1; i < m; i++)
      for (j = 0; j < i; j++)
        d[j + m*i] = ( c[j + m*i] + c[j + m*(i-1)] )/2.0;
  }
}
```
Two Difference Operators: sections Version

The example demonstrates the use of the `sections` directive. The logic is identical to the preceding `for` example, but uses `sections` instead of `for`. Here the speedup is limited to 2 because there are only two units of work whereas in the example above there are \( n-1 + m-1 \) units of work.

**Example**

```c
void sections1(float a[], float b[], float c[], float d[],
int n, int m)
{
    int i, j;
    #pragma omp parallel shared(a,b,c,d,n,m) private(i,j)
    {
        #pragma omp sections nowait
        {
            #pragma omp section
            for (i = 1; i < n; i++)
                for (j = 0; j < i; j++)
                    b[j + n*i] = ( a[j + n*i] + a[j + n*(i-1)] )/2.0;
            #pragma omp section
            for (i = 1; i < m; i++)
                for (j = 0; j < i; j++)
                    d[j + m*i] = ( c[j + m*i] + c[j + m*(i-1)] )/2.0;
        }
    }
}
```
Updating a Shared Scalar

This example demonstrates how to use a single construct to update an element of the shared array `a`. The optional nowait after the first loop is omitted because it is necessary to wait at the end of the loop before proceeding into the single construct.

Example

```c
void sp_1a(float a[], float b[], int n)
{
    int i;
    #pragma omp parallel shared(a,b,n) private(i)
    {
        #pragma omp for
        for (i = 0; i < n; i++)
            a[i] = 1.0 / a[i];
        #pragma omp single
        a[0] = MIN( a[0], 1.0 );
        #pragma omp for nowait
        for (i = 0; i < n; i++)
            b[i] = b[i] / a[i];
    }
}
```

Libraries, Directives, Clauses, and Environmental Variables

OpenMP* Environment Variables

The Intel® Compiler supports OpenMP* environment variables (with the OMP_ prefix) and extensions in the form of Intel-specific environment variables (with the KMP_ prefix).
**OpenMP Environment Variables**

The syntax examples assume bash on Linux* and Mac OS* X. Use the `set` command for Windows*.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Default</th>
<th>Description and Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sets the maximum number of threads to use for OpenMP* parallel regions if no other value is specified in the application.</td>
</tr>
<tr>
<td>OMP_NUM_THREADS</td>
<td>Number of processors visible to the operating system.</td>
<td>This environment variable applies to both <code>-openmp</code> and <code>-parallel</code> (Linux and Mac OS X) or <code>/openmp</code> and <code>/parallel</code> (Windows). Example syntax: <code>export OMP_NUM_THREADS=value</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sets the run-time schedule type and an optional chunk size.</td>
</tr>
<tr>
<td>OMP_SCHEDULE</td>
<td>STATIC, no chunk size specified</td>
<td>Example syntax: <code>export OMP_SCHEDULE=&quot;kind[,chunk_size]&quot;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enables (1) or disables (0) the dynamic adjustment of the number of threads.</td>
</tr>
<tr>
<td>OMP_DYNAMIC</td>
<td>0</td>
<td>Example syntax: <code>export OMP_DYNAMIC=value</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enables (1) or disables (0) nested parallelism.</td>
</tr>
<tr>
<td>OMP_NESTED</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

*Note: Linux, Mac OS X, Windows, and OpenMP are trademarks of their respective companies.*
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Default</th>
<th>Description and Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OMP_STACKSIZE</strong></td>
<td>IA-32 architecture: 2M</td>
<td>Example syntax:</td>
</tr>
<tr>
<td></td>
<td>Intel® 64 and IA-32</td>
<td>export OMP_NESTED=value</td>
</tr>
<tr>
<td></td>
<td>Architectures: 4M</td>
<td>Sets the number of bytes to allocate for each OpenMP thread to use as the private stack for the thread.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recommended size is 16M.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use the optional suffixes: B (bytes), K (Kilobytes), M (Megabytes), G (Gigabytes), or T (Terabytes) to specify the units. If only value is specified and B, K, M, G, or T is not specified, then size is assumed to be K (Kilobytes).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This variable does not affect the native operating system threads created by the user program nor the thread executing the sequential part of an OpenMP program or parallel programs created using <code>-parallel</code> (Linux and Mac OS X) or <code>/Qparallel</code> (Windows).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>kmp_set_stacksize_s()</code> routine must be called from sequential part, before first parallel region is created.</td>
</tr>
</tbody>
</table>

The `kmp_{set, get}_stacksize_s()` routines set/retrieve the value.
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Default</th>
<th>Description and Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Otherwise, calling kmp_set_stacksize_s() has no effect.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Related env variables:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KMP_STACKSIZE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KMP_STACKSIZE overrides</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OMP_STACKSIZE.</td>
</tr>
<tr>
<td>Example syntax:</td>
<td></td>
<td>export OMP_STACKSIZE=value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limits the number of simultaneously executing threads in an OpenMP program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If this limit is reached and another native operating system thread encounters OpenMP API calls or constructs, the program can abort with an error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If this limit is reached when an OpenMP parallel region begins, a one-time warning message might be generated indicating that the number of threads in the team was reduced, but the program will continue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This environment variable is only used for programs compiled with the following options: -openmp or -openmp-profile or -parallel (Linux and Mac OS X) and</td>
</tr>
</tbody>
</table>

OMP_MAX_ACTIVE_LEVELS  | No enforced limit         |
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Default</th>
<th>Description and Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OMP THREAD LIMIT</td>
<td>No enforced limit</td>
</tr>
</tbody>
</table>

/Oopenmp or /Oopenmp-profile or /Oparallel (Windows).

omp_get_thread_limit() routine returns the value of the limit.

Related env variable: KMP_ALL_THREADS. OMP THREAD LIMIT overrides KMP_ALL_THREADS.

Example syntax:
export OMP THREAD LIMIT=value

Limits the number of simultaneously executing threads in an OpenMP* program.

If this limit is reached and another native operating system thread encounters OpenMP* API calls or constructs, the program can abort with an error message. If this limit is reached when an OpenMP parallel region begins, a one-time warning message might be generated indicating that the number of threads in the team was reduced, but the program will continue.
This environment variable is only used for programs compiled with the following options: -openmp or -openmp-profile or -parallel (Linux and Mac OS X) and /Qopenmp or /Qopenmp-profile or /Qparallel (Windows).

omp_get_thread_limit() routine returns the value of the limit.

Related environment variable: KMP_ALL_THREADS. Its value overrides OMP_THREAD_LIMIT.

Example syntax:
export OMP_THREAD_LIMIT=value

### Intel Environment Variables Extensions

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP_ALL_THREADS</td>
<td>No enforced limit</td>
<td>Limits the number of simultaneously executing threads in an OpenMP* program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If this limit is reached and another native operating system thread encounters OpenMP* API calls or constructs, the program can</td>
</tr>
</tbody>
</table>
### Description

Abort with an error message. If this limit is reached when an OpenMP parallel region begins, a one-time warning message might be generated indicating that the number of threads in the team was reduced, but the program will continue.

This environment variable is only used for programs compiled with the following options: `-openmp` or `-openmp-profile` (Linux and Mac OS X) and `/Qopenmp` or `/Qopenmp-profile` (Windows).

Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.

Use the optional character suffixes: `s` (seconds), `m` (minutes), `h` (hours), or `d` (days) to specify the units.

Specify `infinite` for an unlimited wait time.

See also the throughput execution mode and the `KMP_LIBRARY` environment variable.

### Variable Name | Default | Description
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP_BLOCKTIME</td>
<td>200 milliseconds</td>
<td>Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping. Use the optional character suffixes: <code>s</code> (seconds), <code>m</code> (minutes), <code>h</code> (hours), or <code>d</code> (days) to specify the units. Specify <code>infinite</code> for an unlimited wait time. See also the throughput execution mode and the <code>KMP_LIBRARY</code> environment variable.</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KMP_LIBRARY</td>
<td>throughput</td>
<td>Selects the OpenMP run-time library execution mode. The options for the variable value are throughput, turnaround, and serial.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enables (1) or disables (0) the printing OpenMP run-time library environment variables during program execution.</td>
</tr>
<tr>
<td>KMP_SETTINGS</td>
<td>0</td>
<td>Two lists of variables are printed: user-defined environment variables settings and effective values of variables used by OpenMP run-time library.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sets the number of bytes to allocate for each OpenMP* thread to use as the private stack for the thread.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recommended size is 16m.</td>
</tr>
<tr>
<td>KMP_STACKSIZE</td>
<td>IA-32 architecture: 2m, Intel® 64 and IA-64 architectures: 4m</td>
<td>Use the optional suffixes: b (bytes), k (kilobytes), m (megabytes), g (gigabytes), or t (terabytes) to specify the units.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This variable does not affect the native operating system threads created by the user program nor the thread executing the sequential part of an OpenMP* program or parallel programs created</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KMP_MONITOR_STACKSIZE</td>
<td>max (32k, system minimum thread stack size)</td>
<td>using <code>-parallel</code> (Linux and Mac OS X) or <code>/parallell</code> (Windows). Sets the number of bytes to allocate for the monitor thread, which is used for book-keeping during program execution. Use the optional suffixes: <code>b</code> (bytes), <code>k</code> (kilobytes), <code>m</code> (megabytes), <code>g</code> (gigabytes), or <code>t</code> (terabytes) to specify the units.</td>
</tr>
<tr>
<td>KMP_VERSION</td>
<td>0</td>
<td>Enables (1) or disables (0) the printing of OpenMP run-time library version information during program execution.</td>
</tr>
<tr>
<td>KMP_AFFINITY</td>
<td>noverbose, respect, granularity=core</td>
<td>Enables run-time library to bind threads to physical processing units. See Thread Affinity Interface for more information on the default and the affect this environment variable has on the parallel environment.</td>
</tr>
<tr>
<td>KMP_CPUINFO_FILE</td>
<td>none</td>
<td>Specifies an alternate file name for file containing machine topology description. The file must be in the same format as <code>/proc/cpuinfo</code>.</td>
</tr>
</tbody>
</table>

GNU Environment Variables Extensions
These environment variables are GNU extensions. They are recognized by the Intel OpenMP compatibility library.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOMP_STACKSIZE</td>
<td>See OMP_STACKSIZE description</td>
<td>OMP_STACKSIZE overrides GOMP_STACKSIZE. KMP_STACKSIZE overrides OMP_STACKSIZE and GOMP_STACKSIZE</td>
</tr>
<tr>
<td>GOMP_CPU_AFFINITY</td>
<td>TBD</td>
<td></td>
</tr>
</tbody>
</table>

**OpenMP* Directives and Clauses Summary**

This is a summary of the OpenMP* pragmas and clauses supported in the Intel® Compiler. For detailed information about the OpenMP API, see the OpenMP Application Program Interface Version 2.5 specification, which is available from the OpenMP web site (http://www.openmp.org/).

**OpenMP Pragmas**

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel</td>
<td>Defines a parallel region.</td>
</tr>
<tr>
<td>task</td>
<td>Indicates that the associated structured block should be executed in parallel as part of the enclosing parallel construct.</td>
</tr>
<tr>
<td>sections</td>
<td>Identifies a non-iterative worksharing construct that specifies a set of structured blocks that are to be divided among threads in a team.</td>
</tr>
<tr>
<td>section</td>
<td>Indicates that the associated structured block should be executed in parallel as part of the enclosing sections construct.</td>
</tr>
<tr>
<td>single</td>
<td>Identifies a construct that specifies that the associated structured block is executed by only one thread in the team.</td>
</tr>
<tr>
<td>parallel for</td>
<td>A shortcut for a parallel region that contains a single for directive.</td>
</tr>
</tbody>
</table>
###Pragma

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTE.</strong></td>
<td>The parallel or for OpenMP directive must be immediately followed by a for statement. If you place other statement or an OpenMP directive between the parallel or for directive and the for statement, the Intel C++ Compiler issues a syntax error.</td>
</tr>
<tr>
<td>for</td>
<td>Identifies an iterative work-sharing construct that specifies a region in which the iterations of the associated loop should be executed in parallel. Each iteration is executed by one of the threads in the team.</td>
</tr>
<tr>
<td>parallel sections</td>
<td>Provides a shortcut form for specifying a parallel region containing a single sections construct.</td>
</tr>
<tr>
<td>master</td>
<td>Identifies a construct that specifies a structured block that is executed by only the master thread of the team.</td>
</tr>
<tr>
<td>critical[name]</td>
<td>Identifies a construct that restricts execution of the associated structured block to a single thread at a time. Each thread waits at the beginning of the critical construct until no other thread is executing a critical construct with the same name argument.</td>
</tr>
<tr>
<td>taskwait</td>
<td>Indicates a wait on the completion of child tasks generated since the beginning of the current task.</td>
</tr>
<tr>
<td>barrier</td>
<td>Synchronizes all the threads in a team. Each thread waits until all of the other threads in that team have reached this point.</td>
</tr>
<tr>
<td>atomic</td>
<td>Ensures that a specific memory location is updated atomically, rather than exposing it to the possibility of multiple, simultaneously writing threads.</td>
</tr>
<tr>
<td>flush [(list)]</td>
<td>Specifies a cross-thread sequence point at which the implementation is required to ensure that all the threads in a team have a consistent view of certain objects in memory. The optional list argument consists of a comma-separated list of variables to be flushed.</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ordered</td>
<td>The structured block following this directive is executed in the order in which iterations would be executed in a sequential loop.</td>
</tr>
<tr>
<td>threadprivate (list)</td>
<td>Makes the named file-scoped, namespace-scoped, or static block-scoped variables specified private to a thread.</td>
</tr>
</tbody>
</table>

**OpenMP Clauses**

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>Declares variables to be private to each thread in a team. Private copies of the variable are initialized from the original object when entering the region.</td>
</tr>
<tr>
<td>firstprivate</td>
<td>Provides a superset of the functionality provided by the private clause.</td>
</tr>
<tr>
<td>lastprivate</td>
<td>Provides a superset of the functionality provided by the private clause. The original object is updated with the value of the private copy from the last sequential iteration of the associated loop, or the lexically last section construct, when exiting the region.</td>
</tr>
<tr>
<td>shared</td>
<td>Shares variables among all the threads in a team.</td>
</tr>
<tr>
<td>default</td>
<td>Enables you to affect the data-scope attributes of variables.</td>
</tr>
<tr>
<td>reduction</td>
<td>Performs a reduction on scalar variables.</td>
</tr>
<tr>
<td>ordered</td>
<td>The structured block following an ordered directive is executed in the order in which iterations would be executed in a sequential loop.</td>
</tr>
<tr>
<td>if (expression)</td>
<td>If the if(expression) clause is present, the enclosed code block is executed in parallel only if the expression evaluates to TRUE. Otherwise the code block is serialized.</td>
</tr>
<tr>
<td></td>
<td>The expression must be scalar logical.</td>
</tr>
<tr>
<td>Clause</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>schedule</td>
<td>Specifies how iterations of the for loop are divided among the threads of the team.</td>
</tr>
<tr>
<td>collapse(n)</td>
<td>Specifies how many loops are associated with the OpenMP loop construct for collapsing.</td>
</tr>
<tr>
<td>copyin</td>
<td>Provides a mechanism to copy the data values of the master thread to the variables used by the threadprivate copies at the beginning of the parallel region.</td>
</tr>
<tr>
<td>copyprivate</td>
<td>Provides a mechanism to use a private variable to broadcast a value from the data environment of one implicit task to the data environments of the other implicit tasks belonging to the parallel region.</td>
</tr>
<tr>
<td>nowait</td>
<td>Indicates that an implementation may omit the barrier at the end of the worksharing region.</td>
</tr>
<tr>
<td>untied</td>
<td>Indicates that a resumed task does not have to be executed by same thread executing it before it was suspended.</td>
</tr>
</tbody>
</table>

**OpenMP* Library Support**

**OpenMP* Run-time Library Routines**

OpenMP* provides several run-time library routines to help you manage your program in parallel mode. Many of these run-time library routines have corresponding environment variables that can be set as defaults. The run-time library routines let you dynamically change these factors to assist in controlling your program. In all cases, a call to a run-time library routine overrides any corresponding environment variable.

This topic provides a summary of the OpenMP run-time library routines. See [OpenMP* Support Overview](#) for additional resources; refer to the OpenMP API Version 3.0 specification for detailed information about using these routines.
Include the appropriate declarations of the routines by adding a statement similar to the following in your source code:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| include "omp_lib.h"

#include <omp.h>

The header files are provided in the ../include (Linux* and Mac OS* X) or ..\include (Windows*) directory of your compiler installation.

The following tables specify the interfaces to these routines. (The names for the routines are in user name space.)

**Execution Environment Routines**

Use these routines to monitor and influence threads and the parallel environment.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void omp_set_num_threads(int nthreads)</td>
<td>Sets the number of threads to use for subsequent parallel regions created by the calling thread.</td>
</tr>
<tr>
<td>int omp_get_num_threads(void)</td>
<td>Returns the number of threads that are being used in the current parallel region. This function does not return the value inherited by the calling thread from the omp_set_num_threads() function.</td>
</tr>
<tr>
<td>int omp_get_max_threads(void)</td>
<td>Returns the number of threads available to subsequent parallel regions created by the calling thread. This function returns the value inherited by the calling thread from the omp_set_num_threads() function.</td>
</tr>
<tr>
<td>int omp_get_thread_num(void)</td>
<td>Returns the thread number of the calling thread, within the context of the current parallel region..</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>int omp_get_num_procs(void)</td>
<td>Returns the number of processors available to the program.</td>
</tr>
<tr>
<td>int omp_in_parallel(void)</td>
<td>Returns TRUE (non zero) if called within the dynamic extent of a parallel region executing in parallel; otherwise returns FALSE (an integer value of zero).</td>
</tr>
<tr>
<td>void omp_set_dynamic(int dynamic_threads)</td>
<td>Enables or disables dynamic adjustment of the number of threads used to execute a parallel region. If dynamic_threads is TRUE, dynamic threads are enabled. If dynamic_threads is FALSE, dynamic threads are disabled. Dynamics threads are disabled by default.</td>
</tr>
<tr>
<td>int omp_get_dynamic(void)</td>
<td>Returns TRUE if dynamic thread adjustment is enabled, otherwise returns FALSE.</td>
</tr>
<tr>
<td>void omp_set_nested(int nested)</td>
<td>Enables or disables nested parallelism. If nested is TRUE, nested parallelism is enabled. If nested is FALSE, nested parallelism is disabled. Nested parallelism is disabled by default.</td>
</tr>
<tr>
<td>int omp_get_nested(void)</td>
<td>Returns TRUE if nested parallelism is enabled, otherwise returns FALSE.</td>
</tr>
<tr>
<td>void omp_set_schedule(omp_sched_t kind, int modifier)</td>
<td>Determines the schedule of a worksharing loop that is applied when 'runtime' is used as schedule kind.</td>
</tr>
<tr>
<td>void omp_get_schedule(omp_sched_t *kind, int *modifier)</td>
<td>Returns the schedule of a worksharing loop that is applied when the 'runtime' schedule is used.</td>
</tr>
</tbody>
</table>
### Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int omp_get_thread_limit(void)</code></td>
<td>Returns the maximum number of simultaneously executing threads in an OpenMP* program.</td>
</tr>
<tr>
<td><code>void omp_set_max_active_levels(int max_active_levels)</code></td>
<td>Limits the number of nested active parallel regions. The call is ignored if negative <code>max_active_levels</code> specified.</td>
</tr>
<tr>
<td><code>int omp_get_max_active_levels(void)</code></td>
<td>Returns the maximum number of nested active parallel regions.</td>
</tr>
<tr>
<td><code>int omp_get_active_level(void)</code></td>
<td>Returns the number of nested, active parallel regions enclosing the task that contains the call.</td>
</tr>
<tr>
<td><code>int omp_get_level(void)</code></td>
<td>Returns the number of nested parallel regions (whether active or inactive) enclosing the task that contains the call, not including the implicit parallel region.</td>
</tr>
<tr>
<td><code>int omp_get_ancestor_thread_num(int level)</code></td>
<td>Returns the thread number of the ancestor at a given nest level of the current thread.</td>
</tr>
<tr>
<td><code>int omp_get_team_size(int level)</code></td>
<td>Returns the size of the thread team to which the ancestor belongs.</td>
</tr>
</tbody>
</table>

### Lock Routines

Use these routines to affect OpenMP locks.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void omp_init_lock(omp_lock_t lock)</code></td>
<td>Initializes the lock associated with <code>lock</code> for use in subsequent calls.</td>
</tr>
<tr>
<td><code>void omp_destroy_lock(omp_lock_t lock)</code></td>
<td>Causes the lock specified by <code>lock</code> to become undefined or uninitialized.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>void omp_set_lock(omp_lock_t lock)</code></td>
<td>Forces the executing thread to wait until the lock associated with <code>lock</code> is available. The thread is granted ownership of the lock when it becomes available.</td>
</tr>
<tr>
<td><code>void omp_unset_lock(omp_lock_t lock)</code></td>
<td>Releases the executing thread from ownership of the lock associated with <code>lock</code>. The behavior is undefined if the executing thread does not own the lock associated with <code>lock</code>.</td>
</tr>
<tr>
<td><code>int omp_test_lock(omp_lock_t lock)</code></td>
<td>Attempts to set the lock associated with <code>lock</code>. If successful, returns TRUE, otherwise returns FALSE.</td>
</tr>
<tr>
<td><code>void omp_init_nest_lock(omp_nest_lock_t lock)</code></td>
<td>Initializes the nested lock associated with <code>lock</code> for use in the subsequent calls.</td>
</tr>
<tr>
<td><code>void omp_destroy_nest_lock(omp_nest_lock_t lock)</code></td>
<td>Causes the nested lock associated with <code>lock</code> to become undefined or uninitialized.</td>
</tr>
<tr>
<td><code>void omp_set_nest_lock(omp_nest_lock_t lock)</code></td>
<td>Forces the executing thread to wait until the nested lock associated with <code>lock</code> is available. The thread is granted ownership of the nested lock when it becomes available. If the thread already owns the lock, then the lock nesting count is incremented.</td>
</tr>
<tr>
<td><code>void omp_unset_nest_lock(omp_nest_lock_t lock)</code></td>
<td>Releases the executing thread from ownership of the nested lock associated with <code>lock</code> if the nesting count is zero; otherwise, the nesting count is decremented. Behavior is undefined if the executing thread does not own the nested lock associated with <code>lock</code>.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>int omp_test_nest_lock(omp_nest_lock_t lock)</td>
<td>Attempts to set the nested lock specified by lock. If successful, returns the nesting count, otherwise returns zero.</td>
</tr>
</tbody>
</table>

### Timing Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double omp_get_wtime(void)</td>
<td>Returns a double precision value equal to the elapsed wall clock time (in seconds) relative to an arbitrary reference time. The reference time does not change during program execution.</td>
</tr>
<tr>
<td>double omp_get_wtick(void)</td>
<td>Returns a double precision value equal to the number of seconds between successive clock ticks.</td>
</tr>
</tbody>
</table>

### Intel Extension Routines to OpenMP*

The Intel® Compiler implements the following group of routines as an extensions to the OpenMP* run-time library:

- Getting and setting the execution environment
- Getting and setting stack size for parallel threads
- Memory allocation
- Getting and setting thread sleep time for the throughput execution mode

The Intel extension routines described in this section can be used for low-level tuning to verify that the library code and application are functioning as intended. These routines are generally not recognized by other OpenMP-compliant compilers, which may cause the link stage to fail in other compiler. These OpenMP routines require that you use the `-openmp-stubs` (Linux* and Mac OS* X) or `/Qopenmp-stubs` (Windows*) command-line option to execute.

See [OpenMP* Run-time Library Routines](#) for details about including support for these declarations in your source, and see [OpenMP* Support Libraries](#) for detailed information about execution environment (mode).
In most cases, environment variables can be used in place of the extension library routines. For example, the stack size of the parallel threads may be set using the OMP_STACKSIZE environment variable rather than the kmp_set_stacksize_s() library routine.

**NOTE.** A run-time call to an Intel extension routine takes precedence over the corresponding environment variable setting.

### Execution Environment Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void kmp_set_defaults(char const *)</code></td>
<td>Sets OpenMP environment variables defined as a list of variables separated by &quot;</td>
</tr>
<tr>
<td><code>void kmp_set_library_throughput()</code></td>
<td>Sets execution mode to throughput, which is the default. Allows the application to determine the runtime environment. Use in multi-user environments.</td>
</tr>
<tr>
<td><code>void kmp_set_library_turnaround()</code></td>
<td>Sets execution mode to turnaround. Use in dedicated parallel (single user) environments.</td>
</tr>
<tr>
<td><code>void kmp_set_library_serial()</code></td>
<td>Sets execution mode to serial.</td>
</tr>
</tbody>
</table>
| `void kmp_set_library(int)` | Sets execution mode indicated by the value passed to the function. Valid values are:  
  - 1 - serial mode  
  - 2 - turnaround mode  
  - 3 - throughput mode  
  Call this routine before the first parallel region is executed. |
| `int kmp_get_library()` | Returns a value corresponding to the current execution mode: 1 (serial), 2 (turnaround), or 3 (throughput). |
Stack Size

For IA-64 architecture it is recommended to always use kmp_set_stacksize_s() and kmp_get_stacksize_s(). The s() variants must be used if you need to set a stack size ≥ 2**31 bytes (2 gigabytes).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t kmp_get_stacksize_s()</td>
<td>Returns the number of bytes that will be allocated for each parallel thread to use as its private stack. This value can be changed with kmp_set_stacksize_s() routine, prior to the first parallel region or via the KMP_STACKSIZE environment variable.</td>
</tr>
<tr>
<td>int kmp_get_stacksize()</td>
<td>Provided for backwards compatibility only. Use kmp_get_stacksize_s() routine for compatibility across different families of Intel processors.</td>
</tr>
<tr>
<td>void kmp_set_stacksize_s(size_t size)</td>
<td>Sets to size the number of bytes that will be allocated for each parallel thread to use as its private stack. This value can also be set via the KMP_STACKSIZE environment variable. In order for to have an effect, it must be called before the beginning of the first (dynamically executed) parallel region in the program.</td>
</tr>
<tr>
<td>void kmp_set_stacksize(int size)</td>
<td>Provided for backward compatibility only. Use kmp_set_stacksize_s() for compatibility across different families of Intel processors.</td>
</tr>
</tbody>
</table>

Memory Allocation

The Intel® compiler implements a group of memory allocation routines as an extension to the OpenMP* run-time library to enable threads to allocate memory from a heap local to each thread. These routines are: kmp_malloc(), kmp_calloc(), and kmp_realloc().
The memory allocated by these routines must also be freed by the `kmp_free()` routine. While it is legal for the memory to be allocated by one thread and freed by a different thread, this mode of operation has a slight performance penalty.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void* kmp_malloc(size_t size)</code></td>
<td>Allocate memory block of size bytes from thread-local heap.</td>
</tr>
<tr>
<td><code>void* kmp_calloc(size_t nelem, size_t elsize)</code></td>
<td>Allocate array of nelem elements of size elsize from thread-local heap.</td>
</tr>
<tr>
<td><code>void* kmp_realloc(void* ptr, size_t size)</code></td>
<td>Reallocate memory block at address ptr and size bytes from thread-local heap.</td>
</tr>
<tr>
<td><code>void* kmp_free(void* ptr)</code></td>
<td>Free memory block at address ptr from thread-local heap.</td>
</tr>
<tr>
<td></td>
<td>Memory must have been previously allocated with kmp_malloc(), kmp_calloc(), or kmp_realloc().</td>
</tr>
</tbody>
</table>

**Thread Sleep Time**

In the throughput execution mode, threads wait for new parallel work at the ends of parallel regions, and then sleep, after a specified period of time. This time interval can be set by the `KMP_BLOCKTIME` environment variable or by the `kmp_set_blocktime()` function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int kmp_get_blocktime(void)</code></td>
<td>Returns the number of milliseconds that a thread should wait, after completing the execution of a parallel region, before sleeping, as set either by the KMP_BLOCKTIME environment variable or by kmp_set_blocktime().</td>
</tr>
<tr>
<td><code>void kmp_set_blocktime(int msec)</code></td>
<td>Sets the number of milliseconds that a thread should wait, after completing the execution of a parallel region, before sleeping. This routine affects the block time setting for the thread.</td>
</tr>
</tbody>
</table>
Function | Description
--- | ---
 | calling thread and any OpenMP team threads formed by the calling thread. The routine does not affect the block time for any other threads.

**OpenMP* Support Libraries**

The Intel® Compiler provides support libraries for OpenMP*. There are several kinds of libraries:

- **Performance**: supports parallel OpenMP execution.
- **Profile**: supports parallel OpenMP execution and allows use of Intel® Thread Profiler.
- **Stubs**: supports serial execution of OpenMP applications.

Each kind of library is available for both dynamic and static linking.

**NOTE.** The use of static OpenMP libraries is not recommended, because they might cause multiple libraries to be linked in an application. The condition is not supported and could lead to unpredictable results.

This section describes the compatibility libraries and legacy libraries provided with the Intel compiler, as well as the selection of run-time execution modes.

**Compatibility Libraries**

To use the Compatibility OpenMP libraries, specify the (default) `/Qopenmp-lib:compat` (Windows OS) or `-openmp-lib compat` (Linux OS and Mac OS X) compiler option during linking.

On Linux and Mac OS X systems, to use dynamically linked libraries during linking, specify `-openmp-link=dynamic` option; to use static linking, specify the `-openmp-link=static` option.

On Windows systems, to use dynamically linked libraries during linking, specify the `/MD` and `/Qopenmp-link:dyanmic` options; to use static linking, specify the `/MT` and `/Qopenmp-link:static` options.

To provide run-time support for dynamically linked applications, the supplied DLL (Windows OS) or shared library (Linux OS and Mac OS X) must be available to the application at run time.

**Performance Libraries**
To use these libraries, specify the `-openmp` (Linux* and Mac OS* X) or `/Qopenmp` (Windows*) compiler option.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Dynamic Link</th>
<th>Static Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>libiomp5.so</td>
<td>libiomp5.a</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>libiomp5.dylib</td>
<td>libiomp5.a</td>
</tr>
<tr>
<td>Windows</td>
<td>libiomp5md.lib</td>
<td>libiomp5mt.lib</td>
</tr>
</tbody>
</table>

**Profile Libraries**

To use these libraries, specify `-openmp-profile` (Linux* and Mac OS* X) or `/Qopenmp-profile` (Windows*) compiler option. These allow you to use Intel® Thread Profiler to analyze OpenMP applications.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Dynamic Link</th>
<th>Static Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>libiompprof5.so</td>
<td>libiompprof5.a</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>libiompprof5.dylib</td>
<td>libiompprof5.a</td>
</tr>
<tr>
<td>Windows</td>
<td>libiompprof5md.lib</td>
<td>libiompprof5mt.lib</td>
</tr>
</tbody>
</table>

**Stubs Libraries**

To use these libraries, specify `-openmp-stubs` (Linux* and Mac OS* X) or `/Qopenmp-stubs` (Windows*) compiler option. These allow you to compile OpenMP applications in serial mode and provide stubs for OpenMP routines and extended Intel-specific routines.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Dynamic Link</th>
<th>Static Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>libiompstubs5.so</td>
<td>libiompstubs5.a</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>libiompstubs5.dylib</td>
<td>libiompstubs5.a</td>
</tr>
<tr>
<td>Windows</td>
<td>libiompstubs5md.lib</td>
<td>libiompstubs5mt.lib</td>
</tr>
</tbody>
</table>
Legacy Libraries

To use the Legacy OpenMP libraries, specify the /Qopenmp-lib:legacy (Windows OS) or -openmp-lib legacy (Linux OS and Mac OS X) compiler options during linking. Legacy libraries are deprecated.

On Linux and Mac OS X systems, to use dynamically linked libraries during linking, specify the -openmp-link:dynamic option; to use static linking, specify the -openmp-link:static option.

On Windows systems, to use dynamically linked libraries during linking, specify the /MD and /Qopenmp-link=dynamic options; to use static linking, specify the /MT and /Qopenmp-link=static options.

To provide run-time support for dynamically linked applications, the supplied DLL (Windows OS) or shared library (Linux OS and Mac OS X) must be available to the application at run time.

Performance Libraries

To use these libraries, specify -openmp (Linux* and Mac OS* X) or /Qopenmp (Windows*) compiler option.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Dynamic Link</th>
<th>Static Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>libguide.so</td>
<td>libguide.a</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>libguide.dylib</td>
<td>libguide.a</td>
</tr>
<tr>
<td>Windows</td>
<td>libguide40.lib</td>
<td>libguide.lib</td>
</tr>
</tbody>
</table>

Profile Libraries

To use these libraries, specify -openmp-profile (Linux* and Mac OS* X) or /Qopenmp-profile (Windows*) compiler option. These allow you to use Intel® Thread Profiler to analyze OpenMP applications.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Dynamic Link</th>
<th>Static Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>libguide_stats.so</td>
<td>libguide_stats.a</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>libguide_stats.dylib</td>
<td>libguide_stats.a</td>
</tr>
<tr>
<td>Operating System</td>
<td>Dynamic Link</td>
<td>Static Link</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Windows</td>
<td>libguide40_stats.lib</td>
<td>libguide_stats.lib</td>
</tr>
<tr>
<td></td>
<td>libguide40_stats.dll</td>
<td></td>
</tr>
</tbody>
</table>

**Stubs Libraries**

To use these libraries, specify `-openmp-stubs` (Linux* and Mac OS* X) or `/Qopenmp-stubs` (Windows*) compiler option. These allow you to compile OpenMP applications in serial mode and provide stubs for OpenMP routines and extended Intel-specific routines.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Dynamic Link</th>
<th>Static Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>libompstub.so</td>
<td>libompstub.a</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>libompstub.dylib</td>
<td>libompstub.a</td>
</tr>
<tr>
<td>Windows</td>
<td>libompstub40.lib</td>
<td>libompstub.lib</td>
</tr>
<tr>
<td></td>
<td>libompstub40.dll</td>
<td></td>
</tr>
</tbody>
</table>

**Execution modes**

The Intel compiler enables you to run an application under different execution modes specified at run time; the libraries support the turnaround, throughput, and serial modes. Use the `KMP_LIBRARY` environment variable to select the modes at run time.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>The throughput mode allows the program to detect its environment conditions (system load) and adjust resource usage to produce efficient execution in a dynamic environment. In a multi-user environment where the load on the parallel machine is not constant or where the job stream is not predictable, it may be better to design and tune for throughput. This minimizes the total time to run multiple jobs simultaneously. In this mode, the worker threads yield to other threads while waiting for more parallel work.</td>
</tr>
</tbody>
</table>
After completing the execution of a parallel region, threads wait for new parallel work to become available. After a certain period of time has elapsed, they stop waiting and sleep. Until more parallel work becomes available, sleeping allows processor and resources to be used for other work by non-OpenMP threaded code that may execute between parallel regions, or by other applications.

The amount of time to wait before sleeping is set either by the `KMP_BLOCKTIME` environment variable or by the `kmp_set_blocktime()` function. A small blocktime value may offer better overall performance if your application contains non-OpenMP threaded code that executes between parallel regions. A larger blocktime value may be more appropriate if threads are to be reserved solely for use for OpenMP execution, but may penalize other concurrently-running OpenMP or threaded applications.

The turnaround mode is designed to keep active all processors involved in the parallel computation, which minimizes execution time of a single job. In this mode, the worker threads actively wait for more parallel work, without yielding to other threads. In a dedicated (batch or single user) parallel environment where all processors are exclusively allocated to the program for its entire run, it is most important to effectively use all processors all of the time.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>turnaround</td>
<td>After completing the execution of a parallel region, threads wait for new parallel work to become available. After a certain period of time has elapsed, they stop waiting and sleep. Until more parallel work becomes available, sleeping allows processor and resources to be used for other work by non-OpenMP threaded code that may execute between parallel regions, or by other applications. The amount of time to wait before sleeping is set either by the <code>KMP_BLOCKTIME</code> environment variable or by the <code>kmp_set_blocktime()</code> function. A small blocktime value may offer better overall performance if your application contains non-OpenMP threaded code that executes between parallel regions. A larger blocktime value may be more appropriate if threads are to be reserved solely for use for OpenMP execution, but may penalize other concurrently-running OpenMP or threaded applications. The turnaround mode is designed to keep active all processors involved in the parallel computation, which minimizes execution time of a single job. In this mode, the worker threads actively wait for more parallel work, without yielding to other threads. In a dedicated (batch or single user) parallel environment where all processors are exclusively allocated to the program for its entire run, it is most important to effectively use all processors all of the time.</td>
</tr>
</tbody>
</table>
Avoid over-allocating system resources. The condition can occur if either too many threads have been specified, or if too few processors are available at run time. If system resources are over-allocated, this mode will cause poor performance. The throughput mode should be used instead if this occurs.

### Using the OpenMP Compatibility Libraries

This section describes the steps needed to set up and use the OpenMP Compatibility Libraries from the command line. On Windows® systems, you can also build applications compiled with the OpenMP Compatibility libraries in the Microsoft Visual Studio® development environment.

For a summary of the support provided by the Compatibility and Legacy libraries provided with Intel compilers, see OpenMP® Source Compatibility and Interoperability with Other Compilers.

For a list of the options and libraries used by the OpenMP libraries, see OpenMP Support Libraries.

Set up your environment for access to the Intel compiler to ensure that the appropriate OpenMP library is available during linking. On Windows systems, you can either execute the appropriate batch (.bat) file or use the command-line window supplied in the compiler program folder that already has the environment set up. On Linux and Mac OS systems, you can source the appropriate script file (see Invoking the Compiler from the Command Line. Using the ifortvars File to Specify Location of Components.

During C/C++ compilation, ensure the version of `omp.h` used when compiling is the version provided by that compiler. For example, on Linux systems when compiling with the GNU C/C++ compiler, use the `omp.h` provided with the GNU C/C++ compiler. Similarly, during Fortran compilation, ensure that the version of `omp_lib.h` or `omp_lib.mod` used when compiling is the version provided by that compiler.

The following table lists the commands used by the various command-line compilers for both C and C++ source files:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>The serial mode forces parallel applications to run as a single thread.</td>
</tr>
</tbody>
</table>
The command for the Intel® Fortran compiler is ifort on Linux, Mac OS X, and Windows operating systems.

For information on the OpenMP libraries and options used by the Intel compiler, see OpenMP Support Libraries.

**Command-Line Examples, Windows OS**

On Windows systems, to use the Compatibility Libraries with Microsoft Visual C++ in the Microsoft Visual Studio* environment, see Using the OpenMP Compatibility Libraries from Visual Studio*.

To compile and link (build) the entire application with one command using the Compatibility libraries, specify the following Intel compiler command:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source, dynamic link</td>
<td>icl /MD /Qopenmp /Qopenmp-lib:compat hello.c</td>
</tr>
<tr>
<td>C++ source, dynamic link</td>
<td>icl /MD /Qopenmp /Qopenmp-lib:compat hello.cpp</td>
</tr>
</tbody>
</table>

By default, the Intel compilers perform a dynamic link of the OpenMP libraries. To perform a static link (not recommended), use the /MT option in place of the /MD option above and add the option /Qopenmp-link:static. The Intel compiler option /Qopenmp-link controls whether the linker uses static or dynamic OpenMP libraries on Windows OS systems (default is /Qopenmp-link:dynamic).

When using Microsoft Visual C++ compiler, you should link with the Intel OpenMP compatibility library. You need to avoid linking the Microsoft OpenMP run-time library (vcomp) and explicitly pass the name of the Intel OpenMP compatibility library as linker options (following /link):

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source, dynamic link</td>
<td>icl /MD /openmp hello.c /link /nodefaultlib:vcomp libiomp5md.lib</td>
</tr>
<tr>
<td>Type of File</td>
<td>Commands</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>C++ source, dynamic link</td>
<td><code>icl /MD /openmp hello.cpp /link /nodefaultlib:vcomp libiomp5md.lib</code></td>
</tr>
</tbody>
</table>

Performing a static link is not recommended, but would require use of the `/MT` option in place of the `/MD` option above.

You can also use both Intel C++ and Visual C++ compilers to compile parts of the application and create object files (object-level interoperability). In this example, the Intel compiler links the entire application:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
</table>
| C source, dynamic link | `icl /MD /openmp hello.cpp /c f1.c f2.c`  
`icl /MD /Qopenmp /Qopenmp-lib:compat /c f3.c f4.c`  
`icl /MD /Qopenmp /Qopenmp-lib:compat f1.obj f2.obj`  
`f3.obj f4.obj /Feapp /link /nodefaultlib:vcomp` |

The first command produces two object files compiled by Visual C++ compiler, and the second command produces two more object files compiled by Intel C++ Compiler. The final command links all four object files into an application.

Alternatively, the third line below uses the Visual C++ linker to link the application and specifies the Compatibility library `libiomp5md.lib` at the end of the third command:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
</table>
| C source, dynamic link | `icl /MD /openmp hello.cpp /c f1.c f2.c`  
`icl /MD /Qopenmp /Qopenmp-lib:compat /c f3.c f4.c`  
`link f1.obj f2.obj f3.obj f4.obj /out:app.exe /nodefaultlib:vcomp libiomp5md.lib` |

The following example shows the use of interprocedural optimization by the Intel compiler on several files, the Visual C++ compiler compiles several files, and the Visual C++ linker links the object files to create the executable:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
</table>
| C source, dynamic link | `icl /Qopenmp /Qopenmp-lib:compat /O3 /Qipo /Qipo-c f1.c f2.c f3.c`  
`cl /MD /openmp /O2 /c f4.c f5.c` |
The first command uses the Intel® C++ Compiler to produce an optimized multi-file object file named `ipo_out.obj` by default (the `/Fe` option is not required; see Using IPO). The second command uses the Visual C++ `cl` command to link all three object files using the Intel compiler OpenMP Compatibility library. Performing a static link (not recommended) requires use of the `/MT` option in place of the `/MD` option in each line above.

**Command-Line Examples, Linux OS and Mac OS X**

To compile and link (build) the entire application with one command using the Compatibility libraries, specify the following Intel compiler command:

**Type of File** | **Commands**
--- | ---
C source | `icc -openmp -openmp-lib=compat hello.c`
C++ source | `icpc -openmp -openmp-lib=compat hello.cpp`

By default, the Intel compilers perform a dynamic link of the OpenMP libraries. To perform a static link (not recommended), add the option `-openmp-link static`. The Intel compiler option `-openmp-link` controls whether the linker uses static or dynamic OpenMP libraries on Linux OS and Mac OS X systems (default is `-openmp-link dynamic`).

You can also use both Intel C++ `icc/icpc` and GNU `gcc/g++` compilers to compile parts of the application and create object files (object-level interoperability). In this example, the GNU compiler compiles the C file `foo.c` (the `gcc` option `-fopenmp` enables OpenMP support), and the Intel compiler links the application using the OpenMP Compatibility library:

**Type of File** | **Commands**
--- | ---
C source | `gcc -fopenmp -c foo.c`
 | `icc -openmp -openmp-lib=compat foo.o`
C++ source | `g++ -fopenmp -c foo.cpp`
 | `icpc -openmp -openmp-lib=compat foo.o`
When using GNU gcc or g++ compiler to link the application with the Intel compiler OpenMP compatibility library, you need to explicitly pass the Intel OpenMP compatibility library name using the \texttt{-l} option, the GNU pthread library using the \texttt{-l} option, and path to the Intel libraries where the Intel C++ compiler is installed using the \texttt{-L} option:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
</table>
| C source     | gcc -fopenmp -c foo.c bar.c  
gcc foo.o bar.o -liomp5 -lpthread -L<icc_dir>/lib |

You can mix object files, but it is easier to use the Intel compiler to link the application so you do not need to specify the \texttt{gcc -l} option, \texttt{-l} option, and the \texttt{-lpthread} option:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
</table>
| C source     | gcc -fopenmp -c foo.c  
icc -openmp -c bar.c  
icc -openmp -openmp=compat foo.o bar.o |

You can mix OpenMP object files compiled with the GNU gcc compiler, the Intel\textsuperscript{®} C++ Compiler, and the Intel\textsuperscript{®} Fortran Compiler. This example uses use the Intel Fortran Compiler to link all the objects:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
</table>
| Mixed C and Fortran sources | ifort -openmp -c foo.f  
icc -openmp -c ibar.c  
gcc -fopenmp -c gbar.c  
ifort -openmp -openmp-lib=compat foo.o ibar.o gbar.o |

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
</table>
| Mixed C and GNU Fortran sources | gfortran -fopenmp -c foo.f  
icc -openmp -c ibar.c  
gcc -fopenmp -c gbar.c |
Alternatively, you could use the Intel compiler to link the application, but need to pass multiple gfortran libraries using the -l options on the link line:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed C and Fortran sources</td>
<td>gfortran -fopenmp -c foo.f</td>
</tr>
<tr>
<td></td>
<td>icc -openmp -c ibar.c</td>
</tr>
<tr>
<td></td>
<td>icc -openmp -openmp-lib=compat foo.o bar.o</td>
</tr>
<tr>
<td></td>
<td>-lgfortranbegin -lgfortran</td>
</tr>
</tbody>
</table>

**Using the OpenMP Compatibility Libraries from Visual Studio**

When using systems running a Windows OS, you can make certain changes in the Visual C++ Visual Studio 2005 development environment to allow you to use the Intel C++ Compiler and Visual C++ to create applications that use the Intel compiler OpenMP Compatibility libraries.

**NOTE.** Microsoft Visual C++ must have the symbol _OPENMP_NOFORCE_MANIFEST defined or it will include the manifest for the vcomp90 dlls. While this may not appear to cause a problem on the build system, it will cause a problem when the application is moved to another system that does not have this DLL installed.

Set the project Property Pages to indicate the Intel OpenMP run-time library location:

1. Open the project’s property pages in from the main menu: Project > Properties (or right click the Project name and select Properties).
2. Select Configuration Properties > Linker > General > Additional Library Directories
3. Enter the path to the Intel compiler libraries. For example, for an IA-32 architecture system, enter: `<Intel_compiler_installation_path>\IA32\LIB`

Make the Intel OpenMP dynamic run-time library accessible at run-time, you must specify the corresponding path:

1. Open the project’s property pages in from the main menu: Project > Properties (or right click the Project name and select Properties).
2. Select Configuration Properties > Debugging > Environment
3. Enter the path to the Intel compiler libraries. For example, for an IA-32 architecture system, enter:

```
PATH=%PATH%;< Intel_compiler_installation_path>\IA32\Bin
```

Add the Intel OpenMP run-time library name to the linker options and exclude the default Microsoft OpenMP run-time library:

1. Open the project's property pages in from the main menu: Project > Properties (or right-click the Project name and select Properties).
2. Select Configuration Properties > Linker > Command Line > Additional Options
3. Enter the OpenMP library name and the Visual C++ linker option `/nodefaultlib`:

**Thread Affinity Interface (Linux* and Windows*)**

The Intel® runtime library has the ability to bind OpenMP threads to physical processing units. The interface is controlled using the `KMP_AFFINITY` environment variable. Depending on the system (machine) topology, application, and operating system, thread affinity can have a dramatic effect on the application speed.

*Thread affinity* restricts execution of certain threads (virtual execution units) to a subset of the physical processing units in a multiprocessor computer. Depending upon the topology of the machine, thread affinity can have a dramatic effect on the execution speed of a program.

Thread affinity is supported on Windows OS systems and versions of Linux OS systems that have kernel support for thread affinity, but is not supported by Mac OS* X. The thread affinity interface is supported only for Intel® processors.

The Intel compiler's OpenMP runtime library has the ability to bind OpenMP threads to physical processing units. There are three types of interfaces you can use to specify this binding, which are collectively referred to as the Intel OpenMP Thread Affinity Interface:

- The **high-level affinity interface** uses an environment variable to determine the machine topology and assigns OpenMP threads to the processors based upon their physical location in the machine. This interface is controlled entirely by the `KMP_AFFINITY` environment variable.

- The **mid-level affinity interface** uses an environment variable to explicitly specify which processors (labeled with integer IDs) are bound to OpenMP threads. This interface provides compatibility with the GNU gcc* GOMP_CPU_AFFINITY environment variable, but you can also invoke it by using the KMP_AFFINITY environment variable. The GOMP_CPU_AFFINITY environment variable is supported on Linux systems only, but users on Windows or Linux systems can use the similar functionality provided by the KMP_AFFINITY environment variable.

- The **low-level affinity interface** uses APIs to enable OpenMP threads to make calls into the OpenMP runtime library to explicitly specify the set of processors on which they are to be run. This interface is similar in nature to sched_setaffinity and related functions on
Linux* systems or to SetThreadAffinityMask and related functions on Windows* systems. In addition, you can specify certain options of the KMP_AFFINITY environment variable to affect the behavior of the low-level API interface. For example, you can set the affinity type KMP_AFFINITY to disabled, which disables the low-level affinity interface, or you could use the KMP_AFFINITY or GOMP_CPU_AFFINITY environment variables to set the initial affinity mask, and then retrieve the mask with the low-level API interface.

The following terms are used in this section

- The total number of processing elements on the machine is referred to as the number of OS thread contexts.
- Each processing element is referred to as an Operating System processor, or OS proc.
- Each OS processor has a unique integer identifier associated with it, called an OS proc ID.
- The term package refers to a single or multi-core processor chip.
- The term OpenMP Global Thread ID (GTID) refers to an integer which uniquely identifies all threads known to the Intel OpenMP runtime library. The thread that first initializes the library is given GTID 0. In the normal case where all other threads are created by the library and when there is no nested parallelism, then n-threads-var - 1 new threads are created with GTIDs ranging from 1 to nthreads-var - 1, and each thread's GTID is equal to the OpenMP thread number returned by function omp_get_thread_num(). The high-level and mid-level interfaces rely heavily on this concept. Hence, their usefulness is limited in programs containing nested parallelism. The low-level interface does not make use of the concept of a GTID, and can be used by programs containing arbitrarily many levels of parallelism.

The **KMP_AFFINITY Environment Variable**

The KMP_AFFINITY environment variable uses the following general syntax:

```
KMP_AFFINITY= [<modifier>,...]<type>[,<permute>][,<offset>]
```

For example, to list a machine topology map, specify KMP_AFFINITY=verbose,none to use a modifier of verbose and a type of none.

The following table describes the supported specific arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifier</td>
<td>noverbose</td>
<td>Optional. String consisting of keyword and specifier.</td>
</tr>
<tr>
<td>Argument</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>granularity=core</td>
<td>granularity=&lt;specifier&gt; takes the following specifiers: fine, thread, and core</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>none</td>
<td>The syntax for <code>&lt;proc-list&gt;</code> is explained in mid-level affinity interface.</td>
</tr>
</tbody>
</table>

Required string. Indicates the thread affinity to use.

- compact
- disabled
- explicit
- none
- scatter
- logical (deprecated; instead use compact, but omit any permute value)
- physical (deprecated; instead use scatter, possibly with an offset value)
Affinity Types

Type is the only required argument.

**type = none (default)**

Does not bind OpenMP threads to particular thread contexts; however, if the operating system supports affinity, the compiler still uses the OpenMP thread affinity interface to determine machine topology. Specify `KMP_AFFINITY=verbose,none` to list a machine topology map.

**type = compact**

Specifying `compact` assigns the OpenMP thread \(<n>+1\) to a free thread context as close as possible to the thread context where the \(<n>\) OpenMP thread was placed. For example, in a topology map, the nearer a node is to the root, the more significance the node has when sorting the threads.

**type = disabled**

Specifying `disabled` completely disables the thread affinity interfaces. This forces the OpenMP run-time library to behave as if the affinity interface was not supported by the operating system. This includes the low-level API interfaces such as `kmp_set_affinity` and `kmp_get_affinity`, which have no effect and will return a nonzero error code.

**type = explicit**

---

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>permute</td>
<td>0</td>
<td>Optional. Positive integer value. Not valid with type values of <code>explicit</code>, <code>none</code>, or <code>disabled</code>.</td>
</tr>
<tr>
<td>offset</td>
<td>0</td>
<td>Optional. Positive integer value. Not valid with type values of <code>explicit</code>, <code>none</code>, or <code>disabled</code>.</td>
</tr>
</tbody>
</table>

The logical and physical types are deprecated but supported for backward compatibility.
Specifying `explicit` assigns OpenMP threads to a list of OS proc IDs that have been explicitly specified by using the proclist= modifier, which is required for this affinity type. See Explicitly Specifying OS Proc IDs (GOMP_CPU_AFFINITY).

**type = scatter**

Specifying `scatter` distributes the threads as evenly as possible across the entire system. `scatter` is the opposite of `compact`; so the leaves of the node are most significant when sorting through the machine topology map.

**Deprecated Types: logical and physical**

Types `logical` and `physical` are deprecated and may become unsupported in a future release. Both are supported for backward compatibility.

For `logical` and `physical` affinity types, a single trailing integer is interpreted as an offset specifier instead of a permute specifier. In contrast, with `compact` and `scatter` types, a single trailing integer is interpreted as a permute specifier.

- Specifying `logical` assigns OpenMP threads to consecutive logical processors, which are also called hardware thread contexts. The type is equivalent to `compact`, except that the permute specifier is not allowed. Thus, `KMP_AFFINITY=logical,n` is equivalent to `KMP_AFFINITY=compact,0,n` (this equivalence is true regardless of the whether or not a granularity=fine modifier is present).

- Specifying `physical` assigns threads to consecutive physical processors (cores). For systems where there is only a single thread context per core, the type is equivalent to `logical`. For systems where multiple thread contexts exist per core, `physical` is equivalent to `compact` with a permute specifier of 1; that is, `KMP_AFFINITY=physical,n` is equivalent to `KMP_AFFINITY=compact,1,n` (regardless of the whether or not a granularity=fine modifier is present). This equivalence means that when the compiler sorts the map it should permute the innermost level of the machine topology map to the outermost, presumably the thread context level. This type does not support the permute specifier.

**Examples of Types `compact` and `scatter`**

The following figure illustrates the topology for a machine with two processors, and each processor has two cores; further, each core has Hyper-Threading Technology (HT Technology) enabled.

The following figure also illustrates the binding of OpenMP thread to hardware thread contexts when specifying `KMP_AFFINITY=granularity=fine,compact`. 
Specifying `scatter` on the same system as shown in the figure above, the OpenMP threads would be assigned the thread contexts as shown in the following figure, which shows the result of specifying `KMP_AFFINITY=granularity=fine,scatter`. 
permute and offset combinations

For both compact and scatter, permute and offset are allowed; however, if you specify only one integer, the compiler interprets the value as a permute specifier. Both permute and offset default to 0.

The permute specifier controls which levels are most significant when sorting the machine topology map. A value for permute forces the mappings to make the specified number of most significant levels of the sort the least significant, and it inverts the order of significance. The root node of the tree is not considered a separate level for the sort operations.

The offset specifier indicates the starting position for thread assignment.

The following figure illustrates the result of specifying KMP_AFFINITY=granularity=fine,compact,0,3.
Consider the hardware configuration from the previous example, running an OpenMP application which exhibits data sharing between consecutive iterations of loops. We would therefore like consecutive threads to be bound close together, as is done with KMP_AFFINITY=compact, so that communication overhead, cache line invalidation overhead, and page thrashing are minimized. Now, suppose the application also had a number of parallel regions which did not utilize all of the available OpenMP threads. It is desirable to avoid binding multiple threads to the same core and leaving other cores not utilized, since a thread normally executes faster on a core where it is not competing for resources with another active thread on the same core. Since a thread normally executes faster on a core where it is not competing for resources with another active thread on the same core, you might want to avoid binding multiple threads to the same core while leaving other cores unused. The following figure illustrates this strategy of using KMP_AFFINITY=granularity=fine,compact,1,0 as a setting.
The OpenMP thread \( n+1 \) is bound to a thread context as close as possible to OpenMP thread \( n \), but on a different core. Once each core has been assigned one OpenMP thread, the subsequent OpenMP threads are assigned to the available cores in the same order, but they are assigned on different thread contexts.

**Modifier Values for Affinity Types**

Modifiers are optional arguments that precede type. If you do not specify a modifier, the `noverbose`, `respect`, and `granularity=core` modifiers are used automatically.

Modifiers are interpreted in order from left to right, and can negate each other. For example, specifying `KMP_AFFINITY=verbose,noverbose,scatter` is therefore equivalent to setting `KMP_AFFINITY=noverbose,scatter`, or just `KMP_AFFINITY=scatter`.

**modifier = noverbose (default)**

Does not print verbose messages.

**modifier = verbose**

Prints messages concerning the supported affinity. The messages include information about the number of packages, number of cores in each package, number of thread contexts for each core, and OpenMP thread bindings to physical thread contexts.
Information about binding OpenMP threads to physical thread contexts is indirectly shown in the form of the mappings between hardware thread contexts and the operating system (OS) processor (proc) IDs. The affinity mask for each OpenMP thread is printed as a set of OS processor IDs.

For example, specifying `KMP_AFFINITY=verbose,scatter` on a dual core system with two processors, with Hyper-Threading Technology (HT Technology) disabled, results in a message listing similar to the following when the program is executed:

```
Verbose,scatter message
...
KMP_AFFINITY: Affinity capable, using global cpuid info
KMP_AFFINITY: Initial OS proc set respected: {0,1,2,3}
KMP_AFFINITY: 4 available OS procs - Uniform topology of
KMP_AFFINITY: 2 packages x 2 cores/pkg x 1 threads/core (4 total cores)
KMP_AFFINITY: OS proc to physical thread map ([] => level not in map):
KMP_AFFINITY: OS proc 0 maps to package 0 core 0 [thread 0]
KMP_AFFINITY: OS proc 2 maps to package 0 core 1 [thread 0]
KMP_AFFINITY: OS proc 1 maps to package 3 core 0 [thread 0]
KMP_AFFINITY: OS proc 3 maps to package 3 core 1 [thread 0]
KMP_AFFINITY: Internal thread 0 bound to OS proc set {0}
KMP_AFFINITY: Internal thread 2 bound to OS proc set {2}
KMP_AFFINITY: Internal thread 3 bound to OS proc set {3}
KMP_AFFINITY: Internal thread 1 bound to OS proc set {1}
```

The verbose modifier generates several standard, general messages. The following table summarizes how to read the messages.

<table>
<thead>
<tr>
<th>Message String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;affinity capable&quot;</td>
<td>Indicates that all components (compiler, operating system, and hardware) support affinity, so thread binding is possible.</td>
</tr>
<tr>
<td>Message String</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;using global cpuid info&quot;</td>
<td>Indicates that the machine topology was discovered by binding a thread to each operating system processor and decoding the output of the cpuid instruction.</td>
</tr>
<tr>
<td>&quot;using local cpuid info&quot;</td>
<td>Indicates that compiler is decoding the output of the cpuid instruction, issued by only the initial thread, and is assuming a machine topology using the number of operating system processors.</td>
</tr>
<tr>
<td>&quot;using /proc/cpuinfo&quot;</td>
<td>Linux* only. Indicates that cpuinfo is being used to determine machine topology.</td>
</tr>
<tr>
<td>&quot;using flat&quot;</td>
<td>Operating system processor ID is assumed to be equivalent to physical package ID. This method of determining machine topology is used if none of the other methods will work, but may not accurately detect the actual machine topology.</td>
</tr>
<tr>
<td>&quot;uniform topology of&quot;</td>
<td>The machine topology map is a full tree with no missing leaves at any level.</td>
</tr>
</tbody>
</table>

The mapping from the operating system processors to thread context ID is printed next. The binding of OpenMP thread context ID is printed next unless the affinity type is none. The thread level is contained in brackets (in the listing shown above). This implies that there is no representation of the thread context level in the machine topology map. For more information, see Determining Machine Topology.

**modifier = granularity**

Binding OpenMP threads to particular packages and cores will often result in a performance gain on systems with Intel processors with Hyper-Threading Technology (HT Technology) enabled; however, it is usually not beneficial to bind each OpenMP thread to a particular thread context on a specific core. Granularity describes the lowest levels that OpenMP threads are allowed to float within a topology map.

This modifier supports the following additional specifiers.
The finest granularity level. Causes each OpenMP thread to be bound to a single thread context. The two specifiers are functionally equivalent.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>Default. Broadest granularity level supported. Allows all the OpenMP threads bound to a core to float between the different thread contexts.</td>
</tr>
<tr>
<td>fine or thread</td>
<td>The finest granularity level. Causes each OpenMP thread to be bound to a single thread context. The two specifiers are functionally equivalent.</td>
</tr>
</tbody>
</table>
Specifying `KMP_AFFINITY=verbose,granularity=core,compact` on the same dual core system with two processors as in the previous section, but with HT Technology enabled, results in a message listing similar to the following when the program is executed:

```
Verbose, granularity=core,compact message

KMP_AFFINITY: Affinity capable, using global cpuid info
KMP_AFFINITY: Initial OS proc set respected:
{0,1,2,3,4,5,6,7}
KMP_AFFINITY: 8 available OS procs - Uniform topology of
KMP_AFFINITY: 2 packages x 2 cores/pkg x 2 threads/core (4 total cores)
KMP_AFFINITY: OS proc to physical thread map ([] => level not in map):
KMP_AFFINITY: OS proc 0 maps to package 0 core 0 thread 0
KMP_AFFINITY: OS proc 4 maps to package 0 core 0 thread 1
KMP_AFFINITY: OS proc 2 maps to package 0 core 1 thread 0
KMP_AFFINITY: OS proc 6 maps to package 0 core 1 thread 1
KMP_AFFINITY: OS proc 1 maps to package 3 core 0 thread 0
KMP_AFFINITY: OS proc 5 maps to package 3 core 0 thread 1
KMP_AFFINITY: OS proc 3 maps to package 3 core 1 thread 0
KMP_AFFINITY: OS proc 7 maps to package 3 core 1 thread 1
KMP_AFFINITY: Internal thread 0 bound to OS proc set {0,4}
KMP_AFFINITY: Internal thread 1 bound to OS proc set {0,4}
KMP_AFFINITY: Internal thread 2 bound to OS proc set {2,6}
KMP_AFFINITY: Internal thread 3 bound to OS proc set {2,6}
KMP_AFFINITY: Internal thread 4 bound to OS proc set {1,5}
KMP_AFFINITY: Internal thread 5 bound to OS proc set {1,5}
KMP_AFFINITY: Internal thread 6 bound to OS proc set {3,7}
KMP_AFFINITY: Internal thread 7 bound to OS proc set {3,7}
```

The affinity mask for each OpenMP thread is shown in the listing (above) as the set of operating system processor to which the OpenMP thread is bound.
The following figure illustrates the machine topology map, for the above listing, with OpenMP thread bindings.
In contrast, specifying `KMP_AFFINITY=verbose,granularity=fine,compact` or `KMP_AFFINITY=verbose,granularity=thread,compact` binds each OpenMP thread to a single hardware thread context when the program is executed:

<table>
<thead>
<tr>
<th><strong>Verbose, granularity=fine,compact message</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP_AFFINITY: Affinity capable, using global cpuid info</td>
</tr>
<tr>
<td>KMP_AFFINITY: Initial OS proc set respected:</td>
</tr>
<tr>
<td>{0,1,2,3,4,5,6,7}</td>
</tr>
<tr>
<td>KMP_AFFINITY: 8 available OS procs - Uniform topology of</td>
</tr>
<tr>
<td>KMP_AFFINITY: 2 packages x 2 cores/pkg x 2 threads/core (4 total cores)</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc to physical thread map ([] =&gt; level not in map):</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 0 maps to package 0 core 0 thread 0</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 4 maps to package 0 core 0 thread 1</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 2 maps to package 0 core 1 thread 0</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 6 maps to package 0 core 1 thread 1</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 1 maps to package 3 core 0 thread 0</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 5 maps to package 3 core 0 thread 1</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 3 maps to package 3 core 1 thread 0</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 7 maps to package 3 core 1 thread 1</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 0 bound to OS proc set {0}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 1 bound to OS proc set {4}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 2 bound to OS proc set {2}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 3 bound to OS proc set {6}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 4 bound to OS proc set {1}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 5 bound to OS proc set {5}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 6 bound to OS proc set {3}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 7 bound to OS proc set {7}</td>
</tr>
</tbody>
</table>

The OpenMP to hardware context binding for this example was illustrated in the first example.
Specifying `granularity=fine` will always cause each OpenMP thread to be bound to a single OS processor. This is equivalent to `granularity=thread`, currently the finest granularity level.

**modifier = respect (default)**

Respect the process' original affinity mask, or more specifically, the affinity mask in place for the thread that initializes the OpenMP run-time library. The behavior differs between Linux and Windows OS:

- **On Windows:** Respect original affinity mask for the process.
- **On Linux:** Respect the affinity mask for the thread that initializes the OpenMP run-time library.
Specifying `KMP_AFFINITY=verbose,compact` for the same system used in the previous example, with HT Technology enabled, and invoking the library with an initial affinity mask of \{4,5,6,7\} (thread context 1 on every core) causes the compiler to model the machine as a dual core, two-processor system with HT Technology disabled.

<table>
<thead>
<tr>
<th>Verbose,compact message</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP_AFFINITY: Affinity capable, using global cpuid info</td>
</tr>
<tr>
<td>KMP_AFFINITY: Initial OS proc set respected:</td>
</tr>
<tr>
<td>{4,5,6,7}</td>
</tr>
<tr>
<td>KMP_AFFINITY: 4 available OS procs - Uniform topology of</td>
</tr>
<tr>
<td>KMP_AFFINITY: 2 packages x 2 cores/pkg x 1 threads/core (4 total cores)</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc to physical thread map ([] =&gt; level not in map):</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 4 maps to package 0 core 0 [thread 1]</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 6 maps to package 0 core 1 [thread 1]</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 5 maps to package 3 core 0 [thread 1]</td>
</tr>
<tr>
<td>KMP_AFFINITY: OS proc 7 maps to package 3 core 1 [thread 1]</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 0 bound to OS proc set {4}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 1 bound to OS proc set {6}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 2 bound to OS proc set {5}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 3 bound to OS proc set {7}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 4 bound to OS proc set {4}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 5 bound to OS proc set {6}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 6 bound to OS proc set {5}</td>
</tr>
<tr>
<td>KMP_AFFINITY: Internal thread 7 bound to OS proc set {7}</td>
</tr>
</tbody>
</table>

Because there are eight thread contexts on the machine, by default the compiler created eight threads for an OpenMP parallel construct.

The brackets around thread 1 indicate that the thread context level is ignored, and is not present in the topology map. The following figure illustrates the corresponding machine topology map.
When using the local `cpuid` information to determine the machine topology, it is not always possible to distinguish between a machine that does not support Hyper-Threading Technology (HT Technology) and a machine that supports it, but has it disabled. Therefore, the compiler does not include a level in the map if the elements (nodes) at that level had no siblings, with the exception that the package level is always modeled. As mentioned earlier, the package level will always appear in the topology map, even if there only a single package in the machine.

**modifier = norespect**

Do not respect original affinity mask for the process. Binds OpenMP threads to all operating system processors.

In early versions of the OpenMP run-time library that supported only the `physical` and `logical` affinity types, `norespect` was the default and was not recognized as a modifier.

The default was changed to `respect` when types `compact` and `scatter` were added; therefore, thread bindings for the `logical` and `physical` affinity types may have changed with the newer compilers in situations where the application specified a partial initial thread affinity mask.

**modifier = nowarnings**

Do not print warning messages from the affinity interface.

**modifier = warnings (default)**

Print warning messages from the affinity interface (default).
Determining Machine Topology

On IA-32 and Intel® 64 architecture systems, if the package has an APIC (Advanced Programmable Interrupt Controller), the compiler will use the `cpuid` instruction to obtain the package id, core id, and thread context id. Under normal conditions, each thread context on the system is assigned a unique APIC ID at boot time. The compiler obtains other pieces of information obtained by using the `cpuid` instruction, which together with the number of OS thread contexts (total number of processing elements on the machine), determine how to break the APIC ID down into the package ID, core ID, and thread context ID.

Normally, all core ids on a package and all thread context ids on a core are contiguous; however, numbering assignment gaps are common for package ids, as shown in the figure above.

On IA-64 architecture systems on Linux* operating systems, the compiler obtains this information from `/proc/cpuinfo`. The package id, core id, and thread context id are obtained from the physical id, core id, and thread id fields from `/proc/cpuinfo`. The core id and thread context id default to 0, but the physical id field must be present in order to determine the machine topology, which is not always the case. If the information contained in `/proc/cpuinfo` is insufficient or erroneous, you may create an alternate specification file and pass it to the OpenMP runtime library by using the KMP_CPUINFO_FILE environment variable, as described in KMP_CPUINFO and `/proc/cpuinfo`.

If the compiler cannot determine the machine topology using either method, but the operating system supports affinity, a warning message is printed, and the topology is assumed to be flat. For example, a flat topology assumes the operating system process N maps to package N, and there exists only one thread context per core and only one core for each package. (This assumption is always the case for processors based on IA-64 architecture running Windows.)

If the machine topology cannot be accurately determined as described above, the user can manually copy `/proc/cpuinfo` to a temporary file, correct any errors, and specify the machine topology to the OpenMP runtime library via the environment variable KMP_CPUINFO_FILE=<temp_filename>, as described in the section KMP_CPUINFO_FILE and `/proc/cpuinfo`.

Regardless of the method used in determining the machine topology, if there is only one thread context per core for every core on the machine, the thread context level will not appear in the topology map. If there is only one core per package for every package in the machine, the core level will not appear in the machine topology map. The topology map need not be a full tree, because different packages may contain a different number of cores, and different cores may support a different number of thread contexts.
The package level will always appear in the topology map, even if there only a single package in the machine.

**KMP_CPUINFO and /proc/cpuinfo**

One of the methods the Intel compiler OpenMP runtime library can use to detect the machine topology on Linux* systems is to parse the contents of /proc/cpuinfo. If the contents of this file (or a device mapped into the Linux file system) are insufficient or erroneous, you can consider copying its contents to a writable temporary file `<temp_file>`, correct it or extend it with the necessary information, and set KMP_CPUINFO_FILE=`<temp_file>`.

If you do this, the OpenMP runtime library will read the `<temp_file>` location pointed to by KMP_CPUINFO_FILE instead of the information contained in /proc/cpuinfo or attempting to detect the machine topology by decoding the APIC IDs. That is, the information contained in the `<temp_file>` overrides these other methods. You can use the KMP_CPUINFO_FILE interface on Windows* systems, where /proc/cpuinfo does not exist.

The content of /proc/cpuinfo or `<temp_file>` should contain a list of entries for each processing element on the machine. Each processor element contains a list of entries (descriptive name and value on each line). A blank line separates the entries for each processor element. Only the following fields are used to determine the machine topology from each entry, either in `<temp_file>` or /proc/cpuinfo:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>Specifies the OS ID for the processing element. The OS ID must be unique. The processor and physical id fields are the only ones that are required to use the interface.</td>
</tr>
<tr>
<td>physical id</td>
<td>Specifies the package ID, which is a physical chip ID. Each package may contain multiple cores. The package level always exists in the Intel compiler OpenMP run-time library’s model of the machine topology.</td>
</tr>
<tr>
<td>core id</td>
<td>Specifies the core ID. If it does not exist, it defaults to 0. If every package on the machine contains only a single core, the core</td>
</tr>
</tbody>
</table>
thread id:

level will not exist in the machine topology map (even if some of the core ID fields are non-zero).

Specifies the thread ID. If it does not exist, it defaults to 0. If every core on the machine contains only a single thread, the thread level will not exist in the machine topology map (even if some thread ID fields are non-zero).

node_\text{n} id:

This is an extension to the normal contents of \text{/proc/cpuinfo} that can be used to specify the nodes at different levels of the memory interconnect on Non-Uniform Memory Access (NUMA) systems. Arbitrarily many levels \text{n} are supported. The node_0 level is closest to the package level; multiple packages comprise a node at level 0. Multiple nodes at level 0 comprise a node at level 1, and so on.

Each entry must be spelled exactly as shown, in lowercase, followed by optional whitespace, a colon (:), more optional whitespace, then the integer ID. Fields other than those listed are simply ignored.

\textbf{NOTE.} It is common for the thread id field to be missing from \text{/proc/cpuinfo} on many Linux variants, and for a field labeled siblings to specify the number of threads per node or number of nodes per package. However, the Intel compiler OpenMP runtime library ignores fields labeled siblings so it can distinguish between the thread id and siblings fields. When this situation arises, the warning message Physical node/pkg/core/thread ids not unique appears (unless the type specified is nowarnings).
The following is a sample entry for an IA-64 architecture system that has been extended to model the different levels of the memory interconnect:

```
Sample /proc/cpuinfo or <temp-file>

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>23</td>
</tr>
<tr>
<td>vendor</td>
<td>GenuineIntel</td>
</tr>
<tr>
<td>arch</td>
<td>IA-64</td>
</tr>
<tr>
<td>family</td>
<td>32</td>
</tr>
<tr>
<td>model</td>
<td>0</td>
</tr>
<tr>
<td>revision</td>
<td>7</td>
</tr>
<tr>
<td>archrev</td>
<td>0</td>
</tr>
<tr>
<td>features</td>
<td>branchlong, 16-byte atomic ops</td>
</tr>
<tr>
<td>cpu number</td>
<td>0</td>
</tr>
<tr>
<td>cpu regs</td>
<td>4</td>
</tr>
<tr>
<td>cpu MHz</td>
<td>1594.000007</td>
</tr>
<tr>
<td>itc MHz</td>
<td>399.000000</td>
</tr>
<tr>
<td>BogoMIPS</td>
<td>3186.68</td>
</tr>
<tr>
<td>siblings</td>
<td>2</td>
</tr>
<tr>
<td>node_3 id</td>
<td>0</td>
</tr>
<tr>
<td>node_2 id</td>
<td>1</td>
</tr>
<tr>
<td>node_1 id</td>
<td>0</td>
</tr>
<tr>
<td>node_0 id</td>
<td>1</td>
</tr>
<tr>
<td>physical id</td>
<td>2563</td>
</tr>
<tr>
<td>core id</td>
<td>1</td>
</tr>
<tr>
<td>thread id</td>
<td>0</td>
</tr>
</tbody>
</table>
```

This example includes the fields from /proc/cpuinfo that affect the functionality of the Intel compiler OpenMP Affinity Interface: processor, physical id, core id, and thread id. Other fields (vendor, arch, ..., siblings) from /proc/cpuinfo are ignored. The four fields node_n are extensions.
Explicitly Specifying OS Processor IDs (GOMP_CPU_AFFINITY)

Instead of allowing the library to detect the hardware topology and automatically assign OpenMP threads to processing elements, the user may explicitly specify the assignment by using a list of operating system (OS) processor (proc) IDs. However, this requires knowledge of which processing elements the OS proc IDs represent.

This list may either be specified by using the proclist modifier along with the explicit affinity type in the KMP_AFFINITY environment variable, or by using the GOMP_CPU_AFFINITY environment variable (for compatibility with gcc) when using the Intel OpenMP compatibility libraries.

On Linux systems when using the Intel OpenMP compatibility libraries enabled by the compiler option -openmp-lib compat, you can use the GOMP_CPU_AFFINITY environment variable to specify a list of OS processor IDs. Its syntax is identical to that accepted by libgomp (assume that <proc_list> produces the entire GOMP_CPU_AFFINITY environment string):

```
<proc_list> := <entry> | <elem>, <list> | <elem>
                <whitespace> <list>
<elem> := <proc_spec> | <range>
<proc_spec> := <proc_id> - <proc_id> | <proc_id> - <proc_id> : <int>
<range> := <positive_int>
```

OS processors specified in this list are then assigned to OpenMP threads, in order of OpenMP Global Thread IDs. If more OpenMP threads are created than there are elements in the list, then the assignment occurs modulo the size of the list. That is, OpenMP Global Thread ID $n$ is bound to list element $n \mod \text{list_size}$.

Consider the machine previously mentioned: a dual core, dual-package machine without Hyper-Threading Technology (HT Technology) enabled, where the OS proc IDs are assigned in the same manner as the example in a previous figure. Suppose that the application creates 6 OpenMP threads instead of 4 (the default), oversubscribing the machine. If GOMP_CPU_AFFINITY=3,0-2, then OpenMP threads are bound as shown in the figure below, just as should happen when compiling with gcc and linking with libgomp:
The same syntax can be used to specify the OS proc ID list in the proclist=[<proc_list>] modifier in the KMP_AFFINITY environment variable string. There is a slight difference: in order to have strictly the same semantics as in the gcc OpenMP runtime library libgomp: the GOMP_CPU_AFFINITY environment variable implies granularity=fine. If you specify the OS proc list in the KMP_AFFINITY environment variable without a granularity= specifier, then the default granularity is not changed. That is, OpenMP threads are allowed to float between the different thread contexts on a single core. Thus GOMP_CPU_AFFINITY=<proc_list> is an alias for KMP_AFFINITY=granularity=fine,proclist=[<proc_list>],explicit

In the KMP_AFFINITY environment variable string, the syntax is extended to handle operating system processor ID sets. The user may specify a set of operating system processor IDs among which an OpenMP thread may execute ("œfloat") enclosed in brackets:

```
<proc_list> ::= <proc_id> | { <float_list> }

<float_list> ::= <proc_id> | <proc_id>, <float_list>
```

This allows functionality similarity to the granularity= specifier, but it is more flexible. The OS processors on which an OpenMP thread executes may exclude other OS processors nearby in the machine topology, but include other distant OS processors. Building upon the
previous example, we may allow OpenMP threads 2 and 3 to "œfloat" between OS processor 1 and OS processor 2 by using
KMP_AFFINITY="granularity=fine,proclist=[3,0,\{1,2\},\{1,2\}],explicit", as shown in the figure below:
If `verbose` were also specified, the output when the application is executed would include:

```
KMP_AFFINITY="granularity=verbose,fine,proclist=[3,0,(1,2),(1,2)],explicit"
```

**Low Level Affinity API**

Instead of relying on the user to specify the OpenMP thread to OS proc binding by setting an environment variable before program execution starts (or by using the `kmp_settings` interface before the first parallel region is reached), each OpenMP thread may determine the desired set of OS procs on which it is to execute and bind to them with the `kmp_set_affinity` API call.

The C/C++ API interfaces follow, where the type name `kmp_affinity_mask_t` is defined in `omp.h`:
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int kmp_set_affinity (kmp_affinity_mask_t *mask) integer function kmp_set_affinity(mask) integer (kind=kmp_affinity_mask_kind) mask</td>
<td>Sets the affinity mask for the current OpenMP thread to *mask, where *mask is a set of OS proc IDs that has been created using the API calls listed below, and the thread will only execute on OS procs in the set. Returns either a zero (0) upon success or a nonzero error code.</td>
</tr>
<tr>
<td>int kmp_get_affinity (kmp_affinity_mask_t *mask) integer kmp_get_affinity(mask) integer (kind=kmp_affinity_mask_kind) mask</td>
<td>Retrieves the affinity mask for the current OpenMP thread, and stores it in *mask, which must have previously been initialized with a call to kmp_create_affinity_mask(). Returns either a zero (0) upon success or a nonzero error code.</td>
</tr>
<tr>
<td>int kmp_get_affinity_max_proc (void) integer function kmp_get_affinity_max_proc()</td>
<td>Returns the maximum OS proc ID that is on the machine, plus 1. All OS proc IDs are guaranteed to be between 0 (inclusive) and kmp_get_affinity_max_proc() (exclusive).</td>
</tr>
<tr>
<td>void kmp_create_affinity_mask (kmp_affinity_mask_t *mask) subroutine kmp_create_affinity_mask(mask) integer (kind=kmp_affinity_mask_kind) mask</td>
<td>Allocates a new OpenMP thread affinity mask, and initializes *mask to the empty set of OS procs. The implementation is free to use an object of kmp_affinity_mask_t either as the set itself, a pointer to the actual set, or an index into a table describing the set. Do not make any assumption as to what the actual representation is.</td>
</tr>
<tr>
<td>void kmp_destroy_affinity_mask (kmp_affinity_mask_t *mask) subroutine kmp_destroy_affinity_mask(mask) integer (kind=kmp_affinity_mask_kind) mask</td>
<td>Deallocates the OpenMP thread affinity mask. For each call to kmp_create_affinity_mask(), there should be a corresponding call to kmp_destroy_affinity_mask().</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>int kmp_set_affinity_mask_proc (int proc, kmp_affinity_mask_t *mask)</td>
<td>Adds the OS proc ID proc to the set *mask, if it is not already. Returns either a zero (0) upon success or a nonzero error code.</td>
</tr>
<tr>
<td>int kmp_unset_affinity_mask_proc (int proc, kmp_affinity_mask_t *mask)</td>
<td>If the OS proc ID proc is in the set *mask, it removes it. Returns either a zero (0) upon success or a nonzero error code.</td>
</tr>
<tr>
<td>int kmp_get_affinity_mask_proc (int proc, kmp_affinity_mask_t *mask)</td>
<td>Returns 1 if the OS proc ID proc is in the set *mask; if not, it returns 0.</td>
</tr>
</tbody>
</table>

Once an OpenMP thread has set its own affinity mask via a successful call to kmp_affinity_set_mask(), then that thread remains bound to the corresponding OS proc set until at least the end of the parallel region, unless reset via a subsequent call to kmp_affinity_set_mask().

Between parallel regions, the affinity mask (and the corresponding OpenMP thread to OS proc bindings) can be considered thread private data objects, and have the same persistence as described in the OpenMP Application Program Interface. For more information, see the OpenMP API specification (http://www.openmp.org), some relevant parts of which are provided below:
"In order for the affinity mask and thread binding to persist between two consecutive active parallel regions, all three of the following conditions must hold:

- Neither parallel region is nested inside another explicit parallel region.
- The number of threads used to execute both parallel regions is the same.
- The value of the dyn-var internal control variable in the enclosing task region is false at entry to both parallel regions."

Therefore, by creating a parallel region at the start of the program whose sole purpose is to set the affinity mask for each thread, the user can mimic the behavior of the KMP_AFFINITY environment variable with low-level affinity API calls, if program execution obeys the three aforementioned rules from the OpenMP specification. Consider again the example presented in the previous figure. To mimic KMP_AFFINITY=compact, in each OpenMP thread with global thread ID \( n \), we need to create an affinity mask containing OS proc IDs \( n \mod c, n \mod (c + c), \) and so on, where \( c \) is the number of cores. This can be accomplished by inserting the following C code fragment into the application that gets executed at program startup time:

```
Example

int main() {
    #pragma omp parallel
    {
        int tmax = omp_get_max_threads();
        int tnum = omp_get_thread_num();
        int nproc = omp_get_num_procs();
        int ncores = nproc / 2;
        int i;
        kmp_affinity_mask_t mask;
        kmp_create_affinity_mask(&mask);
        for (i = tnum % ncores; i < tmax; i += ncores) {
            kmp_set_affinity_mask_proc(i, &mask);
        }
        if (kmp_set_affinity(&mask) != 0) <error>;
    }
```
This program fragment was written with knowledge about the mapping of the OS proc IDs to the physical processing elements of the target machine. On another machine, or on the same machine with a different OS installed, the program would still run, but the OpenMP thread to physical processing element bindings could differ.

**Intel(R) Workqueueing Model** product="ClassicC"

**Intel® Workqueueing Model Overview**

The workqueueing model implemented by the Intel® compiler extends OpenMP* to parallelize a large range of applications.

**Workqueueing Model**

Programs with irregular patterns of dynamic data structures or with complicated control structures like recursion are hard to parallelize efficiently. The workqueueing model allows the user to exploit irregular parallelism, beyond that possible with OpenMP. The model is an integral part of the compiler, more particularly the OpenMP parallelizer.

The workqueueing model lets you parallelize control structures that are beyond the scope of those supported by the OpenMP model, while attempting to fit into the framework defined by OpenMP. In particular, the workqueueing model is a flexible mechanism for specifying units of work that are not pre-computed at the start of the worksharing construct. For single, for, and sections constructs all work units that can be executed are known at the time the construct begins execution. The workqueueing pragmas, taskq and task, relax this restriction by specifying an environment (taskq) and the units of work (task) separately.

Many control structures exhibit the pattern of separated work iteration and work creation; these structures are naturally parallelized with the workqueueing model. Some common cases are C++ iterators, while loops, and recursive functions.

The taskq pragma specifies the environment within which the enclosed units of work (tasks) are to be executed. From among all the threads that encounter a taskq pragma, one is chosen to execute it initially. Conceptually, the taskq pragma causes an empty queue to be created by the chosen thread, and then the code inside the taskq block is executed in a single-threaded mode.
All the other threads wait for work to be enqueued on the conceptual queue. The `task` pragma specifies a unit of work, potentially executed by a different thread. When a `task` pragma is encountered lexically within a `taskq` block, the code inside the task block is conceptually enqueued on the queue associated with the `taskq`. The conceptual queue is disbanded when all work enqueued on it finishes, and when the end of the `taskq` block is reached.

To preserve sequential semantics, there is an implicit barrier at the completion of the `taskq`. You must ensure either that no dependencies exist or that dependencies are appropriately synchronized, either between the task blocks, or between code in a task block and code in the `taskq` block outside of the task blocks.

See Workqueuing Constructs for more information.

**Workqueuing Constructs**

### `taskq` Pragma

The `taskq` pragma specifies the environment within which the enclosed units of work (tasks) are to be executed. From among all the threads that encounter a `taskq` pragma, one is chosen to execute it initially. Conceptually, the `taskq` pragma causes an empty queue to be created by the chosen thread, and then the code inside the `taskq` block is executed single-threaded.

All the other threads wait for work to be enqueued on the conceptual queue. The `task` pragma specifies a unit of work, potentially executed by a different thread. When a `task` pragma is encountered lexically within a `taskq` block, the code inside the `task` block is conceptually enqueued on the queue associated with the `taskq`. The conceptual queue is disbanded when all work enqueued on it finishes, and when the end of the `taskq` block is reached.

### Control Structures

Many control structures exhibit the pattern of separated work iteration and work creation, and are naturally parallelized with the workqueuing model. Some common cases are:

- **while loops**
- **C++ iterators**
- **recursive functions**

### while Loops
If the computation in each iteration of a while loop is independent, the entire loop becomes the environment for the taskq pragma, and the statements in the body of the while loop become the units of work to be specified with the task pragma. The conditional in the while loop and any modifications to the control variables are placed outside of the task blocks and executed sequentially to enforce the data dependencies on the control variables.

**C++ Iterators**

C++ Standard Template Library (STL) iterators are very much like the while loops just described, whereby the operations on the data stored in the STL are very distinct from the act of iterating over all the data. If the operations are data-independent, they can be done in parallel as long as the iteration over the work is sequential. This type of while loop parallelism is a generalization of the standard OpenMP* worksharing for loops. In the worksharing for loops, the loop increment operation is the iterator and the body of the loop is the unit of work. However, because the for loop iteration variable frequently has a closed form solution, it can be computed in parallel and the sequential step avoided.

**Recursive Functions**

Recursive functions also can be used to specify parallel iteration spaces. The mechanism is similar to specifying parallelism using the sections pragma, but is much more flexible because it allows arbitrary code to sit between the taskq and the task pragmas, and because it allows recursive nesting of the function to build a conceptual tree of taskq queues. The recursive nesting of the taskq pragmas is a conceptual extension of OpenMP worksharing constructs to behave more like nested OpenMP parallel regions. Just like nested parallel regions, each nested workqueueing construct is a new instance and is encountered by exactly one thread. However, the major difference is that nested workqueueing constructs do not cause new threads or teams to be formed, but rather re-use the threads from the team. This permits very easy multi-algorithmic parallelism in dynamic environments, such that the number of threads need not be committed at each level of parallelism, but instead only at the top level. From that point on, if a large amount of work suddenly appears at an inner level, the idle threads from the outer level can assist in getting that work finished. For example, it is very common in server environments to dedicate a thread to handle each incoming request, with a large number of threads awaiting incoming requests. For a particular request, its size may not be obvious at the time the thread begins handling it. If the thread uses nested workqueueing constructs, and the scope of the request becomes large after the inner construct is started, the threads from the outer construct can easily migrate to the inner construct to help finish the request.

Since the workqueueing model is designed to preserve sequential semantics, synchronization is inherent in the semantics of the taskq block. There is an implicit team barrier at the completion of the taskq block for the threads that encountered the taskq construct to ensure that all of the tasks specified inside of the taskq block have finished execution. This taskq barrier enforces
the sequential semantics of the original program. Just like the OpenMP worksharing constructs, it is assumed you are responsible for ensuring that either no dependences exist or that dependencies are appropriately synchronized between the task blocks, or between code in a task block and code in the taskq block outside of the task blocks.

The syntax, semantics, and allowed clauses are designed to resemble OpenMP worksharing constructs. Most of the clauses allowed on OpenMP worksharing constructs have a reasonable meaning when applied to the workqueueing pragmas.

**taskq Construct**

The construct uses syntax similar to the following example:

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
</table>
| #pragma intel omp taskq  
| [clause[,clause]...]  
| structured-block |

where clause can be any of the following:

- private (variable-list)
- firstprivate (variable-list)
- lastprivate (variable-list)
- reduction (operator :variable-list)
- ordered
- nowait

The following table lists the supported clause options:

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>The private clause creates a private, default-constructed version for each object in variable-list for the taskq. It also implies captureprivate on each enclosed task. The original object referenced by each variable has an indeterminate value upon entry to the construct, must not be modified</td>
</tr>
<tr>
<td>Clause</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>firstprivate</td>
<td>The <code>firstprivate</code> clause creates a private, copy-constructed version for each object in <code>variable-list</code> for the task. It also implies <code>captureprivate</code> on each enclosed task. The original object referenced by each variable must not be modified within the dynamic extent of the construct and has an indeterminate value upon exit from the construct.</td>
</tr>
<tr>
<td>lastprivate</td>
<td>The <code>lastprivate</code> clause creates a private, default-constructed version for each object in <code>variable-list</code> for the task. It also implies <code>captureprivate</code> on each enclosed task. The original object referenced by each variable has an indeterminate value upon entry to the construct, must not be modified within the dynamic extent of the construct, and is copy-assigned the value of the object from the last enclosed task after that task completes execution.</td>
</tr>
<tr>
<td>reduction</td>
<td>The <code>reduction</code> clause performs a reduction operation with the given operator in enclosed task constructs for each object in <code>variable-list</code>. <code>operator</code> and <code>variable-list</code> are defined the same as in the OpenMP Specifications.</td>
</tr>
<tr>
<td>ordered</td>
<td>The <code>ordered</code> clause performs ordered constructs in enclosed task constructs in original sequential execution order. The</td>
</tr>
</tbody>
</table>
**Description**

The `taskq` directive, to which the `ordered` is bound, must have an `ordered` clause present.

The `nowait` clause removes the implied barrier at the end of the `taskq`. Threads may exit the `taskq` construct before completing all the `task` constructs queued within it.

### task Construct

The construct uses syntax similar to the following example:

#### Syntax

```
#pragma intel omp task
clause[,...]clause...
structured-block
```

where `clause` can be any of the following:

- `private( variable-list )`
- `captureprivate( variable-list )`

The following table lists the supported clause options:

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>private</code></td>
<td>The <code>private</code> clause creates a private, default-constructed version for each object in <code>variable-list</code> for the <code>task</code>. The original object referenced by the variable has an indeterminate value upon entry to the construct, must not be modified within the dynamic extent of the construct, and has an indeterminate value upon exit from the construct.</td>
</tr>
</tbody>
</table>
The `captureprivate` clause creates a private, copy-constructed version for each object in `variable-list` for the `task` at the time the `task` is enqueued. The original object referenced by each variable retains its value but must not be modified within the dynamic extent of the `task` construct.

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>captureprivate</td>
<td>The <code>captureprivate</code> clause creates a private, copy-constructed version for each object in <code>variable-list</code> for the <code>task</code> at the time the <code>task</code> is enqueued. The original object referenced by each variable retains its value but must not be modified within the dynamic extent of the <code>task</code> construct.</td>
</tr>
</tbody>
</table>

**Combined parallel and taskq Construct**

The combined construct uses syntax similar to the following example:

**Syntax**

```plaintext
#pragma intelomp parallel taskq
[clause[[,]clause]...]
structured-block
```

where clause can be any of the following:

- `if(scalar-expression)`
- `num_threads(integer-expression)`
- `copyin(variable-list)`
- `default(shared | none)`
- `shared(variable-list)`
- `private(variable-list)`
- `firstprivate(variable-list)`
- `lastprivate(variable-list)`
- `reduction(operator : variable-list)`
- `ordered`

Clause descriptions are the same for the parallel and taskq construct.
Workqueuing Example Function

The `test1` function is a natural candidate to be parallelized using the workqueuing model. You can express the parallelism by annotating the loop with a parallel `taskq` pragma and the work in the loop body with a `task` pragma. The parallel `taskq` pragma specifies an environment for the `while` loop in which to enqueue the units of work specified by the enclosed `task` pragma. Thus, the loop's control structure and the enqueuing are executed single-threaded, while the other threads in the team participate in dequeuing the work from the `taskq` queue and executing it. The `captureprivate` clause ensures that a private copy of the link pointer `p` is captured at the time each task is being enqueued, hence preserving the sequential semantics.

Example

```c++
void test1(LIST p){
    #pragma intel omp parallel taskq shared(p){
        while (p != NULL) {
            #pragma intel omp task captureprivate(p){
                do_work1(p);
            }
            p = p->next;
        }
    }
}
```
Auto-parallelization Overview

The auto-parallelization feature of the Intel® compiler automatically translates serial portions of the input program into equivalent multithreaded code. Automatic parallelization determines the loops that are good worksharing candidates, performs the dataflow analysis to verify correct parallel execution, and partitions the data for threaded code generation as needed in programming with OpenMP® directives. The OpenMP and auto-parallelization applications provide the performance gains from shared memory on multiprocessor and dual core systems.

The auto-parallelization feature of the Intel® compiler automatically translates serial portions of the input program into equivalent multithreaded code. The auto-parallelizer analyzes the dataflow of the loops in the application source code and generates multithreaded code for those loops which can safely and efficiently be executed in parallel.

This behavior enables the potential exploitation of the parallel architecture found in symmetric multiprocessor (SMP) systems.

Automatic parallelization frees developers from having to:

- find loops that are good worksharing candidates
- perform the dataflow analysis to verify correct parallel execution
- partition the data for threaded code generation as is needed in programming with OpenMP® directives.

The parallel run-time support provides the same run-time features as found in OpenMP, such as handling the details of loop iteration modification, thread scheduling, and synchronization.

While OpenMP directives enable serial applications to transform into parallel applications quickly, a programmer must explicitly identify specific portions of the application code that contain parallelism and add the appropriate compiler directives.

Auto-parallelization, which is triggered by the -parallel (Linux* OS and Mac OS* X) or /Qparallel (Windows* OS) option, automatically identifies those loop structures that contain parallelism. During compilation, the compiler automatically attempts to deconstruct the code sequences into separate threads for parallel processing. No other effort by the programmer is needed.

**NOTE.** IA-64 architecture only: Specifying these options implies -opt-mem-bandwith1 (Linux) or /Qopt-mem-bandwidth1 (Windows).
Serial code can be divided so that the code can execute concurrently on multiple threads. For example, consider the following serial code example.

**Example 1: Original Serial Code**

```c
void ser(int *a, int *b, int *c)
{
    for (int i=0; i<100; i++)
        a[i] = a[i] + b[i] * c[i];
}
```

The following example illustrates one method showing how the loop iteration space, shown in the previous example, might be divided to execute on two threads.

**Example 2: Transformed Parallel Code**

```c
void par(int *a, int *b, int *c)
{
    int i;
    // Thread 1
    for (i=0; i<50; i++)
        a[i] = a[i] + b[i] * c[i];
    // Thread 2
    for (i=50; i<100; i++)
        a[i] = a[i] + b[i] * c[i];
}
```

**Auto-Vectorization and Parallelization**

Auto-vectorization detects low-level operations in the program that can be done in parallel, and then converts the sequential program to process 2, 4, 8 or up to 16 elements in one operation, depending on the data type. In some cases auto-parallelization and vectorization can be combined for better performance results.
The following example demonstrates how code can be designed to explicitly benefit from parallelization and vectorization. Assuming you compile the code shown below using `-parallel -xSSE3` (Linux*) or `/Qparallel /QxSSE3` (Windows*), the compiler will parallelize the outer loop and vectorize the innermost loop.

### Example

```c
#include <stdio.h>
#define ARR_SIZE 500 //Define array

int main()
{
    int matrix[ARR_SIZE][ARR_SIZE];
    int arrA[ARR_SIZE]={10};
    int arrB[ARR_SIZE]={30};
    int i, j;
    for(i=0;i<ARR_SIZE;i++)
    {
        for(j=0;j<ARR_SIZE;j++)
        {
            matrix[i][j] = arrB[i]*(arrA[i]%2+10);
        }
    }
}
```

Compiling the example code with the correct options, the compiler should report results similar to the following:

```
vectorization.c(18) : (col. 6) remark: LOOP WAS VECTORIZED.
vectorization.c(16) : (col. 3) remark: LOOP WAS AUTO-PARALLELIZED.
```

Auto-vectorization can help improve performance of an application that runs on systems based on Pentium®, Pentium with MMX™ technology, Pentium II, Pentium III, and Pentium 4 processors.

With the right choice of options, you can:

- Increase the performance of your application with minimum effort
• Use compiler features to develop multithreaded programs faster

Additionally, with the relatively small effort of adding OpenMP directives to existing code you can transform a sequential program into a parallel program.
The following example demonstrates one method of using the OpenMP pragmas within code.

Example

```c
#include <stdio.h>
#define ARR_SIZE 100 //Define array
void foo(int ma[][ARR_SIZE], int mb[][ARR_SIZE], int *a, int *b, int *c);
int main()
{
  int arr_a[ARR_SIZE];
  int arr_b[ARR_SIZE];
  int arr_c[ARR_SIZE];
  int i, j;
  int matrix_a[ARR_SIZE][ARR_SIZE];
  int matrix_b[ARR_SIZE][ARR_SIZE];
  #pragma omp parallel for
  // Initialize the arrays and matrices.
  for(i=0; i<ARR_SIZE; i++)
  {
    arr_a[i]= i;
    arr_b[i]= i;
    arr_c[i]= ARR_SIZE-i;
    for(j=0; j<ARR_SIZE; j++)
    {
      matrix_a[i][j]= j;
      matrix_b[i][j]= i;
    }
  }
  foo(matrix_a, matrix_b, arr_a, arr_b, arr_c);
}
```
Example

```c
void foo(int ma[][ARR_SIZE], int mb[][ARR_SIZE], int *a, int *b, int *c)
{
    int i, num, arr_x[ARR_SIZE];
    #pragma omp parallel for private(num)
    // Expresses the parallelism using the OpenMP pragma: parallel for.
    // The pragma guides the compiler generating multithreaded code.
    // Array arr_X, mb, b, and c are shared among threads based on OpenMP
    // data sharing rules. Scalar num is specified as private
    // for each thread.
    for(i=0;i<ARR_SIZE;i++)
    {
        num = ma[b[i]][c[i]];  
        arr_x[i]= mb[a[i]][num];
        printf("Values: %d\n", arr_x[i]); //prints values 0-ARR_SIZE-1
    }
}
```

Auto-Parallelization Options Quick Reference

These options are supported on IA-32, Intel® 64, and IA-64 architectures.

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-parallel</td>
<td>/Qparallel</td>
<td>Enables the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.</td>
</tr>
</tbody>
</table>

IA-64 architecture only:
Auto-parallelization: Enabling, Options, Directives, and Environment Variables

To enable the auto-parallelizer, use the `-parallel` (Linux* and Mac OS* X) or `/Qparallel` (Windows*) option. This option detects parallel loops capable of being executed safely in parallel and automatically generates multi-threaded code for these loops.

**NOTE.** You might need to set the KMP STACKSIZE environment variable to an appropriately large size to enable parallelization with this option.
An example of the command using auto-parallelization is as follows:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icc -c -parallel prog.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /c /Qparallel prog.cpp</td>
</tr>
</tbody>
</table>

Auto-parallelization Directives Format and Syntax

Auto-parallelization Environment Variables

Auto-parallelization uses the following OpenMP* environment variables.

- OMP_NUM_THREADS
- OMP_SCHEDULE
- KMP_STACKSIZE

See OpenMP* Environment Variables for more information about the default settings and how to use these variables.

Programming with Auto-parallelization

The auto-parallelization feature implements some concepts of OpenMP*, such as the worksharing construct (with the PARALLEL FOR directive). See for worksharing construct. This section provides details on auto-parallelization.

Guidelines for Effective Auto-parallelization Usage

A loop can be parallelized if it meets the following criteria:

- The loop is countable at compile time: this means that an expression representing how many times the loop will execute (also called "the loop trip count") can be generated just before entering the loop.
- There are no FLOW (READ after WRITE), OUTPUT (WRITE after WRITE) or ANTI (WRITE after READ) loop-carried data dependencies. A loop-carried data dependency occurs when the same memory location is referenced in different iterations of the loop. At the compiler's discretion, a loop may be parallelized if any assumed inhibiting loop-carried dependencies can be resolved by run-time dependency testing.
The compiler may generate a run-time test for the profitability of executing in parallel for loop with loop parameters that are not compile-time constants.

**Coding Guidelines**

Enhance the power and effectiveness of the auto-parallelizer by following these coding guidelines:

- Expose the trip count of loops whenever possible; specifically use constants where the trip count is known and save loop parameters in local variables.
- Avoid placing structures inside loop bodies that the compiler may assume to carry dependent data, for example, procedure calls, ambiguous indirect references or global references.

**Auto-parallelization Data Flow**

For auto-parallelization processing, the compiler performs the following steps:

1. **Data flow analysis**: Computing the flow of data through the program.
2. **Loop classification**: Determining loop candidates for parallelization based on correctness and efficiency, as shown by threshold analysis.
3. **Dependency analysis**: Computing the dependency analysis for references in each loop nest.
4. **High-level parallelization**: Analyzing dependency graph to determine loops which can execute in parallel, and computing run-time dependency.
5. **Data partitioning**: Examining data reference and partition based on the following types of access: **SHARED**, **PRIVATE**, and **FIRSTPRIVATE**.
6. **Multithreaded code generation**: Modifying loop parameters, generating entry/exit per threaded task, and generating calls to parallel run-time routines for thread creation and synchronization.

**Programming for Multithread Platform Consistency**

For applications where most of the computation is carried out in simple loops, Intel compilers may be able to generate a multithreaded version automatically. This information applies to applications built for deployment on symmetric multiprocessors (SMP), systems with Hyper-Threading Technology (HT Technology) enabled, and dual core processor systems.

The compiler can analyze dataflow in loops to determine which loops can be safely and efficiently executed in parallel. Automatic parallelization can sometimes result in shorter execution times. Compiler enabled auto-parallelization can help reduce the time spent performing several common tasks:
• searching for loops that are good candidates for parallel execution
• performing dataflow analysis to verify correct parallel execution
• adding parallel compiler directives manually

Parallelization is subject to certain conditions, which are described in the next section. If `-openmp` and `-parallel` (Linux* and Mac OS* X) or `/Qopenmp` and `/Qparallel` (Windows*) are both specified on the same command line, the compiler will only attempt to parallelize those functions that do not contain OpenMP* directives.

The following program contains a loop with a high iteration count:

```
#include <math.h>

void no_dep()
{
    int a, n = 100000000;
    float c[n];
    for (int i = 0; i < n; i++) {
        a = 2 * i - 1;
        c[i] = sqrt(a);
    }
}
```

Dataflow analysis confirms that the loop does not contain data dependencies. The compiler will generate code that divides the iterations as evenly as possible among the threads at runtime. The number of threads defaults to the number of processors but can be set independently using the `OMP_NUM_THREADS` environment variable. The increase in parallel speed for a given loop depends on the amount of work, the load balance among threads, the overhead of thread creation and synchronization, etc., but generally will be less than the number of threads. For a whole program, speed increases depend on the ratio of parallel to serial computation.

For builds with separate compiling and linking steps, be sure to link the OpenMP* runtime library when using automatic parallelization. The easiest way to do this is to use the Intel® compiler driver for linking.

**Parallelizing Loops**

Three requirements must be met for the compiler to parallelize a loop.
1. The number of iterations must be known before entry into a loop so that the work can be divided in advance. A while-loop, for example, usually cannot be made parallel.

2. There can be no jumps into or out of the loop.

3. The loop iterations must be independent.

In other words, correct results must not logically depend on the order in which the iterations are executed. There may, however, be slight variations in the accumulated rounding error, as, for example, when the same quantities are added in a different order. In some cases, such as summing an array or other uses of temporary scalars, the compiler may be able to remove an apparent dependency by a simple transformation.

Potential aliasing of pointers or array references is another common impediment to safe parallelization. Two pointers are aliased if both point to the same memory location. The compiler may not be able to determine whether two pointers or array references point to the same memory location. For example, if they depend on function arguments, run-time data, or the results of complex calculations. If the compiler cannot prove that pointers or array references are safe and that iterations are independent, the compiler will not parallelize the loop, except in limited cases when it is deemed worthwhile to generate alternative code paths to test explicitly for aliasing at run-time. If you know parallelizing a particular loop is safe and that potential aliases can be ignored, you can instruct the compiler to parallelize the loop using the `#pragma parallel` pragma.

An alternative way in C to assert that a pointer is not aliased is to use the `restrict` keyword in the pointer declaration, along with the `-restrict` (Linux* OS and Mac OS X) or `/Qrestrict` (Windows) command-line option. The compiler will never parallelize a loop that it can prove to be unsafe.

The compiler can only effectively analyze loops with a relatively simple structure. For example, the compiler cannot determine the thread safety of a loop containing external function calls because it does not know whether the function call might have side effects that introduce dependences. You can invoke interprocedural optimization with the `-ipo` (Linux* OS and Mac OS X) or `/Qipo` (Windows) compiler option. Using this option gives the compiler the opportunity to analyze the called function for side effects.

When the compiler is unable to parallelize automatically loops you know to be parallel use OpenMP*. OpenMP* is the preferred solution because you, as the developer, understand the code better than the compiler and can express parallelism at a coarser granularity. On the other hand, automatic parallelization can be effective for nested loops, such as those in a matrix multiply. Moderately coarse-grained parallelism results from threading of the outer loop, allowing the inner loops to be optimized for fine-grained parallelism using vectorization or software pipelining.

If a loop can be parallelized, it’s not always the case that it should be parallelized. The compiler uses a threshold parameter to decide whether to parallelize a loop. The `-par-threshold` (Linux* OS and Mac OS X) or `/Qpar-threshold` (Windows) compiler option adjusts this behavior. The threshold ranges from 0 to 100, where 0 instructs the compiler to always parallelize
a safe loop and 100 instructs the compiler to only parallelize those loops for which a performance gain is highly probable. Use the -par-report (Linux* OS and Mac OS X) or /Qpar-report (Windows) compiler option to determine which loops were parallelized. The compiler will also report which loops could not be parallelized indicate a probably reason why it could not be parallelized. See Auto-parallelization: Threshold Control and Diagnostics for more information on the using these compiler options.

The following example illustrates using the options in combination. Assume you have the following code:

```c
void add (int k, float *a, float *b)
{
    for (int i = 1; i < 10000; i++)
        a[i] = a[i+k] + b[i];
}
```

Entering a command-line compiler command similar to the following will result in the compiler issuing parallelization messages:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -c -parallel -par-report3 add.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /c /Qparallel /Qpar-report3 add.cpp</td>
</tr>
</tbody>
</table>

The compiler might report results similar to those listed below:

```
add.cpp
    procedure:
    add serial loop: line 2
        anti data dependence assumed from line 2 to line 2, due to "a"
        flow data dependence assumed from line 2 to line 2, due to "a"
```
Because the compiler does not know the value of \( k \), the compiler assumes the iterations depend on each other, for example if \( k \) equals -1, even if the actual case is otherwise. You can override the compiler inserting `#pragma parallel`:

**Example**

```c
void add(int k, float *a, float *b)
{
    #pragma parallel
    for (int i = 0; i < 10000; i++)
        a[i] = a[i+k] + b[i];
}
```

As the developer, it's your responsibility to not call this function with a value of \( k \) that is less than 10000; passing a value less than 10000 could to incorrect results.

**Thread Pooling**

Thread pools offer an effective approach to managing threads. A thread pool is a group of threads waiting for work assignments. In this approach, threads are created once during an initialization step and terminated during a finalization step. This simplifies the control logic for checking for failures in thread creation midway through the application and amortizes the cost of thread creation over the entire application. Once created, the threads in the thread pool wait for work to become available. Other threads in the application assign tasks to the thread pool. Typically, this is a single thread called the thread manager or dispatcher. After completing the task, each thread returns to the thread pool to await further work. Depending upon the work assignment and thread pooling policies employed, it is possible to add new threads to the thread pool if the amount of work grows. This approach has the following benefits:

- Possible runtime failures midway through application execution due to inability to create threads can be avoided with simple control logic.
- Thread management costs from thread creation are minimized. This in turn leads to better response times for processing workloads and allows for multithreading of finer-grained workloads.

A typical usage scenario for thread pools is in server applications, which often launch a thread for every new request. A better strategy is to queue service requests for processing by an existing thread pool. A thread from the pool grabs a service request from the queue, processes it, and returns to the queue to get more work.

Thread pools can also be used to perform overlapping asynchronous I/O. The I/O completion ports provided with the Win32* API allow a pool of threads to wait on an I/O completion port and process packets from overlapped I/O operations.
OpenMP* is strictly a fork/join threading model. In some OpenMP implementations, threads are created at the start of a parallel region and destroyed at the end of the parallel region. OpenMP applications typically have several parallel regions with intervening serial regions. Creating and destroying threads for each parallel region can result in significant system overhead, especially if a parallel region is inside a loop; therefore, the Intel OpenMP implementation uses thread pools. A pool of worker threads is created at the first parallel region. These threads exist for the duration of program execution. More threads may be added automatically if requested by the program. The threads are not destroyed until the last parallel region is executed.

Thread pools can be created on Windows and Linux using the thread creation API. For instance, a custom thread pool using Win32 threads may be created as follows:

```c
Example

// Initialization method/function
{
 DWORD tid;
 // Create initial pool of threads
 for (int i = 0; i < MIN_THREADS; i++)
 {
  HANDLE *ThHandle = CreateThread (NULL,0,CheckPoolQueue,NULL,0,&tid);
  if (ThHandle == NULL)
   // Handle Error
  else
   RegisterPoolThread (ThHandle);
 }
}
```

The function `CheckPoolQueue` executed by each thread in the pool is designed to enter a wait state until work is available on the queue. The thread manager can keep track of pending jobs in the queue and dynamically increase the number of threads in the pool based on the demand.
Using Parallelism: Automatic Vectorization

Automatic Vectorization Overview

The automatic vectorizer (also called the auto-vectorizer) is a component of the Intel® compiler that automatically uses SIMD instructions in the MMX™, Intel® Streaming SIMD Extensions (Intel® SSE, SSE2, SSE3 and SSE4 Vectorizing Compiler and Media Accelerators) and Supplemental Streaming SIMD Extensions (SSSE3) instruction sets. The vectorizer detects operations in the program that can be done in parallel, and then converts the sequential operations like one SIMD instruction that processes 2, 4, 8 or up to 16 elements in parallel, depending on the data type.

Automatic vectorization is supported on IA-32 and Intel® 64 architectures.

The section discusses the following topics, among others:

• High-level discussion of compiler options used to control or influence vectorization
• Vectorization Key Programming Guidelines
• Loop parallelization and vectorization
• Descriptions of the C++ language features to control vectorization
• Discussion and general guidelines on vectorization levels:
  • automatic vectorization
  • vectorization with user intervention
• Examples demonstrating typical vectorization issues and resolutions

The compiler supports a variety of directives that can help the compiler to generate effective vector instructions.


Automatic Vectorization Options Quick Reference

These options are supported on IA-32 and Intel® 64 architectures.
<table>
<thead>
<tr>
<th>Linux* OS and Mac OS* X</th>
<th>Windows* OS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-x</td>
<td>/Qx</td>
<td>Generates specialized code to run exclusively on processors with the extensions specified as the processor value. See Targeting IA-32 and Intel® 64 Architecture Processors Automatically for more information about using the option. Generates, in a single binary, code specialized to the extensions specified as the processor value and also generic IA-32 architecture code. The generic code is usually slower. See Targeting Multiple IA-32 and Intel® 64 Architecture Processors for Run-time Performance for more information about using the option.</td>
</tr>
<tr>
<td>-ax</td>
<td>/Qax</td>
<td>Enables or disables vectorization and transformations enabled for vectorization. The default is that vectorization is enabled. Supported for IA-32 and Intel® 64 architectures only. Controls the diagnostic messages from the vectorizer. See Vectorization Report.</td>
</tr>
<tr>
<td>-vec</td>
<td>/Qvec</td>
<td>Vectorization within the Intel® compiler depends upon ability of the compiler to disambiguate memory references. Certain options may enable the compiler to do better vectorization.</td>
</tr>
<tr>
<td>-vec-report</td>
<td>/Qvec-report</td>
<td></td>
</tr>
</tbody>
</table>
Refer to Quick Reference Lists for a complete listing of the quick reference topics.

Programming Guidelines for Vectorization

The goal of vectorizing compilers is to exploit single-instruction multiple data (SIMD) processing automatically. Users can help however by supplying the compiler with additional information; for example, by using directives.

Guidelines

You will often need to make some changes to your loops. Follow these guidelines for loop bodies.

Use:

• straight-line code (a single basic block)
• vector data only; that is, arrays and invariant expressions on the right hand side of assignments. Array references can appear on the left hand side of assignments.
• only assignment statements

Avoid:

• function calls
• unvectorizable operations (other than mathematical)
• mixing vectorizable types in the same loop
• data-dependent loop exit conditions

To make your code vectorizable, you will often need to make some changes to your loops. However, you should make only the changes needed to enable vectorization and no others. In particular, you should avoid these common changes:

• loop unrolling; the compiler does it automatically.
• decomposing one loop with several statements in the body into several single-statement loops.

Restrictions

There are a number of restrictions that you should be consider. Vectorization depends on two major factors: hardware and style of source code.
The compiler is limited by restrictions imposed by the underlying hardware. In the case of Streaming SIMD Extensions, the vector memory operations are limited to stride-1 accesses with a preference to 16-byte-aligned memory references. This means that if the compiler abstractly recognizes a loop as vectorizable, it still might not vectorize it for a distinct target architecture.

The style in which you write source code can inhibit optimization. For example, a common problem with global pointers is that they often prevent the compiler from being able to prove that two memory references refer to distinct locations. Consequently, this prevents certain reordering transformations.

Many stylistic issues that prevent automatic vectorization by compilers are found in loop structures. The ambiguity arises from the complexity of the keywords, operators, data references, and memory operations within the loop bodies.

However, by understanding these limitations and by knowing how to interpret diagnostic messages, you can modify your program to overcome the known limitations and enable effective vectorization. The following sections summarize the capabilities and restrictions of the vectorizer with respect to loop structures.

### Vectorization and Loops

Combine the `-parallel` (Linux* and Mac OS* X) or `/Qparallel` (Windows*) and `-x` (Linux) or `/Qx` (Windows) options to instructs the compiler to attempt both automatic loop parallelization and automatic loop vectorization in the same compilation.

In most cases, the compiler will consider outermost loops for parallelization and innermost loops for vectorization. If deemed profitable, however, the compiler may even apply loop parallelization and vectorization to the same loop.

See Guidelines for Effective Auto-parallelization Usage and Programming Guidelines for Vectorization.
In some rare cases successful loop parallelization (either automatically or by means of OpenMP* directives) may affect the messages reported by the compiler for a non-vectorizable loop in a non-intuitive way; for example, in the cases where -vec-report2 (Linux and Mac OS X) or /Qvec-report2 (Windows) option indicating loops were not successfully vectorized. (See Vectorization Report.)

**Types of Vectorized Loops**

For integer loops, the 64-bit MMX™ technology and 128-bit Intel® Streaming SIMD Extensions (Intel® SSE) provide SIMD instructions for most arithmetic and logical operators on 32-bit, 16-bit, and 8-bit integer data types.

Vectorization may proceed if the final precision of integer wrap-around arithmetic will be preserved. A 32-bit shift-right operator, for instance, is not vectorized in 16-bit mode if the final stored value is a 16-bit integer. Also, note that because the MMX™ and Intel® SSE instruction sets are not fully orthogonal (shifts on byte operands, for instance, are not supported), not all integer operations can actually be vectorized.

For loops that operate on 32-bit single-precision and 64-bit double-precision floating-point numbers, Intel® SSE provides SIMD instructions for the following arithmetic operators: addition (+), subtraction (-), multiplication (*), and division (/).

Additionally, the Streaming SIMD Extensions provide SIMD instructions for the binary **MIN** and **MAX** and unary **SQRT** operators. SIMD versions of several other mathematical operators (like the trigonometric functions **SIN**, **COS**, and **TAN**) are supported in software in a vector mathematical run-time library that is provided with the Intel® compiler of which the compiler takes advantage.

**Statements in the Loop Body**

The vectorizable operations are different for floating-point and integer data.

**Integer Array Operations**

The statements within the loop body may contain **char**, **unsigned char**, **short**, **unsigned short**, **int**, and **unsigned int**. Calls to functions such as **sqrt** and **fabs** are also supported. Arithmetic operations are limited to addition, subtraction, bitwise **AND**, **OR**, and **XOR** operators, division (16-bit only), multiplication (16-bit only), min, and max. You can mix data types only if the conversion can be done without a loss of precision. Some example operators where you can mix data types are multiplication, shift, or unary operators.
Other Operations

No statements other than the preceding floating-point and integer operations are allowed. In particular, note that the special \texttt{\_m64} and \texttt{\_m128} datatypes are not vectorizable. The loop body cannot contain any function calls. Use of the Streaming SIMD Extensions intrinsics (\texttt{\_mm\_add\_ps}) are not allowed.

Data Dependency

Data dependency relations represent the required ordering constraints on the operations in serial loops. Because vectorization rearranges the order in which operations are executed, any auto-vectorizer must have at its disposal some form of data dependency analysis.

An example where data dependencies prohibit vectorization is shown below. In this example, the value of each element of an array is dependent on the value of its neighbor that was computed in the previous iteration.

Example 1: Data-dependent Loop

```
int i;
void dep(float *data)
{
  for (i=1; i<100; i++)
    data[i] = data[i-1]*0.25 + data[i]*0.5 + data[i+1]*0.25;
}
```
The loop in the above example is not vectorizable because the WRITE to the current element \( \text{DATA}(I) \) is dependent on the use of the preceding element \( \text{DATA}(I-1) \), which has already been written to and changed in the previous iteration. To see this, look at the access patterns of the array for the first two iterations as shown below.

### Example 2: Data-dependency Vectorization Patterns

```
for(i=0; i<100; i++)
a[i]=b[i];
```

has access pattern

- read \( b[0] \)
- write \( a[0] \)
- read \( b[1] \)
- write \( a[1] \)
- i=1: READ \( \text{data}[0] \)
- \( \text{READ data}[1] \)
- \( \text{READ data}[2] \)
- WRITE \( \text{data}[1] \)
- i=2: READ \( \text{data}[1] \)
- \( \text{READ data}[2] \)
- \( \text{READ data}[3] \)
- WRITE \( \text{data}[2] \)
```

In the normal sequential version of this loop, the value of \( \text{DATA}(1) \) read from during the second iteration was written to in the first iteration. For vectorization, it must be possible to do the iterations in parallel, without changing the semantics of the original loop.

### Data dependency Analysis

Data dependency analysis involves finding the conditions under which two memory accesses may overlap. Given two references in a program, the conditions are defined by:

- whether the referenced variables may be aliases for the same (or overlapping) regions in memory
for array references, the relationship between the subscripts

For IA-32 architecture, data dependency analyzer for array references is organized as a series of tests, which progressively increase in power as well as in time and space costs.

First, a number of simple tests are performed in a dimension-by-dimension manner, since independency in any dimension will exclude any dependency relationship. Multidimensional arrays references that may cross their declared dimension boundaries can be converted to their linearized form before the tests are applied.

Some of the simple tests that can be used are the fast greatest common divisor (GCD) test and the extended bounds test. The GCD test proves independency if the GCD of the coefficients of loop indices cannot evenly divide the constant term. The extended bounds test checks for potential overlap of the extreme values in subscript expressions.

If all simple tests fail to prove independency, the compiler will eventually resort to a powerful hierarchical dependency solver that uses Fourier-Motzkin elimination to solve the data dependency problem in all dimensions.

Loop Constructs

Loops can be formed with the usual for and while constructs. The loops must have a single entry and a single exit to be vectorized. The following examples illustrate loop constructs that can and cannot be vectorized.

Example: Vectorizable structure

```c
void vec(float a[], float b[], float c[])
{
    int i = 0;
    while (i < 100) {
        // The if branch is inside body of loop.
        a[i] = b[i] * c[i];
        if (a[i] < 0.0)
            a[i] = 0.0;
        i++;
    }
}
```

```c
for (int i = 0; i < 100; i++)
    if (a[i] < 0.0)
        a[i] = 0.0;
```
The following example shows a loop that cannot be vectorized because of the inherent potential for an early exit from the loop.

<table>
<thead>
<tr>
<th>Example: Non-vectorizable structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>void no_vec(float a[], float b[], float c[])</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>int i = 0;</td>
</tr>
<tr>
<td>while (i &lt; 100) {</td>
</tr>
<tr>
<td>if (i &lt; 50)</td>
</tr>
<tr>
<td>// The next statement is a second exit</td>
</tr>
<tr>
<td>// that allows an early exit from the loop.</td>
</tr>
<tr>
<td>break;</td>
</tr>
<tr>
<td>++i;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

**Loop Exit Conditions**

Loop exit conditions determine the number of iterations a loop executes. For example, fixed indexes for loops determine the iterations. The loop iterations must be countable; in other words, the number of iterations must be expressed as one of the following:

- A constant
- A loop invariant term
- A linear function of outermost loop indices
In the case where a loop's exit depends on computation, the loops are not countable. The examples below show loop constructs that are countable and non-countable.

**Example: Countable Loop**

```cpp
void cnt1(float a[], float b[], float c[],
           int n, int lb)
{
    // Exit condition specified by "N-lb+1"
    int cnt=n, i=0;
    while (cnt >= lb) {
        // lb is not affected within loop.
        a[i] = b[i] * c[i];
        cnt--;
        i++;
    }
}
```

The following example demonstrates a different countable loop construct.

**Example: Countable Loop**

```cpp
void cnt2(float a[], float b[], float c[],
           int m, int n)
{
    // Number of iterations is 
     \((n-m+2)/2\).
    int i=0, l;
    for (l=m; l<n; l+=2) {
        a[i] = b[i] * c[i];
        i++;
    }
}
```
The following examples demonstrates a loop construct that is non-countable due to dependency loop variant count value.

### Example: Non-Countable Loop

```c
void no_cnt(float a[], float b[], float c[])
{
    int i=0;
    // Iterations dependent on a[i].
    while (a[i]>0.0) {
        a[i] = b[i] * c[i];
        i++;
    }
}
```

### Strip-mining and Cleanup

Strip-mining, also known as loop sectioning, is a loop transformation technique for enabling SIMD-encodings of loops, as well as a means of improving memory performance. By fragmenting a large loop into smaller segments or strips, this technique transforms the loop structure in two ways:

- It increases the temporal and spatial locality in the data cache if the data are reusable in different passes of an algorithm.
- It reduces the number of iterations of the loop by a factor of the length of each vector, or number of operations being performed per SIMD operation. In the case of Streaming SIMD Extensions, this vector or strip-length is reduced by 4 times: four floating-point data items per single Streaming SIMD Extensions single-precision floating-point SIMD operation are processed.

First introduced for vectorizers, this technique consists of the generation of code when each vector operation is done for a size less than or equal to the maximum vector length on a given vector machine.
The compiler automatically strip-mines your loop and generates a cleanup loop. For example, assume the compiler attempts to strip-mine the following loop:

**Example 1: Before Vectorization**

```c
i=0;
while(i<n)
{
    // Original loop code
    a[i]=b[i]+c[i];
    ++i;
}
```

The compiler might handle the strip mining and loop cleaning by restructuring the loop in the following manner:

**Example 2: After Vectorization**

```
// The vectorizer generates the following two loops
```
Loop Blocking

It is possible to treat loop blocking as strip-mining in two or more dimensions. Loop blocking is a useful technique for memory performance optimization. The main purpose of loop blocking is to eliminate as many cache misses as possible. This technique transforms the memory domain into smaller chunks rather than sequentially traversing through the entire memory domain. Each chunk should be small enough to fit all the data for a given computation into the cache, thereby maximizing data reuse.
Consider the following example, loop blocking allows arrays A and B to be blocked into smaller rectangular chunks so that the total combined size of two blocked (A and B) chunks is smaller than cache size, which can improve data reuse.

**Example 3: Original loop**

```c
#include <time.h>
#include <stdio.h>
#define MAX 7000

void add(int a[][MAX], int b[][MAX]);

int main()
{
    int i, j;
    int A[MAX][MAX];
    int B[MAX][MAX];
    time_t start, elapsed;
    int sec;
    // Initialize array
    for(i=0; i<MAX; i++)
    {
        for(j=0; j<MAX; j++)
        {
            A[i][j]=j;
            B[i][j]=j;
        }
    }
    start = time(NULL);
    add(A, B);
    elapsed = time(NULL);
    sec = elapsed - start;
```
Example 3: Original loop

```c
void add(int a[][MAX], int b[][MAX])
{
    int i, j;
    for(i=0; i<MAX; i++)
    {
        for(j=0; j<MAX; j++)
        {
            a[i][j] = a[i][j] + b[j][i]; //Adds two matrices
        }
    }
}

printf("Time %d", sec); //List time taken to complete add function
```
The following example illustrates loop blocking the add function (from the previous example). In order to benefit from this optimization you might have to increase the cache size.

**Example 4: Transformed Loop after blocking**

```c
#include <stdio.h>
#include <time.h>
#define MAX 7000
void add(int a[][MAX], int b[][MAX]);
int main()
{
#if define BS 8  //Block size is selected as the loop-blocking factor.
int i, j;
int A[MAX][MAX];
int B[MAX][MAX];
time_t start, elapsed;
int sec;
//initialize array
for(i=0;i<MAX;i++)
{
    for(j=0;j<MAX;j++)
    {
        A[i][j]=j;
        B[i][j]=j;
    }
}
start= time(NULL);
add(A, B);
elapsed= time(NULL);
sec = elapsed - start;
```
**Example 4: Transformed Loop after blocking**

```c
printf("Time %d",sec); //Display time taken to complete loop

void add(int a[][MAX], int b[][MAX])
{
    int i, j, ii, jj;
    for(i=0;i<MAX;i+=BS)
    {
        for(j=0; j<MAX;j+=BS)
        {
            for(ii=i; ii<i+BS; ii++)//outer loop
            {
                for(jj=j; jj<j+BS; jj++) //Array B experiences one cache miss
                {
                    a[ii][jj] = a[ii][jj] + b[jj][ii]; //Add the two arrays
                }
            }
        }
    }
}
```
Loop Interchange and Subscripts: Matrix Multiply

Loop interchange need unit-stride constructs to be vectorized. Matrix multiplication is commonly written as shown in the following example:

**Example: Typical Matrix Multiplication**

```c
void matmul_slow(float *a[], float *b[], float *c[])
{
    int N = 100;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

The use of $B(K,J)$ is not a stride-1 reference and therefore will not normally be vectorizable. If the loops are interchanged, however, all the references will become stride-1 as shown in the following example.

**Example: Matrix Multiplication with Stride-1**

```c
void matmul_fast(float *a[], float *b[], float *c[])
{
    int N = 100;
    for (int i = 0; i < N; i++)
        for (int k = 0; k < N; k++)
            for (int j = 0; j < N; j++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

Interchanging is not always possible because of dependencies, which can lead to different results.
Language Support and Directives

This topic addresses specific language features that better help to vectorize code.

The `__declspec(align(n))` declaration enables you to overcome hardware alignment constraints. The `restrict` qualifier and the pragmas address the stylistic issues due to lexical scope, data dependency, and ambiguity resolution.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| `__declspec(align(n))` | Directs the compiler to align the variable to an `n`-byte boundary. Address of the variable is `address mod n=0`.
| `__declspec(align(n,off))` | Directs the compiler to align the variable to an `n`-byte boundary with offset `off` within each `n`-byte boundary. Address of the variable is `address mod n=off`.
| `restrict` | Permits the disambiguator flexibility in alias assumptions, which enables more vectorization.
| `__assume_aligned(a, n)` | Instructs the compiler to assume that array `a` is aligned on an `n`-byte boundary; used in cases where the compiler has failed to obtain alignment information.
| `#pragma ivdep` | Instructs the compiler to ignore assumed vector dependencies.
| `#pragma vector {aligned|unaligned|always}` | Specifies how to vectorize the loop and indicates that efficiency heuristics should be ignored.
| `#pragma novector` | Specifies that the loop should never be vectorized,}
Interprocedural Optimization (IPO) Overview

Interprocedural Optimization (IPO) allows the compiler to analyze your code to determine where you can benefit from specific optimizations. In many cases, the optimizations that can be applied are related to the specific architectures.

The compiler might apply the following optimizations for the listed architectures:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-32, Intel® 64, and IA-64 architectures</td>
<td>• inlining&lt;br&gt;• constant propagation&lt;br&gt;• mod/ref analysis&lt;br&gt;• alias analysis&lt;br&gt;• forward substitution&lt;br&gt;• routine key-attribute propagation&lt;br&gt;• address-taken analysis&lt;br&gt;• partial dead call elimination&lt;br&gt;• symbol table data promotion&lt;br&gt;• common block variable coalescing&lt;br&gt;• dead function elimination&lt;br&gt;• unreferenced variable removal&lt;br&gt;• whole program analysis&lt;br&gt;• array dimension padding&lt;br&gt;• common block splitting&lt;br&gt;• stack frame alignment&lt;br&gt;• structure splitting and field reordering&lt;br&gt;• formal parameter alignment analysis&lt;br&gt;• C++ class hierarchy analysis&lt;br&gt;• indirect call conversion</td>
</tr>
</tbody>
</table>
Optimization Architecture

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-32 and Intel® 64 architectures</td>
<td>• specialization</td>
</tr>
<tr>
<td>IA-64 architecture only</td>
<td>• Passing arguments in registers to optimize calls and register usage</td>
</tr>
<tr>
<td></td>
<td>• removing redundant EXTEND instructions</td>
</tr>
<tr>
<td></td>
<td>• short data section allocation</td>
</tr>
<tr>
<td></td>
<td>• prefetch analysis</td>
</tr>
</tbody>
</table>

IPO is an automatic, multi-step process: compilation and linking; however, IPO supports two compilation models: single-file compilation and multi-file compilation.

Single-file compilation, which uses the -ip (Linux® OS and Mac OS® X) or /Qip (Windows® OS) option, results in one, real object file for each source file being compiled. During single-file compilation the compiler performs inline function expansion for calls to procedures defined within the current source file.

The compiler performs some single-file interprocedural optimization at the default optimization level: -O2 (Linux® and Mac OS® X) or /O2 (Windows®); additionally some inlining for the -O1 (Linux® and Mac OS® X) or /O1 (Windows®) optimization level, like inlining functions marked with inlining pragmas or attributes (GNU C and C++) and C++ class member functions with bodies included in the class declaration.

Multi-file compilation, which uses the -ipo (Linux and Mac OS X) or /Qipo (Windows) option, results in one or more mock object files rather than normal object files. (See the Compilation section below for information about mock object files.) Additionally, the compiler collects information from the individual source files that make up the program. Using this information, the compiler performs optimizations across functions and procedures in different source files. Inlining is the most powerful optimization supported by IPO. See Inline Function Expansion.

**NOTE.** Inlining and other optimizations are improved by profile information. For a description of how to use IPO with profile information for further optimization, see Profile an Application.

Mac OS® X: Intel®-based systems running Mac OS X do not support a multiple object compilation model.
Compilation

As each source file is compiled with IPO, the compiler stores an intermediate representation (IR) of the source code in a mock object file, which includes summary information used for optimization. The mock object files contain the IR, instead of the normal object code. Mock object files can be ten times larger, and in some cases more, than the size of normal object files.

During the IPO compilation phase only the mock object files are visible. The Intel compiler does not expose the real object files during IPO unless you also specify the `-ipo-c` (Linux and Mac OS X) or `/Qipo-c` (Windows) option.

Linkage

When you link with the `-ipo` (Linux and Mac OS X) or `/Qipo` (Windows) option the compiler is invoked a final time. The compiler performs IPO across all object files that have an IR equivalent. The mock objects must be linked with the Intel compiler or by using the Intel linking tools. The compiler calls the linkers indirectly by using aliases (or wrappers) for the native linkers, so you must modify make files to account for the different linking tool names. For information on using the linking tools, see Using IPO; see the Linking Tools and Options topic for detailed information.

CAUTION. Linking the mock object files with `ld` (Linux and Mac OS X) or `link.exe` (Windows) will cause linkage errors. You must use the Intel linking tools to link mock object files.

During the compilation process, the compiler first analyzes the summary information and then produces mock object files for source files of the application.

Whole program analysis

The compiler supports a large number of IPO optimizations that either can be applied or have the effectiveness greatly increased when the whole program condition is satisfied.

Whole program analysis, when it can be done, enables many interprocedural optimizations. During the analysis process, the compiler reads all Intermediate Representation (IR) in the mock file, object files, and library files to determine if all references are resolved and whether or not a given symbol is defined in a mock object file. Symbols that are included in the IR in a mock object file for both data and functions are candidates for manipulation based on the results of whole program analysis.
There are two types of whole program analysis: object reader method and table method. Most optimizations can be applied if either type of whole program analysis determine that the whole program conditions exists; however, some optimizations require the results of the object reader method, and some optimizations require the results of table method.

**NOTE.** The IPO report provides details about whether whole program analysis was satisfied and indicate the method used during IPO compilation.

In the first type of whole program analysis, the object reader method, the object reader emulates the behavior of the native linker and attempts to resolve the symbols in the application. If all symbols are resolved correctly, the whole program condition is satisfied. This type of whole program analysis is more likely to detect the whole program condition.

Often the object files and libraries accessed by the compiler do not represent the whole program; there are many dependencies to well-known libraries. IPO linking, whole program analysis, determines whether or not the whole program can be detected using the available compiler resources.

The second type of whole program analysis, the table method, is where the compiler analyzes the mock object files and generates a call-graph.

The compiler contains detailed tables about all of the functions for all important language-specific libraries, like libc. In this second method, the compiler constructs a call-graph for the application. The compiler then compares the function table and application call-graph. For each unresolved function in the call-graph, the compiler attempts to resolve the calls. If the compiler can resolve the functions call, the whole program condition exists.

**Interprocedural Optimization (IPO) Quick Reference**

IPO is a two step process: compile and link. See Using IPO.

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ipo or -ipoN</td>
<td>/Qipo</td>
<td>Enables interprocedural optimization for multi-file compilations. Normally, multi-file compilations result in a single object file only. Passing an integer value for ( N ) allows you...</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>-ipo-separate</td>
<td>/Qipo-separate</td>
<td>to specify number of true object files to generate; the default value is 0, which means the compiler determines the appropriate number of object files to generate. (See IPO for Large Programs.)</td>
</tr>
<tr>
<td>-ip</td>
<td>/Qip</td>
<td>Instructs the compiler to generate a separate, real object file for each mock object file. Using this option overrides any integer value passed for ipo(N). (See IPO for Large Programs for specifics.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enables interprocedural optimizations for single file compilations. Instructs the compiler to generate a separate, real object file for each source file.</td>
</tr>
</tbody>
</table>

Additionally, the compiler supports options that provide support for compiler-directed or developer-directed inline function expansion.

Refer to Quick Reference Lists for a complete listing of the quick reference topics.

**Using IPO**

This topic discusses how to use IPO from a command line. For specific information on using IPO from within an Integrated Development Environment (IDE), refer to the appropriate section in Building Applications.

**Compiling and Linking Using IPO**

The steps to enable IPO for compilations targeted for IA-32, Intel® 64, and IA-64 architectures are the same: compile and link.

First, compile your source files with -ipo (Linux* and Mac OS* X) or /Qipo (Windows*) as demonstrated below:

---

1275
The output of the above example command differs according to operating system:

- **Linux and Mac OS X**: The commands produce `a.o`, `b.o`, and `c.o` object files.
- **Windows**: The commands produce `a.obj`, `b.obj`, and `c.obj` object files.

Use `-c` (Linux and Mac OS X) or `/c` (Windows) to stop compilation after generating `.o` or `.obj` files. The output files contain compiler intermediate representation (IR) corresponding to the compiled source files. (See the section below on capturing the intermediate IPO output.)

Second, link the resulting files. The following example command will produce an executable named `app`:

The command invoke the compiler on the objects containing IR and creates a new list of objects to be linked. Alternately, you can use the `xild` (Linux and Mac OS X) or `xilink` (Windows) tool, with the appropriate linking options.

### Combining the Steps

The separate compile and link commands demonstrated above can be combined into a single command, as shown in the following examples:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td><code>icpc -ipo -o app a.cpp b.cpp c.cpp</code></td>
</tr>
<tr>
<td>Windows*</td>
<td><code>icl /Qipo /Feapp a.cpp b.cpp c.cpp</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td><code>icpc -ipo -o app a.o b.o c.o</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>icl /Feapp a.obj b.obj c.obj</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td><code>icpc -ipo -o app a.cpp b.cpp c.cpp</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>icl /Qipo /Feapp a.cpp b.cpp c.cpp</code></td>
</tr>
</tbody>
</table>
The icl/icpc command, shown in the examples above, calls GCC ld (Linux and Mac OS X) or Microsoft* link.exe (Windows only) to link the specified object files and produce the executable application, which is specified by the -o (Linux and Mac OS X) or /Fe (Windows) option.

**NOTE.** Linux and Mac OS X: Using icpc allows the compiler to use standard C++ libraries automatically; icc will not use the standard C++ libraries automatically.

The Intel linking tools emulate the behavior of compiling at -O0 (Linux and Mac OS X) and /Od (Windows).

Multi-file IPO is applied only to the source files that have intermediate representation (IR); otherwise, the object file passes to link stage.

### Capturing Intermediate IPO Output

The -ipo-c (Linux and Mac OS X) or /Qipo-c (Windows*) and -ipo-S (Linux and Mac OS X) or /Qipo-S (Windows) options are useful for analyzing the effects of multi-file IPO, or when experimenting with multi-file IPO between modules that do not make up a complete program.

- **Use the -ipo-c option to optimize across files and produce an object file.** The option performs optimizations as described for the -ipo option but stops prior to the final link stage, leaving an optimized object file. The default name for this file is ipo_out.s (Linux and Mac OS X) or ipo_out.obj (Windows).
- **Use the -ipo-S option to optimize across files and produce an assembly file.** The option performs optimizations as described for -ipo, but stops prior to the final link stage, leaving an optimized assembly file. The default name for this file is ipo_out.s (Linux) or ipo_out.asm (Windows).

For both options, you can use the -o (Linux and Mac OS X) or /Fe (Windows) option to specify a different name.

These options generate multiple outputs if multi-object IPO is being used. The name of the first file is taken from the value of the -o (Linux and Mac OS X) or /Fe (Windows) option.

The names of subsequent files are derived from the first file with an appended numeric value to the file name. For example, if the first object file is named foo.o (Linux and Mac OS X) or foo.obj (Windows), the second object file will be named foo1.o or foo1.obj.

You can use the object file generated with the -ipo-c (Linux and Mac OS X) or /Qipo-c (Windows) option, but you will not get the same benefits from the optimizations applied that would otherwise if the whole program analysis condition had been satisfied.
The object file created using the `-ipo-c` option is a real object file, in contrast to the mock file normally generated using IPO; however, the generated object file is significantly different than the mock object file. It does not contain the IR information needed to fully optimize the application using IPO.

The compiler generates a message indicating the name of each object or assembly file it generates. These files can be added to the real link step to build the final application.

/-auto-ilp32 (Linux) or /Qauto-ilp32 (Windows) for Intel® 64 and IA-64 architectures

On systems based on IA-64 architecture, the `-auto-ilp32` (Linux) or `/Qauto-ilp32` (Windows) option specifies interprocedural analysis over the whole program. The option enables specific interprocedural whole-program analysis and then continues with the next statement. Because this optimization requires interprocedural analysis over the whole program, you must use this option with the `-ipo` (Linux) or `/Qipo` (Windows) option.

On systems based on Intel® 64 architecture, this option has no effect unless `-xSSE3` or `-axSSE3` (Linux and Mac OS X) or `/QxSSE3` or `/QaxSSE3` (Windows) is also specified.

IPO-Related Performance Issues

There are some general optimization guidelines for IPO that you should keep in mind:

- **Large IPO compilations** might trigger internal limits of other compiler optimization phases.
- Combining IPO and PGO can be a key technique for C++ applications. The `-O3`, `-ipo`, and `-prof-use` (Linux* and Mac OS* X) or `/O3`, `/Qipo`, `/Qprof-use` (Windows*) options can result in significant performance gains.
- IPO benefits C more than C++, since C++ compilations include intra-file inlining by default.
- Applications where the compiler does not have sufficient intermediate representation (IR) coverage to do whole program analysis might not perform as well as those where IR information is complete.

In addition to the general guidelines, there are some practices to avoid while using IPO. The following list summarizes the activities to avoid:

- Do not use the link phase of an IPO compilation using mock object files produced for your application by a different compiler. The Intel® Compiler cannot inspect mock object files generated by other compilers for optimization opportunities.
- Do not link mock files with the `-prof-use` (Linux* and Mac OS* X) or `/Qprof-use` (Windows*) option unless the mock files were also compiled with the `-prof-use` (Linux and Mac OS X) or `/Qprof-use` (Windows) option.
• Update make files to call the appropriate Intel linkers when using IPO from scripts. For Linux and Mac OS X, replaces all instances of `ld` with `xild`; for Windows, replace all instances of `link` with `xilink`.

• Update make file to call the appropriate Intel archiver. Replace all instances of `ar` with `xiar`.

**IPO for Large Programs**

In most cases, IPO generates a single object file for the link-time compilation. This behavior is not optimal for very large programs, perhaps even making it impossible to use `-ipo` (Linux* and Mac OS* X) or `/Qipo` (Windows*) on the application.

The compiler provides two methods to avoid this problem. The first method is an automatic size-based heuristic, which causes the compiler to generate multiple object files for large link-time compilations. The second method is to manually instruct the compiler to perform multi-object IPO.

• Use the `-ipoN` (Linux and Mac OS X) or `/QipoN` (Windows) option and pass an integer value in the place of `N`.

• Use the `-ipo-separate` (Linux and Mac OS X) or `/Qipo-separate` (Windows) option.

The number of true object files generated by the link-time compilation is invisible to the user unless either the `-ipo-c` or `-ipo-S` (Linux and Mac OS X) or `/Qipo-c` or `/Qipo-S` (Windows) option is used.

Regardless of the method used, it is best to use the compiler defaults first and examine the results. If the defaults do not provide the expected results then experiment with generating more object files.

You can use the `-ipo-jobs` (Linux and Mac OS X) or `/Qipo-jobs` (Windows) option to control the number of commands, or jobs, executed during parallel builds.

**Using `-ipoN` or `/QipoN` to Create Multiple Object Files**

If you specify `-ipo0` (Linux and Mac OS X) or `/Qipo0` (Windows), which is the same as not specifying a value, the compiler uses heuristics to determine whether to create one or more object files based on the expected size of the application. The compiler generates one object file for small applications, and two or more object files for large applications. If you specify any value greater than 0, the compiler generates that number of object files, unless the value you pass a value that exceeds the number of source files. In that case, the compiler creates one object file for each source file then stops generating object files.
The following example commands demonstrate how to use `-ipo2` (Linux and Mac OS X) or `/Qipo2` (Windows) to compile large programs.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -ipo2 -c a.cpp b.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Qipo2 /c a.cpp b.cpp</td>
</tr>
</tbody>
</table>

Because the example command shown above, the compiler generates object files using an OS-dependent naming convention. On Linux and Mac OS X, the example command results in object files named `ipo_out.o`, `ipo_out1.o`, `ipo_out2.o`, and `ipo_out3.o`. On Windows, the file names follow the same convention; however, the file extensions will be `.obj`.

Link the resulting object files as shown in Using IPO or Linking Tools and Options. Linking Tools and Options.

Creating the Maximum Number of Object Files

Using `-ipo-separate` (Linux and Mac OS X) or `/Qipo-separate` (Windows) allows you to force the compiler to generate the maximum number of true object files that the compiler will support during multiple object compilation.

For example, if you passed example commands similar to the following:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -ipo-separate -ipo-c a.o b.o c.o</td>
</tr>
<tr>
<td>Windows</td>
<td>icl a.obj b.obj c.obj /Qipo-separate /Qipo-c</td>
</tr>
</tbody>
</table>

The compiler will generate multiple object file, which use the same naming convention discussed above. The first object file contains global variables. The other object files contain code for the functions or routines used the source files.

Link the resulting object files as shown in Using IPO or Linking Tools and Options. Linking Tools and Options.
Considerations for Large Program Compilation

For many large programs, compiling with IPO can result in a single, large object file. Compiling
to produce large objects can create problems for efficient compilation. During compilation, the
compiler attempts to swap the memory usage during compiles; a large object file might result
in poor swap usage, which could result in out-of-memory message or long compilation times.
Using multiple, relatively small object files during compilation causes the system to use resources
more efficiently.

Understanding Code Layout and Multi-Object IPO

One of the optimizations performed during an IPO compilation is code layout. The analysis
performed by the compiler during multi-file IPO determines a layout order for all of the routines
for which it has intermediate representation (IR) information. For a multi-object IPO compilation,
the compiler must tell the linker about the desired order.

If you are generating an executable in the link step, the compiler does all of this automatically.
However, if you are generating object files instead of an executable, the compiler generates a
layout script, which contains the correct information needed to optimally link the executable
when you are ready to create it.

This linking tool script must be taken into account if you use either -ipo-c or -ipo-S (Linux*)
or /Qipo-c or /Qipo-S (Windows*). With these options, the IPO compilation and actual linking
are done by different invocations of the compiler. When this occurs, the compiler issues a
message indicating that it is generating an explicit linker script, ipo_layout.script.

The Windows linker (link.exe) automatically collates these sections lexicographically in the desired
order.

The compiler first puts each routine in a named text section that varies depending on the
operating system:

Windows:

- The first routine is placed in .text$00001, the second is placed in .text$00002, and so on.

Linux:

- The first routine is placed in .text00001, the second is placed in .text00002, and so on.
  It then generates a linker script that tells the linker to first link contributions from
  .text00001, then .text00002.
When *ipo_layout.script* is generated, you should modify your link command if you want to use the script to optimize code layout:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--script=ipo_layout.script</code></td>
</tr>
</tbody>
</table>

If your application already requires a custom linker script, you can place the necessary contents of *ipo_layout.script* in your script.

The layout-specific content of *ipo_layout.script* is at the beginning of the description of the `.text` section. For example, to describe the layout order for 12 routines:

<table>
<thead>
<tr>
<th>Example output</th>
</tr>
</thead>
</table>
| `.text :
{
 *(.text00001) *(.text00002) *(.text00003) *(.text00004) *(.text00005)
 *(.text00006) *(.text00007) *(.text00008) *(.text00009) *(.text00010)
 *(.text00011) *(.text00012)
 ...` |

For applications that already require a linker script, you can add this section of the `.text` section description to the customized linker script. If you add these lines to your linker script, it is desirable to add additional entries to account for future development. The addition is harmless, since the "r;*(')" syntax makes these contributions optional.

If you choose to not use the linker script your application will still build, but the layout order will be random. This may have an adverse affect on application performance, particularly for large applications.
Creating a Library from IPO Objects

**Linux* and Mac OS* X**

Libraries are often created using a library manager such as *lib*. Given a list of objects, the library manager will insert the objects into a named library to be used in subsequent link steps.

**Example**

```
xiar cru user.a a.o b.o
```

The above command creates a library named *user.a* containing the *a.o* and *b.o* objects.

If the objects have been created using `-ipo-c` then the archive will not only contain a valid object, but the archive will also contain intermediate representation (IR) for that object file. For example, the following example will produce *a.o* and *b.o* that may be archived to produce a library containing both object code and IR for each source file.

**Example**

```
icc -ipo -c a.cpp b.cpp
```

This program will invoke the compiler on the IR saved in the object file and generate a valid object that can be inserted into a library.

Using *xiar* is the same as specifying *xild* `-lib`.

**Mac OS X Only**

When using *xilibtool*, specify `-static` to generate static libraries, or specify `-dynamiclib` to create dynamic libraries. For example, the following example command will create a static library named *mylib.a* that includes the *a.o*, *b.o*, and *c.o* objects.

**Example**

```
xilibtool -static -o mylib.a a.o b.o c.o
```
Alternately, the following example command will create a dynamic library named `mylib.dylib` that includes the `a.o`, `b.o`, and `c.o` objects.

**Example**

```
xilibtool -dynamic -o mylib.dylib a.o b.o c.o
```

Specifying `xilibtool` is the same as specifying `xild -libtool`.

**Windows* Only**

Create libraries using `xilib` or `xilink -lib` to create libraries of IPO mock object files and link them on the command line.

For example, assume that you create three mock object files by using a command similar to the following:

**Example**

```
icl /c /Qipo a.obj b.obj c.obj
```

Further assume `a.obj` contains the main subprogram. You can enter commands similar to the following to link the objects.

**Example**

```
xilib -out:main.lib b.obj c.obj
```

or

```
xilink -lib -out:main.lib b.obj c.obj
```

You can link the library and the main program object file by entering a command similar to the following:

**Example**

```
xilink -out:result.exe a.obj main.lib
```
Requesting Compiler Reports with the xi* Tools

The compiler option `-opt-report` (Linux* and Mac OS* X) or `/Qopt-report` (Windows*) generates optimization reports with different levels of detail. Related compiler options described in Compiler Reports Quick Reference allow you to specify the phase, direct output to a file (instead of stderr), and request reports from all routines with names containing a string as part of their name.

The xi* tools are used with interprocedural optimization (IPO) during the final stage of IPO compilation. You can request compiler reports to be generated during the final IPO compilation by using certain options. The supported xi* tools are:

- Linker tools: xilink (Windows) and xild (Linux and Mac OS X)
- Library tools: xilib (Windows), xiar (Linux and Mac OS X), xilibtool (Mac OS X)

The following tables lists the compiler report options for the xi* tools. These options are equivalent to the corresponding compiler options, but occur during the final IPO compilation.

<table>
<thead>
<tr>
<th>Tool Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-opt-report[=n]</code></td>
<td>Enables optimization report generation with different levels of detail, directed to stderr. Valid values for n are 0 through 3. By default, when you specify this option without passing a value the compiler will generate a report with a medium level of detail.</td>
</tr>
<tr>
<td><code>-opt-report-file=file</code></td>
<td>Generates an optimization report and directs the report output to the specified file name. If you omit the path, the file is created in the current directory. To create the file in a different directory, specify the full path to the output file and its file name.</td>
</tr>
<tr>
<td><code>-opt-report-phase=name</code></td>
<td>Specifies the optimization phase name to use when generating reports. If you do not specify a phase the compiler defaults to all. You can request a list of all available phase by using the <code>-opt-report-help</code> option.</td>
</tr>
<tr>
<td><code>-opt-report-routine=string</code></td>
<td>Generates reports from all routines with names containing a string as part of their name. If not specified, the compiler will generate reports on all routines.</td>
</tr>
<tr>
<td><code>-opt-report-help</code></td>
<td>Displays the optimization phases available.</td>
</tr>
</tbody>
</table>
To understand the compiler reports, use the links provided in Compiler Reports Overview.

**Inline Expansion of Functions**

**Inline Function Expansion**

Because inline function expansion does not require that the applications meet the criteria for
whole program analysis normally require by IPO, this optimization is one of the primary
optimizations used in Interprocedural Optimization (IPO). For function calls that the compiler
believes are frequently executed, the Intel® compiler often decides to replace the instructions
of the call with code for the function itself.

In the compiler, inline function expansion typically favors relatively small user functions over
functions that are relatively large. This optimization improves application performance by
performing the following:

- Removing the need to set up parameters for a function call
- Eliminating the function call branch
- Propagating constants

Function inlining can improve execution time by removing the runtime overhead of function
calls; however, function inlining can increase code size, code complexity, and compile times.
In general, when you instruct the compiler to perform function inlining, the compiler can examine
the source code in a much larger context, and the compiler can find more opportunities to apply
optimizations.

Specifying `-ip` (Linux* and Mac OS* X) or `/Qip` (Windows*), single-file IP, causes the compiler
to perform inline function expansion for calls to procedures defined within the current source
file; in contrast, specifying `-ipo` (Linux and Mac OS X) or `/Qipo` (Windows), multi-file IPO,
causes the compiler to perform inline function expansion for calls to procedures defined in other
files.

**CAUTION.** Using the `-ip` and `-ipo` (Linux and Mac OS X) or `/Qip` and `/Qipo` (Windows)
options can in some cases significantly increase compile time and code size.

**Selecting Routines for Inlining**

The compiler attempts to select the routines whose inline expansions will provide the greatest
benefit to program performance. The selection is done using the default heuristics. The inlining
heuristics used by the compiler differ based on whether or not you use Profile-Guided
Optimizations (PGO): `-prof-use` (Linux and Mac OS X) or `/Qprof-use` (Windows).
When you use PGO with -ip or -ipo (Linux and Mac OS X) or /Qip or /Qipo (Windows), the compiler uses the following guidelines for applying heuristics:

- The default heuristic focuses on the most frequently executed call sites, based on the profile information gathered for the program.
- The default heuristic always inlines very small functions that meet the minimum inline criteria.

PGO (Windows)

Combining IPO and PGO produces better results than using IPO alone. PGO produces dynamic profiling information that can usually provide better optimization opportunities than the static profiling information used in IPO.

The compiler uses characteristics of the source code to estimate which function calls are executed most frequently. It applies these estimates to the PGO-based guidelines described above. The estimation of frequency, based on static characteristics of the source, is not always accurate.

Avoid using static profile information when combining PGO and IPO; with static profile information, the compiler can only estimate the application performance for the source files being used. Using dynamically generated profile information allows the compiler to accurately determine the real performance characteristics of the application.

Inline Expansion of Library Functions

If one of your functions has the same name as one of the compiler-supplied inline library functions, the compiler might choose to replace the function with inline assembly code. If you have a function call with the same name as one of the supplied inline functions, then instead of calling the user-defined function the compiler assumes that the functions is the well-known function call and will replace the function call with supplied assembly code.

Use the -fno-builtin (Linux and Mac OS X) or /Oi- (Windows) option to ensure that the user-supplied function is used; these options disables inlining of all intrinsics.

Inlining and Function Preemption (Linux)

You must specify -fPIC or -fPIC to use function preemption. By default the compiler does not generate the position-independent code needed for preemption.
## Compiler Directed Inline Expansion of User Functions

Without directions from the user, the compiler attempts to estimate what functions should be inlined to optimize application performance. See [Criteria for Inline Function Expansion](#) for more information.

The following options are useful in situations where an application can benefit from user function inlining but does not need specific direction about inlining limits. Except where noted, these options are supported on IA-32, Intel® 64, and IA-64 architectures.

<table>
<thead>
<tr>
<th><em><em>Linux</em> and Mac OS</em> X**</th>
<th><strong>Windows</strong>*</th>
<th><strong>Effect</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-inline-level</code></td>
<td><code>/Ob</code></td>
<td>Specifies the level of inline function expansion. Depending on the value specified, the option can disable or enable inlining. By default, the option enables inlining of any function if the compiler believes the function can be inlined. For more information, see the following topic:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>/Ob</code> compiler option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>-inline-level</code> compiler option</td>
</tr>
<tr>
<td><code>-ip-no-inlining</code></td>
<td><code>/Qip-no-inlining</code></td>
<td>Disables only inlining normally enabled by the following options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Linux and Mac OS X: <code>-ip</code> or <code>-ipo</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows: <code>/Qip</code>, <code>/Qipo</code>, or <code>/Ob2</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>No other IPO optimization are disabled. For more information, see the following topic:</td>
</tr>
</tbody>
</table>

*IP and MacOS X are Apple's trademarks.*
<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ip-no-pinlining</td>
<td>/Qip-no-pinlining</td>
<td>Disables partial inlining normally enabled by the following options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Linux and Mac OS X: <strong>-ip</strong> or <strong>-ipo</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows: <strong>/Qip</strong> or <strong>/Qipo</strong></td>
</tr>
<tr>
<td>-fno-builtin</td>
<td>/Oi-</td>
<td>No other IPO optimization are disabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see the following topic:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>/Qip-no-pinlining</strong> compiler option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>-ip-no-pinlining</strong> compiler option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disables inlining for intrinsic functions. Disables the by-name recognition support of intrinsic functions and the resulting optimizations. Use this option if you redefine standard library routines with your own version and your version of the routine has the same name as the standard library routine.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>By default, the compiler automatically inlines (expands) a number of standard and math library functions at the point of the</td>
</tr>
</tbody>
</table>
For more information, see the following topic:

- **-fno-builtin compiler option**

Keeps source information for inlined functions. The additional source code can be used by the Intel® Debugger to track the user-defined call stack while using inlining.

To use this option you must also specify an additional option to enable debugging:

- **Linux:** `-g`
- **Mac OS X:** This option is not supported.
- **Windows:** `/debug`

For more information, see the following topic:

- `/Qinline-debug-info` compiler option
- `-inline-debug-info` compiler option

### Developer Directed Inline Expansion of User Functions

In addition to the options that support compiler directed inline expansion of user functions, the compiler also provides compiler options that allow you to more precisely direct when and if inline function expansion occurs.
The compiler measures the relative size of a routine in an abstract value of intermediate language units, which is approximately equivalent to the number of instructions that will be generated. The compiler uses the intermediate language unit estimates to classify routines and functions as relatively small, medium, or large functions. The compiler then uses the estimates to determine when to inline a function; if the minimum criteria for inlining is met and all other things are equal, the compiler has an affinity for inlining relatively small functions and not inlining relative large functions.

The following developer directed inlining options provide the ability to change the boundaries used by the inlining optimizer to distinguish between small and large functions. These options are supported on IA-32, Intel® 64, and IA-64 architectures.

In general, you should use the **-inline-factor** (Linux* and Mac OS* X) and **/Qinline-factor** (Windows*) option before using the individual inlining options listed below; this single option effectively controls several other upper-limit options.

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-inline-factor</strong></td>
<td><strong>/Qinline-factor</strong></td>
<td>Controls the multiplier applied to all inlining options that define upper limits: <code>inline-max-size</code>, <code>inline-max-total-size</code>, <code>inline-max-per-routine</code>, and <code>inline-max-per-compile</code>. While you can specify an individual increase in any of the upper-limit options, this single option provides an efficient means of controlling all of the upper-limit options with a single command. By default, this option uses a multiplier of 100, which corresponds to a factor of 1. Specifying 200 implies a factor of 2, and so on. Experiment with the multiplier carefully. You could increase the upper limits to allow too much inlining, which might result in your system running out of memory.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
</table>
| -inline-forceinline | /Qinline-forceinline | For more information, see the following topic:  
  - /Qinline-factor  
  - /Qinline-factor  

Instructs the compiler to force inlining of functions suggested for inlining whenever the compiler is capable doing so. Typically, the compiler targets functions that have been marked for inlining based on the presence of keywords, like `__forceinline`, in the source code; however, all such directives in the source code are treated only as suggestions for inlining. The option instructs the compiler to view the inlining suggestion as mandatory and inline the marked function if it can be done legally.  

For more information, see the following topic:  
  - /Qinline-forceinline compiler option  
  - /Qinline-forceinline compiler option  

<p>| -inline-min-size | /Qinline-min-size | Redefines the maximum size of small routines; routines that are equal to or smaller than the value specified are more likely to be inlined. |</p>
<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{-inline-max-size}</td>
<td>\texttt{/Qinline-max-size}</td>
<td>Redefines the minimum size of large routines; routines that are equal to or larger than the value specified are less likely to be inlined.</td>
</tr>
<tr>
<td>\texttt{-inline-max-total-size}</td>
<td>\texttt{/Qinline-max-total-size}</td>
<td>Limits the expanded size of inlined functions.</td>
</tr>
<tr>
<td>\texttt{-inline-max-per-routine}</td>
<td>\texttt{/Qinline-max-per-routine}</td>
<td>Limits the number of times inlining can be applied within a routine.</td>
</tr>
</tbody>
</table>

For more information, see the following topic:

- \texttt{/Qinline-min-size} compiler option
- \texttt{-inline-min-size} compiler option
- \texttt{/Qinline-max-size} compiler option
- \texttt{-inline-max-size} compiler option
- \texttt{/Qinline-max-total-size} compiler option
- \texttt{-inline-max-total-size} compiler option
- \texttt{/Qinline-max-per-routine} compiler option
- \texttt{-inline-max-per-routine} compiler option
<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Qinline-max-per-routine compiler option</td>
<td>/Qinline-max-per-compile /Qinline-max-per-compile</td>
<td>Limits the number of times inlining can be applied within a compilation unit. The compilation unit limit depends on the whether or not you specify the -ipo (Linux and Mac OS X) or /Qipo (Windows) option. If you enable IPO, all source files that are part of the compilation are considered one compilation unit. For compilations not involving IPO each source file is considered an individual compilation unit. For more information, see the following topic:</td>
</tr>
<tr>
<td>-inline-max-per-compile</td>
<td>/Qinline-max-per-compile</td>
<td>-inline-max-per-routine compiler option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-inline-max-per-compile compiler option</td>
</tr>
</tbody>
</table>
Profile-Guided Optimizations Overview

Profile-guided Optimization (PGO) improves application performance by reorganizing code layout to reduce instruction-cache problems, shrinking code size, and reducing branch mispredictions. PGO provides information to the compiler about areas of an application that are most frequently executed. By knowing these areas, the compiler is able to be more selective and specific in optimizing the application.

PGO consists of three phases or steps.

1. Step one is to instrument the program. In this phase, the compiler creates and links an instrumented program from your source code and special code from the compiler.

2. Step two is to run the instrumented executable. Each time you execute the instrumented code, the instrumented program generates a dynamic information file, which is used in the final compilation.

3. Step three is a final compilation. When you compile a second time, the dynamic information files are merged into a summary file. Using the summary of the profile information in this file, the compiler attempts to optimize the execution of the most heavily traveled paths in the program.

See Profile-guided Optimization Quick Reference for information about the supported options and Profile an Application for specific details about using PGO from a command line.

PGO enables the compiler to take better advantage of the processor architecture, more effective use of instruction paging and cache memory, and make better branch predictions. PGO provides the following benefits:

- Use profile information for register allocation to optimize the location of spill code.
- Improve branch prediction for indirect function calls by identifying the most likely targets. (Some processors have longer pipelines, which improves branch prediction and translates into high performance gains.)
- Detect and do not vectorize loops that execute only a small number of iterations, reducing the run time overhead that vectorization might otherwise add.

Interprocedural Optimization (IPO) and PGO can affect each other; using PGO can often enable the compiler to make better decisions about function inlining, which increases the effectiveness of interprocedural optimizations. Unlike other optimizations, such as those strictly for size or speed, the results of IPO and PGO vary. This variability is due to the unique characteristics of each program, which often include different profiles and different opportunities for optimizations.
Performance Improvements with PGO

PGO works best for code with many frequently executed branches that are difficult to predict at compile time. An example is the code with intensive error-checking in which the error conditions are false most of the time. The infrequently executed (cold) error-handling code can be relocated so the branch is rarely predicted incorrectly. Minimizing cold code interleaved into the frequently executed (hot) code improves instruction cache behavior.

When you use PGO, consider the following guidelines:

- Minimize changes to your program after you execute the instrumented code and before feedback compilation. During feedback compilation, the compiler ignores dynamic information for functions modified after that information was generated. (If you modify your program the compiler issues a warning that the dynamic information does not correspond to a modified function.)

- Repeat the instrumentation compilation if you make many changes to your source files after execution and before feedback compilation.

- Know the sections of your code that are the most heavily used. If the data set provided to your program is very consistent and displays similar behavior on every execution, then PGO can probably help optimize your program execution.

- Different data sets can result in different algorithms being called. The difference can cause the behavior of your program to vary for each execution. In cases where your code behavior differs greatly between executions, PGO may not provide noticeable benefits. If it takes multiple data sets to accurately characterize application performance then execute the application with all data sets then merge the dynamic profiles; this technique should result in an optimized application.

You must insure the benefit of the profiled information is worth the effort required to maintain up-to-date profiles.

Profile-Guided Optimization (PGO) Quick Reference

Profile-Guided Optimization consists of three phases (or steps):

1. Generating instrumented code by compiling with the `-prof-gen` (Linux* OS and Mac OS* X) or `/Qprof-gen` (Windows* OS) option when creating the instrumented executable.

2. Running the instrumented executable, which produces dynamic-information (.dyn) files.

3. Compiling the application using the profile information using the `-prof-use` (Linux and Mac OS X) or `/Qprof-use` (Windows) option.
The figure illustrates the phases and the results of each phase.

See Profile an Application for details about using each phase.

The following table lists the compiler options used in PGO:

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>-prof-gen</td>
<td>/Qprof-gen</td>
<td>Instruments a program for profiling to get the execution counts of each basic block. The option is used in phase 1 (instrumenting the code) to instruct the compiler to produce instrumented code for your object files in preparation for instrumented execution. By default, each instrumented execution creates one dynamic-information (dyn) file for each executable and (on Windows OS) one for each DLL invoked by the application. You can specify keywords, such as -prof-gen=default (Linux and Mac OS X) or /Qprof-gen:default (Windows). The keywords control the amount of source information gathered during phase 2 (run the instrumented executable). The prof-gen keywords are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Specify default (or omit the keyword) to request profiling information for use with the prof-use option and optimization</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Effect</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>-prof-use</td>
<td>/Qprof-use</td>
<td>when the instrumented application is run (phase 2).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Specify srcpos or globdata to request additional profiling information for the code coverage and test prioritization tools when the instrumented application is run (phase 2). The phase 1 compilation creates an spi file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Specify globdata to request additional profiling information for data ordering optimization when the instrumented application is run (phase 2). The phase 1 compilation creates an spi file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you are performing a parallel make, this option will not affect it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Instructs the compiler to produce a profile-optimized executable and merges available dynamic-information (dyn) files into a pgopti.dpi file. This option implicitly invokes the profmerge tool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The dynamic-information files are produced in phase 2 when you run the instrumented executable.</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Effect</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you perform multiple executions of the instrumented program to create additional dynamic-information files that are newer than the current summary pgopti.dpi file, this option merges the dynamic-information files again and overwrites the previous pgopti.dpi file (you can set the environment variable PROF_NO_CLOBBER to prevent the previous dpi file from being overwritten). When you compile with prof-use, all dynamic information and summary information files should be in the same directory (current directory or the directory specified by the prof-dir option). If you need to use certain profmerge options not available with compiler options (such as specifying multiple directories), use the profmerge tool. For example, you can use profmerge to create a new summary dpi file before you compile with the prof-use option to create the optimized application. You can specify keywords, such as -prof-gen=weighted (Linux and Mac OS X) or /Qprof-gen:weighted (Windows). If you omit the weighted keyword, the merged dynamic-information</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Effect</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(dyn) files will be weighted proportionally to the length of time each application execution runs. If you specify the weighted keyword, the profiler applies an equal weighting (regardless of execution times) to the dyn file values to normalize the data counts. This keyword is useful when the execution runs have different time durations and you want them to be treated equally.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When you use prof-use, you can also specify the prof-file option to rename the summary dpi file and the prof-dir option to specify the directory for dynamic-information (dyn) and summary (dpi) files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Using this option with -prof-func-groups allows you to control function grouping behavior.</td>
</tr>
<tr>
<td>-no-fnsplit</td>
<td>/Qfnsplit-</td>
<td>Disables function splitting. Function splitting is enabled by the prof-use option in phase 3 to improve code locality by splitting routines into different sections: one section to contain the cold or very infrequently executed (cold) code, and one section to contain the rest of the frequently executed (hot) code.</td>
</tr>
</tbody>
</table>
code. You may want to disable function splitting for the following reasons:

- Improve debugging capability. In the debug symbol table, it is difficult to represent a split routine, that is, a routine with some of its code in the hot code section and some of its code in the cold code section.

- Account for the cases when the profile data does not represent the actual program behavior, that is, when the routine is actually used frequently rather than infrequently.

This option is supported on IA-32 architecture for Windows OS and on IA-64 architecture for Windows and Linux OS. It is not supported on other platforms (Intel® 64 architecture, Mac OS X, and Linux on IA-32 architecture).

Windows: This option behaves differently on systems based on IA-32 architecture than it does on systems based on IA-64 architecture.

IA-32 architecture, Windows OS:

- The option completely disables function splitting, placing all the code in one section.

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>code. You may want</td>
<td></td>
<td>to</td>
</tr>
<tr>
<td>disable function</td>
<td></td>
<td>splitting for</td>
</tr>
<tr>
<td>splitting for</td>
<td></td>
<td>the following reasons:</td>
</tr>
<tr>
<td>the following</td>
<td></td>
<td>reasons:</td>
</tr>
<tr>
<td>reasons:</td>
<td></td>
<td>- Improve debugging</td>
</tr>
<tr>
<td>- Improve debugging</td>
<td></td>
<td>capability. In the debug</td>
</tr>
<tr>
<td>capability. In the</td>
<td></td>
<td>symbol table, it is difficult</td>
</tr>
<tr>
<td>symbol table, it is</td>
<td></td>
<td>to represent a split</td>
</tr>
<tr>
<td>is difficult to</td>
<td></td>
<td>routine, that is, a routine</td>
</tr>
<tr>
<td>represent a split</td>
<td></td>
<td>with some of its code in</td>
</tr>
<tr>
<td>routine, that is,</td>
<td></td>
<td>the hot code section and</td>
</tr>
<tr>
<td>a routine with some</td>
<td></td>
<td>some of its code in the</td>
</tr>
<tr>
<td>of its code in the</td>
<td></td>
<td>cold code section.</td>
</tr>
<tr>
<td>cold code section.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Account for the</td>
<td></td>
<td>the cases when the profile</td>
</tr>
<tr>
<td>cases when the</td>
<td></td>
<td>data does not represent the</td>
</tr>
<tr>
<td>profile data does</td>
<td></td>
<td>actual program behavior, that</td>
</tr>
<tr>
<td>does not represent</td>
<td></td>
<td>is, when the routine is</td>
</tr>
<tr>
<td>the actual</td>
<td></td>
<td>actually used frequently</td>
</tr>
<tr>
<td>program behavior,</td>
<td></td>
<td>rather than infrequently.</td>
</tr>
<tr>
<td>when the profile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data does not</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This option is</td>
<td></td>
<td>supported on</td>
</tr>
<tr>
<td>supported on</td>
<td></td>
<td>IA-32 architecture for</td>
</tr>
<tr>
<td>IA-32 architecture</td>
<td></td>
<td>Windows OS and on IA-64</td>
</tr>
<tr>
<td>for Windows OS and</td>
<td></td>
<td>architecture for Windows and</td>
</tr>
<tr>
<td>for Windows and</td>
<td></td>
<td>Linux OS. It is not supported</td>
</tr>
<tr>
<td>Linux OS. It is</td>
<td></td>
<td>on other platforms (Intel® 64</td>
</tr>
<tr>
<td>not supported on</td>
<td></td>
<td>architecture, Mac OS X, and</td>
</tr>
<tr>
<td>other platforms (</td>
<td></td>
<td>Linux on IA-32 architecture).</td>
</tr>
<tr>
<td>Intel® 64 architecture, Mac OS X, and</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux on IA-32</td>
<td></td>
<td>architecture).</td>
</tr>
<tr>
<td>architecture).</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows: This option</td>
<td></td>
<td>behaves differently on</td>
</tr>
<tr>
<td>behaves differently</td>
<td></td>
<td>systems based on IA-32</td>
</tr>
<tr>
<td>differently on</td>
<td></td>
<td>architecture than it does on</td>
</tr>
<tr>
<td>systems based on</td>
<td></td>
<td>systems based on IA-64</td>
</tr>
<tr>
<td>systems based on</td>
<td></td>
<td>architecture.</td>
</tr>
<tr>
<td>IA-32 architecture,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows OS:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The option completely</td>
<td></td>
<td></td>
</tr>
<tr>
<td>disables function</td>
<td></td>
<td>splitting, placing all the</td>
</tr>
<tr>
<td>splitting, placing</td>
<td></td>
<td>code in one section.</td>
</tr>
<tr>
<td>all the code in one</td>
<td></td>
<td>section.</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Effect</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>--------</td>
</tr>
</tbody>
</table>
| **-prof-func-groups** | **/Qprof-func-order** | IA-64 architecture, Linux and Windows OS:  
- The option disables the splitting within a routine but enables function grouping, an optimization in which entire routines are placed either in the cold code section or the hot code section. Function grouping does not degrade debugging capability.  

Enables ordering of program routines using profile information when specified with **prof-use** *(phase 3)*. The instrumented program *(phase 1)* must have been compiled with the **prof-gen** option **srcpos** keyword. Not valid for multi-file compilation with the **ipo** options.  

Mac OS X: Not supported.  
IA-64 architecture: Not supported.  
For more information, see *Using Function Ordering, Function Order Lists, Function Grouping, and Data Ordering Optimizations*.  

**-prof-data-order** | **/Qprof-data-order** | Enables ordering of static program data items based on profiling information when specified with **prof-use**. The instrumented program *(phase 1)* must have been compiled |
<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Effect</th>
</tr>
</thead>
</table>
| **-prof-src-dir**   | **/Qprof-src-dir** | with the `prof-gen` option `srcpos` keyword. Not valid for multi-file compilation with the `ipo` options.  

Mac OS X: Not supported.  

For more information, see [Using Function Ordering, Function Order Lists, Function Grouping, and Data Ordering Optimizations](#).  

Controls whether full directory information for the source directory path is stored in or read from dynamic-information (dyn) files. When used during phase 1 compilation (`prof-gen`), this determines whether the full path is added into dyn file created during instrumented application execution. When used during `profmerge` or phase 3 compilation (`prof-use`), this determines whether the full path for source file names is used or ignored when reading the dyn or dpi files.  

Using the default `-prof-src-dir` (Linux and Mac OS X) or `/Qprof-src-dir` (Windows) uses the full directory information and also enables the use of the `prof-src-root` and `prof-src-cwd` options.  

---

1303
<table>
<thead>
<tr>
<th><strong>Linux</strong> and Mac OS X</th>
<th><strong>Windows</strong></th>
<th><strong>Effect</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-prof-dir</strong></td>
<td><strong>/Qprof-dir</strong></td>
<td>If you specify -no-prof-src-dir (Linux and Mac OS X) or /Qprof-src-dir (Windows), only the file name (and not the full path) is stored or used. If you do this, all dyn or dpi files must be in the current directory and the prof-src-root and prof-src-cwd options are ignored.</td>
</tr>
<tr>
<td><strong>-prof-src-root or</strong></td>
<td><strong>/Qprof-src-root</strong></td>
<td>Specifies the directory in which dynamic information (dyn) files are created in, read from, and stored; otherwise, the dyn files are created in or read from the current directory used during compilation. For example, you can use this option when compiling in phase 1 (prof-gen option) to define where dynamic information files will be created when running the instrumented executable in phase 2. You also can use this option when compiling in phase 3 (prof-use option) to define where the dynamic information files will be read from and a summary file (dpi) created.</td>
</tr>
<tr>
<td><strong>-prof-src-cwd</strong></td>
<td><strong>/Qprof-src-cwd</strong></td>
<td>Specifies a directory path prefix for the root directory where the user's application files are stored:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• To specify the directory prefix root where source files are stored, specify</td>
</tr>
<tr>
<td><em><em>Linux</em> and Mac OS</em> X**</td>
<td><strong>Windows</strong>*</td>
<td><strong>Effect</strong></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the <code>-prof-src-root</code> (Linux and Mac OS X) or <code>/Qprof-src-root</code> (Windows) option.</td>
</tr>
<tr>
<td><code>-prof-file</code></td>
<td><code>/Qprof-file</code></td>
<td>To use the current working directory, specify the <code>-prof-src-cwd</code> (Linux and Mac OS X) or <code>/Qprof-src-cwd</code> (Windows) option. This option is ignored if you specify <code>-no-prof-src-dir</code> (Linux and Mac OS X) or <code>/Qprof-src-dir</code> (Windows).</td>
</tr>
<tr>
<td><code>-prof-gen-sampling</code></td>
<td><code>/Qprof-gen-sampling</code></td>
<td>Specifies file name for profiling summary file. If this option is not specified, the name of the file containing summary information will be <code>pgopti.dpi</code>.</td>
</tr>
<tr>
<td><code>-ssp</code></td>
<td><code>/Qssp</code></td>
<td>IA-32 architecture. Prepares application executables for hardware profiling (sampling) and causes the compiler to generate source code mapping information. Mac OS X: This option is not supported.</td>
</tr>
</tbody>
</table>

Refer to [Quick Reference Lists](#) for a complete listing of the quick reference topics.
Profile an Application

Profiling an application includes the following three phases:

- Instrumentation compilation and linking
- Instrumented execution
- Feedback compilation

This topic provides detailed information on how to profile an application by providing sample commands for each of the three phases (or steps).

1. Instrumentation compilation and linking

Use `-prof-gen` (Linux* and Mac OS* X) or `/Qprof-gen` (Windows*) to produce an executable with instrumented information included.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -prof-gen -prof-dir/usr/profiled a1.cpp a2.cpp a3.cpp</td>
</tr>
<tr>
<td></td>
<td>icpc a1.o a2.o a3.o</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Qprof-gen /Qprof-dir:\profiled a1.cpp a2.cpp a3.cpp</td>
</tr>
<tr>
<td></td>
<td>icl a1.obj a2.obj a3.obj</td>
</tr>
</tbody>
</table>

Use the `-prof-dir` (Linux and Mac OS X) or `/Qprof-dir` (Windows) option if the application includes the source files located in multiple directories; using the option insures the profile information is generated in one consistent place. The example commands demonstrate how to combine these options on multiple sources.

The compiler gathers extra information when you use the `-prof-gen=srcpos` (Linux and Mac OS X) or `/Qprof-gen:srcpos` (Windows) option; however, the extra information is collected to provide support only for specific Intel tools, like the code-coverage tool. If you do not expect to use such tools, do not specify `-prof-gen=srcpos` (Linux and Mac OS X) or `/Qprof-gen:srcpos` (Windows); the extended option does not provide better optimization and could slow parallel compile times.

2. Instrumented execution
Run your instrumented program with a representative set of data to create one or more dynamic information files.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>./a1.out</td>
</tr>
<tr>
<td>Windows</td>
<td>a1.exe</td>
</tr>
</tbody>
</table>

Executing the instrumented applications generates dynamic information file that has a unique name and .dyn suffix. A new dynamic information file is created every time you execute the instrumented program.

You can run the program more than once with different input data.

3. Feedback compilation

Before this step, copy all .dyn and .dpi files into the same directory. Compile and link the source files with -prof-use (Linux and Mac OS X) or /Qprof-use (Windows); the option instructs the compiler to use the generated dynamic information to guide the optimization:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>icpc -prof-use -ipo</td>
</tr>
<tr>
<td></td>
<td>-prof-dir/usr/profiled</td>
</tr>
<tr>
<td></td>
<td>a1.cpp a2.cpp a3.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Qprof-use /Qipo</td>
</tr>
<tr>
<td></td>
<td>/Qprof-dirc:\profiled</td>
</tr>
<tr>
<td></td>
<td>a1.cpp a2.cpp a3.cpp</td>
</tr>
</tbody>
</table>

This final phase compiles and links the sources files using the data from the dynamic information files generated during instrumented execution (phase 2).

In addition to the optimized executable, the compiler produces a pgopti.dpi file.

Most of the time, you should specify the default optimizations, -02 (Linux and Mac OS X) or /O2 (Windows), for phase 1, and specify more advanced optimizations, -ipo (Linux) or /Qipo (Windows), during the final (phase 3) compilation. For example, the example shown above used -02 (Linux and Mac OS X) or /O2 (Windows) in step 1 and -ipo (Linux or Mac OS X) or /Qipo (Windows) in step 3.

**NOTE.** The compiler ignores the -ipo or -ip (Linux and Mac OS X) or /Qipo or /Qip (Windows) option during phase 1 with -prof-gen (Linux and Mac OS X) or /Qprof-gen (Windows).
PGO Tools

PGO Tools Overview
This section describes the tools that take advantage or support the Profile-guided Optimizations (PGO) available in the compiler.

• code coverage tool
• test prioritization tool
• profmerge tool
• proforder tool

In addition to the tools, this section also contains information on using Software-based Speculative Precomputation, which will allow you to optimize applications using profiling- and sampling-feedback methods on IA-32 architectures.

code coverage Tool
The code coverage tool provides software developers with a view of how much application code is exercised when a specific workload is applied to the application. To determine which code is used, the code coverage tool uses Profile-guided Optimization (PGO) options and optimizations. The major features of the code coverage tool are:

• Visually presenting code coverage information for an application with a customizable code coverage coloring scheme
• Displaying dynamic execution counts of each basic block of the application
• Providing differential coverage, or comparison, profile data for two runs of an application

The information about using the code coverage tool is separated into the following sections:

• code coverage tool Requirements
• Visually Presenting Code Coverage for an Application
• Excluding Code from Coverage Analysis
• Exporting Coverage Data

The tool analyzes static profile information generated by the compiler, as well as dynamic profile information generated by running an instrumented form of the application binaries on the workload. The tool can generate the in HTML-formatted report and export data in both text-, and XML-formatted files. The reports can be further customized to show color-coded, annotated, source-code listings that distinguish between used and unused code.
The code coverage tool is available on IA-32, Intel® 64, and IA-64 architectures on Linux* and Windows*. The tool is available on IA-32 and Intel® 64 architectures on Mac OS* X.

You can use the tool in a number of ways to improve development efficiency, reduce defects, and increase application performance:

- During the project testing phase, the tool can measure the overall quality of testing by showing how much code is actually tested.
- When applied to the profile of a performance workload, the code coverage tool can reveal how well the workload exercises the critical code in an application. High coverage of performance-critical modules is essential to taking full advantage of the Profile-Guided Optimizations that Intel Compilers offer.
- The tool provides an option, useful for both coverage and performance tuning, enabling developers to display the dynamic execution count for each basic block of the application.
- The code coverage tool can compare the profile of two different application runs. This feature can help locate portions of the code in an application that are unrevealed during testing but are exercised when the application is used outside the test space, for example, when used by a customer.

**code coverage tool Requirements**

To run the code coverage tool on an application, you must have following items:

- The application sources.
- The .spi file generated by the Intel® compiler when compiling the application for the instrumented binaries using the -prof-gen=srcpos (Linux and Mac OS X) or /Qprof-gen:srcpos (Windows) options.
- A pgopti.dpi file that contains the results of merging the dynamic profile information (.dyn) files, which is most easily generated by the profmerge tool. This file is also generated implicitly by the Intel® compilers when compiling an application with -prof-use (Linux and Mac OS X) or /Qprof-use (Windows) options with available .dyn and .dpi files.

See Understanding Profile-guided Optimization and Example of Profile-guided Optimization for general information on creating the files needed to run this tool.

**Using the Tool**

The tool uses the following syntax:

<table>
<thead>
<tr>
<th>Tool Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>codecov [-codecov_option]</code></td>
</tr>
</tbody>
</table>
where -codecov_option is one or more optional parameters specifying the tool option passed to the tool. The available tool options are listed in code coverage tools Options (below). If you do not use any additional tool options, the tool will provide the top-level code coverage for the entire application.

In general, you must perform the following steps to use the code coverage tool:

1. Compile the application using -prof-gen=srcpos (Linux and Mac OS X) or /Qprof-gen:srcpos (Windows).
   This step generates an instrumented executable and a corresponding static profile information (pgopti.spi) file.

2. Run the instrumented application.
   This step creates the dynamic profile information (.dyn) file. Each time you run the instrumented application, the compiler generates a unique .dyn file either in the current directory or the directory specified by prof_dir.

3. Use the profmerge tool to merge all the .dyn files into one .dpi (pgopti.dpi) file.
   This step consolidates results from all runs and represents the total profile information for the application, generates an optimized binary, and creates the dpi file needed by the code coverage tool.
   You can use the profmerge tool to merge the .dyn files into a .dpi file without recompiling the application. The profmerge tool can also merge multiple .dpi files into one .dpi file using the profmerge -a option. Select the name of the output .dpi file using the profmerge -prof_dpi option.

   CAUTION. The profmerge tool merges all .dyn files that exist in the given directory. Make sure unrelated .dyn files, which may remain from unrelated runs, are not present. Otherwise, the profile information will be skewed with invalid profile data, which can result in misleading coverage information and adverse performance of the optimized code.

4. Run the code coverage tool. (The valid syntax and tool options are shown below.)
   This step creates a report or exported data as specified. If no other options are specified, the code coverage tool creates a single HTML file (CODE_COVERAGE.HTML) and a sub-directory (CodeCoverage) in the current directory. Open the file in a web browser to view the reports.
NOTE. Windows* only: Unlike the compiler options, which are preceded by forward slash ("/"), the tool options are preceded by a hyphen ("-").

The code coverage tool allows you to name the project and specify paths to specific, necessary files. The following example demonstrates how to name a project and specify .dpi and .spi files to use:

<table>
<thead>
<tr>
<th>Example: specify .dpi and .spi files</th>
</tr>
</thead>
<tbody>
<tr>
<td>codecov -prj myProject -spi ggopti.spi -dpi ggopti.dpi</td>
</tr>
</tbody>
</table>

The tool can add a contact name and generate an email link for that contact at the bottom of each HTML page. This provides a way to send an electronic message to the named contact. The following example demonstrates how to add specify a contact and the email links:

<table>
<thead>
<tr>
<th>Example: add contact information</th>
</tr>
</thead>
<tbody>
<tr>
<td>codecov -prj myProject -mname JoeSmith -maddr <a href="mailto:js@company.com">js@company.com</a></td>
</tr>
</tbody>
</table>

This following example demonstrates how to use the tool to specify the project name, specify the dynamic profile information file, and specify the output format and file name.

<table>
<thead>
<tr>
<th>Example: export data to text</th>
</tr>
</thead>
<tbody>
<tr>
<td>codecov -prj test1 -dpi test1.dpi -txtbcvrg test1_bcvrg.txt</td>
</tr>
</tbody>
</table>

code coverage tool Options

The tool uses the options listed in the table:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-bcolor color</td>
<td>#FFFF99</td>
<td>Specifies the HTML color name for code in the uncovered blocks.</td>
</tr>
<tr>
<td>-beginblkdsbl string</td>
<td></td>
<td>Specifies the comment that marks the beginning of the code fragment to be ignored by the coverage tool.</td>
</tr>
<tr>
<td>Option</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-ccolor color</td>
<td>#FFFFFF</td>
<td>Specifies the HTML color name or code of the covered code.</td>
</tr>
<tr>
<td>-comp file</td>
<td></td>
<td>Specifies the file name that contains the list of files being (or not) displayed.</td>
</tr>
<tr>
<td>-counts</td>
<td></td>
<td>Generates dynamic execution counts.</td>
</tr>
<tr>
<td>-demang</td>
<td></td>
<td>Demangles both function names and their arguments.</td>
</tr>
<tr>
<td>-dpi file</td>
<td>pgopti.dpi</td>
<td>Specifies the file name of the dynamic profile information file (.dpi).</td>
</tr>
<tr>
<td>-endblkdsbl string</td>
<td></td>
<td>Specifies the comment that marks the end of the code fragment to be ignored by the coverage tool.</td>
</tr>
<tr>
<td>-fcolor color</td>
<td>#FFCCCC</td>
<td>Specifies the HTML color name for code of the uncovered functions.</td>
</tr>
<tr>
<td>-help, -h</td>
<td></td>
<td>Prints tool option descriptions.</td>
</tr>
<tr>
<td>-icolor color</td>
<td>#FFFFFF</td>
<td>Specifies the HTML color name or code of the information lines, such as basic-block markers and dynamic counts.</td>
</tr>
<tr>
<td>-maddr string</td>
<td>Nobody</td>
<td>Sets the email address of the web-page owner.</td>
</tr>
<tr>
<td>Option</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-mname string</td>
<td>Nobody</td>
<td>Sets the name of the web-page owner.</td>
</tr>
<tr>
<td>-nopartial</td>
<td></td>
<td>Treats partially covered code as fully covered code.</td>
</tr>
<tr>
<td>-nopmeter</td>
<td></td>
<td>Turns off the progress meter. The meter is enabled by default.</td>
</tr>
<tr>
<td>-onelinedabl</td>
<td></td>
<td>Specifies the comment that marks individual lines of code or the whole functions to be ignored by the coverage tool.</td>
</tr>
<tr>
<td>-pcolor color</td>
<td>#FAFAD2</td>
<td>Specifies the HTML color name or code of the partially covered code.</td>
</tr>
<tr>
<td>-prj string</td>
<td></td>
<td>Sets the project name.</td>
</tr>
<tr>
<td>-ref</td>
<td></td>
<td>Finds the differential coverage with respect to ref_dpi_file.</td>
</tr>
<tr>
<td>-spi file</td>
<td>pgopti.spi</td>
<td>Specifies the file name of the static profile information file (.spi).</td>
</tr>
<tr>
<td>-srcroot dir</td>
<td></td>
<td>Specifies a different top level project directory than was used during compiler instrumentation run to use for relative paths to source files in place of absolute paths.</td>
</tr>
<tr>
<td>Option</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-txtbcvrg file</td>
<td></td>
<td>Export block-coverage for covered functions as text format. The file parameter must be in the form of a valid file name.</td>
</tr>
<tr>
<td>-txtbcvrgfull file</td>
<td></td>
<td>Export block-coverage for entire application in text and HTML formats. The file parameter must be in the form of a valid file name.</td>
</tr>
<tr>
<td>-txtdcg file</td>
<td></td>
<td>Generates the dynamic call-graph information in text format. The file parameter must be in the form of a valid file name.</td>
</tr>
<tr>
<td>-txtfcvrg file</td>
<td></td>
<td>Export function coverage for covered function in text format. The file parameter must by in the form of a valid file name.</td>
</tr>
<tr>
<td>-ucolor color</td>
<td>#FFFFFF</td>
<td>Specifies the HTML color name or code of the unknown code.</td>
</tr>
<tr>
<td>-xcolor color</td>
<td>#90EE90</td>
<td>Specifies the HTML color of the code ignored.</td>
</tr>
<tr>
<td>-xmlbcvrg file</td>
<td></td>
<td>Export the block-coverage for the covered function in XML format. The file parameter must by in the form of a valid file name.</td>
</tr>
<tr>
<td>Option</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-xmlbcvrgfull</td>
<td>file</td>
<td>Export function coverage for entire application in XML format in addition to HTML output. The file parameter must be in the form of a valid file name.</td>
</tr>
<tr>
<td>-xmlfcvrg</td>
<td>file</td>
<td>Export function coverage for covered function in XML format. The file parameter must be in the form of a valid file name.</td>
</tr>
</tbody>
</table>

**Visually Presenting Code Coverage for an Application**

Based on the profile information collected from running the instrumented binaries when testing an application, the Intel® compiler will create HTML-formatted reports using the code coverage tool. These reports indicate portions of the source code that were or were not exercised by the tests. When applied to the profile of the performance workloads, the code coverage information shows how well the training workload covers the application’s critical code. High coverage of performance-critical modules is essential to taking full advantage of the profile-guided optimizations.

The code coverage tool can create two levels of coverage:

- Top level (for a group of selected modules)
- Individual module source views

**Top Level Coverage**

The top-level coverage reports the overall code coverage of the modules that were selected. The following options are provided:

- Select the modules of interest
- For the selected modules, the tool generates a list with their coverage information. The information includes the total number of functions and blocks in a module and the portions that were covered.
• By clicking on the title of columns in the reported tables, the lists may be sorted in ascending or descending order based on:
  • basic block coverage
  • function coverage
  • function name

By default, the code coverage tool generates a single HTML file (CODE_COVERAGE.HTML) and a subdirectory (CodeCoverage) in the current directory. The HTML file defines a frameset to display all of the other generated reports. Open the HTML file in a web-browser. The tool places all other generated report files in a CodeCoverage subdirectory.

If you choose to generate the html-formatted version of the report, you can view coverage source of that particular module directly from a browser. The following figure shows the top-level coverage report.

The coverage tool creates a frame set that allows quick browsing through the code to identify uncovered code. The top frame displays the list of uncovered functions while the bottom frame displays the list of covered functions. For uncovered functions, the total number of basic blocks of each function is also displayed. For covered functions, both the total number of blocks and the number of covered blocks as well as their ratio (that is, the coverage rate) are displayed.

For example, 66.67(4/6) indicates that four out of the six blocks of the corresponding function were covered. The block coverage rate of that function is thus 66.67%. These lists can be sorted based on the coverage rate, number of blocks, or function names. Function names are linked to the position in source view where the function body starts. So, just by one click, you can see the least-covered function in the list and by another click the browser displays the body of the function. You can scroll down in the source view and browse through the function body.

**Individual Module Source View**

Within the individual module source views, the tool provides the list of uncovered functions as well as the list of covered functions. The lists are reported in two distinct frames that provide easy navigation of the source code. The lists can be sorted based on:

• Number of blocks within uncovered functions
• Block coverage in the case of covered functions
• Function names

**Setting the Coloring Scheme for the Code Coverage**

The tool provides a visible coloring distinction of the following coverage categories: covered code, uncovered basic blocks, uncovered functions, partially covered code, and unknown code. The default colors that the tool uses for presenting the coverage information are shown in the tables that follows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covered code</td>
<td>#FFFFFF</td>
<td>Indicates code was exercised by the tests. You can override the default color with the -ccolor tool option.</td>
</tr>
<tr>
<td>Uncovered basic block</td>
<td>#FFFF99</td>
<td>Indicates the basic blocks that were not exercised by any of the tests. However, these blocks were within functions that were executed during the tests. You can override the default color with the -bcolor tool option.</td>
</tr>
<tr>
<td>Uncovered function</td>
<td>#FFCCCC</td>
<td>Indicates functions that were never called during the tests. You can override the default color with the -fcolor tool option.</td>
</tr>
<tr>
<td>Partially covered code</td>
<td>#FAFAD2</td>
<td>Indicates that more than one basic block was generated for the code at this position. Some of the blocks were covered and some were not. You can override the default color with the -pcolor tool option.</td>
</tr>
</tbody>
</table>
Indicates code that was specifically marked to be ignored. You can override this default color using the \texttt{-xcolor} tool option.

Indicates basic-block markers and dynamic counts. You can override the default color with the \texttt{-icolor} tool option.

Indicates that no code was generated for this source line. Most probably, the source at this position is a comment, a header-file inclusion, or a variable declaration. You can override the default color with the \texttt{-ucolor} tool option.

The default colors can be customized to be any valid HTML color name or hexadecimal value using the options mentioned for each coverage category in the table above.

For code coverage colored presentation, the coverage tool uses the following heuristic: source characters are scanned until reaching a position in the source that is indicated by the profile information as the beginning of a basic block. If the profile information for that basic block indicates that a coverage category changes, then the tool changes the color corresponding to the coverage condition of that portion of the code, and the coverage tool inserts the appropriate color change in the HTML-formatted report files.

\textbf{NOTE.} You need to interpret the colors in the context of the code. For instance, comment lines that follow a basic block that was never executed would be colored in the same color as the uncovered blocks. Another example is the closing brackets in C/C++ applications.

\textbf{Dynamic Counters}
The coverage tool can be configured to generate the information about the dynamic execution counts. This ability can display the dynamic execution count of each basic block of the application and is useful for both coverage and performance tuning.

The custom configuration requires using the `-counts` option. The counts information is displayed under the code after a "^" sign precisely under the source position where the corresponding basic block begins.

If more than one basic block is generated for the code at a source position (for example, for macros), then the total number of such blocks and the number of the blocks that were executed are also displayed in front of the execution count. For example, line 11 in the code is an `if` statement:

```
Example

11  if ((N == 1).OR. (N == 0))
     ^ 10 (1/2)
12  printf("%d\n", N)
     ^
```

The coverage lines under code lines 11 and 12 contain the following information:

- The `IF` statement in line 11 was executed 10 times.
- Two basic blocks were generated for the `IF` statement in line 11.
- Only one of the two blocks was executed, hence the partial coverage color.
- Only seven out of the ten times variable n had a value of 0 or 1.

In certain situations, it may be desirable to consider all the blocks generated for a single source position as one entity. In such cases, it is necessary to assume that all blocks generated for one source position are covered when at least one of the blocks is covered. This assumption can be configured with the `-nopartial` option. When this option is specified, decision coverage is disabled, and the related statistics are adjusted accordingly. The code lines 11 and 12 indicate that the `printf` statement in line 12 was covered. However, only one of the conditions in line 11 was ever true. With the `-nopartial` option, the tool treats the partially covered code (like the code on line 11) as covered.

**Differential Coverage**

Using the code coverage tool, you can compare the profiles from two runs of an application: a reference run and a new run identifying the code that is covered by the new run but not covered by the reference run. Use this feature to find the portion of the applications code that is not
covered by the applications tests but is executed when the application is run by a customer. It can also be used to find the incremental coverage impact of newly added tests to an applications test space.

**Generating Reference Data**

Create the dynamic profile information for the reference data, which can be used in differential coverage reporting later, by using the `-ref` option. The following command demonstrate a typical command for generating the reference data:

```
Example: generating reference data
codecov -prj Project_Name -dpi customer.dpi -ref appTests.dpi
```

The coverage statistics of a differential-coverage run shows the percentage of the code exercised on a new run but missed in the reference run. In such cases, the tool shows only the modules that included the code that was not covered. Keep this in mind when viewing the coloring scheme in the source views.

The code that has the same coverage property (covered or not covered) on both runs is considered as covered code. Otherwise, if the new run indicates that the code was executed while in the reference run the code was not executed, then the code is treated as uncovered. On the other hand, if the code is covered in the reference run but not covered in the new run, the differential-coverage source view shows the code as covered.

**Running Differential Coverage**

To run the code coverage tool for differential coverage, you must have the application sources, the `.spi` file, and the `.dpi` file, as described in the code coverage tool Requirements section (above).

Once the required files are available, enter a command similar to the following begin the process of differential coverage analysis:

```
Example
codecov -prj Project_Name -spi pgopti.spi -dpi pgopti.dpi
```

Specify the `.dpi` and `.spi` files using the `-spi` and `-dpi` options.
Excluding Code from Coverage Analysis

The code coverage tool allows you to exclude portions of your code from coverage analysis. This ability can be useful during development; for example, certain portions of code might include functions used for debugging only. The test case should not include tests for functionality that will unavailable in the final application.

Another example of code that can be excluded is code that might be designed to deal with internal errors unlikely to occur in the application. In such cases, not having a test case lack of a test case is preferred. You might want to ignore infeasible (dead) code in the coverage analysis. The code coverage tool provides several options for marking portions of the code infeasible (dead) and ignoring the code at the file level, function level, line level, and arbitrary code boundaries indicated by user-specific comments. The following sections explain how to exclude code at different levels.

Including and Excluding Coverage at the File Level

The code coverage tool provides the ability to selectively include or exclude files for analysis. Create a component file and add the appropriate string values that indicate the file and directory name for code you want included or excluded. Pass the file name as a parameter of the \(-\text{comp}\) option. The following example shows the general command:

<table>
<thead>
<tr>
<th>Example: specifying a component file</th>
</tr>
</thead>
<tbody>
<tr>
<td>codecov -comp file</td>
</tr>
</tbody>
</table>

where \textit{file} is the name of a text file containing strings that ask as file and directory name masks for including and excluding file-level analysis. For example, assume that the following:

- You want to include all files with the string "source" in the file name or directory name.
- You create a component text file named \texttt{myComp.txt} with the selective inclusion string.

Once you have a component file, enter a command similar to the following:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>codecov -comp myComp.txt</td>
</tr>
</tbody>
</table>

In this example, individual files name including the string "source" (like source1.c and source2.c) and files in directories where the name contains the string "source" (like source/file1.c and source2\file2.c ) are include in the analysis.
Excluding files is done in the same way; however, the string must have a tilde (~) prefix. The inclusion and exclusion can be specified in the same component file.

For example, assume you want to analyze all individual files or files contained in a directory where the name included the string "source", and you wanted to exclude all individual file and files contained in directories where the name included the string "skip". You would add content similar to the following to the component file (myComp.txt) and pass it to the -comp option:

<table>
<thead>
<tr>
<th>Example: inclusion and exclusion strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
</tr>
<tr>
<td>~skip</td>
</tr>
</tbody>
</table>

Entering the `codecov -comp myComp.txt` command with both instructions in the component file will instruct the tool to include individual files where the name contains "source" (like source1.c and source2.c) and directories where the name contains "source" (like source/file1.c and source2/file2.c), and exclude any individual files where the name contains "skip" (likeskipthis1.c and skipthis2.c) or directories where the name contains "skip" (like skipthes1/debug1.c and skipthose2/debug2.c).

**Excluding Coverage at the Line and Function Level**

You can mark individual lines for exclusion my passing string values to the -onelinedsbl option. For example, assume that you have some code similar to the following:

<table>
<thead>
<tr>
<th>Sample code</th>
</tr>
</thead>
<tbody>
<tr>
<td>printf (&quot;internal error 123 - please report!\n&quot;); // NO_COVER</td>
</tr>
<tr>
<td>printf (&quot;internal error 456 - please report!\n&quot;); /* INF IA-32 architecture */</td>
</tr>
</tbody>
</table>

If you wanted to exclude all functions marked with the comments NO_COVER or INF IA-32 architecture, you would enter a command similar to the following:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>codecov -onelinedsbl NO_COVER -onelinedsbl &quot;INF IA-32 architecture&quot;</td>
</tr>
</tbody>
</table>

You can specify multiple exclusion strings simultaneously, and you can specify any string values for the markers; however, you must remember the following guidelines when using this option:

- Inline comments must occur at the end of the statement.
- The string must be a part of an inline comment.
An entire function can be excluded from coverage analysis using the same methods. For example, the following function will be ignored from the coverage analysis when you issue example command shown above.

```
Sample code

    void dumpInfo (int n)
    {
        // NO_COVER
        ...
    }
```

Additionally, you can use the code coverage tool to color the infeasible code with any valid HTML color code by combining the `-onelinedsbl` and `-xcolor` options. The following example commands demonstrate the combination:

```
Example: combining tool options

codecov -onelinedsbl INF -xcolor lightgreen
codecov -onelinedsbl INF -xcolor #CCFFCC
```

Excluding Code by Defining Arbitrary Boundaries

The code coverage tool provides the ability to arbitrarily exclude code from coverage analysis. This feature is most useful where the excluded code either occur inside of a function or spans several functions.

Use the `-beginblkdsbl` and `-endblkdsbl` options to mark the beginning and end, respectively, of any arbitrarily defined boundary to exclude code from analysis. Remember the following guidelines when using these options:

- Inline comments must occur at the end of the statement.
- The string must be a part of an inline comment.
For example assume that you have the following code:

```c
void div (int m, int n)
{
    if (n == 0)
        /* BEGIN_INF */
        {
            printf (internal error 314 please report\n);
            recover ();

        } /* END_INF */
    else {
        ...
    }
}
...
// BINF
Void recover ()
{
    ...
}
// EINF
```

The following example commands demonstrate how to use the `-beginblkdsbl` option to mark the beginning and the `-endblkdsbl` option to mark the end of code to exclude from the sample shown above.

```bash
codecov -xcolor #ccFFCC -beginblkdsbl BINF -endblkdsbl EINF
```
Example: arbitrary code marker commands

```
codecov -xcolor #ccFFCC -beginblkdsbl "BEGIN_INF" -endblkdsbl "END_INF"
```

Notice that you can combine these options in combination with the `-xcolor` option.

**Exporting Coverage Data**

The code coverage tool provides specific options to extract coverage data from the dynamic profile information (.dpi files) that result from running instrumented application binaries under various workloads. The tool can export the coverage data in various formats for post-processing and direct loading into databases: the default HTML, text, and XML. You can choose to export data at the function and basic block levels.

There are two basic methods for exporting the data: quick export and combined export. Each method has associated options supported by the tool:

- **Quick export**: The first method is to export the data coverage to text- or XML-formatted files without generating the default HTML report. The application sources need not be present for this method. The code coverage tool creates reports and provides statistics only about the portions of the application executed. The resulting analysis and reporting occurs quickly, which makes it practical to apply the coverage tool to the dynamic profile information (the .dpi file) for every test case in a given test space instead of applying the tool to the profile of individual test suites or the merge of all test suites. The `-xmlfcvrg`, `-txtfcvrg`, `-xmlbcvrg`, and `-txtbcvrg` options support the first method.

- **Combined export**: The second method is to generate the default HTML and simultaneously export the data to text- and XML-formatted files. This process is slower than first method since the application sources are parsed and reports generated. The `-xmlbcvrgfull` and `-txtbcvrgfull` options support the second method.

These export methods provide the means to quickly extend the code coverage reporting capabilities by supplying consistently formatted output from the code coverage tool. You can extend these by creating additional reporting tools on top of these report files.

**Quick Export**

The profile of covered functions of an application can be exported quickly using the `-xmlfcvrg`, `-txtfcvrg`, `-xmlbcvrg`, and `-txtbcvrg` options. When using any of these options, specify the output file that will contain the coverage report. For example, enter a command similar to the following to generate a report of covered functions in XML formatted output:
Example: quick export of function data

codecov -prj test1 -dpi test1.dpi -xmlfcvrg test1_fcvrg.xml

The resulting report will show how many times each function was executed and the total number of blocks of each function together with the number of covered blocks and the block coverage of each function. The following example shows some of the content of a typical XML report.

<table>
<thead>
<tr>
<th>XML-formatted report example</th>
</tr>
</thead>
</table>

```
<PROJECT name = "test1">
  <MODULE name = "D:\SAMPLE.C">
    <FUNCTION name="f0" freq="2">
      <BLOCKS total="6" covered="5" coverage="83.33%"></BLOCKS>
    </FUNCTION>
    ...
  </MODULE>
  <MODULE name = "D:\SAMPLE2.C">
    ...
  </MODULE>
</PROJECT>
```

In the above example, we note that function f0, which is defined in file sample.c, has been executed twice. It has a total number of six basic blocks, five of which are executed, resulting in an 83.33% basic block coverage.
You can also export the data in text format using the `-txtfcvrg` option. The generated text report, using this option, for the above example would be similar to the following example:

### Text-formatted report example

<table>
<thead>
<tr>
<th>Covered Functions in File: &quot;D:\SAMPLE.C&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;f0&quot; 2 6 5 83.33</td>
</tr>
<tr>
<td>&quot;f1&quot; 1 6 4 66.67</td>
</tr>
<tr>
<td>&quot;f2&quot; 1 6 3 50.00</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

In the text formatted version of the report, the each line of the report should be read in the following manner:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>function name</td>
<td>execution frequency</td>
<td>line number of the start of the function</td>
<td>column number of the start of the function definition</td>
<td>percentage of basic-block coverage of the function</td>
</tr>
</tbody>
</table>

Additionally, the tool supports exporting the block level coverage data using the `-xmlbcvrg` option. For example, enter a command similar to the following to generate a report of covered blocks in XML formatted output:

### Example: quick export of block data to XML

```bash
codecov -prj test1 -dpi test1.dpi -xmlbcvrg test1_bcvg.xml
```
The example command shown above would generate XML-formatted results similar to the following:

```
<PROJECT name = "test1">
  <MODULE name = "D:\SAMPLE.cpp">
    <FUNCTION name="f0" freq="2">
      ...
      <BLOCK line="11" col="2">
        <INSTANCE id="1" freq="1"> </INSTANCE>
      </BLOCK>
      <BLOCK line="12" col="3">
        <INSTANCE id="1" freq="2"> </INSTANCE>
        <INSTANCE id="2" freq="1"> </INSTANCE>
      </BLOCK>
    </FUNCTION>
  </MODULE>
</PROJECT>
```

In the sample report, notice that one basic block is generated for the code in function f0 at the line 11, column 2 of the file sample.cpp. This particular block has been executed only once. Also notice that there are two basic blocks generated for the code that starts at line 12, column 3 of file. One of these blocks, which has id = 1, has been executed two times, while the other block has been executed only once. A similar report in text format can be generated through the `-txtbcvrg` option.

**Combined Exports**

The code coverage tool has also the capability of exporting coverage data in the default HTML format while simultaneously generating the text- and XML-formatted reports.

Use the `-xmlbcvrgfull` and `-txtbcvrgfull options` to generate reports in all supported formatted in a single run. These options export the basic-block level coverage data while simultaneously generating the HTML reports. These options generate more complete reports since they include analysis on functions that were not executed at all. However, exporting the coverage data using these options requires access to application source files and take much longer to run.

**Dynamic Call Graphs**
Using the `-txtdcg` option the tool can provide detailed information about the dynamic call graphs in an application. Specify an output file for the dynamic call-graph report. The resulting call graph report contains information about the percentage of static and dynamic calls (direct, indirect, and virtual) at the application, module, and function levels.

**test prioritization Tool**

The test prioritization tool enables the profile-guided optimizations on IA-32, Intel® 64, and IA-64 architectures, on Linux* and Windows*, to select and prioritize test for an application based on prior execution profiles. The tool is available on IA-32 and Intel® 64 architectures on Mac OS* X.

The tool offers a potential of significant time saving in testing and developing large-scale applications where testing is the major bottleneck.

Development often requires changing applications modules. As applications change, developers can have a difficult time retaining the quality of their functional and performance tests so they are current and on-target. The test prioritization tool lets software developers select and prioritize application tests as application profiles change.

The information about the tool is separated into the following sections:

- Features and benefits
- Requirements and syntax
- Usage model
- Tool options
- Running the tool

**Features and Benefits**

The test prioritization tool provides an effective testing hierarchy based on the code coverage for an application. The following list summarizes the advantages of using the tool:

- Minimizing the number of tests that are required to achieve a given overall coverage for any subset of the application: the tool defines the smallest subset of the application tests that achieve exactly the same code coverage as the entire set of tests.
- Reducing the turn-around time of testing: instead of spending a long time on finding a possibly large number of failures, the tool enables the users to quickly find a small number of tests that expose the defects associated with the regressions caused by a change set.
- Selecting and prioritizing the tests to achieve certain level of code coverage in a minimal time based on the data of the tests' execution time.
See Understanding Profile-guided Optimization and Example of Profile-guided Optimization for general information on creating the files needed to run this tool.

**test prioritization Tool Requirements**

The test prioritization tool needs the following items to work:

- The .spi file generated by Intel® compilers when compiling the application for the instrumented binaries with the `-prof-gen=srcpos` (Linux* and Mac OS* X) or `/Qprof-gen:srcpos` (Windows*) option.
- The .dpi files generated by the `profmerge` tool as a result of merging the dynamic profile information .dyn files of each of the application tests. (Run the profmerge tool on all .dyn files that are generated for each individual test and name the resulting .dpi in a fashion that uniquely identifies the test.)

---

**CAUTION.** The profmerge tool merges all .dyn files that exist in the given directory. Make sure unrelated .dyn files, which may remain from unrelated runs, are not present. Otherwise, the profile information will be skewed with invalid profile data, which can result in misleading coverage information and adverse performance of the optimized code;

- User-generated file containing the list of tests to be prioritized. For successful instrumented code run, you should:
  - Name each test .dpi file so the file names uniquely identify each test.
  - Create a .dpi list file, which is a text file that contains the names of all .dpi test files. Each line of the .dpi list file should include one, and only one .dpi file name. The name can optionally be followed by the duration of the execution time for a corresponding test in the `dd:hh:mm:ss` format.

  For example: `Test1.dpi 00:00:60:35` states that `Test1` lasted 0 days, 0 hours, 60 minutes and 35 seconds.

  The execution time is optional. However, if it is not provided, then the tool will not prioritize the test for minimizing execution time. It will prioritize to minimize the number of tests only.

The tool uses the following general syntax:

```plaintext
Tool Syntax

tselect -dpi_list file
```
-dpi_list is a required tool option that sets the path to the list file containing the list of the all .dpi files. All other tool commands are optional.

**NOTE.** Windows* only: Unlike the compiler options, which are preceded by forward slash ("/"), the tool options are preceded by a hyphen ("-").

**Usage Model**

The following figure illustrates a typical test prioritization tool usage model.

---

**test prioritization tool Options**

The tool uses the options that are listed in the following table:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-help</td>
<td></td>
<td>Prints tool option descriptions.</td>
</tr>
<tr>
<td>-dpi_list file</td>
<td></td>
<td>Required. Specifies the file name of the file that contains the names of the dynamic profile information (.dpi) files. Each line of the file must contain only one .dpi file name, which can be optionally followed by its execution time. The name must uniquely identify the test.</td>
</tr>
<tr>
<td>-spi file</td>
<td>pgopti.spi</td>
<td>Specifies the file name of the static profile information file (.SPI).</td>
</tr>
<tr>
<td>-o file</td>
<td></td>
<td>Specifies the file name of the output report file.</td>
</tr>
</tbody>
</table>

---

1331
<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-comp file</td>
<td></td>
<td>Specifies the file name that contains the list of files of interest.</td>
</tr>
<tr>
<td>-cutoff file</td>
<td></td>
<td>Instructs the tool to terminate when the cumulative block coverage reaches a preset percentage, as specified by value, of pre-computed total coverage. Value must be greater than 0.0 (for example, 99.00) but not greater than 100. Value can be set to 100.</td>
</tr>
<tr>
<td>-nototal</td>
<td></td>
<td>Instructs the tool to ignore the pre-compute total coverage process.</td>
</tr>
<tr>
<td>-mintime</td>
<td></td>
<td>Instructs the tool to minimize testing execution time. The execution time of each test must be provided on the same line of dpi_list file, after the test name in dd:hh:mm:ss format.</td>
</tr>
<tr>
<td>-srcbasedir dir</td>
<td></td>
<td>Specifies a different top level project directory than was used during compiler instrumentation run with the prof-src-root compiler option to support relative paths to source files in place of absolute paths.</td>
</tr>
<tr>
<td>Option</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-verbose</td>
<td></td>
<td>Instructs the tool to generate more logging information about program progress.</td>
</tr>
</tbody>
</table>

**Running the tool**

The following steps demonstrate one simple example for running the tool on IA-32 architectures.

1. Specify the directory by entering a command similar to the following:

   **Example**
   ```
   set prof-dir=c:\myApp\prof-dir
   ```

2. Compile the program and generate instrumented binary by issuing commands similar to the following:

   **Operating System** | **Command**
   ---------------------|-----------------|
   Linux and Mac OS X   | `icpc -prof-gen=srcpos myApp.cpp`
   Windows              | `icl /Qprof-gen:srcpos myApp.cpp`

   This commands shown above compiles the program and generates instrumented binary `myApp` as well as the corresponding static profile information `pgopti.spi`.

3. Make sure that unrelated `.dyn` files are not present by issuing a command similar to the following:

   **Example**
   ```
   rm prof-dir /*.dyn
   ```

4. Run the instrumented files by issuing a command similar to the following:

   **Example**
   ```
   myApp < data1
   ```
The command runs the instrumented application and generates one or more new dynamic profile information files that have an extension .dyn in the directory specified by the \texttt{-prof-dir} step above.

5. Merge all .dyn file into a single file by issuing a command similar to the following:

\textbf{Example}

\texttt{profmerge -prof_dpi Test1.dpi}

The \texttt{profmerge} tool merges all the .dyn files into one file (Test1.dpi) that represents the total profile information of the application on Test1.

6. Again make sure there are no unrelated .dyn files present a second time by issuing a command similar to the following:

\textbf{Example}

\texttt{rm prof-dir \*.dyn}

7. Run the instrumented application and generate one or more new dynamic profile information files that have an extension .dyn in the directory specified the \texttt{prof-dir} step above by issuing a command similar to the following:

\textbf{Example}

\texttt{myApp < data2}

8. Merge all .dyn files into a single file, by issuing a command similar to the following

\textbf{Example}

\texttt{profmerge -prof_dpi Test2.dpi}

At this step, the \texttt{profmerge} tool merges all the .dyn files into one file (Test2.dpi) that represents the total profile information of the application on Test2.

9. Make sure that there are no unrelated .dyn files present for the final time, by issuing a command similar to the following:

\textbf{Example}

\texttt{rm prof-dir \*.dyn}
10. Run the instrumented application and generates one or more new dynamic profile information files that have an extension .dyn in the directory specified by -prof-dir by issuing a command similar to the following:

**Example**

myApp < data3

11. Merge all .dyn file into a single file, by issuing a command similar to the following:

**Example**

profmerge -prof_dpi Test3.dpi

At this step, the profmerge tool merges all the .dyn files into one file (Test3.dpi) that represents the total profile information of the application on Test3.

12. Create a file named tests_list with three lines. The first line contains Test1.dpi, the second line contains Test2.dpi, and the third line contains Test3.dpi.

**Tool Usage Examples**

When these items are available, the test prioritization tool may be launched from the command line in prof-dir directory as described in the following examples.

**Example 1: Minimizing the Number of Tests**

The following example describes how minimize the number of test runs.

**Example Syntax**

```
tselect -dpi_list tests_list -spi pgopti.spi
```

where the `-spi` option specifies the path to the .spi file.
The following sample output shows typical results.

<table>
<thead>
<tr>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of tests = 3</td>
</tr>
<tr>
<td>Total block coverage ~ 52.17</td>
</tr>
<tr>
<td>Total function coverage ~ 50.00</td>
</tr>
<tr>
<td>num</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

In this example, the results provide the following information:

- By running all three tests, we achieve 52.17% block coverage and 50.00% function coverage.
- Test3 by itself covers 45.65% of the basic blocks of the application, which is 87.50% of the total block coverage that can be achieved from all three tests.
- By adding Test2, we achieve a cumulative block coverage of 52.17% or 100% of the total block coverage of Test1, Test2, and Test3.
- Elimination of Test1 has no negative impact on the total block coverage.

**Example 2: Minimizing Execution Time**

Suppose we have the following execution time of each test in the `tests_list` file:

<table>
<thead>
<tr>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1.dpi 00:00:60:35</td>
</tr>
<tr>
<td>Test2.dpi 00:00:10:15</td>
</tr>
<tr>
<td>Test3.dpi 00:00:30:45</td>
</tr>
</tbody>
</table>

The following command minimizes the execution time by passing the `-mintime` option:

<table>
<thead>
<tr>
<th>Sample Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>tselect -dpi_list tests_list -spi pgopti.spi -mintime</td>
</tr>
</tbody>
</table>
The following sample output shows possible results:

<table>
<thead>
<tr>
<th>num</th>
<th>elapsedTime</th>
<th>%RatCvrg</th>
<th>%BlkCvrg</th>
<th>%FncCvrg</th>
<th>Test Name</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10:15</td>
<td>75.00</td>
<td>39.13</td>
<td>25.00</td>
<td>Test2.dpi</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>41:00</td>
<td>100.00</td>
<td>52.17</td>
<td>50.00</td>
<td>Test3.dpi</td>
<td></td>
</tr>
</tbody>
</table>

In this case, the results indicate that the running all tests sequentially would require one hour, 45 minutes, and 35 seconds, while the selected tests would achieve the same total block coverage in only 41 minutes.

The order of tests is based on minimizing time (first Test2, then Test3) could be different than when prioritization is done based on minimizing the number of tests. See Example 1 shown above: first Test3, then Test2. In Example 2, Test2 is the test that gives the highest coverage per execution time, so Test2 is picked as the first test to run.

**Using Other Options**

The `-cutoff` option enables the tool to exit when it reaches a given level of basic block coverage. The following example demonstrates how to the option:

```
tselect -dpi_list tests_list -spi pgopti.spi -cutoff 85.00
```

If the tool is run with the cutoff value of 85.00, as in the above example, only Test3 will be selected, as it achieves 45.65% block coverage, which corresponds to 87.50% of the total block coverage that is reached from all three tests.

The tool does an initial merging of all the profile information to figure out the total coverage that is obtained by running all the tests. The `-nototal` option enables you to skip this step. In such a case, only the absolute coverage information will be reported, as the overall coverage remains unknown.
profmerge and proforder Tools

profmerge Tool

Use the profmerge tool to merge dynamic profile information (.dyn) files and any specified summary files (.dpi). The compiler executes profmerge automatically during the feedback compilation phase when you specify -prof-use (Linux* and Mac OS* X) or /Qprof-use (Windows*).

The command-line usage for profmerge is as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>profmerge [-prof_dir dir_name]</td>
</tr>
</tbody>
</table>

The tool merges all .dyn files in the current directory, or the directory specified by -prof_dir, and produces a summary file: pgopti.dpi.

**NOTE.** The spelling of tools options may differ slightly from compiler options. Tools options use an underscore (for example -prof_dir) instead of the hyphen used by compiler options (for example -prof-dir or /Qprof-dir) to join words. Also, on Windows* OS systems, the tool options are preceded by a hyphen (“-”) unlike Windows compiler options, which are preceded by a forward slash (“/”).

You can use profmerge tool to merge .dyn files into a .dpi file without recompiling the application. Thus, you can run the instrumented executable file on multiple systems to generate dyn files, and optionally use profmerge with the -prof_dpi option to name each summary dpi file created from the multiple dyn files.

Since the profmerge tool merges all the .dyn files that exist in the given directory, make sure unrelated .dyn files are not present; otherwise, profile information will be based on invalid profile data, which can negatively impact the performance of optimized code.

profmerge Options

The profmerge tool supports the following options:

<table>
<thead>
<tr>
<th>Tool Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-help</td>
<td>Lists supported options.</td>
</tr>
<tr>
<td>Tool Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-nologo</td>
<td>Disables version information. This option is supported on Windows* only.</td>
</tr>
<tr>
<td>-exclude_funcs functions</td>
<td>Excludes functions from the profile. The list items must be separated by a</td>
</tr>
<tr>
<td></td>
<td>comma (&quot;,&quot;); you can use a period (&quot;,&quot; ) as a wild card character in</td>
</tr>
<tr>
<td></td>
<td>function names.</td>
</tr>
<tr>
<td>-prof_dir dir dir</td>
<td>Specifies the directory from which to read .dyn and .dpi files, and write</td>
</tr>
<tr>
<td></td>
<td>the .dpi file. Alternatively, you can set the environment variable PROF_DIR.</td>
</tr>
<tr>
<td>-prof_dpi file file</td>
<td>Specifies the name of the .dpi file being generated.</td>
</tr>
<tr>
<td>-dump</td>
<td>Displays profile information.</td>
</tr>
<tr>
<td>-src_old dir -src_new dir</td>
<td>Changes the directory path stored within the .dpi file.</td>
</tr>
<tr>
<td>-src_no_dir</td>
<td>Uses only the file name and not the directory name when reading dyn/dpi</td>
</tr>
<tr>
<td></td>
<td>records. If you specify -src_no_dir, the directory name of the source file</td>
</tr>
<tr>
<td></td>
<td>will be ignored when deciding which profile data records correspond to a</td>
</tr>
<tr>
<td></td>
<td>specific application routine, and the -src-root option is ignored.</td>
</tr>
<tr>
<td>-src-root dir dir</td>
<td>Specifies a directory path prefix for the root directory where the user's</td>
</tr>
<tr>
<td></td>
<td>application files are stored. This option is ignored if you specify</td>
</tr>
<tr>
<td></td>
<td>-src_no_dir.</td>
</tr>
<tr>
<td>-a file1.dpi ... fileN.dpi</td>
<td>Specifies and merges available .dpi files.</td>
</tr>
<tr>
<td>Tool Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-verbose</td>
<td>Instructs the tool to display full information during merge.</td>
</tr>
<tr>
<td>-weighted</td>
<td>Instructs the tool to apply an equal weighting (regardless of execution times) to the dyn file values to normalize the data counts. This keyword is useful when the execution runs have different time durations and you want them to be treated equally.</td>
</tr>
</tbody>
</table>

**Relocating source files using profmerge**

The Intel® compiler uses the full path to the source file for each routine to look up the profile summary information associated with that routine. By default, this prevents you from:

- Using the profile summary file (.dpi) if you move your application sources.
- Sharing the profile summary file with another user who is building identical application sources that are located in a different directory.

You can disable the use of directory names when reading dyn/dpi file records by specifying the profmerge option -src_no_dir. This profmerge option is the same as the compiler option -no-prof-src-dir (Linux and Mac OS X) and /Qprof-src-dir- (Windows).

To enable the movement of application sources, as well as the sharing of profile summary files, you can use the profmerge option -src-root to specify a directory path prefix for the root directory where the application files are stored. Alternatively, you can specify the option pair -src_old -src_new to modify the data in an existing summary dpi file. For example:

**Example: relocation command syntax**

```
profmerge -prof_dir <dir1> -src_old <dir2> -src_new <dir3>
```

where `<dir1>` is the full path to dynamic information file (.dpi), `<dir2>` is the old full path to source files, and `<dir3>` is the new full path to source files. The example command (above) reads the pgopti.dpi file, in the location specified in `<dir1>`. For each function represented in the pgopti.dpi file, whose source path begins with the `<dir2>` prefix, profmerge replaces that prefix with `<dir3>`. The pgopti.dpi file is updated with the new source path information.

You can run profmerge more than once on a given pgopti.dpi file. For example, you may need to do this if the source files are located in multiple directories:
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Command Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>profmerge -prof_dir -src_old /src/prog_1 -src_new /src/prog_2</td>
</tr>
<tr>
<td></td>
<td>profmerge -prof_dir -src_old /proj_1 -src_new /proj_2</td>
</tr>
<tr>
<td>Windows</td>
<td>profmerge -src_old &quot;c:/program files&quot; -src_new &quot;e:/program files&quot;</td>
</tr>
<tr>
<td></td>
<td>profmerge -src_old c:/proj/application -src_new d:/app</td>
</tr>
</tbody>
</table>

In the values specified for `-src_old` and `-src_new`, uppercase and lowercase characters are treated as identical. Likewise, forward slash (`/`) and backward slash (`\`) characters are treated as identical.

**NOTE.** Because the source relocation feature of profmerge modifies the `pgopti.dpi` file, consider making a backup copy of the file before performing the source relocation.

**proforder Tool**

The proforder tool is used as part of the feedback compilation phase, to improve program performance. Use proforder to generate a function order list for use with the `/ORDER` linker option. The tool uses the following syntax:

**Syntax**

```
proforder [-prof_dir dir] [-o file]
```

where `dir` is the directory containing the profile files (`.dpi` and `.spi`), and `file` is the optional name of the function order list file. The default name is `proford.txt`.

**NOTE.** The spelling of tools options may differ slightly from compiler options. Tools options use an underscore (for example `-prof_dir`) instead of the hyphen used by compiler options (for example `-prof-dir` or `/Qprof-dir`) to join words. Also, on Windows* OS systems, the tool options are preceded by a hyphen ("-") unlike Windows compiler options, which are preceded by a forward slash ("/").
proforder Options

The proforder tool supports the following options:

<table>
<thead>
<tr>
<th>Tool Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-help</td>
<td></td>
<td>Lists supported options.</td>
</tr>
<tr>
<td>-nologo</td>
<td></td>
<td>Disables version information. This option is supported on Windows* only.</td>
</tr>
<tr>
<td>-omit_static</td>
<td></td>
<td>Instructs the tool to omit static functions from function ordering.</td>
</tr>
<tr>
<td>-prof_dir dir</td>
<td>dir</td>
<td>Specifies the directory where the .spi and .dpi file reside.</td>
</tr>
<tr>
<td>-prof_dpi file</td>
<td>file</td>
<td>Specifies the name of the .dpi file.</td>
</tr>
<tr>
<td>-prof_file string</td>
<td></td>
<td>Selects the .dpi and .spi files that include the substring value in the file name matching the values passed as string.</td>
</tr>
<tr>
<td>-prof_spi file</td>
<td></td>
<td>Specifies the name of the .spi file.</td>
</tr>
<tr>
<td>-o file</td>
<td>proford.txt</td>
<td>Specifies an alternate name for the output file.</td>
</tr>
</tbody>
</table>

Using Function Order Lists, Function Grouping, Function Ordering, and Data Ordering Optimizations

Instead of doing a full multi-file interprocedural build of your application by using the compiler option -ipo (Linux* OS) or /Qipo (Windows* OS), you can obtain some of the benefits by having the compiler and linker work together to make global decisions about where to place the functions and data in your application. These optimizations are not supported on Mac OS* X systems.
The following table lists each optimization, the type of functions or global data it applies to, and the operating systems and architectures that it is supported on.

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Type of Function or Data</th>
<th>Supported OS and Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function Order Lists:</strong></td>
<td>Specifies the order in which the linker should link the non-static routines (functions) of your program. This optimization can improve application performance by improving code locality and reduce paging. Also see Comparison of Function Order Lists and IPO Code Layout.</td>
<td>Windows OS: IA-32, Intel® 64, and IA-64 architectures. Linux OS: not supported.</td>
</tr>
<tr>
<td><strong>Function Grouping:</strong></td>
<td>Specifies that the linker should place the extern and static routines (functions) of your program into hot or cold program sections. This optimization can improve application performance by improving code locality and reduce paging.</td>
<td>Linux OS: IA-32 and Intel 64 architectures. Windows OS: not supported.</td>
</tr>
<tr>
<td><strong>Function Ordering:</strong></td>
<td>Enables ordering of static and extern routines using profile information. Specifies the order in which the linker should link the routines (functions) of your program. This optimization can improve application performance by improving code locality and reduce paging.</td>
<td>Linux and Windows OS: IA-32, Intel 64, and IA-64 architectures.</td>
</tr>
<tr>
<td><strong>Data Ordering:</strong></td>
<td>Enables ordering of static global data items based on profiling information. Specifies the</td>
<td>Linux and Windows OS, IA-32, Intel 64, and IA-64 architectures.</td>
</tr>
</tbody>
</table>
You can use only one of the function-related ordering optimizations listed above. However, you can use the Data Ordering optimization with any one of the function-related ordering optimizations listed above, such as Data Ordering with Function Ordering, or Data Ordering with Function Grouping. In this case, specify the prof-gen option keyword globdata (needed for Data Ordering) instead of srcpos (needed for function-related ordering).

The following sections show the commands needed to implement each of these optimizations: function order list, function grouping, function ordering, and data ordering. For all of these optimizations, omit the -ipo (Linux* OS) or /Qipo (Windows OS) or equivalent compiler option.

Generating a Function Order List (Windows OS)

This section provides an example of the process for generating a function order list. Assume you have a C++ program that consists of the following files: file1.cpp and file2.cpp. Additionally, assume you have created a directory for the profile data files called c:\profdata. You would enter commands similar to the following to generate and use a function order list for your Windows application.

1. Compile your program using the /Qprof-gen:srcpos option. Use the /Qprof-dir option to specify the directory location of the profile files. This step creates an instrumented executable.

   **Example commands**

   ```
   icl /Femyprog /Qprof-gen=srcpos /Qprof-dir c:\profdata file1.cpp
   file2.cpp
   ```

2. Run the instrumented program with one or more sets of input data. Change your directory to the directory where the executables are located. The program produces a .dyn file each time it is executed.
3. Before this step, copy all .dyn and .dpi files into the same directory. Merge the data from one or more runs of the instrumented program by using the profmerge tool to produce the pgopti.dpi file. Use the /prof_dir option to specify the directory location of the .dyn files.

Example commands

profmerge /prof_dir c:\profdata

4. Generate the function order list using the proforder tool. (By default, the function order list is produced in the file proford.txt.)

Example commands

proforder /prof_dir c:\profdata /o myprog.txt

5. Compile the application with the generated profile feedback by specifying the ORDER option to the linker. Use the /Qprof-dir option to specify the directory location of the profile files.

Example commands

icl /Femyprog /Qprof-dir c:\profdata file1.cpp file2.cpp /link
-ORDER:@myprog.txt

Using Function Grouping (Linux OS)

This section provides a general example of the process for using the function grouping optimization. Assume you have a C++ program that consists of the following files: file1.cpp and file2.cpp. Additionally, assume you have created a directory for the profile data files called profdata. You would enter commands similar to the following to use a function grouping for your Linux application.

1. Compile your program using the -prof-gen option. Use the -prof-dir option to specify the directory location of the profile files. This step creates an instrumented executable.
Example commands

```
icc -o myprog -prof-gen -prof-dir ./profdata file1.cpp file2.cpp
```

2. Run the instrumented program with one or more sets of input data. Change your directory to the directory where the executables are located. The program produces a .dyn file each time it is executed.

Example commands

```
./myprog
```

3. Copy all .dyn and .dpi files into the same directory. If needed, you can merge the data from one or more runs of the instrumented program by using the profmerge tool to produce the pgopti.dpi file.

4. Compile the application with the generated profile feedback by specifying the -prof-func-group option to request the function grouping as well as the -prof-use option to request feedback compilation. Again, use the -prof-dir option to specify the location of the profile files.

Example commands

```
icl /Femyprog file1.cpp file2.cpp -prof-func-group -prof-use -prof-dir ./profdata
```

Using Function Ordering

This section provides an example of the process for using the function ordering optimization. Assume you have a C++ program that consists of the following files: file1.cpp and file2.cpp. Additionally, assume you have created a directory for the profile data files called c:\profdata (on Windows) or ./profdata (on Linux). You would enter commands similar to the following to generate and use function ordering for your application.

1. Compile your program using the -prof-gen=srcpos (Linux) or /Qprof-gen:srcpos (Windows) option. Use the -prof-dir (Linux) or /Qprof-dir (Windows) option to specify the directory location of the profile files. This step creates an instrumented executable.
2. Run the instrumented program with one or more sets of input data. Change your directory to the directory where the executables are located. The program produces a .dyn file each time it is executed.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icc -o myprog -prof-gen=srcpos -prof-dir ./profdata file1.cpp file2.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Femyprog /Qprof-gen:srcpos /Qprof-dir c:\profdata file1.cpp file2.cpp</td>
</tr>
</tbody>
</table>

3. Copy all .dyn and .dpi files into the same directory. If needed, you can merge the data from one or more runs of the instrumented program by using the profmerge tool to produce the pgopti.dpi file.

4. Compile the application with the generated profile feedback by specifying the -prof-func-order (Linux) or /Qprof-func-order (Windows) option to request the function ordering as well as the -prof-use (Linux) or /Qprof-use (Windows) option to request feedback compilation. Again, use the -prof-dir (Linux) or /Qprof-dir (Windows) option to specify the location of the profile files.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icpc -o myprog -prof-dir ./profdata file1.cpp file2.cpp -prof-func-order -prof-use</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Femyprog /Qprof-dir c:\profdata file1.cpp file2.cpp /Qprof-func-order /Qprof-use</td>
</tr>
</tbody>
</table>
Using Data Ordering

This section provides an example of the process for using the data order optimization. Assume you have a C++ program that consists of the following files: file1.cpp and file2.cpp. Additionally, assume you have created a directory for the profile data files called c:\profdata (on Windows) or ./profdata (on Linux). You would enter commands similar to the following to use data ordering for your application.

1. Compile your program using the -prof-gen=globdata (Linux) or /Qprof-gen:globdata (Windows) option. Use the -prof-dir (Linux) or /Qprof-dir (Windows) option to specify the directory location of the profile files. This step creates an instrumented executable.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icc -o myprog -prof-gen=globdata -prof-dir ./profdata file1.cpp file2.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icl /Femyprog /Qprof-gen=globdata /Qprof-dir c:\profdata file1.cpp file2.cpp</td>
</tr>
</tbody>
</table>

2. Run the instrumented program with one or more sets of input data. If you specified a location other than the current directory, change your directory to the directory where the executables are located. The program produces a .dyn file each time it is executed.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>./myprog</td>
</tr>
<tr>
<td>Windows</td>
<td>myprog.exe</td>
</tr>
</tbody>
</table>

3. Copy all .dyn and .dpi files into the same directory. If needed, you can merge the data from one or more runs of the instrumented program by using the profmerge tool to produce the pgopti.dpi file.

4. Compile the application with the generated profile feedback by specifying the -prof-data-order (Linux) or /Qprof-data-order option to request the data ordering as well as the -prof-use (Linux) or /Qprof-use (Windows) option to request feedback compilation. Again, use the -prof-dir (Linux) or /Qprof-dir (Windows) option to specify the location of the profile files.
Comparison of Function Order Lists and IPO Code Layout

The Intel® compiler provides two methods of optimizing the layout of functions in the executable:

- Using a function order list
- Using the /Qipo (Windows) compiler option

Each method has its advantages. A function order list, created with proforder, lets you optimize the layout of non-static functions: that is, external and library functions whose names are exposed to the linker.

The linker cannot directly affect the layout order for static functions because the names of these functions are not available in the object files.

The compiler cannot affect the layout order for functions it does not compile, such as library functions. The function layout optimization is performed automatically when IPO is active.

Alternately, using the /Qipo (Windows) option allows you to optimize the layout of all static or extern functions compiled with the Intel® C++ Compiler. The compiler cannot affect the layout order for functions it does not compile, such as library functions. The function layout optimization is performed automatically when IPO is active.

Function Order List Effects

<table>
<thead>
<tr>
<th>Function Type</th>
<th>IPO Code Layout</th>
<th>Function Ordering with proforder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>X</td>
<td>No effect</td>
</tr>
<tr>
<td>Extern</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Library</td>
<td>No effect</td>
<td>X</td>
</tr>
</tbody>
</table>

Function Order List Usage Guidelines (Windows*)
Use the following guidelines to create a function order list:

- The order list only affects the order of non-static functions.
- You must compile with /Gy to enable function-level linking. (This option is active if you specify either option /01 or /02.)

**PGO API Support**

**API Support Overview**

The Profile Information Generation Support (Profile IGS) lets you control the generation of profile information during the instrumented execution phase of profile-guided optimizations.

A set of functions and an environment variable comprise the Profile IGS. The remaining topics in this section describe the associated functions and environment variables.

The compiler sets a define for _PGO_INSTRUMENT when you compile with either -prof-gen (Linux* and Mac OS* X) or /Qprof-gen (Windows*). Without instrumentation, the Profile IGS functions cannot provide PGO API support.

Normally, profile information is generated by an instrumented application when it terminates by calling the standard exit() function.

To ensure that profile information is generated, the functions described in this section may be necessary or useful in the following situations:

- The instrumented application exits using a non-standard exit routine.
- The instrumented application is a non-terminating application: exit() is never called.
- The application requires control of when the profile information is generated.

You can use the Profile IGS functions in your application by including a header file at the top of any source file where the functions may be used.

```c
#include <pgouser.h>
```

**The Profile IGS Environment Variable**

The environment variable for Profile IGS is INTEL_PROF_DUMP_INTERVAL. This environment variable may be used to initiate Interval Profile Dumping in an instrumented user application.
**See Also**
- **PGO API Support**
- **_PGOPTI_Set_Interval_Prof_Dump()**

**PGO Environment Variables**

The environment variables determine the directory in which to store dynamic information files, control the creation of one or multiple dyn files to collect profiling information, and determine whether to overwrite `pgopti.dpi`.

The environment variables are described in the table below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTEL_PROF_DUMP_CUMULATIVE</strong></td>
<td>When using interval profile dumping (initiated by <code>INTEL_PROF_DUMP_INTERVAL</code> or the function <code>_PGOPTI_Set_Interval_Prof_Dump</code>) during the execution of an instrumented user application, allows creation of a single .dyn file to contain profiling information instead of multiple .dyn files. If this environment variable is not set, executing an instrumented user application creates a new .dyn file for each interval. Setting this environment variable is useful for applications that do not terminate or those that terminate abnormally (bypass the normal exit code).</td>
</tr>
<tr>
<td><strong>INTEL_PROF_DUMP_INTERVAL</strong></td>
<td>Initiates interval profile dumping in an instrumented user application. This environment variable may be used to initiate Interval Profile Dumping in an instrumented application. See <a href="#">Interval Profile Dumping</a> for more information.</td>
</tr>
<tr>
<td><strong>PROF_DIR</strong></td>
<td>Specifies the directory in which dynamic information files are created. This variable applies to all three phases of the profiling process.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PROF_DUMP_INTERVAL</td>
<td>Deprecated. Please use INTEL_PROF_DUMP_INTERVAL instead.</td>
</tr>
<tr>
<td>PROF_NO_CLOBBER</td>
<td>Alters the feedback compilation phase slightly. By default, during the feedback compilation phase, the compiler merges data from all dynamic information files and creates a new pgopti.dpi file if the .dyn files are newer than an existing pgopti.dpi file. When this variable is set the compiler does not overwrite the existing pgopti.dpi file. Instead, the compiler issues a warning. You must remove the pgopti.dpi file if you want to use additional dynamic information files.</td>
</tr>
</tbody>
</table>

See the appropriate operating system documentation for instructions on how to specify environment variables and their values.

**Dumping Profile Information**

The _PGOPTI_Prof_Dump_All() function dumps the profile information collected by the instrumented application. The prototype of the function call is listed below.

**Syntax**

```c
void _PGOPTI_Prof_Dump_All(void);
```

An older version of this function, _PGOPTI_Prof_Dump(), which will also dump profile information is still available; the older function operates much like _PGOPTI_Prof_Dump_All(), except on Linux when used in connection with shared libraries (.so) and _exit() to terminate a program. When _PGOPTI_Prof_Dump_All() is called before _exit() to terminate the program, the new function insures that a .dyn file is created for all shared libraries needing to create a .dyn file. Use _PGOPTI_Prof_Dump_All() on Linux to insure portability and correct functionality.

The profile information is generated in a .dyn file (generated in phase 2 of PGO).
**Recommended usage**

Insert a single call to this function in the body of the function which terminates the user application. Normally, `_PGOPTI_Prof_Dump_All()` should be called just once. It is also possible to use this function in conjunction with `_PGOPTI_Prof_Reset()` function to generate multiple .dyn files (presumably from multiple sets of input data).

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| #include <pgouser.h>  
void process_data(int foo) {}  
int get_input_data() { return 1; }  
int main(void)  
{  
// Selectively collect profile information for the portion  
// of the application involved in processing input data.  
int input_data = get_input_data();  
while (input_data) {  
 _PGOPTI_Prof_Reset();  
 process_data(input_data);  
 _PGOPTI_Prof_Dump_All();  
 input_data = get_input_data();  
}  
return 0;  
} |

**Interval Profile Dumping**

The `_PGOPTI_Set_Interval_Prof_Dump()` function activates Interval Profile Dumping and sets the approximate frequency at which dumps occur. This function is used in non-terminating applications.

The prototype of the function call is listed below.
Syntax

```
void _PGOPTI_Set_Interval_Prof_Dump(int interval);
```

This function is used in non-terminating applications.

The `interval` parameter specifies the time interval at which profile dumping occurs and is measured in milliseconds. For example, if interval is set to 5000, then a profile dump and reset will occur approximately every 5 seconds. The interval is approximate because the time-check controlling the dump and reset is only performed upon entry to any instrumented function in your application.

Setting the interval to zero or a negative number will disable interval profile dumping, and setting a very small value for the interval may cause the instrumented application to spend nearly all of its time dumping profile information. Be sure to set interval to a large enough value so that the application can perform actual work and substantial profile information is collected.

The following example demonstrates one way of using interval profile dumping in non-terminating code.

**Example**

```c
#include <stdio.h>
// The next include is to access
// _PGOPTI_Set_Interval_Prof_Dump_All
#include <pgouser.h>
int returnValue()
{ return 100; }
int main()
{
    int ans;
    printf("CTRL-C to quit.\n");
    _PGOPTI_Set_Interval_Prof_Dump(5000);
    while (1)
        ans = returnValue();
}
```

You can compile the code shown above by entering commands similar to the following:
When compiled, the code shown above will dump profile information a .dyn file about every five seconds until the program is ended.

You can use the profmerge tool to merge the .dyn files.

**Recommended usage**

Call this function at the start of a non-terminating user application to initiate interval profile dumping. Note that an alternative method of initiating interval profile dumping is by setting the environment variable `INTEL_PROF_DUMP_INTERVAL` to the desired interval value prior to starting the application.

Using interval profile dumping, you should be able to profile a non-terminating application with minimal changes to the application source code.

**Resetting the Dynamic Profile Counters**

The `_PGOPTI_Prof_Reset()` function resets the dynamic profile counters. The prototype of the function call is listed below.

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>void _PGOPTI_Prof_Reset(void);</td>
</tr>
</tbody>
</table>

**Recommended usage**

Use this function to clear the profile counters prior to collecting profile information on a section of the instrumented application. See the example under Dumping Profile Information.

**Dumping and Resetting Profile Information**

The `_PGOPTI_Prof_Dump_And_Reset()` function dumps the profile information to a new .dyn file and then resets the dynamic profile counters. Then the execution of the instrumented application continues.
The prototype of the function call is listed below.

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>void _PGOPTI_Prof_DUMP_And_Reset(void);</td>
</tr>
</tbody>
</table>

This function is used in non-terminating applications and may be called more than once. Each call will dump the profile information to a new .dyn file.

**Recommended usage**

Periodic calls to this function enables a non-terminating application to generate one or more profile information files (.dyn files). These files are merged during the feedback phase (phase 3) of profile-guided optimizations. The direct use of this function enables your application to control precisely when the profile information is generated.
Using High-Level Optimization (HLO)

High-Level Optimizations (HLO) Overview

HLO exploits the properties of source code constructs (for example, loops and arrays) in applications developed in high-level programming languages. Within HLO, loop transformation techniques include:

- Loop Permutation or Interchange
- Loop Distribution
- Loop Fusion
- Loop Unrolling
- Data Prefetching
- Scalar Replacement
- Unroll and Jam
- Loop Blocking or Tiling
- Partial-Sum Optimization
- Loadpair Optimization
- Predicate Optimization
- Loop Versioning with Runtime Data-Dependence Check (IA-64 architecture only)
- Loop Versioning with Low Trip-Count Check
- Loop Reversal
- Profile-Guided Loop Unrolling
- Loop Peeling
- Data Transformation: Malloc Combining and Memset Combining
- Loop Rerolling
- Memset and Memcpy Recognition
- Statement Sinking for Creating Perfect Loopnests

While the default optimization level, `-O2` (Linux* OS and Mac OS* X) or `/O2` (Windows* OS) option, performs some high-level optimizations (for example, prefetching, complete unrolling, etc.), specifying `-O3` (Linux and Mac OS X) or `/O3` (Windows) provides the best chance for performing loop transformations to optimize memory accesses; the scope of optimizations enabled by these options is different for IA-32 architecture, Intel® 64, and IA-64 architectures.
Applications for the IA-32 and Intel® 64 architectures

In conjunction with the vectorization options, -ax and -x (Linux and Mac OS X) or /Qax and /Qx (Windows), the -O3 (Linux and Mac OS X) or /O3 (Windows) option causes the compiler to perform more aggressive data dependency analysis than the default -O2 (Linux and Mac OS X) or /O2 (Windows).

Compiler prefetching is disabled in favor of the prefetching support available in the processors.

Applications for the IA-32 and IA-64 architectures

The -O3 (Linux and Mac OS X) or /O3 (Windows) option enables the -O2 (Linux and Mac OS X) or /O2 (Windows) option and adds more aggressive optimizations (like loop transformations); O3 optimizes for maximum speed, but may not improve performance for some programs.

Applications for the IA-64 architecture

The -ivdep-parallel (Linux) or /Qivdep-parallel (Windows) option implies there is no loop-carried dependency in the loop where an ivdep pragma is specified. (This strategy is useful for sparse matrix applications.)

Tune applications for IA-64 architecture by following these general steps:

1. Compile your program with -O3 (Linux) or /O3 (Windows) and -ipo (Linux) or /Qipo (Windows). Use profile guided optimization whenever possible.
2. Identify hot spots in your code.
3. Generate a high-level optimization report.
4. Check why loops are not software pipelined.
5. Make the changes indicated by the results of the previous steps.
6. Repeat these steps until you achieve the desired performance.

General Application Tuning

In general, you can use the following strategies to tune applications for multiple architectures:

- Use #pragma ivdep to indicate there is no dependence. You might need to compile with the -ivdep-parallel (Linux and Mac OS X) or /Qivdep-parallel (Windows) option to absolutely specify no loop carried dependence.
• Use `#pragma swp` to enable software pipelining (useful for lop-sided controls and unknown loop count).

• Use `#pragma loop count(n)` when needed.

• Use of `-ansi-alias` (Linux and Mac OS X) or `/Qansi-alias` (Windows) is helpful.

• Add the restrict keyword to insure there is no aliasing. Compile with `-restrict` (Linux) or `/Qrestrict` (Windows).

• Use `-fargument-alias` (Linux and Mac OS X) or `/Qalias-args-` (Windows) to indicate arguments are not aliased.

• Use `#pragma distribute point` to split large loops (normally this is done automatically).

• For C code, do not use unsigned int for loop indexes. HLO may skip optimization due to possible subscripts overflow. If upper bounds are pointer references, assign it to a local variable whenever possible.

• Check that the prefetch distance is correct. Use `#pragma prefetch` to override the distance when it is needed.

**Loop Unrolling**

The benefits of loop unrolling are as follows:

• Unrolling eliminates branches and some of the code.

• Unrolling enables you to aggressively schedule (or pipeline) the loop to hide latencies if you have enough free registers to keep variables live.

• For processors based on the IA-32 architectures, the processor can correctly predict the exit branch for an inner loop that has 16 or fewer iterations, if that number of iterations is predictable and there are no conditional branches in the loop. Therefore, if the loop body size is not excessive, and the probable number of iterations is known, unroll inner loops for the processors, until they have a maximum of 16 iterations.

• A potential limitation is that excessive unrolling, or unrolling of very large loops, can lead to increased code size.

The `-unroll[n]` (Linux* and Mac OS* X) or `/Qunroll:[n]` (Windows*) option controls how the Intel® compiler handles loop unrolling.

Refer to **Applying Optimization Strategies** for more information.
Specifies the maximum number of times you want to unroll a loop. The following examples unrolls a loop four times:

(Windows)

icpc -unroll 4 a.cpp
icl /Qunroll:4 a.cpp

NOTE. The compilers for IA-64 architecture recognizes only \( n = 0 \); any other value is ignored.

Omitting a value for \( n \) lets the compiler decide whether to perform unrolling or not. This is the default; the compiler uses default heuristics or defines \( n \).

Passing 0 as \( n \) disables loop unrolling; the following examples disables loop unrolling:

(Windows)

icpc -unroll 0 a.cpp
icl /Qunroll:0 a.cpp

-\texttt{unrolln} \quad \texttt{/Qunroll:n}
Loop Independence

Loop independence is important since loops that are independent can be parallelized. Independent loops can be parallelized in a number of ways, from the course-grained parallelism of OpenMP*, to fine-grained Instruction Level Parallelism (ILP) of vectorization and software pipelining.

Loops are considered independent when the computation of iteration Y of a loop can be done independently of the computation of iteration X. In other words, if iteration 1 of a loop can be computed and iteration 2 simultaneously could be computed without using any result from iteration 1, then the loops are independent.

Occasionally, you can determine if a loop is independent by comparing results from the output of the loop with results from the same loop written with a decrementing index counter.

<table>
<thead>
<tr>
<th>Linux and Mac OS X</th>
<th>Windows</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-funroll-all-loops</td>
<td>No equivalent</td>
<td>Instructs the compiler to unroll all loops even if the number of iterations is uncertain when the loop is entered.</td>
</tr>
</tbody>
</table>
For example, the loop shown in example 1 might be independent if the code in example 2 generates the same result.

```
Example
#define MAX 1024
void loop_indep1(int a[MAX],int b[MAX])
{
    for (int j=0;j<MAX;j++)
        a[j] = b[j];
}
#define MAX 1024
void loop_indep2(int a[MAX],int b[MAX])
{
    for (int j=MAX;j>0;j--)
        a[j] = b[j];
}
```

When loops are dependent, improving loop performance becomes much more difficult. Loops can be dependent in several, general ways.

- Flow Dependency
- Anti Dependency
- Output Dependency
- Reductions

The following sections illustrate the different loop dependencies.
Flow Dependency - Read After Write

Cross-iteration flow dependence is created when variables are written then read in different iterations, as shown in the following example:

**Example**

```c
void flow_dep(double A[])
{
    for (int j=1; j<1024; j++)
        A[j]=A[j-1];
}
```

The above example is equivalent to the following lines for the first few iterations:

**Sample Iterations**

```c
```

Recurrence relations feed information forward from one iteration to the next.

**Example**

```c
void time_stepping_loops(double a[], double b[])
{
    for(int j=1; j<MAX; j++) {
        a[j] = a[j-1] + b[j];
    }
}
```

Most recurrences cannot be made fully parallel. Instead, look for a loop further out or further in to parallelize. You might be able to get more performance gains through unrolling.
Anti Dependency - Write After Read

Cross-iteration anti-dependence is created when variables are read then written in different iterations, as shown in the following example:

Example

```c
void anti_dep1(double A[])
{
    for (int j=1; j<1024; j++)
        A[j]=A[j+1];
}
```

The above example is equivalent to the following lines for the first few iterations:

Sample Iterations

```c
```

Output Dependency - Write After Write

Cross-iteration output dependence is where variables are written then rewritten in a different iteration. The following example illustrates this type of dependency:

Example

```c
void anti_dep2(double A[], double B[], double C[])
{
    for (int j=1; j<1024; j++) {
        A[j]=B[j];
        A[j+1]=C[j];
    }
}
```

The above example is equivalent to the following lines for the first few iterations:
### Sample Iterations

<table>
<thead>
<tr>
<th>Sample Iterations</th>
</tr>
</thead>
</table>

### Reductions

The Intel® compiler can successfully vectorize or software pipeline (SWP) most loops containing reductions on simple math operators like multiplication (*), addition (+), subtraction (-), and division (/). Reductions collapse array data to scalar data by using associative operations:

#### Example

```c
void reduction(double * sum, double c[])
{
    for (int j=0; j<MAX; j++) {
        *sum = *sum + c[j];
    }
}
```

The compiler might occasionally misidentify a reduction and report flow-, anti-, output-dependencies or sometimes loop-carried memory-dependency-edges; in such cases, the compiler will not vectorize or SWP the loop. In such cases, recognize that the programming construct is simply a reduction, and direct the compiler through the use of pragmas to vectorize or SWP the loop; you can pragmas (like, #pragma ivdep or #pragma swp) to help the compiler in these cases.

### Prefetching with Options

The goal of prefetch insertion optimization is to reduce cache misses by providing hints to the processor about when data should be loaded into the cache. The prefetch optimization is enabled or disabled by the `-opt-prefetch` (Linux® and Mac OS® X) or `/Qopt-prefetch` (Windows®) compiler option. This option also allows you to specify the level of software prefetching.
To facilitate compiler optimization:

- Minimize use of global variables and pointers.
- Minimize use of complex control flow.
- Choose data types carefully and avoid type casting.

In addition to the `-opt-prefetch` (Linux and Mac OS X) or `/Qopt-prefetch` (Windows) option, an intrinsic subroutine `mm_prefetch` and compiler directive `prefetch` are also available.

The architecture affects the option behavior. For more information about the differences, see the following topic:

- `-opt-prefetch` compiler option

**See Also**
- Using High-Level Optimization (HLO)
- Other Resources
Prefetching Support

Data prefetching refers to loading data from a relatively slow memory into a relatively fast cache before the data is needed by the application. Data prefetch behavior depends on the architecture:

- IA-64 architecture: The Intel® compiler generally issues prefetch instructions when you specify `-O1`, `-O2`, and `-O3` (Linux*) or `/O1`, `/O2`, and `/O3` (Windows*).
- IA-32 and Intel® 64 architectures: The processor identifies simple, regular data access patterns and performs a hardware prefetch. The compiler will only issue prefetch instructions for more complicated data access patterns where a hardware prefetch is not expected.

Issuing prefetches improves performance in most cases; however, there are cases where issuing prefetch instructions might slow application performance. Experiment with prefetching; it can be helpful to turn prefetching on or off with a compiler option while leaving all other optimizations unaffected to isolate a suspected prefetch performance issue. See Prefetching with Options for information on using compiler options for prefetching data.

There are two primary methods of issuing prefetch instructions. One is by using compiler directives and the other is by using compiler intrinsics.

**prefetch and noprefetch Pragmas**

The `prefetch` and `noprefetch` directives are supported by Itanium® processors only. These directives assert that the data prefetches be generated or not generated for some memory references. This affects the heuristics used in the compiler. The general syntax for these pragmas is shown below:

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma noprefetch</td>
</tr>
<tr>
<td>#pragma prefetch</td>
</tr>
<tr>
<td>#pragma prefetch a,b</td>
</tr>
</tbody>
</table>


If loop includes expression $A(j)$, placing `prefetch A` in front of the loop, instructs the compiler to insert prefetches for $A(j + d)$ within the loop. $d$ is the number of iterations ahead to prefetch the data and is determined by the compiler. This directive is supported with optimization levels of `-O1` (Linux*) or `/O1` (Windows*) or higher. Remember that `-O2` or `/O2` is the default optimization level.

**Example**

```cpp
#pragma noprefetch b
#pragma prefetch a
for(i=0; i<m; i++)
{
    a[i]=b[i]+1;
}
```
The following example, which is for IA-64 architecture only, demonstrates how to use the `prefetch`, `noprefetch`, and `memref_control` pragmas together:

```c
#define SIZE 10000
int prefetch(int *a, int *b)
{
    int i, sum = 0;
    #pragma memref_control a:12
    #pragma noprefetch a
    #pragma prefetch b
    for (i = 0; i<SIZE; i++)
    {
        sum += a[i] * b[i];
    }
    return sum;
}
#include <stdio.h>
int main()
{
    int i, arr1[SIZE], arr2[SIZE];
    for (i = 0; i<SIZE; i++)
    {
        arr1[i] = i;
        arr2[i] = i;
    }
    printf("Demonstrating the use of prefetch, noprefetch, \\
            and memref_control pragma together.\n\n");
    prefetch(arr1, arr2);
    return 0;
}
```
Intrinsics

Before inserting compiler intrinsics, experiment with all other supported compiler options and pragmas. Compiler intrinsics are less portable and less flexible than either a compiler option or compiler pragmas.

Pragmas enable compiler optimizations while intrinsics perform optimizations. As a result, programs with pragmas are more portable, because the compiler can adapt to different processors, while the programs with intrinsics may have to be rewritten/ported for different processors. This is because intrinsics are closer to assembly programming.

Some prefetching intrinsics are:

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__lfetch</td>
<td>Generate the lfetch.lfhint instruction.</td>
</tr>
<tr>
<td>__lfetch_fault</td>
<td>Generate the lfetch.fault.lfhint instruction.</td>
</tr>
<tr>
<td>__lfetch_excl</td>
<td>Generate the lfetch.excl.lfhint instruction.</td>
</tr>
<tr>
<td>__lfetch_fault_excl</td>
<td>Generate the lfetch.fault.excl.lfhint instruction.</td>
</tr>
<tr>
<td>__mm_prefetch</td>
<td>Loads one cache line of data from address a to a location closer to the processor.</td>
</tr>
</tbody>
</table>

See Operating System Related Intrinsics and Cacheability Support Using Streaming SIMD Extensions in the Compiler Reference for more information about these intrinsics.
The following example demonstrates how to generate an `lfetch.nt2` instruction using prefetch intrinsics:

```c
Example
for (i=i0; i!=i1; i+=is) {
    float sum = b[i];
    int ip = srow[i];
    int c = col[ip];
    for(; ip<srow[i+1]; c=col[++ip])
        lfetch(2, &value[ip+40]);
    // mm_prefetch(&value[ip+40], 2);
    sum -= value[ip] * x[c];
    y[i] = sum;
}
```

For Intel® Streaming SIMD Extensions-enabled processors you could also use the following Intel® SSE intrinsics:

- `_mm_prefetch`
- `_mm_stream_pi`
- `_mm_stream_ps`
- `_mm_sfence`

You can find more information about IA-64 architecture instructions by referring to the hardware and software programming resources listed in Other Resources.

**About Register Allocation**

The Intel® Compiler for IA-32 and Intel® 64 architectures contains an advanced, region-based register allocator. Register allocation can be influenced using the `–opt-ra-region-strategy` (Linux® and Mac OS® X) and `/Qopt-ra-region-strategy` (Windows®) option.
The register allocation high-level strategy when compiling a routine is to partition the routine into regions, assign variables to registers or memory within each region, and resolve discrepancies at region boundaries. The overall quality of the allocation depends heavily on the region partitioning.

By default, the Intel Compiler selects the best region partitioning strategy, but the -opt-ra-region-strategy (Linux* and Mac OS X) and /Qopt-ra-region-strategy (Windows) option allows you to experiment with the other available allocation strategies, which might result in better performance in some cases. The option provides several different arguments that allow you to specify the following allocation strategies:

- routine = a region for each routine
- trace = a region for each trace
- loop = a region for each loop
- block = a region for each block
- default = the compiler selects best allocation strategy

See the /Qopt-ra-region-strategy (Windows) -opt-ra-region-strategy (Linux and Mac OS X) compiler option for additional information.

The option can affect compile time. Register allocation is a relatively costly operation, and the time spent in register allocation tends to grow as the number of regions increases. Expect relative compile time to increase in the order listed, from shortest to longest:

1. routine-based regions (shortest)
2. loop-based regions
3. trace-based regions
4. block-based regions (longest)

Trace-based regions tend to work very well when profile guided optimizations are enabled. The allocator is able to construct traces that accurately reflect the hot paths through the routine.

In the absence of profile information, loop-based regions tend to work well because the execution profile of a program tends to match its loop structure. In programs where the execution profile does not match the loop structure, routine- or block-based regions can produce better allocations.

Block-based regions provide maximum flexibility to the allocator and in many cases can produce the best allocation; however, the allocator is sometimes over-aggressive with block-based regions about allocating variables to registers; the behavior can lead to poor allocations in the IA-32 and Intel® 64 architectures where registers can be scarce resources.
Register Allocation Example Scenarios

Consider the following example, which illustrates a control flow that results from a simple if-then-else statement within a loop.

There are 3 variables in this loop: \( i \), \( s \), and \( n \). For this example, assume there are only two registers available to hold these three variables; one, or more, variable will need to be stored in memory for at least part of the loop.

The best choice depends on which path through the loop is more frequently executed. For example, if B1, B2, B4 is the hot path, keeping variables \( i \) and \( n \) in registers along that path and saving and restoring one of them in B3 to free up a register for \( s \) is the best strategy. That scenario avoids all memory accesses on the hot path. If B1, B3, B4 is the hot path, the best strategy is to keep variables \( i \) and \( s \) in registers and to store \( n \) in memory, since there are no assignments to \( n \) along the path. This strategy results in a single memory read on the hot path.

If both paths are executed with equal frequency, the best strategy is to save and restore either \( i \) or \( n \) around B3 just like in the B1, B2, B4 case. That case avoids all memory accesses on one path and results in a single memory write and a single memory read on the other path.

The compiler generates two significantly different allocations for this example depending on the region strategy; the preferred result depends on the runtime behavior of the program, which may not be known at compile time.
With a routine- or a loop-based strategy, all four blocks will be allocated together. The compiler picks a variable to store in memory based on expected costs. In this example, the allocator will probably select variable $n$, resulting in a memory write in B2 and a memory read in B3.

With a trace-based strategy, the compiler uses estimates of execution frequency to select the most frequently executed path through the loop. When profile guided optimizations are enabled the estimates are based on the concrete information about the runtime behavior of the instrumented program. If the PGO information accurately reflects typical application behavior, the compiler produces highly accurate traces. In other cases, the traces are not necessarily an accurate reflection of the hot paths through the code.

Suppose in this example that the compiler selects B1, B2, B4 path as the hot trace. The compiler will assign these three blocks to one region, and B3 will be in a separate region. There are only two variables in the larger region, so both may be kept in registers. In the region containing just B3 either $i$ or $n$ are stored in memory, and the compiler makes an arbitrary choice between the two variables. In a block-based strategy, each block is a region. In the B1, B2, B4 path there are sufficient registers to hold the two variables: $i$ and $n$. The region containing B3 is treated just like the trace-based case; either $i$ or $n$ will be stored in memory.
Understanding Run-time Performance

The information in this topic assumes that you are using a performance optimization methodology and have analyzed the application type you are optimizing.

After profiling your application to determine where best to spend your time, attempt to discover what optimizations and what limitations have been imposed by the compiler. Use the compiler reports to determine what to try next.

Depending on what you discover from the reports you may be able to help the compiler through options, pragmas, and slight code modifications to take advantage of key architectural features to achieve the best performance.

The compiler reports can describe what actions have been taken and what actions cannot be taken based on the assumptions made by the compiler. Experimenting with options and pragmas allows you to use an understanding of the assumptions and suggest a new optimization strategy or technique.

Helping the Compiler

You can help the compiler in some important ways:

- Read the appropriate reports to gain an understanding of what the compiler is doing for you and the assumptions the compiler has made with respect to your code.
- Use specific options, intrinsics, libraries, and pragmas to get the best performance from your application.

For example, if your code is attempting to constantly compute square roots of single precision values, you can gain performance by using the appropriate intrinsic for single precision data type; for example, sqrtf() instead of sqrt() in C.

See Applying Optimization Strategies for other suggestions.
Memory Aliasing For IA-64 Architectures

Memory aliasing is the single largest issue affecting the optimizations in the Intel® compiler for IA-64 architecture based systems. Memory aliasing is writing to a given memory location with more than one pointer. The compiler is cautious to not optimize too aggressively in these cases; if the compiler optimizes too aggressively, unpredictable behavior can result (for example, incorrect results, abnormal termination, etc.).

Since the compiler usually optimizes on a module-by-module, function-by-function basis, the compiler does not have an overall perspective with respect to variable use for global variables or variables that are passed into a function; therefore, the compiler usually assumes that any pointers passed into a function are likely to be aliased. The compiler makes this assumption even for pointers you know are not aliased. This behavior means that perfectly safe loops do not get pipelined or vectorized, and performance suffers.

There are several ways to instruct the compiler that pointers are not aliased:

- Use a comprehensive compiler option, such as `-fno-alias` (Linux*) or `/Oa` (Windows*). These options instruct the compiler that no pointers in any module are aliased, placing the responsibility of program correctness directly with the developer.

- Use a less comprehensive option, like `-fno-fnalias` (Linux) or `/Ow` (Windows). These options instruct the compiler that no pointers passed through function arguments are aliased. Function arguments are a common example of potential aliasing that you can clarify for the compiler. You may know that the arguments passed to a function do not alias, but the compiler is forced to assume so. Using these options tells the compiler it is now safe to assume that these function arguments are not aliased. This option is still a somewhat bold statement to make, as it affects all functions in the module(s) compiled with the `-fno-nalias` (Linux) or `/Ow` (Windows) option.

- Use the `ivdep` pragma. Alternatively, you might use a pragma that applies to a specified loop in a function. This is more precise than specifying an entire function. The pragma asserts that, for a given loop, there are no vector dependencies. Essentially, this is the same as saying that no pointers are aliasing in a given loop.

- Use of keyword `restrict`. An even more precise method of disambiguating pointers is the `restrict` keyword. The `restrict` keyword is used to identify individual pointers as not being aliased. You would use the `restrict` keyword to tell the compiler that a given memory location is not written to by any other pointer.
The following example demonstrates using the restrict keyword to tell the compiler that the memory address pointed to by `z` is not written to by any other pointer. With this new information the compiler can then vectorize or software pipeline the loop as follows:

### Example

```c
// One-dimension array.
void restrict1(int *x, int *y, int * restrict z)
{
    int A = 42;
    int i;
    double temp;
    for(i=0;i<100;i++) {
        z[i] = A * x[i] + y[i];
    }
}

// Two-dimension array.
void restrict2(int a[][100], int b[restrict][100]) {
    /* ... */
}
```

**CAUTION.** To use the restrict keyword as in the example above, you must also use the `-restrict` (Linux*) or `/Qrestrict` (Windows*) option on the compile line.
Non-Unit Stride Memory Access

Another issue that can have considerable impact on performance is accessing memory in a non-Unit Stride fashion. This means that as your inner loop increments consecutively, you access memory from non adjacent locations. For example, consider the following matrix multiplication code:

```c
// Non-Unit Stride access problem with b[k][j]
void non_unit_stride(int **a, int **b, int **c)
{
    int A = 42;
    for(int i=0; i<A; i++)
        for(int j=0; j<A; j++)
            for(int k=0; k<A; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

Notice that \( c[i][j] \) and \( a[i][k] \) both access consecutive memory locations when the inner-most loops associated with the array are incremented. The \( b \) array however, with its loops with indexes \( k \) and \( j \), does not access Memory Unit Stride. When the loop reads \( b[k=0][j=0] \) and then the \( k \) loop increments by one to \( b[k=1][j=0] \).

Loop transformation (sometimes called loop interchange) helps to address this problem. While the compiler is capable of doing loop interchange automatically, it does not always recognize the opportunity.

The memory access pattern for the example code listed above is illustrated in the following figure:
Assume you modify the example code listed above by making the following changes to introduce loop interchange:

```c
// After loop interchange of k and j loops.
void unit_stride(int **a, int **b, int **c)
{
    int A = 42;
    for(int i=0; i<A; i++)
        for(int k=0; k<A; k++)
            for(int j=0; j<A; j++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

After the loop interchange the memory access pattern might look the following figure:

---

**Understanding Data Alignment**

Aligning data on boundaries can help performance. The Intel® compiler attempts to align data on boundaries for you. However, as in all areas of optimization, coding practices can either help or hinder the compiler and can lead to performance problems.

Always attempt to optimize using compiler options first.

To avoid performance problems you should keep the following guidelines in mind, which are separated by architecture:

**IA-32, Intel® 64, and IA-64 architectures:**

- Do not access or create data at large intervals that are separated by exactly $2^n$ (for example, 1 KB, 2 KB, 4 KB, 16 KB, 32 KB, 64 KB, 128 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, etc.).
- Align data so that memory accesses does not cross cache lines (for example, 32 bytes, 64 bytes, 128 bytes).
- Use `__mm_malloc(size,alignment,[offset])` to force allocated structures to be enforce the rules above.
- Use Application Binary Interface (ABI) for the Itanium® compiler to insure that ITP pointers are 16-byte aligned.

IA-32 and Intel® 64 architectures:
- Align data to correspond to the SIMD or Streaming SIMD Extension registers sizes.
- Use either \texttt{__assume \_aligned()} or \texttt{#pragma vector aligned} to instruct the compiler that the data is aligned.

IA-64 architecture:
- Avoid using packed structures.
- Avoid casting pointers of small data elements to pointers of large data elements.
- Do computations on unpacked data, then repack data if necessary, to correctly output the data.
- Use \texttt{__unaligned} keyword on pointers to unaligned data to cause the structure to be accessed one byte at a time. This is a slow alternative.

In general, keeping data in cache has a better performance impact than keeping the data aligned. Try to use techniques that conform to the rules listed above.

**Timing Your Application**

You can start collecting information about your application performance by timing your application. More sophisticated and helpful data can be collected by using performance analysis tools.

**Considerations on Timing Your Application**

One of the performance indicators is your application timing. The following considerations apply to timing your application:

- Run program timings when other users are not active. Your timing results can be affected by one or more CPU-intensive processes also running while doing your timings.
- Try to run the program under the same conditions each time to provide the most accurate results, especially when comparing execution times of a previous version of the same program. Use the same system (processor model, amount of memory, version of the operating system, and so on) if possible.
- If you do need to change systems, you should measure the time using the same version of the program on both systems, so you know each system's effect on your timings.
• For programs that run for less than a few seconds, run several timings to ensure that the results are not misleading. Certain overhead functions like loading libraries might influence short timings considerably.

• If your program displays a lot of text, consider redirecting the output from the program. Redirecting output from the program will change the times reported because of reduced screen I/O.

Timings that show a large amount of system time may indicate a lot of time spent doing , which might be worth investigating.

• For programs that run for less than a few seconds, run several timings to ensure that the results are not misleading. Overhead functions like loading shared libraries might influence short timings considerably.

Use the \texttt{time} command and specify the name of the executable program to provide the following:

• The elapsed, real, or "wall clock" time, which will be greater than the total charged actual CPU time.

• Charged actual CPU time, shown for both system and user execution. The total actual CPU time is the sum of the actual user CPU time and actual system CPU time.

\section*{Applying Optimization Strategies}

The compiler may or may not apply the following optimizations to your loop: Interchange, Unrolling, Cache Blocking, Loop Distribution, Loop Fusion, and LoadPair. These transformations are discussed in the following sections, including how to transform loops manually and how to control them with or internal options.

\subsection*{Loop Interchange}

Loop Interchange is a nested loop transformation applied by \texttt{High-level Optimization (HLO)} that swaps the order of execution of two nested loops. Typically, the transformation is performed to provide sequential Unit Stride access to array elements used inside the loop to improve cache locality. The compiler \texttt{-O3} (Linux* and Mac OS* X) or \texttt{/O3} (Windows*) optimization looks for opportunities to apply loop interchange for you.

\begin{center}
\textbf{26}
\end{center}
The following is an example of a loop interchange.

```c
#define NUM 1024
void loop_interchange(
    double a[][NUM], double b[][NUM],
    double c[][NUM] )
{
    int i,j,k;
    // Loop before Loop Interchange
    for(i=0;i<NUM;i++)
        for(j=0;j<NUM;j++)
            for(k=0;k<NUM;k++)
                c[i][j] =c[i][j] + a[i][k] * b[k][j];
    // Loop after Loop Interchange of k & j loops
    for(i=0;i<NUM;i++)
        for(k=0;k<NUM;k++)
            for(j=0;j<NUM;j++)
                c[i][j] =c[i][j] + a[i][k] * b[k][j];
}
```

See discussion under Non-Unit Stride Memory Access for more detail.

**Unrolling**

Loop unrolling is a loop transformation generally used by HLO that can take better advantage of Instruction-Level Parallelism (ILP), keeping as many functional units busy doing useful work as possible during single loop iteration. In loop unrolling, you add more work to the inside of the loop while doing fewer loop iterations in exchange.

There are pragmas and internal options to control unroll behavior.
Pragma | Description
--- | ---
#pragma unroll | Specifying unroll by itself allows the compiler to determine the unroll factor.
#pragma unroll(n) | Specifying `-unroll n` (Linux* OS and Mac OS X) or `/Qunroll:n` (Windows) instructs the compiler to unroll the loop n times.
#pragma nounroll | Specifies nounroll instructs the compiler not to unroll a specified loop.

Generally, you should unroll loops by factors of cache line sizes; experiment with the number. Consider the following loop:

**Example**

```c
#define NUM 1025
void loop_unroll_before(
    double a[][NUM], double b[][NUM],
    double c[][NUM])
{
    int i,j;
    int N,M;
    N=NUM;
    M=5;
    for(i=0;i<N; i++)
        for (j=0;j<M; j++)
            a[i][j] = b[i][j] + c[i][j];
}
```
Assume you want to unroll the “i” or outer loop by a factor 4, but you notice that 4 does not evenly divide N of 1025. Unrolling in this case is difficult; however, you might use a “post conditioning loop” to take care of the unusual case as follows:

Example

```c
#define NUM 1025
void loop_unroll_after(
  double a[][NUM], double b[][NUM],
  double c[][NUM])
{
  int i, j, K;
  int N, M;
  N = NUM;
  M = 5;
  K = N % 4;
  // Main part of loop.
  for (i = 0; i < N - K; i += 4)
    for (j = 0; j < M; j++) {
      a[i][j] = b[i][j] + c[i][j];
      a[i+1][j] = b[i+1][j] + c[i+1][j];
      a[i+2][j] = b[i+2][j] + c[i+2][j];
      a[i+3][j] = b[i+3][j] + c[i+3][j];
    }
  // Post conditioning part of loop.
  for (i = N - K + 1; i < N; i += 4)
    for (j = 0; j < M; j++)
      a[i][j] = b[i][j] + c[i][j];
}
```
Post conditioning is preferred over pre-conditioning because post conditioning will preserve the data alignment and avoid the cost of memory alignment access penalties.

**Cache Blocking**

Cache blocking involves structuring data blocks so that they conveniently fit into a portion of the L1 or L2 cache. By controlling data cache locality, an application can minimize performance delays due to memory bus access. The application controls the behavior by dividing a large array into smaller blocks of memory so a thread can make repeated accesses to the data while the data is still in cache.

For example, image processing and video applications are well suited to cache blocking techniques because an image can be processed on smaller portions of the total image or video frame. Compilers often use the same technique, by grouping related blocks of instructions close together so they execute from the L2 cache.

The effectiveness of the cache blocking technique depends on data block size, processor cache size, and the number of times the data is reused. Cache sizes vary based on processor. An application can detect the data cache size using the CPUID instruction and dynamically adjust cache blocking tile sizes to maximize performance. As a general rule, cache block sizes should target approximately one-half to three-quarters the size of the physical cache. For systems that are Hyper-Threading Technology (HT Technology) enabled target one-quarter to one-half the physical cache size. (See Designing for Hyper-Threading Technology for more other design considerations.)
Cache blocking is applied in HLO and is used on large arrays where the arrays cannot all fit into cache simultaneously. This method is one way of pulling a subset of data into cache (in a small region), and using this cached data as effectively as possible before the data is replaced by new data from memory.

Example

```c
#define NUM 1024

void cache_blocking_before(
    double a[][NUM][NUM], double b[][NUM][NUM],
    double c[][NUM][NUM], int N )
{
    int i,j,k;
    N=1000;
    for (i=0; i < N; i++)
        for (j=0; j < N; j++)
            for (k=0; k < N; k++)
                a[i][j][k] = a[i][j][k] + b[i][j][k];
}

#define NUM 1024

void cache_blocking_after(
    double a[][NUM][NUM], double b[][NUM][NUM],
    double c[][NUM][NUM], int N )
{
    int i,j,k,u,v;
    N=1000;
    for (v=0; v<N; v+=20)
        for (u=0; u<N; u+=20)
            for (k=v; k< v+20; k++)
                for (j=u;j< u+20;j++)
                    for (i=0;i < N; i++)
```

Intel® C++ Compiler User and Reference Guides
The cache block size is set to 20. The goal is to read in a block of cache, do every bit of computing we can with the data in this cache, then load a new block of data into cache. There are 20 elements of $A$ and 20 elements of $B$ in cache at the same time and you should do as much work with this data as you can before you increment to the next cache block.

Blocking factors will be different for different architectures. Determine the blocking factors experimentally. For example, different blocking factors would be required for single precision versus double precision. Typically, the overall impact to performance can be significant.
Loop Distribution

Loop distribution is a high-level loop transformation that splits a large loop into two smaller loops. It can be useful in cases where optimizations like software-pipelining (SWP) or vectorization cannot take place due to excessive register usage. By splitting a loop into smaller segments, it may be possible to get each smaller loop or at least one of the smaller loops to SWP or vectorize. An example is as follows:

**Example**

```c
#define NUM 1024
void loop_distribution_before(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] )
{
    int i;
    // Before distribution or splitting the loop.
    for (i=0; i< NUM; i++) {
        a[i] = a[i] + i;
        b[i] = b[i] + i;
        c[i] = c[i] + i;
        x[i] = x[i] + i;
        y[i] = y[i] + i;
        z[i] = z[i] + i;
    }
}
#define NUM 1024
void loop_distribution_after(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] )
{
    int i;
```
### Example

```c
// After distribution or splitting the loop.
for (i=0; i< NUM; i++) {
    a[i] = a[i] + i;
    b[i] = b[i] + i;
    c[i] = c[i] + i;
}
for (i=0; i< NUM; i++) {
    x[i] = x[i] + i;
    y[i] = y[i] + i;
    z[i] = z[i] + i;
}
```

There are pragmas to suggest distributing loops to the compiler as follows:

### Example

```c
#pragma distribute point
```

---

1389
Placed outside a loop, the compiler will attempt to distribute the loop based on an internal heuristic. The following is an example of using the pragma outside the loop:

**Example**

```c
#define NUM 1024
void loop_distributionPragma1(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] )
{
int i;
// Before distribution or splitting the loop
#pragma distribute point
for (i=0; i< NUM; i++) {
    a[i] = a[i] + i;
    b[i] = b[i] + i;
    c[i] = c[i] + i;
    x[i] = x[i] + i;
    y[i] = y[i] + i;
    z[i] = z[i] + i;
}
}
```
Placed within a loop, the compiler will attempt to distribute the loop at that point. All loop-carried dependencies will be ignored. The following example uses the pragma within a loop to precisely indicate where the split should take place:

**Example**

```c
#define NUM 1024

void loop_distributionPragma2(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM]
{
    int i;
    // After distribution or splitting the loop.
    for (i=0; i< NUM; i++) {
        a[i] = a[i] +i;
        b[i] = b[i] +i;
        c[i] = c[i] +i;
        #pragma distribute point
        x[i] = x[i] +i;
        y[i] = y[i] +i;
        z[i] = z[i] +i;
    }
}
```

**Loop Fusion**

Loop Fusion is the inverse of Loop Distribution. The idea in loop fusion is to join two loops that have the same trip count in order to reduce loop overhead. The `-O3` (Linux) or `/O3` (Windows) option will attempt loop fusion if the opportunity is present.
Load Pair (Itanium® Compiler)

Load pairs (ldfp) are instructions that load two contiguous single or double precision values from memory in one move. Load pairs can significantly improve performance.

Manual Loop Transformations

There might be cases where these manual transformations are called acceptable or even preferred. As a general rule, you should let the compiler transform loops for you. Manually transform loops as a last resort; use this strategy only in cases where you are attempting to gain performance increases.

Manual loop transformations have many disadvantages, which include the following:

- Application code becomes harder to maintain over time.
- New compiler features can cause you to lose any optimization you gain by manually transforming the loop.
- Architectural requirements might restrict your code to a specific architecture unintentionally.

The HLO report can give you an idea of what loop transformations have been applied by the compiler.

Experimentation is a critical component in manually transforming loops. You might try to apply a loop transformation that the compiler ignored. Sometimes, it is beneficial to apply a manual loop transformation that the compiler has already applied with -O3 (Linux) or /O3 (Windows).

Optimizing the Compilation Process

Symbol Visibility Attribute Options (Linux* and Mac OS* X)

Applications that do not require symbol preemption or position-independent code can obtain a performance benefit by taking advantage of the generic ABI visibility attributes.

Global Symbols and Visibility Attributes

A global symbol is a symbol that is visible outside the compilation unit in which it is declared (compilation unit is a single-source file with the associated include files). Each global symbol definition or reference in a compilation unit has a visibility attribute that controls how it may be referenced from outside the component in which it is defined.

The values for visibility are defined and described in the following topic:
• `-fvisibility` compiler option

**NOTE.** Visibility applies to both references and definitions. A symbol reference's visibility attribute is an assertion that the corresponding definition will have that visibility.

**Symbol Preemption and Optimization**

Sometimes programmers need to use some of the functions or data items from a shareable object, but at the same time, they need to replace other items with definitions of their own. For example, an application may need to use the standard run-time library shareable object, libc.so, but to use its own definitions of the heap management routines `malloc` and `free`.

**NOTE.** In this case it is important that calls to `malloc` and `free` within libc.so use the user's definition of the routines and not the definitions in libc.so. The user's definition should then override, or preempt, the definition within the shareable object.

This functionality of redefining the items in shareable objects is called symbol preemption. When the run-time loader loads a component, all symbols within the component that have default visibility are subject to preemption by symbols of the same name in components that are already loaded. Note that since the main program image is always loaded first, none of the symbols it defines will be preempted (redefined).

The possibility of symbol preemption inhibits many valuable compiler optimizations because symbols with default visibility are not bound to a memory address until run-time. For example, calls to a routine with default visibility cannot be inlined because the routine might be preempted if the compilation unit is linked into a shareable object. A preemptable data symbol cannot be accessed using GP-relative addressing because the name may be bound to a symbol in a different component; and the GP-relative address is not known at compile time.

Symbol preemption is a rarely used feature and has negative consequences for compiler optimization. For this reason, by default the compiler treats all global symbol definitions as non-preemptable (protected visibility). Global references to symbols defined in another compilation unit are assumed by default to be preemptable (default visibility). In those rare cases where all global definitions as well as references need to be preemptable, you can override this default.
Specifying Symbol Visibility Explicitly

The Intel® compiler has visibility attribute options that provide command-line control of the visibility attributes in addition to a source syntax to set the complete range of these attributes. The options ensure immediate access to the feature without depending on header file modifications. The visibility options cause all global symbols to get the visibility specified by the option. There are two variety of options to specify symbol visibility explicitly.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fvisibility=keyword</td>
</tr>
<tr>
<td>-fvisibility-keyword= file</td>
</tr>
</tbody>
</table>

The first form specifies the default visibility for global symbols. The second form specifies the visibility for symbols that are in a file (this form overrides the first form).

Specifying Visibility without the Symbol File

This option sets the visibility for symbols not specified in a visibility list file and that do not have VISIBILITY attribute in their declaration. If no symbol file option is specified, all symbols will get the specified attribute. Command line example:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>icc -fvisibility=protected a.c</td>
</tr>
</tbody>
</table>

You can set the default visibility for symbols using one of the following command line options:

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fvisibility=extern</td>
</tr>
<tr>
<td>-fvisibility=default</td>
</tr>
<tr>
<td>-fvisibility=protected</td>
</tr>
<tr>
<td>-fvisibility=hidden</td>
</tr>
<tr>
<td>-fvisibility=internal</td>
</tr>
</tbody>
</table>
Part IV

Floating-point Operations

Topics:

- Overview: Floating-point Operations
- Understanding Floating-point Operations
- Tuning Performance
- Understanding IEEE Floating-point Operations
Overview: Floating-point Operations

This section introduces the floating-point support in the Intel® C++ Compiler and provides information about using floating-point operations in your applications.

The following table lists some possible starting points:

<table>
<thead>
<tr>
<th>If you are trying to...</th>
<th>Then start with...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understand the programming trade-offs in floating-point applications</td>
<td>Programming Trade-offs in Floating-Point Applications</td>
</tr>
<tr>
<td>Use the -fp-model (Linux* and Mac OS* X) or /fp (Windows*) option</td>
<td>Using the -fp-model or /fp Option</td>
</tr>
<tr>
<td>Set the flush-to-zero (FTZ) or denormals-are-zero (DAZ) flags</td>
<td>Setting the FTZ and DAZ Flags</td>
</tr>
<tr>
<td>Tune the performance of floating-point applications for consistency</td>
<td>Overview: Tuning Performance of Floating-Point Applications</td>
</tr>
</tbody>
</table>
Understanding Floating-point Operations

Using the -fp-model (/fp) Option

The -fp-model (Linux* and Mac OS* X) or /fp (Windows*) option allows you to control the optimizations on floating-point data. You can use this option to tune the performance, level of accuracy, or result consistency for floating-point applications across platforms and optimization levels.

For applications that do not require support for denormalized numbers, the -fp-model or /fp option can be combined with the -ftz (Linux* and Mac OS* X) or -Qftz (Windows*) option to flush denormalized results to zero in order to obtain improved runtime performance on processors based on all Intel architectures (IA-32, Intel® 64, and IA-64 architectures).

You can use keywords to specify the semantics to be used. Possible values of the keywords are as follows:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>precise</td>
<td>Enables value-safe optimizations on floating-point data.</td>
</tr>
<tr>
<td>fast[=1</td>
<td>2]</td>
</tr>
<tr>
<td>strict</td>
<td>Enables precise and except, disables contractions, and enables</td>
</tr>
<tr>
<td></td>
<td><code>pragma stdc fenv_access</code></td>
</tr>
<tr>
<td>source</td>
<td>Rounds intermediate results to source-defined precision and enables</td>
</tr>
<tr>
<td></td>
<td>value-safe optimizations.</td>
</tr>
<tr>
<td>double</td>
<td>Rounds intermediate results to 53-bit (double) precision and enables</td>
</tr>
<tr>
<td></td>
<td>value-safe optimizations.</td>
</tr>
<tr>
<td>extended</td>
<td>Rounds intermediate results to 64-bit (extended) precision and enables</td>
</tr>
<tr>
<td></td>
<td>value-safe optimizations.</td>
</tr>
<tr>
<td>[no-]except</td>
<td>Determines whether strict floating-point exception semantics are used.</td>
</tr>
</tbody>
</table>

The default value of the option is -fp-model fast=1 or /fp:fast=1, which means that the compiler uses more aggressive optimizations on floating-point calculations.
NOTE. Using the default option keyword -fp-model fast or /fp:fast, you may get significant differences in your result depending on whether the compiler uses x87 or SSE2 instructions to implement floating-point operations. Results are more consistent when the other option keywords are used.

Several examples are provided to illustrate the usage of the keywords. These examples show:

- A small example of source code
  Note that the same source code is considered in all the included examples.
- The semantics that are used to interpret floating-point calculations in the source code
- One or more possible ways the compiler may interpret the source code
  Note that there are several ways the compiler may interpret the code; we show just some of these possibilities.

**-fp-model fast or /fp:fast**

Example source code:
```c
float t0, t1, t2;
...
t0 = 4.0f + 0.1f + t1 + t2;
```

When this option is specified, the compiler applies the following semantics:

- Additions may be performed in any order
- Intermediate expressions may use single, double, or extended double precision
- The constant addition may be pre-computed, assuming the default rounding mode
Using these semantics, some possible ways the compiler may interpret the original code are given below:

```c
float t0, t1, t2;
...
```

```c
t0 = (float)((double)t1 + (double)t2) + 4.1f;
```

```c
float t0, t1, t2;
...
```

```c
t0 = (t1 + t2) + 4.1f;
```

```c
float t0, t1, t2;
...
```

```c
t0 = (t1 + 4.1f) + t2;
```

**-fp-model extended or /fp:extended**

This setting is equivalent to `-fp-model precise` on Linux* operating systems based on the IA-32 architecture and `-fp-model precise` or `/fp:precise` on systems based on the IA-64 architecture.

Example source code:

```c
float t0, t1, t2;
...
```

```c
t0 = 4.0f + 0.1f + t1 + t2;
```

When this option is specified, the compiler applies the following semantics:

- Additions are performed in program order
- Intermediate expressions use extended double precision
- The constant addition may be pre-computed, assuming the default rounding mode

Using these semantics, a possible way the compiler may interpret the original code is shown below:

```c
float t0, t1, t2;
...
```

```c
t0 = (float)((((long double)4.1 + (long double)t1) + (long double)t2));
```
-fp-model source or /fp:source

This setting is equivalent to -fp-model precise or /fp:precise on systems based on the Intel® 64 architecture.

Example source code:
```c
float t0, t1, t2;
...
t0 = 4.0f + 0.1f + t1 + t2;
```

When this option is specified, the compiler applies the following semantics:
- Additions are performed in program order
- Intermediate expressions use the precision specified in the source code, that is, single-precision
- The constant addition may be pre-computed, assuming the default rounding mode

Using these semantics, a possible way the compiler may interpret the original code is shown below:
```c
float t0, t1, t2;
...
t0 = ((4.1f + t1) + t2);
```

-fp-model double or /fp:double

This setting is equivalent to -fp-model precise or /fp:precise on Windows systems based on the IA-32 architecture.

Example source code:
```c
float t0, t1, t2;
...
t0 = 4.0f + 0.1f + t1 + t2;
```

When this option is specified, the compiler applies the following semantics:
- Additions are performed in program order
- Intermediate expressions use double precision
- The constant addition may be pre-computed, assuming the default rounding mode
Using these semantics, a possible way the compiler may interpret the original code is shown below:

```c
float t0, t1, t2;
...
```

```c
t0 = (float)(((double)4.1 + (double)t1) + (double)t2);
```

**-fp-model strict or /fp:strict**

Example source code:
```c
float t0, t1, t2;
...
```

```c
t0 = 4.0f + 0.1f + t1 + t2;
```

When this option is specified, the compiler applies the following semantics:

- Additions are performed in program order
- Expression evaluation matches expression evaluation under keyword `precise`.
- The constant addition is not pre-computed because there is no way to tell what rounding mode will be active when the program runs.

Using these semantics, a possible way the compiler may interpret the original code is shown below:
```c
float t0, t1, t2;
...
```

```c
t0 = (float)(((long double)4.0f + (long double)0.1f) + (long double)t1) + (long double)t2);
```

**See Also**
- Understanding Floating-point Operations
- `-fp-model` compiler option

**Setting the FTZ and DAZ Flags**

In Intel® processors, the flush-to-zero (FTZ) and denormals-are-zero (DAZ) flags in the MXCSR register are used to control floating-point calculations. The Intel® Streaming SIMD (Single Instruction Multiple Data) Extensions (Intel® SSE) and the Intel® SSE 2 instructions, including
scalar and vector instructions, benefit from enabling the FTZ and DAZ flags respectively. Floating-point computations using these Intel® SSE instructions are accelerated when the FTZ and DAZ flags are enabled and thus the performance of the application improves.

You can use the `-ftz` (Linux* and Mac OS* X) or `/Qftz` (Windows*) option to flush denormal results to zero when the application is in the gradual underflow mode. This option may improve performance if the denormal values are not critical to your application’s behavior. The `-ftz` and `/Qftz` options, when applied to the main program, set the FTZ and the DAZ hardware flags. The `-no-ftz` and `/Qftz-` options leave the flags as they are.

The following table describes how the compiler processes denormal values based on the status of the FTZ and DAZ flags:

<table>
<thead>
<tr>
<th>Flag</th>
<th>When set to ON, the compiler...</th>
<th>When set to OFF, the compiler...</th>
<th>Supported on</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTZ (flush-to-zero)</td>
<td>Sets denormal results from floating-point calculations to zero</td>
<td>Does not change the denormal results</td>
<td>IA-64, Intel® 64 architectures, and some IA-32 architectures</td>
</tr>
<tr>
<td>DAZ (denormals-are-zero)</td>
<td>Treats denormal values used as input to floating-point instructions as zero</td>
<td>Does not change the denormal instruction inputs</td>
<td>Intel® 64 architecture and some IA-32 architecture</td>
</tr>
</tbody>
</table>

- FTZ and DAZ are not supported on all IA-32 architectures. The FTZ flag is supported only on IA-32 architectures that support Intel® SSE instructions.
- On systems based on the IA-64 architecture, FTZ always works, while on systems based on the IA-32 and Intel® 64 architectures, FTZ only applies to Intel® SSE instructions. Hence, if your application happens to generate denormals using x87 instructions, FTZ does not apply.
- DAZ and FTZ flags are not compatible with IEEE Standard 754, so you should only consider enabling them when strict compliance to the IEEE standard is not required.

Options `-ftz` and `/Qftz` are performance options. Setting these options does not guarantee that all denormals in a program are flushed to zero. They only cause denormals generated at run-time to be flushed to zero.

On Intel®64 and IA-32 systems, the compiler, by default, inserts code into the main routine to set the FTZ and DAZ flags. When `-ftz` or `/Qftz` option is used on IA-32 systems with the option `-msse2` or `/arch:sse2`, the compiler will insert code to conditionally set FTZ/DAZ flags based on a run-time processor check. The `-no-ftz` (Linux* and Mac OS* X) or `/Qftz-` (Windows) will prevent the compiler from inserting any code that might set FTZ or DAZ flags.
When `-ftz` or `/Qftz` is used in combination with an Intel® SSE-enabling option on systems based on the IA-32 architecture (for example, `-msse2` or `/arch:sse2`), the compiler will insert code in the main routine to set FTZ and DAZ. When `-ftz` or `/Qftz` is used without such an option, the compiler will insert code to conditionally set FTZ/DAZ based on a run-time processor check. `-no-ftz` (Linux and Mac OS X) or `/Qftz-` (Windows) will prevent the compiler from inserting any code that might set FTZ or DAZ.

The `-ftz` or `/Qftz` option only has an effect when the main program is being compiled. It sets the FTZ/DAZ mode for the process. The initial thread and any threads subsequently created by that process will operate in the FTZ/DAZ mode.

On systems based on the IA-64 architecture, optimization option O3 sets `-ftz` and `/Qftz`; optimization option O2 sets `-no-ftz` (Linux) and `/Qftz-` (Windows). On systems based on the IA-32 and Intel® 64 architectures, every optimization option O level, except O0, sets `-ftz` and `/Qftz`.

If this option produces undesirable results of the numerical behavior of your program, you can turn the FTZ/DAZ mode off by using `-no-ftz` or `/Qftz-` in the command line while still benefiting from the O3 optimizations.

For some non-Intel processors, you can set the flags manually with the following macros:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable FTZ</td>
<td><code>_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON)</code></td>
</tr>
<tr>
<td>Enable DAZ</td>
<td><code>_MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON)</code></td>
</tr>
</tbody>
</table>

The prototypes for these macros are in `xmmintrin.h` (FTZ) and `pmmintrin.h` (DAZ).

See Also

- Understanding Floating-point Operations

Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture
Overview: Tuning Performance

This section describes several programming guidelines that can help you improve the performance of a floating-point applications:

- Avoid exceeding representable ranges during computation; handling these cases can have a performance impact.
- Use a single-precision type (for example, float) unless the extra precision and/or range obtained through double or long double is required. Greater precision types increase memory size and bandwidth requirements. See Using Efficient Data Types section.
- Reduce the impact of denormal exceptions for all supported architectures.
- Avoid mixed data type arithmetic expressions.

Handling Floating-point Array Operations in a Loop Body

Following the guidelines below will help autovectorization of the loop.

- Statements within the loop body may contain float or double operations (typically on arrays). The following arithmetic operations are supported: addition, subtraction, multiplication, division, negation, square root, MAX, MIN, and mathematical functions such as SIN and COS.
- Writing to a single-precision scalar/array and a double scalar/array within the same loop decreases the chance of autovectorization due to the differences in the vector length (that is, the number of elements in the vector register) between float and double types. If autovectorization fails, try to avoid using mixed data types.

**NOTE.** The special __m64 and __m128 datatypes are not vectorizable. The loop body cannot contain any function calls. Use of the Intel® Streaming SIMD Extensions intrinsics (for example, mm_add_ps) are not allowed.

See Also

- Tuning Performance
- Programming Guidelines for Vectorization
Reducing the Impact of Denormal Exceptions

Denormalized floating-point values are those that are too small to be represented in the normal manner; that is, the mantissa cannot be left-justified. Denormal values require hardware or operating system interventions to handle the computation, so floating-point computations that result in denormal values may have an adverse impact on performance.

There are several ways to handle denormals to increase the performance of your application:

- Scale the values into the normalized range
- Use a higher precision data type with a larger range
- Flush denormals to zero

For example, you can translate them to normalized numbers by multiplying them using a large scalar number, doing the remaining computations in the normal space, then scaling back down to the denormal range. Consider using this method when the small denormal values benefit the program design.

Consider using a higher precision data type with a larger range; for example, by converting variables declared as `float` to be declared as `double`. Understand that making the change can potentially slow down your program. Storage requirements will increase, which will increase the amount of time for loading and storing data from memory. Higher precision data types can also decrease the potential throughput of SSE operations.

If you change the declaration of a variable you might also need to change the libraries you call to use the variable; for example, `cosd()` instead of `cos()`. Another strategy that might result in increased performance is to increase the amount of precision of intermediate values using the `-fp-model [double|extended]` option. However, this strategy might not eliminate all denormal exceptions, so you must experiment with the performance of your application.

If you change the type declaration of a variable, you might also need to change associated library calls, unless these are generic; for example, `cosd()` instead of `cos()`. Another strategy that might result in increased performance is to increase the amount of precision of intermediate values using the `-fp-model [double|extended]` option. However, this strategy might not eliminate all denormal exceptions, so you must experiment with the performance of your application. You should verify that the gain in performance from eliminating denormals is greater than the overhead of using a data type with higher precision and greater dynamic range.

Finally, in many cases denormal numbers can be treated safely as zero without adverse effects on program results. Depending on the target architecture, use flush-to-zero (FTZ) options.
IA-32 and Intel® 64 Architectures

These architectures take advantage of the FTZ (flush-to-zero) and DAZ (denormals-are-zero) capabilities of Intel® Streaming SIMD Extensions (Intel® SSE) instructions.

On Intel® 64 and IA-32-based systems, the compiler, by default, inserts code into the main routine to enable FTZ and DAZ at optimization levels higher than -O0. To enable FTZ and DAZ at -O0, compile the source file containing main() using the -ftz or /Qftz option. When -ftz or /Qftz option is used on IA-32-based systems with the option -mia32 or /arch:IA32, the compiler inserts code to conditionally enable FTZ and DAZ flags based on a run-time processor check.

NOTE. After using flush-to-zero, ensure that your program still gives correct results when treating denormalized values as zero.

IA-64 Architecture

Enable the FTZ mode by using the -ftz (Linux and Mac OS X) or /Qftz (Windows) option on the source file containing main(). The -O3 (Linux and Mac OS X) or /O3 (Windows) option automatically enables -ftz or /Qftz.

NOTE. After using flush-to-zero, ensure that your program still gives correct results when treating denormalized values as zero.

See Also

- Tuning Performance
- Setting the FTZ and DAZ Flags

Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture

Avoiding Mixed Data Type Arithmetic Expressions

Avoid mixing integer and floating-point (float, double, or long double) data in the same computation. Expressing all numbers in a floating-point arithmetic expression (assignment statement) as floating-point values eliminates the need to convert data between fixed and floating-point formats. Expressing all numbers in an integer arithmetic expression as integer values also achieves this. This improves run-time performance.
For example, assuming that \( I \) and \( J \) are both int variables, expressing a constant number (2.) as an integer value (2) eliminates the need to convert the data. The following examples demonstrate inefficient and efficient code.

### Examples

<table>
<thead>
<tr>
<th>Example 1: Inefficient Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>int I, J;</td>
</tr>
<tr>
<td>I = J / 2.;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example 2: Efficient Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>int I, J;</td>
</tr>
<tr>
<td>I = J / 2;</td>
</tr>
</tbody>
</table>

### Special Considerations for Auto-Vectorization of the Innermost Loops

Auto-vectorization of an innermost loop packs multiple data elements from consecutive loop iterations into a vector register, each of which is 128-bit in size.

Consider a loop that uses different sized data, for example, REAL and DOUBLE PRECISION. For REAL data, the compiler tries to pack data elements from four (4) consecutive iterations (32 bits x 4 = 128 bits). For DOUBLE PRECISION data, the compiler tries to pack data elements from two (2) consecutive iterations (64 bits x 2 = 128 bits). Because of the mismatched number of iterations, the compiler sometimes fails to perform auto-vectorization of the loop, after trying to automatically remedy the situation.

If your attempt to auto-vectorize an innermost loop fails, it is a good practice to try using the same sized data. INTEGER and REAL are considered same sized data since both are 32-bit in size.
### Examples

#### Example 1: Non-autovectorizable code

```fortran
DOUBLE PRECISION A(N), B(N)
REAL C(N), D(N)
DO I=1, N
    A(I)=D(I)
    C(I)=B(I)
ENDDO
```

#### Example 2: Auto-vectorizable after automatic distribution into two loops

```fortran
DOUBLE PRECISION A(N), B(N)
REAL C(N), D(N)
DO I=1, N
    A(I)=B(I)
    C(I)=D(I)
ENDDO
```

#### Example 3: Auto-vectorizable as one loop

```fortran
REAL A(N), B(N)
REAL C(N), D(N)
DO I=1, N
    A(I)=B(I)
    C(I)=D(I)
ENDDO
```
Using Efficient Data Types

In cases where more than one data type can be used for a variable, consider selecting the data types based on the following hierarchy, listed from most to least efficient:

- char
- short
- int
- long
- long long
- float
- double
- long double

However, keep in mind that in an arithmetic expression, you should avoid mixing integer and floating-point data.

You can use integer data types (int, int long, etc.) in loops to improve floating point performance. Convert the data type to integer data types, process the data, then convert the data to the old type.

Checking the Floating-point Stack State

On systems based on the IA-32 architectures, when an application calls a function that returns a floating-point value, the returned floating-point value is supposed to be on the top of the floating-point stack. If the return value is not used, the compiler must pop the value off of the floating-point stack in order to keep the floating-point stack in the correct state.

On systems based on Intel(R) 64 architectures, floating-point values are usually returned in the xmm0 register. The floating-point stack is used only when the return value is a long double on Linux* and Mac OS* X systems.

If the application calls a function without defining or incorrectly defining the function's prototype, the compiler cannot determine if the function must return a floating-point value. Consequently, the return value is not popped off the floating-point stack if it is not used. This can cause the floating-point stack to overflow.

The overflow of the stack results in two undesirable situations:

- A NaN value gets involved in the floating-point calculations
• The program results become unpredictable; the point where the program starts making errors can be arbitrarily far away from the point of the actual error.

For systems based on the IA-32 and Intel® 64 architectures, the `-fp-stack-check` (Linux* and Mac OS* X) or `/Qfp-stack-check` (Windows*) option checks whether a program makes a correct call to a function that should return a floating-point value. If an incorrect call is detected, the option places a code that marks the incorrect call in the program. The `-fp-stack-check` (Linux* and Mac OS* X) or `/Qfp-stack-check` (Windows*) option marks the incorrect call and makes it easy to find the error.

**NOTE.** The `-fp-stack-check` (Linux* and Mac OS* X) and the `/Qfp-stack-check` (Windows*) option causes significant code generation after every function/subroutine call to ensure that the floating-point stack is maintained in the correct state. Therefore, using this option slows down the program being compiled. Use the option only as a debugging aid to find floating point stack underflow/overflow problems, which can be otherwise hard to find.

**See Also**
• Tuning Performance
• `-fp-stack-check, /Qfp-stack-check option`
Overview: Understanding IEEE Floating-point Standard

This version of Intel® Compiler uses a close approximation to the IEEE floating-point standard (ANSI/IEEE Std 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*, 1985) unless otherwise stated. This standard is common to many microcomputer-based systems due to the availability of fast processors that implement the required characteristics.

This section outlines the characteristics of the standard and its implementation for the Intel Compilers. Except as noted, the description includes both the IEEE standard and the Intel Compiler implementation.

Floating-point Formats

The IEEE Standard 754 specifies values and requirements for floating-point representation (such as base 2). The standard outlines requirements for two formats: basic and extended, and for two word-lengths within each format: single and double.

On IA-32 and Intel® 64 architecture-based systems, the Intel C/C++ Compiler supports the GCC decimal floating point feature and the following decimal floating types: _Decimal32, _Decimal64, and _Decimal128. This feature is supported for applications written in C only. These three decimal formats are defined in IEEE 754-2008 standard. See [http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1312.pdf](http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1312.pdf).

Special Values

The following lists special values that the Intel® Compiler supports and provides a brief description.

Signed zero

The sign of zero is the same as the sign of a nonzero number. Comparisons, however, consider +0 to be equal to -0. A signed zero is useful in certain numerical analysis algorithms, but in most applications the sign of zero is invisible.
Denormalized numbers

Denormalized numbers (denormals) fill the gap between the smallest positive normalized number and the smallest negative number. Otherwise only (+/-) 0 occurs in that interval. Denormalized numbers extend the range of computable results by allowing for gradual underflow.

Systems based on the IA-32 architecture support a Denormal Operand status flag, which, when set, means at least one of the input operands to a floating-point operation is a denormal. The Underflow status flag is set when a number loses precision and becomes a denormal.

Signed infinity

Infinities are the result of arithmetic in the limiting case of operands with arbitrarily large magnitude. They provide a way to continue when an overflow occurs. The sign of an infinity is simply the sign you obtain for a finite number in the same operation as the finite number approaches an infinite value.

By retrieving the status flags, you can differentiate between an infinity that results from an overflow and one that results from division by zero. Intel® Compiler treats infinity as signed by default. The output value of infinity is +Infinity or -Infinity.

Not a Number

Not a Number (NaN) results from an invalid operation. For instance 0/0 and \( \sqrt{-1} \) result in NaN. In general, an operation involving a NaN produces another NaN. Because the fraction of a NaN is unspecified, there are many possible NaNs.

Intel® Compiler treats all NaNs identically, but provides two different types:

- Signaling NaN, which has an initial fraction bit of 0 (zero). This usually raises an invalid exception when used in an operation.
- Quiet NaN, which has an initial fraction bit of 1.

The floating-point hardware changes a signaling NAN into a quiet NAN during many arithmetic operations, including the assignment operation. An invalid exception may be raised but the resulting floating-point value will be a quiet NaN. Fortran binary and unformatted input and output do not change the internal representations of the values as they are handled. Therefore, signaling and quiet NaNs may be read into real data and output to files in binary form.

The output value of NaN is NaN.
Part V

Intrinsics Reference

Topics:

- Overview: Intrinsics Reference
- Intrinsics for All Intel Architectures
- Intrinsics for IA-64 Architecture
- Data Alignment, Memory Allocation Intrinsics, and Inline Assembly
- MMX(TM) Technology Intrinsics
- Intrinsics for Intel(R) Streaming SIMD Extensions
- Intrinsics for Intel(R) Streaming SIMD Extensions 2
- Intrinsics for Intel(R) Streaming SIMD Extensions 3
- Intrinsics for Intel(R) Supplemental Streaming SIMD Extensions 3
- Intrinsics for Intel(R) Streaming SIMD Extensions 4
- Intrinsic Performance Across Intel Architectures
- Intrinsics for Advanced Encryption Standard Implementation
- Intrinsics for Converting Half Floats
Intrinsics are assembly-coded functions that allow you to use C++ function calls and variables in place of assembly instructions.

Intrinsics are expanded inline eliminating function call overhead. Providing the same benefit as using inline assembly, intrinsics improve code readability, assist instruction scheduling, and help reduce debugging.

Intrinsics provide access to instructions that cannot be generated using the standard constructs of the C and C++ languages.

**Intrinsics for Intel® C++ Compilers**

The Intel® C++ Compiler enables easy implementation of assembly instructions through the use of intrinsics. Intrinsics are provided for the following instructions:

- Intel® Advanced Vector Extensions instructions
- Carry-less multiplication instruction and Advanced Encryption Standard Extensions instructions
- Half-float conversion instructions
- MMX™ Technology instructions
- Intel® Streaming SIMD Extensions (SSE) instructions
- Intel® Streaming SIMD Extensions 2 (SSE2) instructions
- Intel® Streaming SIMD Extensions 3 (SSE3) instructions
- Supplemental Streaming SIMD Extensions 3 (SSSE3) instructions
- Intel® Streaming SIMD Extensions 4 (SSE4) instructions

The Short Vector Math Library intrinsics are documented in this reference. These intrinsics do not have any corresponding assembly instructions.

The Intel C++ Compiler provides IA-64 architecture-specific intrinsics and intrinsics that work across IA-32, Intel® 64, and IA-64 architectures. Most intrinsics map directly to a corresponding assembly instruction, some map to several assembly instructions.

Availability of Intrinsics on Intel Processors

Not all Intel processors support all intrinsics. For information on which intrinsics are supported on Intel processors, visit http://processorfinder.intel.com.

The Processor Spec Finder tool links directly to all processor documentation and the data sheets list the features, including intrinsics, supported by each processor.

Details about Intrinsics

All instructions use the following features:

- Registers
- Data Types

Registers

Intel® processors provide special register sets for different instructions.

The MMX™ instructions use eight 64-bit registers (mm0 to mm7) which are aliased on the floating-point stack registers.

The Intel® Streaming SIMD Extensions (Intel® SSE) and the Advanced Encryption Standard (AES) instructions use eight 128-bit registers (xmm0 to xmm7).

The Intel® Advanced Vector Extension (Intel® AVX) instructions use 256-bit registers which are extensions of the 128-bit SIMD registers.

Because each of these registers can hold more than one data element, the processor can process more than one data element simultaneously. This processing capability is also known as single-instruction multiple data processing (SIMD).

For each computational and data manipulation instruction in the new extension sets, there is a corresponding C intrinsic that implements that instruction directly. This frees you from managing registers and assembly programming. Further, the compiler optimizes the instruction scheduling so that your executable runs faster.

NOTE. The MM and XMM registers are the SIMD registers used by the IA-32 architecture-based platforms to implement MMX technology and Intel® SSE or Intel® SSE2 intrinsics. On the IA-64 architecture, the MMX and Intel® SSE intrinsics use the 64-bit general registers and the 64-bit significand of the 80-bit floating-point register.
## Data Types

Intrinsic functions use new C data types as operands, representing the new registers that are used as the operands to these intrinsic functions.

The following table details for which instructions each of the new data types are available. A 'Yes' indicates that the data type is available for that group of intrinsics; an 'NA' indicates that the data type is not available for that group of intrinsics.

<table>
<thead>
<tr>
<th>Data Types --&gt;</th>
<th>__m64</th>
<th>__m128</th>
<th>__m128d</th>
<th>__m128i</th>
<th>__m256</th>
<th>__m256d</th>
<th>__m256i</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technology</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMX™ Technology Intrinsics</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Intel® Streaming SIMD Extensions Intrinsics</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Intel® Streaming SIMD Extensions 2 Intrinsics</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Intel® Streaming SIMD Extensions 3 Intrinsics</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Advanced Encryption Standard Intrinsics + Carry-less Multiplication Intrinsic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Half-Float Intrinsics</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Intel® Advanced Vector Extensions Intrinsics</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
__m64 Data Type
The __m64 data type is used to represent the contents of an MMX register, which is the register that is used by the MMX technology intrinsics. The __m64 data type can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

__m128 Data Types
The __m128 data type is used to represent the contents of a Streaming SIMD Extension register used by the Streaming SIMD Extension intrinsics. The __m128 data type can hold four 32-bit floating-point values.

The __m128d data type can hold two 64-bit floating-point values.

The __m128i data type can hold sixteen 8-bit, eight 16-bit, four 32-bit, or two 64-bit integer values.

The compiler aligns __m128d and __m128i local and global data to 16-byte boundaries on the stack. To align integer, float, or double arrays, you can use thedeclspec align statement.

__m256 Data Types
The __m256 data type is used to represent the contents of the extended SSE register - the YMM register, used by the Intel® AVX intrinsics. The __m256 data type can hold eight 32-bit floating-point values.

The __m256d data type can hold four 64-bit double precision floating-point values.

The __m256i data type can hold thirty-two 8-bit, sixteen 16-bit, eight 32-bit, or four 64-bit integer values.

Data Types Usage Guidelines
These data types are not basic ANSI C data types. You must observe the following usage restrictions:

• Use data types only on either side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (+, -, etc).
• Use data types as objects in aggregates, such as unions, to access the byte elements and structures.
• Use data types only with the respective intrinsics described in this documentation.
Accessing __m128i Data

To access 8-bit data:

```c
#define _mm_extract_epi8(x, imm) \\
    (((imm) & 0x1) == 0) ? \\
    _mm_extract_epi16((x), (imm) >> 1) & 0xff : \\
    _mm_extract_epi16(_mm_srli_epi16((x), 8), (imm) >> 1))
```

For 16-bit data, use the following intrinsic:

```c
int _mm_extract_epi16(__m128i a, int imm)
```

To access 32-bit data:

```c
#define _mm_extract_epi32(x, imm) \\
    _mm_cvtsi128_si32(_mm_srli_si128((x), 4 * (imm)))
```

To access 64-bit data (Intel® 64 architecture only):

```c
#define _mm_extract_epi64(x, imm) \\
    _mm_cvtsi128_si64(_mm_srli_si128((x), 8 * (imm)))
```

Naming and Usage Syntax

Most intrinsic names use the following notational convention:

```
_mm_<intrin_op>_<suffix>
```

The following table explains each item in the syntax.

| <intrin_op> | Indicates the basic operation of the intrinsic; for example, add for addition and sub for subtraction. |
| <suffix>    | Denotes the type of data the instruction operates on. The first one or two letters of each suffix denote whether the data is packed (p), extended packed (ep), or scalar (s). The remaining letters and numbers denote the type, with notation as follows: |
|            | • s single-precision floating point |
|            | • d double-precision floating point |
|            | • i128 signed 128-bit integer |
A number appended to a variable name indicates the element of a packed object. For example, \( r_0 \) is the lowest word of \( r \). Some intrinsics are "composites" because they require more than one instruction to implement them.

The packed values are represented in right-to-left order, with the lowest value being used for scalar operations. Consider the following example operation:

\[
\begin{align*}
\text{double } a[2] &= \{1.0, 2.0\}; \\
\text{__m128d } t &= \text{_mm_load_pd}(a);
\end{align*}
\]

The result is the same as either of the following:

\[
\begin{align*}
\text{__m128d } t &= \text{_mm_set_pd}(2.0, 1.0); \\
\text{__m128d } t &= \text{_mm_setr_pd}(1.0, 2.0);
\end{align*}
\]

In other words, the xmm register that holds the value \( t \) appears as follows:

\[\begin{array}{c|c|c}
127 & 2.0 & 1.0 \\
\end{array}\]

The "scalar" element is 1.0. Due to the nature of the instruction, some intrinsics require their arguments to be immediates (constant integer literals).

References

See the following publications and internet locations for more information about intrinsics and the Intel architectures that support them. You can find all publications on the Intel website.
<table>
<thead>
<tr>
<th>Internet Location or Publication</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.intel.com/software/products">http://www.intel.com/software/products</a></td>
<td>Technical resource center for hardware designers and developers; contains links to product pages and documentation.</td>
</tr>
<tr>
<td>Intel® Itanium® processor website</td>
<td>Intel website for the Itanium® processor; select the &quot;Technical Documentation&quot; tab for documentation.</td>
</tr>
<tr>
<td>Intel® Itanium® Architecture Software Developer's Manuals, Volume 3: Instruction Set Reference</td>
<td>Contains information and details about Itanium® instructions.</td>
</tr>
<tr>
<td>Intel® 64 and IA-32 architecture manuals</td>
<td>Intel website for Intel(R) 64 and IA-32 architecture manuals.</td>
</tr>
<tr>
<td>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M</td>
<td>Describes the format of the instruction set of Intel® 64 and IA-32 architectures and covers the instructions from A to M.</td>
</tr>
<tr>
<td>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z</td>
<td>Describes the format of the instruction set of Intel® 64 and IA-32 architectures and covers the instructions from N to Z.</td>
</tr>
</tbody>
</table>
Overview: Intrinsic Performance across Intel Architectures

Most of the intrinsics documented in this section function across all Intel architectures, namely IA-32, Intel® 64, and IA-64 architectures. Some of them function across two Intel architectures.

The intrinsics are offered as a convenience to the programmer and are categorized as follows:

- **Integer Arithmetic Intrinsics**
- **Floating-Point Intrinsics**
- **String and Block Copy Intrinsics** (only for IA-32 and Intel® 64 architectures)
- **Miscellaneous Intrinsics**

### Integer Arithmetic Intrinsics

The following table lists and describes integer arithmetic intrinsics that you can use across all Intel architectures.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int abs(int)</code></td>
<td>Returns the absolute value of an integer.</td>
</tr>
<tr>
<td><code>long labs(long)</code></td>
<td>Returns the absolute value of a long integer.</td>
</tr>
<tr>
<td><code>unsigned long _lrotl(unsigned long value, int shift)</code></td>
<td>Implements 64-bit left rotate of <code>value</code> by <code>shift</code> positions.</td>
</tr>
<tr>
<td><code>unsigned long _lrorr(unsigned long value, int shift)</code></td>
<td>Implements 64-bit right rotate of <code>value</code> by <code>shift</code> positions.</td>
</tr>
<tr>
<td><code>unsigned int _rotr(unsigned int value, int shift)</code></td>
<td>Implements 32-bit right rotate of <code>value</code> by <code>shift</code> positions.</td>
</tr>
<tr>
<td><code>unsigned int _rotl(unsigned int value, int shift)</code></td>
<td>Implements 32-bit left rotate of <code>value</code> by <code>shift</code> positions.</td>
</tr>
</tbody>
</table>
Intrinsic Syntax | Description
--- | ---
unsigned short _rotwl(unsigned short value, int shift) | Implements 16-bit left rotate of value by shift positions. These intrinsics are not supported on IA-64 architecture-based platforms.
unsigned short _rotwr(unsigned short value, int shift) | Implements 16-bit right rotate of value by shift positions. These intrinsics are not supported on IA-64 architecture-based platforms.

**NOTE.** Passing a constant shift value in the rotate intrinsics results in higher performance.

**Floating-point Intrinsics**

The following table lists and describes floating point intrinsics that you can use across all Intel architectures.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double fabs(double)</td>
<td>Returns the absolute value of a floating-point value.</td>
</tr>
<tr>
<td>double log(double)</td>
<td>Returns the natural logarithm ln(x), x&gt;0, with double precision.</td>
</tr>
<tr>
<td>float logf(float)</td>
<td>Returns the natural logarithm ln(x), x&gt;0, with single precision.</td>
</tr>
<tr>
<td>double log10(double)</td>
<td>Returns the base 10 logarithm log10(x), x&gt;0, with double precision.</td>
</tr>
<tr>
<td>float log10f(float)</td>
<td>Returns the base 10 logarithm log10(x), x&gt;0, with single precision.</td>
</tr>
<tr>
<td>double exp(double)</td>
<td>Returns the exponential function with double precision.</td>
</tr>
<tr>
<td>float expf(float)</td>
<td>Returns the exponential function with single precision.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>double pow(double, double)</td>
<td>Returns the value of x to the power y with double precision.</td>
</tr>
<tr>
<td>float powf(float, float)</td>
<td>Returns the value of x to the power y with single precision.</td>
</tr>
<tr>
<td>double sin(double)</td>
<td>Returns the sine of x with double precision.</td>
</tr>
<tr>
<td>float sinf(float)</td>
<td>Returns the sine of x with single precision.</td>
</tr>
<tr>
<td>double cos(double)</td>
<td>Returns the cosine of x with double precision.</td>
</tr>
<tr>
<td>float cosf(float)</td>
<td>Returns the cosine of x with single precision.</td>
</tr>
<tr>
<td>double tan(double)</td>
<td>Returns the tangent of x with double precision.</td>
</tr>
<tr>
<td>float tanf(float)</td>
<td>Returns the tangent of x with single precision.</td>
</tr>
<tr>
<td>double acos(double)</td>
<td>Returns the inverse cosine of x with double precision.</td>
</tr>
<tr>
<td>float acosf(float)</td>
<td>Returns the inverse cosine of x with single precision.</td>
</tr>
<tr>
<td>double acosh(double)</td>
<td>Compute the inverse hyperbolic cosine of the argument with double precision.</td>
</tr>
<tr>
<td>float acoshf(float)</td>
<td>Compute the inverse hyperbolic cosine of the argument with single precision.</td>
</tr>
<tr>
<td>double asin(double)</td>
<td>Compute inverse sine of the argument with double precision.</td>
</tr>
<tr>
<td>float asinf(float)</td>
<td>Compute inverse sine of the argument with single precision.</td>
</tr>
<tr>
<td>double asinh(double)</td>
<td>Compute inverse hyperbolic sine of the argument with double precision.</td>
</tr>
<tr>
<td>float asinhf(float)</td>
<td>Compute inverse hyperbolic sine of the argument with single precision.</td>
</tr>
<tr>
<td>double atan(double)</td>
<td>Compute inverse tangent of the argument with double precision.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>double atanh(double)</td>
<td>Compute inverse hyperbolic tangent of the argument with double precision. 900896.</td>
</tr>
<tr>
<td>float atanhf(float)</td>
<td>Compute inverse hyperbolic tangent of the argument with single precision. 480042.</td>
</tr>
<tr>
<td>double cabs(double complex z)</td>
<td>Computes absolute value of complex number. The intrinsic argument is a complex number made up of two double precision elements, one real and one imaginary. The input parameter z is made up of two values of double type passed together as a single argument.</td>
</tr>
<tr>
<td>float cabsf(float complex z)</td>
<td>Computes absolute value of complex number. The intrinsic argument is a complex number made up of two single precision elements, one real and one imaginary. The input parameter z is made up of two values of float type passed together as a single argument.</td>
</tr>
<tr>
<td>double ceil(double)</td>
<td>Computes smallest integral value of double precision argument not less than the argument. 16311637.</td>
</tr>
<tr>
<td>float ceilf(float)</td>
<td>Computes smallest integral value of single precision argument not less than the argument. 16311637.</td>
</tr>
<tr>
<td>double cosh(double)</td>
<td>Computes the hyperbolic cosine of double precision argument. 322522.</td>
</tr>
<tr>
<td>float coshf(float)</td>
<td>Computes the hyperbolic cosine of single precision argument. 160202.</td>
</tr>
<tr>
<td>float fabsf(float)</td>
<td>Computes absolute value of single precision argument. 900896.</td>
</tr>
<tr>
<td>double floor(double)</td>
<td>Computes the largest integral value of the double precision argument not greater than the argument. 322522.</td>
</tr>
<tr>
<td>float floorf(float)</td>
<td>Computes the largest integral value of the single precision argument not greater than the argument. 160202.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>double fmod(double)</td>
<td>Computes the floating-point remainder of the division of the first argument by the second argument with double precision.</td>
</tr>
<tr>
<td>float fmodf(float)</td>
<td>Computes the floating-point remainder of the division of the first argument by the second argument with single precision.</td>
</tr>
<tr>
<td>double hypot(double, double)</td>
<td>Computes the length of the hypotenuse of a right angled triangle with double precision.</td>
</tr>
<tr>
<td>float hypotf(float, float)</td>
<td>Computes the length of the hypotenuse of a right angled triangle with single precision.</td>
</tr>
<tr>
<td>double rint(double)</td>
<td>Computes the integral value represented as double using the IEEE rounding mode.</td>
</tr>
<tr>
<td>float rintf(float)</td>
<td>Computes the integral value represented with single precision using the IEEE rounding mode.</td>
</tr>
<tr>
<td>double sinh(double)</td>
<td>Computes the hyperbolic sine of the double precision argument.</td>
</tr>
<tr>
<td>float sinhf(float)</td>
<td>Computes the hyperbolic sine of the single precision argument.</td>
</tr>
<tr>
<td>float sqrtf(float)</td>
<td>Computes the square root of the single precision argument.</td>
</tr>
<tr>
<td>double tanh(double)</td>
<td>Computes the hyperbolic tangent of the double precision argument.</td>
</tr>
<tr>
<td>float tanhf(float)</td>
<td>Computes the hyperbolic tangent of the single precision argument.</td>
</tr>
</tbody>
</table>

**String and Block Copy Intrinsics**

The following table lists and describes string and block copy intrinsics that you can use on systems based on IA-32 and Intel® 64 architectures.
NOTE. On systems based on IA-64 architectures, you can perform string and block copy operations using the String and Block Copy intrinsics implemented as regular function calls.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char * _strset(char *, _int32)</td>
<td>Sets all characters in a string to a fixed value.</td>
</tr>
<tr>
<td>int memcmp(const void *cs, const void *ct, size_t n)</td>
<td>Compares two regions of memory. Return &lt;0 if cs&lt;ct, 0 if cs=ct, or &gt;0 if cs&gt;ct.</td>
</tr>
<tr>
<td>void *memcpy(void *s, const void *ct, size_t n)</td>
<td>Copies from memory. Returns s.</td>
</tr>
<tr>
<td>void *memset(void * s, int c, size_t n)</td>
<td>Sets memory to a fixed value. Returns s.</td>
</tr>
<tr>
<td>char *strcat(char * s, const char *ct)</td>
<td>Appends to a string. Returns s.</td>
</tr>
<tr>
<td>int(strcmp)</td>
<td>Compares two strings. Return &lt;0 if cs&lt;ct, 0 if cs=ct, or &gt;0 if cs&gt;ct.</td>
</tr>
<tr>
<td>char *strcpy(char * s, const char *ct)</td>
<td>Copies a string. Returns s.</td>
</tr>
<tr>
<td>size_t strlen(const char * cs)</td>
<td>Returns the length of string cs.</td>
</tr>
<tr>
<td>int strncmp(char *, char *, int)</td>
<td>Compare two strings, but only specified number of characters.</td>
</tr>
<tr>
<td>int strncpy(char *, char *, int)</td>
<td>Copies a string, but only specified number of characters.</td>
</tr>
</tbody>
</table>

**Synchronization Primitives**

The synchronization primitive intrinsics provide a variety of operations. Besides performing these operations, each intrinsic has two key properties:

- the function performed is guaranteed to be atomic
• associated with each intrinsic are certain memory barrier properties that restrict the movement of memory references to visible data across the intrinsic operation by either the compiler or the processor

For the following intrinsics, `<type>` is either a 32-bit or 64-bit integer.

**Atomic Fetch-and-op Operations**

- `<type>` __sync_fetch_and_add(<type> *ptr,<type> val)
- `<type>` __sync_fetch_and_and(<type> *ptr,<type> val)
- `<type>` __sync_fetch_and_nand(<type> *ptr,<type> val)
- `<type>` __sync_fetch_and_or(<type> *ptr,<type> val)
- `<type>` __sync_fetch_and_sub(<type> *ptr,<type> val)
- `<type>` __sync_fetch_and_xor(<type> *ptr,<type> val)

**Atomic Op-and-fetch Operations**

- `<type>` __sync_add_and_fetch(<type> *ptr,<type> val)
- `<type>` __sync_sub_and_fetch(<type> *ptr,<type> val)
- `<type>` __sync_or_and_fetch(<type> *ptr,<type> val)
- `<type>` __sync_and_and_fetch(<type> *ptr,<type> val)
- `<type>` __sync_nand_and_fetch(<type> *ptr,<type> val)
- `<type>` __sync_xor_and_fetch(<type> *ptr,<type> val)

**Atomic Compare-and-swap Operations**

- `<type>` __sync_val_compare_and_swap(<type> *ptr, <type> old_val, <type> new_val)
- `int __sync_bool_compare_and_swap(<type> *ptr, <type> old_val, <type> new_val)`

**Atomic Synchronize Operation**

- `void __sync_synchronize (void);`
Atomic Lock-test-and-set Operation

\[
\text{type} \_\text{sync} \_\text{lock} \_\text{test} \_\text{and} \_\text{set}(\text{type} \ *\text{ptr}, \text{type} \ \text{val})
\]

Atomic Lock-release Operation

\[
\text{void} \_\text{sync} \_\text{lock} \_\text{release}(\text{type} \ *\text{ptr})
\]

Miscellaneous Intrinsics

The following tables list and describe intrinsics that you can use across all Intel architectures, except where noted.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_abnormal_termination(void)</td>
<td>Can be invoked only by termination handlers. Returns TRUE if the termination handler is invoked as a result of a premature exit of the corresponding try-finally region.</td>
</tr>
<tr>
<td>__cpuid</td>
<td>Queries the processor for information about processor type and supported features. The Intel® C++ Compiler supports the Microsoft* implementation of this intrinsic. See the Microsoft documentation for details.</td>
</tr>
<tr>
<td>void *_alloca(int)</td>
<td>Allocates memory in the local stack frame. The memory is automatically freed upon return from the function.</td>
</tr>
<tr>
<td>int _bit_scan_forward(int x)</td>
<td>Returns the bit index of the least significant set bit of x. If x is 0, the result is undefined.</td>
</tr>
<tr>
<td>int _bit_scan_reverse(int)</td>
<td>Returns the bit index of the most significant set bit of x. If x is 0, the result is undefined.</td>
</tr>
<tr>
<td>int _bswap(int)</td>
<td>Reverses the byte order of x. Swaps 4 bytes; bits 0-7 are swapped with bits 24-31, bits 8-15 are swapped with bits 16-23.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>__int64 _bswap64(__int64 x)</td>
<td>Reverses the byte order of x. Swaps 8 bytes; bits 0-7 are swapped with bits 56-63, bits 8-15 are swapped with bits 48-55, bits 16-23 are swapped with bits 40-47, and bits 24-31 are swapped with bits 32-39.</td>
</tr>
<tr>
<td>unsigned int __cacheSize(unsigned int cacheLevel)</td>
<td>__cacheSize(n) returns the size in kilobytes of the cache at level n. 1 represents the first-level cache. 0 is returned for a non-existent cache level. For example, an application may query the cache size and use it to select block sizes in algorithms that operate on matrices.</td>
</tr>
<tr>
<td>_exception_code(void)</td>
<td>Returns the exception code.</td>
</tr>
<tr>
<td>_exception_info(void)</td>
<td>Returns the exception information.</td>
</tr>
<tr>
<td>void _enable(void)</td>
<td>Enables the interrupt.</td>
</tr>
<tr>
<td>void _disable(void)</td>
<td>Disables the interrupt.</td>
</tr>
<tr>
<td>int _in_byte(int)</td>
<td>Intrinsic that maps to the IA-32 instruction IN. Transfer data byte from port specified by argument.</td>
</tr>
<tr>
<td>int _in_dword(int)</td>
<td>Intrinsic that maps to the IA-32 instruction IN. Transfer double word from port specified by argument.</td>
</tr>
<tr>
<td>int _in_word(int)</td>
<td>Intrinsic that maps to the IA-32 instruction IN. Transfer word from port specified by argument.</td>
</tr>
<tr>
<td>int _inp(int)</td>
<td>Same as _in_byte</td>
</tr>
<tr>
<td>int _inpd(int)</td>
<td>Same as _in_dword</td>
</tr>
<tr>
<td>int _inpw(int)</td>
<td>Same as _in_word</td>
</tr>
<tr>
<td>int _out_byte(int, int)</td>
<td>Intrinsic that maps to the IA-32 instruction OUT. Transfer data byte in second argument to port specified by first argument.</td>
</tr>
</tbody>
</table>
Intrinsic | Description |
--- | --- |
`int _out_dword(int, int)` | Intrinsic that maps to the IA-32 instruction OUT. Transfer double word in second argument to port specified by first argument. |
`int _out_word(int, int)` | Intrinsic that maps to the IA-32 instruction OUT. Transfer word in second argument to port specified by first argument. |
`int _outp(int, int)` | Same as `_out_byte` |
`int _outpw(int, int)` | Same as `_out_word` |
`int _outpd(int, int)` | Same as `_out_dword` |
`int _popcnt32(int x)` | Returns the number of set bits in x. |
`int _popcnt64(__int64 x)` | Returns the number of set bits in x. |
`__int64 _rdpmc(int p)` | Returns the current value of the 40-bit performance monitoring counter specified by p. |

**Intrinsics for IA-32 and Intel® 64 Architectures Only**

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__int64 _rdtsc(void)</code></td>
<td>Returns the current value of the processor's 64-bit time stamp counter. This intrinsic is not implemented on systems based on IA-64 architecture.</td>
</tr>
<tr>
<td><code>int _setjmp(jmp_buf)</code></td>
<td>A fast version of <code>setjmp()</code>, which bypasses the termination handling. Saves the callee-save registers, stack pointer and return address. This intrinsic is not implemented on systems based on IA-64 architecture.</td>
</tr>
</tbody>
</table>
Overview: Intrinsics for IA-64 Instructions

This section lists and describes the native intrinsics for IA-64 instructions. These intrinsics cannot be used on the IA-32 architecture. The IA-64 intrinsics give programmers access to IA-64 instructions that cannot be generated using the standard constructs of the C and C++ languages.

The prototypes for these intrinsics are in the `ia64intrin.h` header file.

The Intel® Itanium® processor does not support Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics. However, you can use the `sse2mmx.h` emulation pack to enable support for Intel® SSE2 instructions on IA-64 architecture.

For information on how to use Intel® SSE2 intrinsics on IA-64 architecture, see Using Streaming SIMD Extensions on IA-64 Architecture.

For information on how to use MMX™ technology intrinsics on IA-64 architecture, see MMX™ Technology Intrinsics on IA-64 Architecture

Native Intrinsics

The prototypes for the native intrinsics are in the `ia64intrin.h` header file.

Table 420: Integer Operations

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding IA-64 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_m64_dep_mr</td>
<td>Deposit</td>
<td>dep</td>
</tr>
<tr>
<td>_m64_dep_mi</td>
<td>Deposit</td>
<td>dep</td>
</tr>
<tr>
<td>_m64_dep_zr</td>
<td>Deposit</td>
<td>dep.z</td>
</tr>
<tr>
<td>_m64_dep_zi</td>
<td>Deposit</td>
<td>dep.z</td>
</tr>
<tr>
<td>_m64_extr</td>
<td>Extract</td>
<td>extr</td>
</tr>
</tbody>
</table>
### Table 421: FSR Operations

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding IA-64 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_m64_extract</td>
<td>Extract</td>
<td>extr.u</td>
</tr>
<tr>
<td>_m64_xma</td>
<td>Multiply and add</td>
<td>xma.l</td>
</tr>
<tr>
<td>_m64_xmau</td>
<td>Multiply and add</td>
<td>xma.lu</td>
</tr>
<tr>
<td>_m64_xmah</td>
<td>Multiply and add</td>
<td>xma.h</td>
</tr>
<tr>
<td>_m64_xmahu</td>
<td>Multiply and add</td>
<td>xma.hu</td>
</tr>
<tr>
<td>_m64_popcnt</td>
<td>Population Count</td>
<td>popcnt</td>
</tr>
<tr>
<td>_m64_shladd</td>
<td>Shift left and add</td>
<td>shladd</td>
</tr>
<tr>
<td>_m64_shrp</td>
<td>Shift right pair</td>
<td>shrp</td>
</tr>
</tbody>
</table>

**void _fsetc(int amask, int omask)**

Sets the control bits of FPSR.sf0. Maps to the fsetc.sf0 r, r instruction. There is no corresponding instruction to read the control bits. Use _mm_getfpsr().

**void _fclrf(void)**

Clears the floating point status flags (the 6-bit flags of FPSR.sf0). Maps to the fclrf.sf0 instruction.

**__int64 _m64_dep_mr(__int64 r, __int64 s, const int pos, const int len)**

The right-justified 64-bit value r is deposited into the value in s at an arbitrary bit position and the result is returned. The deposited bit field begins at bit position pos and extends to the left (toward the most significant bit) the number of bits specified by len.

**__int64 _m64_dep_mi(const int v, __int64 s, const int p, const int len)**

1438
The sign-extended value \( v \) (either all 1s or all 0s) is deposited into the value in \( s \) at an arbitrary bit position and the result is returned. The deposited bit field begins at bit position \( p \) and extends to the left (toward the most significant bit) the number of bits specified by \( \text{len} \).

\[
\_\text{int64} \_\text{m64\_dep\_zr}(\_\text{int64} \ s, \ \text{const int} \ \text{pos}, \ \text{const int} \ \text{len})
\]

The right-justified 64-bit value \( s \) is deposited into a 64-bit field of all zeros at an arbitrary bit position and the result is returned. The deposited bit field begins at bit position \( \text{pos} \) and extends to the left (toward the most significant bit) the number of bits specified by \( \text{len} \).

\[
\_\text{int64} \_\text{m64\_dep\_zi}(\text{const int} \ v, \ \text{const int} \ \text{pos}, \ \text{const int} \ \text{len})
\]

The sign-extended value \( v \) (either all 1s or all 0s) is deposited into a 64-bit field of all zeros at an arbitrary bit position and the result is returned. The deposited bit field begins at bit position \( \text{pos} \) and extends to the left (toward the most significant bit) the number of bits specified by \( \text{len} \).

\[
\_\text{int64} \_\text{m64\_extr}(\_\text{int64} \ r, \ \text{const int} \ \text{pos}, \ \text{const int} \ \text{len})
\]

A field is extracted from the 64-bit value \( r \) and is returned right-justified and sign extended. The extracted field begins at position \( \text{pos} \) and extends \( \text{len} \) bits to the left. The sign is taken from the most significant bit of the extracted field.

\[
\_\text{int64} \_\text{m64\_extru}(\_\text{int64} \ r, \ \text{const int} \ \text{pos}, \ \text{const int} \ \text{len})
\]

A field is extracted from the 64-bit value \( r \) and is returned right-justified and zero extended. The extracted field begins at position \( \text{pos} \) and extends \( \text{len} \) bits to the left.

\[
\_\text{int64} \_\text{m64\_xmal}(\_\text{int64} \ a, \ _\text{int64} \ b, \ _\text{int64} \ c)
\]

The 64-bit values \( a \) and \( b \) are treated as signed integers and multiplied to produce a full 128-bit signed result. The 64-bit value \( c \) is zero-extended and added to the product. The least significant 64 bits of the sum are then returned.

\[
\_\text{int64} \_\text{m64\_xmalu}(\_\text{int64} \ a, \ _\text{int64} \ b, \ _\text{int64} \ c)
\]

The 64-bit values \( a \) and \( b \) are treated as signed integers and multiplied to produce a full 128-bit unsigned result. The 64-bit value \( c \) is zero-extended and added to the product. The least significant 64 bits of the sum are then returned.

\[
\_\text{int64} \_\text{m64\_xmah}(\_\text{int64} \ a, \ _\text{int64} \ b, \ _\text{int64} \ c)
\]

The 64-bit values \( a \) and \( b \) are treated as signed integers and multiplied to produce a full 128-bit signed result. The 64-bit value \( c \) is zero-extended and added to the product. The most significant 64 bits of the sum are then returned.

\[
\_\text{int64} \_\text{m64\_xmahu}(\_\text{int64} \ a, \ _\text{int64} \ b, \ _\text{int64} \ c)
\]
The 64-bit values \(a\) and \(b\) are treated as unsigned integers and multiplied to produce a full 128-bit unsigned result. The 64-bit value \(c\) is zero-extended and added to the product. The most significant 64 bits of the sum are then returned.

\[\_\text{int64} \_\text{m64\_popcnt}(\_\text{int64} \ a)\]

The number of bits in the 64-bit integer \(a\) that have the value 1 are counted, and the resulting sum is returned.

\[\_\text{int64} \_\text{m64\_shladd}(\_\text{int64} \ a, \ \text{const int} \ \text{count}, \ \_\text{int64} \ b)\]

\(a\) is shifted to the left by \(\text{count}\) bits and then added to \(b\). The result is returned.

\[\_\text{int64} \_\text{m64\_shrp}(\_\text{int64} \ a, \ \_\text{int64} \ b, \ \text{const int} \ \text{count})\]

\(a\) and \(b\) are concatenated to form a 128-bit value and shifted to the right \(\text{count}\) bits. The least significant 64 bits of the result are returned.

### Lock and Atomic Operation Related Intrinsics

The prototypes for these intrinsics are in the `ia64intrin.h` header files.

<table>
<thead>
<tr>
<th>Intrinsic Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unsigned \_\text{int64} \_\text{InterlockedExchange8}(\text{volatile unsigned char *Target, unsigned \_\text{int64} value})</code></td>
<td>Map to the <code>xchg1</code> instruction. Atomically write the least significant byte of its 2nd argument to address specified by its 1st argument.</td>
</tr>
<tr>
<td><code>unsigned \_\text{int64} \_\text{InterlockedCompareExchange8\_rel}(\text{volatile unsigned char *Destination, unsigned \_\text{int64 Exchange, unsigned \_\text{int64 Comparand})</code></td>
<td>Compare and exchange atomically the least significant byte at the address specified by its 1st argument. Maps to the <code>cmpxchg1\_rel</code> instruction with appropriate setup.</td>
</tr>
<tr>
<td><code>unsigned \_\text{int64} \_\text{InterlockedCompareExchange8\_acq}(\text{volatile unsigned char *Destination, unsigned \_\text{int64 Exchange, unsigned \_\text{int64 Comparand})</code></td>
<td>Same as the previous intrinsic, but using acquire semantic.</td>
</tr>
<tr>
<td><code>unsigned \_\text{int64} \_\text{InterlockedExchange16}(\text{volatile unsigned short *Target, unsigned \_\text{int64 value})</code></td>
<td>Map to the <code>xchg2</code> instruction. Atomically write the least significant word of its 2nd argument to address specified by its 1st argument.</td>
</tr>
<tr>
<td><strong>Intrinsic Prototype</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange16_rel(volatile unsigned short *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Compare and exchange atomically the least significant word at the address specified by its 1st argument. Maps to the cmpxchg2.rel instruction with appropriate setup.</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange16_acq(volatile unsigned short *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Same as the previous intrinsic, but using acquire semantic.</td>
</tr>
<tr>
<td>int _InterlockedIncrement(volatile int *addend)</td>
<td>Atomically increment by one the value specified by its argument. Maps to the fetchadd4 instruction.</td>
</tr>
<tr>
<td>int _InterlockedDecrement(volatile int *addend)</td>
<td>Atomically decrement by one the value specified by its argument. Maps to the fetchadd4 instruction.</td>
</tr>
<tr>
<td>int _InterlockedExchange(volatile int *Target, long value)</td>
<td>Do an exchange operation atomically. Maps to the xchg4 instruction.</td>
</tr>
<tr>
<td>int _InterlockedCompareExchange(volatile int *Destination, int Exchange, int Comparand)</td>
<td>Do a compare and exchange operation atomically. Maps to the cmpxchg4 instruction with appropriate setup.</td>
</tr>
<tr>
<td>int _InterlockedExchangeAdd(volatile int *addend, int increment)</td>
<td>Use compare and exchange to do an atomic add of the increment value to the addend. Maps to a loop with the cmpxchg4 instruction to guarantee atomicity.</td>
</tr>
<tr>
<td>int _InterlockedAdd(volatile int *addend, int increment)</td>
<td>Same as the previous intrinsic, but returns new value, not the original one.</td>
</tr>
<tr>
<td><strong>Intrinsic Prototype</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>void * _InterlockedCompareExchange-Pointer(void * volatile *Destination, void *Exchange, void *Comparand)</td>
<td>Map the <code>exch8</code> instruction; Atomically compare and exchange the pointer value specified by its first argument (all arguments are pointers)</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedExchangeU(volatile unsigned int *Target, unsigned __int64 value)</td>
<td>Atomically exchange the 32-bit quantity specified by the 1st argument. Maps to the <code>xchg4</code> instruction.</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange_rel(volatile unsigned int *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Maps to the <code>cmpxchg4.rel</code> instruction with appropriate setup. Atomically compare and exchange the value specified by the first argument (a 64-bit pointer).</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange_acq(volatile unsigned int *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Same as the previous intrinsic, but map the <code>cmpxchg4.acq</code> instruction.</td>
</tr>
<tr>
<td>void _ReleaseSpinLock(volatile int *x)</td>
<td>Release spin lock.</td>
</tr>
<tr>
<td>__int64 _InterlockedIncrement64(volatile __int64 *addend)</td>
<td>Increment by one the value specified by its argument. Maps to the <code>fetchadd</code> instruction.</td>
</tr>
<tr>
<td>__int64 _InterlockedDecrement64(volatile __int64 *addend)</td>
<td>Decrement by one the value specified by its argument. Maps to the <code>fetchadd</code> instruction.</td>
</tr>
<tr>
<td>__int64 _InterlockedExchange64(volatile __int64 *Target, __int64 value)</td>
<td>Do an exchange operation atomically. Maps to the <code>xchg</code> instruction.</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedExchangeU64(volatile unsigned __int64 *Target, unsigned __int64 value)</td>
<td>Same as <code>InterlockedExchange64</code> (for unsigned quantities).</td>
</tr>
<tr>
<td>Intrinsic Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange64_rel(volatile unsigned __int64 *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Maps to the <code>cmpxchg.rel</code> instruction with appropriate setup. Atomically compare and exchange the value specified by the first argument (a 64-bit pointer).</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange64_acq(volatile unsigned __int64 *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Maps to the <code>cmpxchg.acq</code> instruction with appropriate setup. Atomically compare and exchange the value specified by the first argument (a 64-bit pointer).</td>
</tr>
<tr>
<td>__int64 _InterlockedCompareExchange64(volatile __int64 *Destination, __int64 Exchange, __int64 Comparand)</td>
<td>Same as the previous intrinsic for signed quantities.</td>
</tr>
<tr>
<td>__int64 _InterlockedExchangeAdd64(volatile __int64 *addend, __int64 increment)</td>
<td>Use compare and exchange to do an atomic add of the increment value to the addend. Maps to a loop with the <code>cmpxchg</code> instruction to guarantee atomicity.</td>
</tr>
<tr>
<td>__int64 _InterlockedAdd64(volatile __int64 *addend, __int64 increment);</td>
<td>Same as the previous intrinsic, but returns the new value, not the original value. See Note.</td>
</tr>
</tbody>
</table>

**NOTE.** `_InterlockedSub64` is provided as a macro definition based on `_InterlockedAdd64`.

```c
#define _InterlockedSub64(target, incr) _InterlockedAdd64((target),(-incr))
```

Uses `cmpxchg` to do an atomic sub of the `incr` value to the `target`. Maps to a loop with the `cmpxchg` instruction to guarantee atomicity.
Load and Store Intrinsics

You can use the load and store intrinsic to force strict memory access ordering of specific data objects. This intended use is for cases when a user suppresses strict memory access ordering by using the `-serialize-volatile-` option.

<table>
<thead>
<tr>
<th>Intrinsic Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void __st1_rel(void *dst, const char value);</td>
<td>Generates an st1.rel instruction.</td>
</tr>
<tr>
<td>void __st2_rel(void *dst, const short value);</td>
<td>Generates an st2.rel instruction.</td>
</tr>
<tr>
<td>void __st4_rel(void *dst, const int value);</td>
<td>Generates an st4.rel instruction.</td>
</tr>
<tr>
<td>void __st8_rel(void *dst, const __int64 value);</td>
<td>Generates an st8.rel instruction.</td>
</tr>
<tr>
<td>unsigned char __ld1_acq(void *src);</td>
<td>Generates an ld1.acq instruction.</td>
</tr>
<tr>
<td>unsigned short __ld2_acq(void *src);</td>
<td>Generates an ld2.acq instruction.</td>
</tr>
<tr>
<td>unsigned int __ld4_acq(void *src);</td>
<td>Generates an ld4.acq instruction.</td>
</tr>
<tr>
<td>unsigned __int64 __ld8_acq(void *src);</td>
<td>Generates an ld8.acq instruction.</td>
</tr>
</tbody>
</table>

Operating System Related Intrinsics

The prototypes for these operating system related intrinsics are in the `ia64intrin.h` header file.
<table>
<thead>
<tr>
<th>Intrinsic Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned __int64 __getReg(const int whichReg)</td>
<td>Gets the value from a hardware register based on the index passed in. Produces a corresponding <code>mov = r</code> instruction. Provides access to the following registers: See Register Names for <code>getReg()</code> and <code>setReg()</code>.</td>
</tr>
<tr>
<td>void __setReg(const int whichReg, unsigned __int64 value)</td>
<td>Sets the value for a hardware register based on the index passed in. Produces a corresponding <code>mov = r</code> instruction. See Register Names for <code>getReg()</code> and <code>setReg()</code>.</td>
</tr>
<tr>
<td>unsigned __int64 __getIndReg(const int whichIndReg, __int64 index)</td>
<td>Return the value of an indexed register. The index is the 2nd argument; the register file is the first argument.</td>
</tr>
<tr>
<td>void __setIndReg(const int whichIndReg, __int64 index, unsigned __int64 value)</td>
<td>Copy a value in an indexed register. The index is the 2nd argument; the register file is the first argument.</td>
</tr>
<tr>
<td>void *__ptr64 _rdteb(void)</td>
<td>Gets TEB address. The TEB address is kept in r13 and maps to the <code>move r=tp</code> instruction</td>
</tr>
<tr>
<td>void __isrlz(void)</td>
<td>Executes the serialize instruction. Maps to the <code>srlz.i</code> instruction.</td>
</tr>
<tr>
<td>void __dsrlz(void)</td>
<td>Serializes the data. Maps to the <code>srlz.d</code> instruction.</td>
</tr>
<tr>
<td>unsigned __int64 __fetchadd4_acq(unsigned int *addend, const int increment)</td>
<td>Map the <code>fetchadd4.acq</code> instruction.</td>
</tr>
<tr>
<td>unsigned __int64 __fetchadd4_rel(unsigned int *addend, const int increment)</td>
<td>Map the <code>fetchadd4.rel</code> instruction.</td>
</tr>
<tr>
<td>Intrinsic Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>unsigned __int64 __fetchadd8_acq(unsigned __int64 *addend, const int increment)</td>
<td>Map the <code>fetchadd8.acq</code> instruction.</td>
</tr>
<tr>
<td>unsigned __int64 __fetchadd8_rel(unsigned __int64 *addend, const int increment)</td>
<td>Map the <code>fetchadd8.rel</code> instruction.</td>
</tr>
<tr>
<td>void __fwb(void)</td>
<td>Flushes the write buffers. Maps to the <code>fwb</code> instruction.</td>
</tr>
<tr>
<td>void __ldfs(const int whichFloatReg, void *src)</td>
<td>Map the <code>ldfs</code> instruction. Load a single precision value to the specified register.</td>
</tr>
<tr>
<td>void __ldfd(const int whichFloatReg, void *src)</td>
<td>Map the <code>ldfd</code> instruction. Load a double precision value to the specified register.</td>
</tr>
<tr>
<td>void __ldfe(const int whichFloatReg, void *src)</td>
<td>Map the <code>ldfe</code> instruction. Load an extended precision value to the specified register.</td>
</tr>
<tr>
<td>void __ldf8(const int whichFloatReg, void *src)</td>
<td>Map the <code>ldf8</code> instruction.</td>
</tr>
<tr>
<td>void __ldf_fill(const int whichFloatReg, void *src)</td>
<td>Map the <code>ldf.fill</code> instruction.</td>
</tr>
<tr>
<td>void __stfs(void *dst, const int whichFloatReg)</td>
<td>Map the <code>stfs</code> instruction.</td>
</tr>
<tr>
<td>void __stfd(void *dst, const int whichFloatReg)</td>
<td>Map the <code>stfd</code> instruction.</td>
</tr>
<tr>
<td>void __stfe(void *dst, const int whichFloatReg)</td>
<td>Map the <code>stfe</code> instruction.</td>
</tr>
<tr>
<td>void __stf8(void *dst, const int whichFloatReg)</td>
<td>Map the <code>stf8</code> instruction.</td>
</tr>
<tr>
<td>Intrinsic Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>void __stf_spill(void *dst, const int whichFloatReg)</code></td>
<td>Map the <code>stf.spill</code> instruction.</td>
</tr>
<tr>
<td><code>void __mf(void)</code></td>
<td>Executes a memory fence instruction. Maps to the <code>mf</code> instruction.</td>
</tr>
<tr>
<td><code>void __mfa(void)</code></td>
<td>Executes a memory fence, acceptance form instruction. Maps to the <code>mf.a</code> instruction.</td>
</tr>
<tr>
<td><code>void __synci(void)</code></td>
<td>Enables memory synchronization. Maps to the <code>sync.i</code> instruction.</td>
</tr>
<tr>
<td><code>unsigned __int64 __thash(__int64)</code></td>
<td>Generates a translation hash entry address. Maps to the <code>thash</code> instruction.</td>
</tr>
<tr>
<td><code>unsigned __int64 __ttag(__int64)</code></td>
<td>Generates a translation hash entry tag. Maps to the <code>ttag</code> instruction.</td>
</tr>
<tr>
<td><code>void __itcd(__int64 pa)</code></td>
<td>Insert an entry into the data translation cache (Map <code>itc.d</code> instruction).</td>
</tr>
<tr>
<td><code>void __itci(__int64 pa)</code></td>
<td>Insert an entry into the instruction translation cache (Map <code>itc.i</code>).</td>
</tr>
<tr>
<td><code>void __itr(__int64 whichTransReg, __int64 pa)</code></td>
<td>Map the <code>itr.d</code> instruction.</td>
</tr>
<tr>
<td><code>void __itri(__int64 whichTransReg, __int64 pa)</code></td>
<td>Map the <code>itr.i</code> instruction.</td>
</tr>
<tr>
<td><code>void __ptce(__int64 va)</code></td>
<td>Map the <code>ptc.e</code> instruction.</td>
</tr>
<tr>
<td><code>void __ptcl(__int64 va, __int64 pagesz)</code></td>
<td>Purges the local translation cache. Maps to the <code>ptc.l</code> instruction.</td>
</tr>
<tr>
<td><code>void __ptcg(__int64 va, __int64 pagesz)</code></td>
<td>Purges the global translation cache. Maps to the <code>ptc.g</code> instruction.</td>
</tr>
<tr>
<td>Intrinsic Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>void __ptcga(__int64 va, __int64 pagesz)</td>
<td>Purges the global translation cache and ALAT. Maps to the ptc.ga r, r instruction.</td>
</tr>
<tr>
<td>void __ptri(__int64 va, __int64 pagesz)</td>
<td>Purges the translation register. Maps to the ptr.i r, r instruction.</td>
</tr>
<tr>
<td>void __ptrd(__int64 va, __int64 pagesz)</td>
<td>Purges the translation register. Maps to the ptr.d r, r instruction.</td>
</tr>
<tr>
<td>__int64 __tpa(__int64 va)</td>
<td>Map the tpa instruction.</td>
</tr>
<tr>
<td>void __invalat(void)</td>
<td>Invalidates ALAT. Maps to the invala instruction.</td>
</tr>
<tr>
<td>void __invala (void)</td>
<td>Same as void __invalat(void)</td>
</tr>
<tr>
<td>void __invala_gr(const int whichGeneralReg)</td>
<td>whichGeneralReg = 0-127</td>
</tr>
<tr>
<td>void __invala_fr(const int whichFloatReg)</td>
<td>whichFloatReg = 0-127</td>
</tr>
<tr>
<td>void __break(const int)</td>
<td>Generates a break instruction with an immediate.</td>
</tr>
<tr>
<td>void __nop(const int)</td>
<td>Generate a nop instruction.</td>
</tr>
<tr>
<td>void __debugbreak(void)</td>
<td>Generates a Debug Break Instruction fault.</td>
</tr>
<tr>
<td>void __fc(void*)</td>
<td>Flushes a cache line associated with the address given by the argument. Maps to the fc instruction.</td>
</tr>
<tr>
<td>void __sum(int mask)</td>
<td>Sets the user mask bits of PSR. Maps to the sum imm24 instruction.</td>
</tr>
<tr>
<td>void __rum(int mask)</td>
<td>Resets the user mask.</td>
</tr>
<tr>
<td>__int64 _ReturnAddress(void)</td>
<td>Get the caller's address.</td>
</tr>
<tr>
<td>Intrinsic Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>void __lfetch(int lfhint, void const *y)</td>
<td>Generate the <code>lfetch.lfhint</code> instruction. The value of the first argument specifies the hint type.</td>
</tr>
<tr>
<td>void __lfetch_fault(int lfhint, void const *y)</td>
<td>Generate the <code>lfetch.fault.lfhint</code> instruction. The value of the first argument specifies the hint type.</td>
</tr>
<tr>
<td>void __lfetch_excl(int lfhint, void const *y)</td>
<td>Generate the <code>lfetch.excl.lfhint</code> instruction. The value {0</td>
</tr>
<tr>
<td>void __lfetch_fault_excl(int lfhint, void const *y)</td>
<td>Generate the <code>lfetch.fault.excl.lfhint</code> instruction. The value of the first argument specifies the hint type.</td>
</tr>
<tr>
<td>unsigned int __cacheSize(unsigned int cacheLevel)</td>
<td>See <code>__cacheSize()</code> intrinsic under Intrinsic for Use Across All IA&gt;Miscellaneous Intrinsics.</td>
</tr>
<tr>
<td>void __memory_barrier(void)</td>
<td>Creates a barrier across which the compiler will not schedule any data access instruction. The compiler may allocate local data in registers across a memory barrier, but not global data.</td>
</tr>
<tr>
<td>void __ssm(int mask)</td>
<td>Sets the system mask. Maps to the <code>ssm imm24</code> instruction.</td>
</tr>
<tr>
<td>void __rsm(int mask)</td>
<td>Resets the system mask bits of PSR. Maps to the <code>rsm imm24</code> instruction.</td>
</tr>
</tbody>
</table>

**Conversion Intrinsics**

The prototypes for these conversion intrinsics are in the `ia64intrin.h` header file.
### Intrinsic Prototype

<table>
<thead>
<tr>
<th>Intrinsic Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__int64 _m_to_int64(__m64 a)</td>
<td>Convert a of type __m64 to type __int64. Translates to <code>nop</code> since both types reside in the same register for systems based on IA-64 architecture.</td>
</tr>
<tr>
<td>__m64 _m_from_int64(__int64 a)</td>
<td>Convert a of type __int64 to type __m64. Translates to <code>nop</code> since both types reside in the same register for systems based on IA-64 architecture.</td>
</tr>
<tr>
<td>__int64 __round_double_to_int64(double d)</td>
<td>Convert its double precision argument to a signed integer.</td>
</tr>
<tr>
<td>unsigned __int64 __getf_exp(double d)</td>
<td>Map the <code>getf.exp</code> instruction and return the 16-bit exponent and the sign of its operand.</td>
</tr>
</tbody>
</table>

## Multimedia Addition Intrinsics

The prototypes for the multimedia addition intrinsics are in the `ia64intrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding IA-64 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_m64_czx1l</td>
<td>Compute Zero Index</td>
<td>czx1.l</td>
</tr>
<tr>
<td>_m64_czx1r</td>
<td>Compute Zero Index</td>
<td>czx1.r</td>
</tr>
<tr>
<td>_m64_czx2l</td>
<td>Compute Zero Index</td>
<td>czx2.l</td>
</tr>
<tr>
<td>_m64_czx2r</td>
<td>Compute Zero Index</td>
<td>czx2.r</td>
</tr>
<tr>
<td>_m64_mix1l</td>
<td>Mix</td>
<td>mix1.l</td>
</tr>
<tr>
<td>_m64_mix1r</td>
<td>Mix</td>
<td>mix1.r</td>
</tr>
<tr>
<td>_m64_mix2l</td>
<td>Mix</td>
<td>mix2.l</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding IA-64 Instruction</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>_m64_mix2r</td>
<td>Mix</td>
<td>mix2.r</td>
</tr>
<tr>
<td>_m64_mix4l</td>
<td>Mix</td>
<td>mix4.1</td>
</tr>
<tr>
<td>_m64_mix4r</td>
<td>Mix</td>
<td>mix4.r</td>
</tr>
<tr>
<td>_m64_mux1</td>
<td>Permutation</td>
<td>mux1</td>
</tr>
<tr>
<td>_m64_mux2</td>
<td>Permutation</td>
<td>mux2</td>
</tr>
<tr>
<td>_m64_padd1uus</td>
<td>Parallel add</td>
<td>padd1.uus</td>
</tr>
<tr>
<td>_m64_padd2uus</td>
<td>Parallel add</td>
<td>padd2.uus</td>
</tr>
<tr>
<td>_m64_pavg1_nraz</td>
<td>Parallel average</td>
<td>pavg1</td>
</tr>
<tr>
<td>_m64_pavg2_nraz</td>
<td>Parallel average</td>
<td>pavg2</td>
</tr>
<tr>
<td>_m64_pavgsub1</td>
<td>Parallel average subtract</td>
<td>pavgsub1</td>
</tr>
<tr>
<td>_m64_pavgsub2</td>
<td>Parallel average subtract</td>
<td>pavgsub2</td>
</tr>
<tr>
<td>_m64_pmpy2r</td>
<td>Parallel multiply</td>
<td>pmpy2.r</td>
</tr>
<tr>
<td>_m64_pmpy2l</td>
<td>Parallel multiply</td>
<td>pmpy2.l</td>
</tr>
<tr>
<td>_m64_pmpyshr2</td>
<td>Parallel multiply and shift right</td>
<td>pmpyshr2</td>
</tr>
<tr>
<td>_m64_pmpyshr2u</td>
<td>Parallel multiply and shift right</td>
<td>pmpyshr2.u</td>
</tr>
<tr>
<td>_m64_pshladd2</td>
<td>Parallel shift left and add</td>
<td>pshladd2</td>
</tr>
<tr>
<td>_m64_pshradd2</td>
<td>Parallel shift right and add</td>
<td>pshradd2</td>
</tr>
<tr>
<td>_m64_psub1uus</td>
<td>Parallel subtract</td>
<td>psub1.uus</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding IA-64 Instruction</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>_m64_psub2uus</td>
<td>Parallel subtract</td>
<td>psub2.uus</td>
</tr>
</tbody>
</table>

__int64 _m64_czx1l(__m64 a)  

The 64-bit value `a` is scanned for a zero element from the most significant element to the least significant element, and the index of the first zero element is returned. The element width is 8 bits, so the range of the result is from 0 - 7. If no zero element is found, the default result is 8.

__int64 _m64_czx1r(__m64 a)  

The 64-bit value `a` is scanned for a zero element from the least significant element to the most significant element, and the index of the first zero element is returned. The element width is 8 bits, so the range of the result is from 0 - 7. If no zero element is found, the default result is 8.

__int64 _m64_czx2l(__m64 a)  

The 64-bit value `a` is scanned for a zero element from the most significant element to the least significant element, and the index of the first zero element is returned. The element width is 16 bits, so the range of the result is from 0 - 3. If no zero element is found, the default result is 4.

__int64 _m64_czx2r(__m64 a)  

The 64-bit value `a` is scanned for a zero element from the least significant element to the most significant element, and the index of the first zero element is returned. The element width is 16 bits, so the range of the result is from 0 - 3. If no zero element is found, the default result is 4.

__m64 _m64_mix1l(__m64 a, __m64 b)  

Interleave 64-bit quantities `a` and `b` in 1-byte groups, starting from the left, as shown in Figure 1, and return the result.
Interleave 64-bit quantities \( a \) and \( b \) in 1-byte groups, starting from the right, as shown in Figure 2, and return the result.

Interleave 64-bit quantities \( a \) and \( b \) in 2-byte groups, starting from the left, as shown in Figure 3, and return the result.

Interleave 64-bit quantities \( a \) and \( b \) in 2-byte groups, starting from the right, as shown in Figure 4, and return the result.
__m64 __m64_mix4l(__m64 a, __m64 b)
Interleave 64-bit quantities $a$ and $b$ in 4-byte groups, starting from the left, as shown in Figure 5, and return the result.

__m64 __m64_mix4r(__m64 a, __m64 b)
Interleave 64-bit quantities $a$ and $b$ in 4-byte groups, starting from the right, as shown in Figure 6, and return the result.

__m64 __m64_mux1(__m64 a, const int n)
Based on the value of $n$, a permutation is performed on $a$ as shown in Figure 7, and the result is returned. Table 1 shows the possible values of $n$. 

1454
Table 427: Table 1. Values of n for m64_mux1 Operation

<table>
<thead>
<tr>
<th></th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>@br cst</td>
<td>0</td>
</tr>
<tr>
<td>@mix</td>
<td>8</td>
</tr>
</tbody>
</table>
Based on the value of \( n \), a permutation is performed on \( a \) as shown in Figure 8, and the result is returned.

\[
\_\_m64 \_m64\text{mux2}(_m64 a, \text{const int} \ n)
\]

\[
\_\_m64 \_m64\text{pavgsub1}(_m64 a, _m64 b)
\]
The unsigned data elements (bytes) of \( b \) are subtracted from the unsigned data elements (bytes) of \( a \) and the results of the subtraction are then each independently shifted to the right by one position. The high-order bits of each element are filled with the borrow bits of the subtraction.

\[
\_\text{m64} \_\text{m64}_\text{pavgsub2}(\_\text{m64} \ a, \_\text{m64} \ b)
\]

The unsigned data elements (double bytes) of \( b \) are subtracted from the unsigned data elements (double bytes) of \( a \) and the results of the subtraction are then each independently shifted to the right by one position. The high-order bits of each element are filled with the borrow bits of the subtraction.

\[
\_\text{m64} \_\text{m64}_\text{pmpy2l}(\_\text{m64} \ a, \_\text{m64} \ b)
\]

Two signed 16-bit data elements of \( a \), starting with the most significant data element, are multiplied by the corresponding two signed 16-bit data elements of \( b \), and the two 32-bit results are returned as shown in Figure 9.

\[
\_\text{m64} \_\text{m64}_\text{pmpy2r}(\_\text{m64} \ a, \_\text{m64} \ b)
\]

Two signed 16-bit data elements of \( a \), starting with the least significant data element, are multiplied by the corresponding two signed 16-bit data elements of \( b \), and the two 32-bit results are returned as shown in Figure 10.
__m64 _m64_pmpyshr2(__m64 a, __m64 b, const int count)
The four signed 16-bit data elements of a are multiplied by the corresponding signed 16-bit data elements of b, yielding four 32-bit products. Each product is then shifted to the right count bits and the least significant 16 bits of each shifted product form 4 16-bit results, which are returned as one 64-bit word.

__m64 _m64_pmpyshr2u(__m64 a, __m64 b, const int count)
The four unsigned 16-bit data elements of a are multiplied by the corresponding unsigned 16-bit data elements of b, yielding four 32-bit products. Each product is then shifted to the right count bits and the least significant 16 bits of each shifted product form 4 16-bit results, which are returned as one 64-bit word.

__m64 _m64_pshladd2(__m64 a, const int count, __m64 b)
a is shifted to the left by count bits and then is added to b. The upper 32 bits of the result are forced to 0, and then bits [31:30] of b are copied to bits [62:61] of the result. The result is returned.

__m64 _m64_pshradd2(__m64 a, const int count, __m64 b)
The four signed 16-bit data elements of a are each independently shifted to the right by count bits (the high order bits of each element are filled with the initial value of the sign bits of the data elements in a); they are then added to the four signed 16-bit data elements of b. The result is returned.
_m64 _m64_padd1uus(_m64 a, _m64 b)
a is added to b as eight separate byte-wide elements. The elements of a are treated as unsigned, while the elements of b are treated as signed. The results are treated as unsigned and are returned as one 64-bit word.

_m64 _m64_padd2uus(_m64 a, _m64 b)
a is added to b as four separate 16-bit wide elements. The elements of a are treated as unsigned, while the elements of b are treated as signed. The results are treated as unsigned and are returned as one 64-bit word.

_m64 _m64_psub1uus(_m64 a, _m64 b)
a is subtracted from b as eight separate byte-wide elements. The elements of a are treated as unsigned, while the elements of b are treated as signed. The results are treated as unsigned and are returned as one 64-bit word.

_m64 _m64_psub2uus(_m64 a, _m64 b)
a is subtracted from b as four separate 16-bit wide elements. The elements of a are treated as unsigned, while the elements of b are treated as signed. The results are treated as unsigned and are returned as one 64-bit word.

_m64 _m64_pavg1_nraz(_m64 a, _m64 b)
The unsigned byte-wide data elements of a are added to the unsigned byte-wide data elements of b and the results of each add are then independently shifted to the right by one position. The high-order bits of each element are filled with the carry bits of the sums.

_m64 _m64_pavg2_nraz(_m64 a, _m64 b)
The unsigned 16-bit wide data elements of a are added to the unsigned 16-bit wide data elements of b and the results of each add are then independently shifted to the right by one position. The high-order bits of each element are filled with the carry bits of the sums.

Miscellaneous Intrinsics

void* __get_return_address(unsigned int level);

This intrinsic yields the return address of the current function. The level argument must be a constant value. A value of 0 yields the return address of the current function. Any other value yields a zero return address. On Linux systems, this intrinsic is synonymous with __builtin_return_address. The name and the argument are provided for compatibility with GCC*.
void __set_return_address(void* addr);
This intrinsic overwrites the default return address of the current function with the address indicated by its argument. On return from the current invocation, program execution continues at the address provided.

void* __get_frame_address(unsigned int level);
This intrinsic returns the frame address of the current function. The level argument must be a constant value. A value of 0 yields the frame address of the current function. Any other value yields a zero return value. On Linux systems, this intrinsic is synonymous with __builtin_frame_address. The name and the argument are provided for compatibility with GCC*.

Register Names for getReg() and setReg()
The register names for getReg() and setReg() functions are in the ia64regs.h header file.

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_IP</td>
<td>1016</td>
</tr>
<tr>
<td>_IA64_REG_PSR</td>
<td>1019</td>
</tr>
<tr>
<td>_IA64_REG_PSR_L</td>
<td>1019</td>
</tr>
</tbody>
</table>

Table 429: General Integer Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_GP</td>
<td>1025</td>
</tr>
<tr>
<td>_IA64_REG_SP</td>
<td>1036</td>
</tr>
<tr>
<td>_IA64_REG_TP</td>
<td>1037</td>
</tr>
</tbody>
</table>

Table 430: Application Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_AR_KR0</td>
<td>3072</td>
</tr>
<tr>
<td>Name</td>
<td>whichReg</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR1</td>
<td>3073</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR2</td>
<td>3074</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR3</td>
<td>3075</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR4</td>
<td>3076</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR5</td>
<td>3077</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR6</td>
<td>3078</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR7</td>
<td>3079</td>
</tr>
<tr>
<td>_IA64_REG_AR_RSC</td>
<td>3088</td>
</tr>
<tr>
<td>_IA64_REG_AR_BSP</td>
<td>3089</td>
</tr>
<tr>
<td>_IA64_REG_AR_BSPSTORE</td>
<td>3090</td>
</tr>
<tr>
<td>_IA64_REG_AR_RNAT</td>
<td>3091</td>
</tr>
<tr>
<td>_IA64_REG_AR_FCR</td>
<td>3093</td>
</tr>
<tr>
<td>_IA64_REG_AR_EFLAG</td>
<td>3096</td>
</tr>
<tr>
<td>_IA64_REG_AR_CSD</td>
<td>3097</td>
</tr>
<tr>
<td>_IA64_REG_AR_SSD</td>
<td>3098</td>
</tr>
<tr>
<td>_IA64_REG_AR_CFLAG</td>
<td>3099</td>
</tr>
<tr>
<td>_IA64_REG_AR_FSR</td>
<td>3100</td>
</tr>
<tr>
<td>_IA64_REG_AR_FIR</td>
<td>3101</td>
</tr>
<tr>
<td>_IA64_REG_AR_FDR</td>
<td>3102</td>
</tr>
<tr>
<td>_IA64_REG_AR_CCV</td>
<td>3104</td>
</tr>
</tbody>
</table>
### Table 431: Control Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_AR_UNAT</td>
<td>3108</td>
</tr>
<tr>
<td>_IA64_REG_AR_FPSR</td>
<td>3112</td>
</tr>
<tr>
<td>_IA64_REG_AR_ITC</td>
<td>3116</td>
</tr>
<tr>
<td>_IA64_REG_AR_PFS</td>
<td>3136</td>
</tr>
<tr>
<td>_IA64_REG_AR_LC</td>
<td>3137</td>
</tr>
<tr>
<td>_IA64_REG_AR_EC</td>
<td>3138</td>
</tr>
<tr>
<td>_IA64_REG_CR_DCR</td>
<td>4096</td>
</tr>
<tr>
<td>_IA64_REG_CR_ITM</td>
<td>4097</td>
</tr>
<tr>
<td>_IA64_REG_CR_IVA</td>
<td>4098</td>
</tr>
<tr>
<td>_IA64_REG_CR_PTA</td>
<td>4104</td>
</tr>
<tr>
<td>_IA64_REG_CR_IPSR</td>
<td>4112</td>
</tr>
<tr>
<td>_IA64_REG_CR_ISR</td>
<td>4113</td>
</tr>
<tr>
<td>_IA64_REG_CR_IIP</td>
<td>4115</td>
</tr>
<tr>
<td>_IA64_REG_CR_IFA</td>
<td>4116</td>
</tr>
<tr>
<td>_IA64_REG_CR_ITIR</td>
<td>4117</td>
</tr>
<tr>
<td>_IA64_REG_CR_IIPA</td>
<td>4118</td>
</tr>
<tr>
<td>_IA64_REG_CR_IFS</td>
<td>4119</td>
</tr>
<tr>
<td>_IA64_REG_CR_IIM</td>
<td>4120</td>
</tr>
<tr>
<td>Name</td>
<td>whichReg</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
</tr>
<tr>
<td>_IA64_REG_CR_IHA</td>
<td>4121</td>
</tr>
<tr>
<td>_IA64_REG_CR_LID</td>
<td>4160</td>
</tr>
<tr>
<td>_IA64_REG_CR_IVR</td>
<td>4161</td>
</tr>
<tr>
<td>_IA64_REG_CR_TPR</td>
<td>4162</td>
</tr>
<tr>
<td>_IA64_REG_CR_EOI</td>
<td>4163</td>
</tr>
<tr>
<td>_IA64_REG_CR_IRR0</td>
<td>4164</td>
</tr>
<tr>
<td>_IA64_REG_CR_IRR1</td>
<td>4165</td>
</tr>
<tr>
<td>_IA64_REG_CR_IRR2</td>
<td>4166</td>
</tr>
<tr>
<td>_IA64_REG_CR_IRR3</td>
<td>4167</td>
</tr>
<tr>
<td>_IA64_REG_CR_ITV</td>
<td>4168</td>
</tr>
<tr>
<td>_IA64_REG_CR_PMV</td>
<td>4169</td>
</tr>
<tr>
<td>_IA64_REG_CR_CM CV</td>
<td>4170</td>
</tr>
<tr>
<td>_IA64_REG_CR_LRR0</td>
<td>4176</td>
</tr>
<tr>
<td>_IA64_REG_CR_LRR1</td>
<td>4177</td>
</tr>
</tbody>
</table>

**Table 432: Indirect Registers for getIndReg() and setIndReg()**

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_INDR_CPUID</td>
<td>9000</td>
</tr>
<tr>
<td>_IA64_REG_INDR_DBR</td>
<td>9001</td>
</tr>
<tr>
<td>_IA64_REG_INDR_IBR</td>
<td>9002</td>
</tr>
<tr>
<td>_IA64_REG_INDR_PKR</td>
<td>9003</td>
</tr>
</tbody>
</table>
Intrinsics for Dual-Core Intel® Itanium® 2 Processor 9000 Sequence

The Dual-Core Intel® Itanium® 2 processor 9000 sequence supports the intrinsics listed in the table below.

These intrinsics each generate IA-64 instructions. The first alpha-numerical chain in the intrinsic name represents the return type, and the second alpha-numerical chain in the intrinsic name represents the instruction the intrinsic generates. For example, the intrinsic _int64_cmp8xchg generates the _int64 return type and the cmp8xchg IA-64 instruction.

Examples of several of these intrinsics are provided at the end of this topic.

For more information about the instructions these intrinsics generate, please see the documentation area of the Itanium® processor website at http://developer.intel.com/products/processor/itanium/index.htm.

**NOTE.** Calling these intrinsics on any previous Itanium® processor causes an illegal instruction fault.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>__cmp8xchg16</td>
<td>Compare and exchange</td>
</tr>
<tr>
<td>__ld16</td>
<td>Load</td>
</tr>
<tr>
<td>__fc_i</td>
<td>Flush cache</td>
</tr>
<tr>
<td>__hint</td>
<td>Provide performance hints</td>
</tr>
</tbody>
</table>
Generates the 16-byte form of the compare and exchange IA-64 instruction.

Returns the original 64-bit value read from memory at the specified address.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>sem</th>
<th>ldhint</th>
<th>addr</th>
<th>xchg_lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literal value between 0 and 1 that specifies the semaphore completer (0==.acq, 1==.rel)</td>
<td>Literal value between 0 and 2 that specifies the load hint completer (0==.none, 1==.nt1, 2==.nta)</td>
<td>The address of the value to read.</td>
<td>The least significant 8 bytes of the exchange value.</td>
</tr>
</tbody>
</table>

The following table describes each implicit argument for this intrinsic.

<table>
<thead>
<tr>
<th>xchg_hi</th>
<th>cmpnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest 8 bytes of the exchange value. Use the <code>__setReg</code> intrinsic to set the <code>&lt;xchg_hi&gt;</code> value in the register AR[CSD]. <code>__setReg (_IA64_REG_AR_CSD, &lt;xchg_hi&gt;);</code></td>
<td>The 64-bit compare value. Use the <code>__setReg</code> intrinsic to set the <code>&lt;cmpnd&gt;</code> value in the register AR[CCV]. <code>__setReg (_IA64_REG_AR_CCV, &lt;cmpnd&gt;);</code></td>
</tr>
</tbody>
</table>
Example:

```c
__int64 foo_cmp8xchg16(__int64 xchg_lo, __int64 xchg_hi, __int64 cmpnd, void* addr)
{
    __int64 old_value;
/**/
// set the highest bits of the exchange value and the comperand value
// respectively in CSD and CCV. Then, call the exchange intrinsic
//
__setReg(_IA64_REG_AR_CSD, xchg_hi);
__setReg(_IA64_REG_AR_CCV, cmpnd);
old_value = __cmp8xchg16(__semtype_acq, __ldhint_none, addr, xchg_lo);
/**/
    return old_value;
}
```

__int64 __ld16(const int <ldtype>, const int <ldhint>, void *<addr>)
Generates the IA-64 instruction that loads 16 bytes from the given address.

Returns the lower 8 bytes of the quantity loaded from <addr>. The higher 8 bytes are loaded in register AR[CSD].

Generates implicit return of the higher 8 bytes to the register AR[CSD]. You can use the __getReg intrinsic to copy the value into a user variable. [foo = __getReg(_IA64_REG_AR_CSD);]

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>ldtype</th>
<th>ldhint</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>A literal value between 0 and 1 that specifies the load type (0==none, 1==.acq).</td>
<td>A literal value between 0 and 2 that specifies the hint completer (0==none, 1==.nt1, 2==.nta).</td>
<td>The address to load from.</td>
</tr>
</tbody>
</table>
Example:

```c
void foo_ld16(__int64* lo, __int64* hi, void* addr)
{
    /**/
    // The following two calls load the 16-byte value at the given address
    // into two (2) 64-bit integers
    // The higher 8 bytes are returned implicitly in the CSD register;
    // The call to __getReg moves that value into a user variable (hi).
    // The instruction generated is a plain ld16
    // ld16 Ra,ar.csd=[Rb]
    *lo = __ld16(__ldtype_none, __ldhint_none, addr);
    *hi = __getReg(_IA64_REG_AR_CSD);
    /**/
}

void __fc_i(void **addr)

Generates the IA-64 instruction that flushes the cache line associated with the specified address
and ensures coherency between instruction cache and data cache.

The following table describes the argument for this intrinsic.

<table>
<thead>
<tr>
<th>cache_line</th>
</tr>
</thead>
<tbody>
<tr>
<td>An address associated with the cache line you want to flush</td>
</tr>
</tbody>
</table>

void __hint(const int <hint_value>)

Generates the IA-64 instruction that provides performance hints about the program being executed.

The following table describes the argument for this intrinsic.

<table>
<thead>
<tr>
<th>hint_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A literal value that specifies the hint. Currently, zero is the only legal value. __hint(0) generates the IA-64 hint@pause instruction.</td>
</tr>
</tbody>
</table>
void __st16(const int <sttype>, const int <sthint>, void *<addr>, __int64 <src_lo>)

Generates the IA-64 instruction to store 16 bytes at the given address.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>sttype</th>
<th>sthint</th>
<th>addr</th>
<th>src_lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>A literal value</td>
<td>A literal value</td>
<td>The address where</td>
<td>The lowest 8 bytes of</td>
</tr>
<tr>
<td>between 0 and 1</td>
<td>between 0 and 1</td>
<td>the 16-byte value is</td>
<td>the 16-byte value to</td>
</tr>
<tr>
<td>that specifies</td>
<td>that specifies</td>
<td>stored.</td>
<td>store.</td>
</tr>
<tr>
<td>the store</td>
<td>the store</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type completer</td>
<td>hint completer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0==.none,</td>
<td>(0==.none,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1==.rel).</td>
<td>1==.nta).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following table describes the implicit argument for this intrinsic.

<table>
<thead>
<tr>
<th>src_hi</th>
</tr>
</thead>
<tbody>
<tr>
<td>The highest 8 bytes of the 16-byte value to store. Use the setReg intrinsic to set the &lt;src_hi&gt; value in the register AR[CSD]. [__setReg(_IA64_REG_AR_CSD, &lt;src_hi&gt;); ]</td>
</tr>
</tbody>
</table>

Example:

```c
void foo_st16(__int64 lo, __int64 hi, void* addr)
{
    /**/
    // first set the highest 64-bits into CSD register. Then call
    // __st16 with the lowest 64-bits as argument
    //
    __setReg(_IA64_REG_AR_CSD, hi);
    __st16(__sttype_none, __sthint_none, addr, lo);
    /**/
}
```
Example of Using Intrinsics Together

The following examples show how to use some of the intrinsics presented above together to generate the corresponding instructions. In all cases, use the __setReg (resp. __getReg) intrinsic to set up implicit arguments (resp. = retrieve implicit return values).

// file foo.c

#include <ia64intrin.h>

void foo_ld16(__int64* lo, __int64* hi, void* addr)
{
    /**/
    // The following two calls load the 16-byte value at the given address
    // into two (2) 64-bit integers
    // The higher 8 bytes are returned implicitly in the CSD register;
    // The call to __getReg moves that value into a user variable (hi).
    // The instruction generated is a plain ld16
    // ld16 Ra,ar.csd=[Rb]
    *lo = __ld16(__ldtype_none, __ldhint_none, addr);
    *hi = __getReg(_IA64_REG_AR_CSD);
    /**/
}

void foo_ld16_acq(__int64* lo, __int64* hi, void* addr)
{
    /**/
    // This is the same as the previous example, except that it uses the
    // __ldtype_acq completer to generate the acquire_from of the ld16:
    // ld16.acq Ra,ar.csd=[Rb]
    //
    *lo = __ld16(__ldtype_acq, __ldhint_none, addr);
    *hi = __getReg(_IA64_REG_AR_CSD);
void foo_st16(__int64 lo, __int64 hi, void* addr)
{
    __setReg(_IA64_REG_AR_CSD, hi);
    __st16(__sttype_none, __sthint_none, addr, lo);
}

__int64 foo_cmp8xchg16(__int64 xchg_lo, __int64 xchg_hi, __int64 cmpnd, void* addr)
{
    __int64 old_value;
    __setReg(_IA64_REG_AR_CSD, xchg_hi);
    __setReg(_IA64_REG_AR_CCV, cmpnd);
    old_value = __cmp8xchg16(__semtype_acq, __ldhint_none, addr, xchg_lo);
    return old_value;
}

// end foo.c
Microsoft*-Compatible Intrinsics for Dual-Core Intel® Itanium® Processor 9000 Sequence

The Dual-Core Intel® Itanium® processor 9000 sequence supports the intrinsics listed in the table below. These intrinsics are also compatible with the Microsoft* compiler. These intrinsics each generate IA-64 instructions. The second alpha-numerical chain in the intrinsic name represents the IA-64 instruction the intrinsic generates. For example, the intrinsic _int64_cmp8xchg generates the cmp8xchg IA-64 instruction.

For more information about the instructions these intrinsics generate, please see the documentation area of the Itanium® processor website.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding IA-64 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__InterlockedCompare64Exchange128</td>
<td>Compare and exchange</td>
<td></td>
</tr>
<tr>
<td>__InterlockedCompare64Exchange128_acq</td>
<td>Compare and Exchange</td>
<td></td>
</tr>
<tr>
<td>__InterlockedCompare64Exchange128_rel</td>
<td>Compare and Exchange</td>
<td></td>
</tr>
<tr>
<td>__load128</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>__load128_acq</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>__store128</td>
<td>Store</td>
<td></td>
</tr>
<tr>
<td>__store128_rel</td>
<td>Store</td>
<td></td>
</tr>
</tbody>
</table>

__int64 __InterlockedCompare64Exchange128(__int64 volatile * <Destination>, __int64 <ExchangeHigh>, __int64 <ExchangeLow>, __int64 <Comparer>)

Generates a compare and exchange IA-64 instruction. Returns the lowest 64-bit value of the destination.

The following table describes each argument for this intrinsic.
__int64 __InterlockedCompare64Exchange128_acq( __int64 volatile *<Destination>, __int64 <ExchangeHigh>, __int64 <ExchangeLow>, __int64 <Comperand>)

Generates a compare and exchange IA-64 instruction. Same as __InterlockedCompare64Exchange128, but this intrinsic uses acquire semantics. Returns the lowest 64-bit value of the destination.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>Destination</th>
<th>ExchangeHigh</th>
<th>ExchangeLow</th>
<th>Comperand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Destination value</td>
<td>Highest 64 bits of the Exchange value</td>
<td>Lowest 64 bits of the Exchange value</td>
<td>Value to compare with Destination</td>
</tr>
</tbody>
</table>

__int64 __InterlockedCompare64Exchange128_rel( __int64 volatile *<Destination>, __int64 <ExchangeHigh>, __int64 <ExchangeLow>, __int64 <Comperand>)

Generates a compare and exchange IA-64 instruction. Same as __InterlockedCompare64Exchange128, but this intrinsic uses release semantics. Returns the lowest 64-bit value of the destination.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>Destination</th>
<th>ExchangeHigh</th>
<th>ExchangeLow</th>
<th>Comperand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Destination value</td>
<td>Highest 64 bits of the Exchange value</td>
<td>Lowest 64 bits of the Exchange value</td>
<td>Value to compare with Destination</td>
</tr>
</tbody>
</table>

__int64 __load128( __int64 volatile *Source, __int64 *<DestinationHigh>)

Generates the IA-64 instruction that atomically reads 128 bits from the memory location. Returns the lowest 64-bit value of the 128-bit loaded value.

The following table describes each argument for this intrinsic.
__int64 __load128_acq( __int64 volatile * <Source>, __int64 *<DestinationHigh>
Generates the IA-64 instruction that atomically reads 128 bits from the memory location. Same as __load128, but the this intrinsic uses acquire semantics. Returns the lowest 64-bit value of the 128-bit loaded value.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>Source</th>
<th>DestinationHigh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Source value</td>
<td>Pointer to the location in memory that stores the highest 64 bits of the 128-bit loaded value</td>
</tr>
</tbody>
</table>

__void __store128( __int64 volatile * <Destination>, __int64 <SourceHigh> __int64 <SourceLow>)
Generates the IA-64 instruction that atomically stores 128 bits at the destination memory location. No returns.

<table>
<thead>
<tr>
<th>Destination</th>
<th>SourceHigh</th>
<th>SourceLow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Destination value</td>
<td>The highest 64 bits of the value to be stored</td>
<td>The lowest 64 bits of the value to be stored</td>
</tr>
</tbody>
</table>

__void __store128_rel( __int64 volatile * <Destination>, __int64 <SourceHigh> __int64 <SourceLow>)
Generates the IA-64 instruction that atomically stores 128 bits at the destination memory location. Same as __store128, but this intrinsic uses release semantics. No returns.
Data Alignment, Memory Allocation Intrinsics, and Inline Assembly

Overview: Data Alignment, Memory Allocation Intrinsics, and Inline Assembly

This section describes features that support usage of the intrinsics. The following topics are described:

- Alignment Support
- Allocating and Freeing Aligned Memory Blocks
- Inline Assembly

Alignment Support

Aligning data improves the performance of intrinsics. When using the Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics, you should align data to 16 bytes in memory operations. Specifically, you must align __m128 objects as addresses passed to the _mm_load and _mm_store intrinsics. If you want to declare arrays of floats and treat them as __m128 objects by casting, you need to ensure that the float arrays are properly aligned.

Use __declspec(align) to direct the compiler to align data more strictly than it otherwise would. For example, a data object of type int is allocated at a byte address which is a multiple of 4 by default. However, by using __declspec(align), you can direct the compiler to instead use an address which is a multiple of 8, 16, or 32 with the following restriction on IA-32 architecture:

- 16-byte addresses can be locally or statically allocated

You can use this data alignment support as an advantage in optimizing cache line usage. By clustering small objects that are commonly used together into a struct, and forcing the struct to be allocated at the beginning of a cache line, you can effectively guarantee that each object is loaded into the cache as soon as any one is accessed, resulting in a significant performance benefit.

The syntax of this extended-attribute is as follows:

align(n)

where n is an integral power of 2, up to 4096. The value specified is the requested alignment.
CAUTION. In this release, __declspec(align(8)) does not function correctly. Use
__declspec(align(16)) instead.

NOTE. If a value is specified that is less than the alignment of the affected data type,
it has no effect. In other words, data is aligned to the maximum of its own alignment or
the alignment specified with __declspec(align).

You can request alignments for individual variables, whether of static or automatic storage
duration. (Global and static variables have static storage duration; local variables have automatic
storage duration by default.) You cannot adjust the alignment of a parameter, nor a field of a
struct or class. You can, however, increase the alignment of a struct (or union or class),
in which case every object of that type is affected.

As an example, suppose that a function uses local variables i and j as subscripts into a
2-dimensional array. They might be declared as follows:
int i, j;

These variables are commonly used together. But they can fall in different cache lines, which
could be detrimental to performance. You can instead declare them as follows:
__declspec(align(16)) struct { int i, j; } sub;

The compiler now ensures that they are allocated in the same cache line. In C++, you can omit
the struct variable name (written as sub in the previous example). In C, however, it is required,
and you must write references to i and j as sub.i and sub.j.

If you use many functions with such subscript pairs, it is more convenient to declare and use
a struct type for them, as in the following example: typedef struct
__declspec(align(16)) { int i, j; } Sub;

By placing the __declspec(align) after the keyword struct, you are requesting the
appropriate alignment for all objects of that type. Note that allocation of parameters is unaffected
by __declspec(align). (If necessary, you can assign the value of a parameter to a local
variable with the appropriate alignment.)

You can also force alignment of global variables, such as arrays:
__declspec(align(16)) float array[1000];
Allocating and Freeing Aligned Memory Blocks

To allocate and free aligned blocks of memory use the _mm_malloc and _mm_free intrinsics. These intrinsics are based on malloc and free, which are in the libirc.a library. You need to include malloc.h. The syntax for these intrinsics is as follows:

```c
void* _mm_malloc (int size, int align)
void _mm_free (void *p)
```

The _mm_malloc routine takes an extra parameter, which is the alignment constraint. This constraint must be a power of two. The pointer that is returned from _mm_malloc is guaranteed to be aligned on the specified boundary.

NOTE. Memory that is allocated using _mm_malloc must be freed using _mm_free. Calling free on memory allocated with _mm_malloc or calling _mm_free on memory allocated with malloc will cause unpredictable behavior.

Inline Assembly

Microsoft Style Inline Assembly

The Intel® C++ Compiler supports Microsoft-style inline assembly with the -use-msasm compiler option. See your Microsoft documentation for the proper syntax.

GNU*-like Style Inline Assembly (IA-32 architecture and Intel(R) 64 architecture only)

The Intel® C++ Compiler supports GNU-like style inline assembly. The syntax is as follows:

```c
asm-keyword [ volatile-keyword ] ( asm-template [ asm-interface ] )
```

The Intel C++ Compiler also supports mixing UNIX and Microsoft style asms. Use the __asm__ keyword for GNU-style ASM when using the -use_msasm switch.

NOTE. The Intel C++ Compiler supports gcc-style inline ASM if the assembler code uses AT&T* System V/386 syntax.
<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asm-keyword</td>
<td>Assembly statements begin with the keyword <code>asm</code>. Alternatively, either <code>__asm</code> or <code>__asm__</code> may be used for compatibility. When mixing UNIX and Microsoft style <code>asm</code>, use the <code>__asm__</code> keyword. The compiler only accepts the <code>__asm__</code> keyword. The <code>asm</code> and <code>__asm</code> keywords are reserved for Microsoft style assembly statements.</td>
</tr>
<tr>
<td>volatile-keyword</td>
<td>If the optional keyword <code>volatile</code> is given, the <code>asm</code> is volatile. Two volatile <code>asm</code> statements are never moved past each other, and a reference to a volatile variable is not moved relative to a volatile <code>asm</code>. Alternate keywords <code>__volatile</code> and <code>__volatile__</code> may be used for compatibility.</td>
</tr>
<tr>
<td>asm-template</td>
<td>The <code>asm-template</code> is a C language ASCII string that specifies how to output the assembly code for an instruction. Most of the template is a fixed string; everything but the substitution-directives, if any, is passed through to the assembler. The syntax for a substitution directive is a <code>%</code> followed by one or two characters.</td>
</tr>
<tr>
<td>asm-interface</td>
<td>The <code>asm-interface</code> consists of three parts:</td>
</tr>
<tr>
<td></td>
<td>1. an optional output-list</td>
</tr>
<tr>
<td></td>
<td>2. an optional input-list</td>
</tr>
<tr>
<td></td>
<td>3. an optional clobber-list</td>
</tr>
<tr>
<td></td>
<td>These are separated by colon (:) characters. If the <code>output-list</code> is missing, but an <code>input-list</code> is given, the input list may be preceded by two colons (::) to take the place of the missing <code>output-list</code>. If the <code>asm-interface</code> is omitted altogether, the <code>asm</code> statement is considered volatile regardless of whether a <code>volatile-keyword</code> was specified.</td>
</tr>
<tr>
<td>output-list</td>
<td>An <code>output-list</code> consists of one or more <code>output-specs</code> separated by commas. For the purposes of substitution in the <code>asm-template</code>, each <code>output-spec</code> is numbered. The first operand in the <code>output-list</code> is numbered 0, the second is 1, and so on. Numbering is continuous through the <code>output-list</code> and into the <code>input-list</code>. The total number of operands is limited to 30 (i.e. 0-29).</td>
</tr>
</tbody>
</table>
Similar to an output-list, an input-list consists of one or more input-specs separated by commas. For the purposes of substitution in the asm-template, each input-spec is numbered, with the numbers continuing from those in the output-list.

A clobber-list tells the compiler that the asm uses or changes a specific machine register that is either coded directly into the asm or is changed implicitly by the assembly instruction. The clobber-list is a comma-separated list of clobber-specs. The input-specs tell the compiler about expressions whose values may be needed by the inserted assembly instruction. In order to describe fully the input requirements of the asm, you can list input-specs that are not actually referenced in the asm-template. Each clobber-spec specifies the name of a single machine register that is clobbered. The register name may optionally be preceded by a %. You can specify any valid machine register name. It is also legal to specify "memory" in a clobber-spec. This prevents the compiler from keeping data cached in registers across the asm statement.

When compiling an assembly statement on Linux*, the compiler simply emits the asm-template to the assembly file after making any necessary operand substitutions. The compiler then calls the GNU assembler to generate machine code. In contrast, on Windows* the compiler itself must assemble the text contained in the asm-template string into machine code. In essence, the compiler contains a built-in assembler.

The compiler's built-in assembler supports the GNU .byte directive but does not support other functionality of the GNU assembler, so there are limitations in the contents of the asm-template. The following assembler features are not currently supported.

- Directives other than the .byte directive
- Symbols*

**NOTE.** * Direct symbol references in the asm-template are not supported. To access a C++ object, use the asm-interface with a substitution directive.

**Example**

Incorrect method for accessing a C++ object:
Proper method for accessing a C++ object:

```
__asm__("addl $5, %0" : "+rm" (x));
```

Additionally, there are some restrictions on the usage of labels. The compiler only allows local
labels, and only references to labels within the same assembly statement are permitted. A local
label has the form 

```
label N:
```

where N is a non-negative integer. N does not have to be unique, even
within the same assembly statement. To reference the most recent definition of label N, use

```
"Nb"
```

To reference the next definition of label N, use 

```
"Nf"
```

In this context, “b” means backward and “f” means forward. For more information, refer to the GNU assembler documentation.

GNU-style inline assembly statements on Windows* use the same assembly instruction format
as on Linux*. This means that destination operands are on the right and source operands are
on the left. This operand order is the reverse of Intel assembly syntax.

Due to the limitations of the compiler's built-in assembler, many assembly statements that
compile and run on Linux* will not compile on Windows*. On the other hand, assembly
statements that compile and run on Windows* should also compile and run on Linux*.
This feature provides a high-performance alternative to Microsoft-style inline assembly statements when portability between Windows*, Linux*, and Mac OS* X is important. Its intended use is in small primitives where high-performance integration with the surrounding C++ code is essential.

```c
#ifdef __WIN64
#define INT64_PRINTF_FORMAT "I64"
#else
#define __int64 long long
#define INT64_PRINTF_FORMAT "L"
#endif
#include <stdio.h>
typedef struct {
  __int64 lo64;
  __int64 hi64;
} my_i128;
#define ADD128(out, in1, in2) \
  __asm__ (."addq %2, %0; adcq %3, %1" : \
    "=r"(out.lo64), "=r"(out.hi64) : \
    "emr" (in2.lo64), "emr"(in2.hi64), \
    "0" (in1.lo64), "1" (in1.hi64));

extern int
main()
{
  my_i128 val1, val2, result;
  val1.lo64 = ~0;
  val1.hi64 = 0;
  val2.hi64 = 65;
  ADD128(result, val1, val2);
  printf("0x%016" INT64_PRINTF_FORMAT "x%016" INT64_PRINTF_FORMAT "x\n",
```
This example, written for Intel(R) 64 architecture, shows how to use a GNU-style inline assembly statement to add two 128-bit integers. In this example, a 128-bit integer is represented as two __int64 objects in the my_i128 structure. The inline assembly statement used to implement the addition is contained in the ADD128 macro, which takes 3 my_i128 arguments representing 3 128-bit integers. The first argument is the output. The next two arguments are the inputs. The example compiles and runs using the Intel Compiler on Linux* or Windows*, producing the following output.

```
0x0000000000000000ffffffffffffffff
+ 0x00000000000000410000000000000001
------------------------------------
+ 0x00000000000000420000000000000000
```

In the GNU-style inline assembly implementation, the asm interface specifies all the inputs, outputs, and side effects of the asm statement, enabling the compiler to generate very efficient code.

```
mov          r13, 0xffffffffffffffff
mov     r12, 0x00000000
add     r13, 1
adc     r12, 65
```

It is worth noting that when the compiler generates an assembly file on Windows*, it uses Intel syntax even though the assembly statement was written using Linux* assembly syntax.
The compiler moves \texttt{in1.lo64} into a register to match the constraint of operand 4.Operand 4's constraint of "0" indicates that it must be assigned the same location as output operand 0. And operand 0's constraint is "=r", indicating that it must be assigned an integer register. In this case, the compiler chooses \texttt{r13}. In the same way, the compiler moves \texttt{in1.hi64} into register \texttt{r12}.

The constraints for input operands 2 and 3 allow the operands to be assigned a register location ("r"), a memory location ("m"), or a constant signed 32-bit integer value ("e"). In this case, the compiler chooses to match operands 2 and 3 with the constant values 1 and 65, enabling the \texttt{add} and \texttt{adc} instructions to utilize the "register-immediate" forms.

The same operation is much more expensive using a Microsoft-style inline assembly statement, because the interface between the assembly statement and the surrounding C++ code is entirely through memory. Using Microsoft assembly, the \texttt{ADD128} macro might be written as follows.

\begin{verbatim}
#define ADD128(out, in1, in2) 
{
   __asm mov rax, in1.lo64
   __asm mov rdx, in1.hi64
   __asm add rax, in2.lo64
   __asm adc rdx, in2.hi64
   __asm mov out.lo64, rax
   __asm mov out.hi64, rdx
}
\end{verbatim}
The compiler must add code before the assembly statement to move the inputs into memory, and it must add code after the assembly statement to retrieve the outputs from memory. This prevents the compiler from exploiting some optimization opportunities. Thus, the following assembly code is produced.

```assembly
mov  QWORD PTR [rsp+32], -1
mov  QWORD PTR [rsp+40], 0
mov  QWORD PTR [rsp+48], 1
mov  QWORD PTR [rsp+56], 65

; Begin ASM
mov  rax, QWORD PTR [rsp+32]
mov  rdx, QWORD PTR [rsp+40]
add  rax, QWORD PTR [rsp+48]
adc  rdx, QWORD PTR [rsp+56]
mov  QWORD PTR [rsp+64], rax
mov  QWORD PTR [rsp+72], rdx

; End ASM
mov  rdx, QWORD PTR [rsp+72]
mov  r8, QWORD PTR [rsp+64]
```

The operation that took only 4 instructions and 0 memory references using GNU-style inline assembly takes 12 instructions with 12 memory references using Microsoft-style inline assembly.
Overview: MMX™ Technology Intrinsics

MMX™ technology is an extension to the Intel architecture (IA) instruction set. The MMX instruction set adds 57 opcodes and a 64-bit quadword data type, and eight 64-bit registers. Each of the eight registers can be directly addressed using the register names \texttt{mm0} to \texttt{mm7}.

The prototypes for MMX technology intrinsics are in the \texttt{mmintrin.h} header file.

Details about MMX(TM) Technology Intrinsics

The MMX™ technology instructions use the following features:

- Registers--Enable packed data of up to 128 bits in length for optimal single-instruction multiple data (SIMD) processing
- Data Types--Enable packing of up to 16 elements of data in one register

Registers

Intel® processors provide special register sets. The MMX instructions use eight 64-bit registers (\texttt{mm0} to \texttt{mm7}) which are aliased on the floating-point stack registers.

Because each of these registers can hold more than one data element, the processor can process more than one data element simultaneously. This processing capability is also known as single-instruction multiple data (SIMD) processing.

For each computational and data manipulation instruction in the new extension sets, there is a corresponding C intrinsic that implements that instruction directly. This frees you from managing registers and assembly programming. Further, the compiler optimizes the instruction scheduling so that your executable runs faster.

\textbf{NOTE.} The \texttt{MM} and \texttt{XMM} registers are the SIMD registers used by the IA-32 architecture-based platforms to implement MMX™ technology and Intel® Streaming SIMD Extensions (Intel® SSE) or Intel® SSE2 intrinsics. On the IA-64 architecture, the MMX and Intel® SSE intrinsics use the 64-bit general registers and the 64-bit significand of the 80-bit floating-point register.
Data Types

Intrinsic functions use four new C data types as operands, representing the new registers that are used as the operands to these intrinsic functions.

__m64 Data Type

The __m64 data type is used to represent the contents of an MMX register, which is the register that is used by the MMX technology intrinsics. The __m64 data type can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

Data Types Usage Guidelines

These data types are not basic ANSI C data types. You must observe the following usage restrictions:

- Use data types only on either side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (+, -, etc).
- Use data types as objects in aggregates, such as unions, to access the byte elements and structures.
- Use data types only with the respective intrinsics described in this documentation.

The EMMS Instruction: Why You Need It

Using EMMS is like emptying a container to accommodate new content. The EMMS instruction clears the MMX™ registers and sets the value of the floating-point tag word to empty.

You should clear the MMX registers before issuing a floating-point instruction because floating-point convention specifies that the floating-point stack be cleared after use. Insert the EMMS instruction at the end of all MMX code segments to avoid a floating-point overflow exception.

Why You Need EMMS to Reset After an MMX™ Instruction
CAUTION. Failure to empty the multimedia state after using an MMX instruction and before using a floating-point instruction can result in unexpected execution or poor performance.

**EMMS Usage Guidelines**

Here are guidelines for when to use the EMMS instruction:
Use `_mm_empty()` after an MMX™ instruction if the next instruction is a floating-point (FP) instruction. For example, you should use the EMMS instruction before performing calculations on `float`, `double` or `long double`. You must be aware of all situations in which your code generates an MMX instruction:

- when using an MMX technology intrinsic
- when using Intel® Streaming SIMD Extensions (Intel® SSE) integer intrinsics that use the `_m64` data type
- when referencing an `_m64` data type variable
- when using an MMX instruction through inline assembly

Use different functions for operations that use floating point instructions and those that use MMX instructions. This action eliminates the need to empty the multimedia state within the body of a critical loop.

Use `_mm_empty()` during runtime initialization of `_m64` and FP data types. This ensures resetting the register between data type transitions.

Do not use `_mm_empty()` before an MMX instruction, since using `_mm_empty()` before an MMX instruction incurs an operation with no benefit (no-op).

Do not use on systems based on IA-64 architecture. There are no special registers (or overlay) for the MMX™ instructions or Intel® SSE on systems based on IA-64 architecture even though the intrinsics are supported.

See the Correct Usage and Incorrect Usage coding examples in the following table.

<table>
<thead>
<tr>
<th>Incorrect Usage</th>
<th>Correct Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_m64 x = _m_padd(y, z);</code></td>
<td><code>_m64 x = _m_padd(y, z);</code></td>
</tr>
<tr>
<td><code>float f = init();</code></td>
<td><code>float f = (_mm_empty(), init());</code></td>
</tr>
</tbody>
</table>

**MMX™ Technology General Support Intrinsics**

The prototypes for MMX™ technology general support intrinsics are in the `mmextrin.h` header file.
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_empty</td>
<td>Empty MM state</td>
<td>EMMS</td>
</tr>
<tr>
<td>_mm_cvtsi32_si64</td>
<td>Convert from int</td>
<td>MOVD</td>
</tr>
<tr>
<td>_mm_cvtsi64_si32</td>
<td>Convert to int</td>
<td>MOVD</td>
</tr>
<tr>
<td>_mm_cvtsi64_m64</td>
<td>Convert from __int64</td>
<td>MOVQ</td>
</tr>
<tr>
<td>_mm_cvtm64_si64</td>
<td>Convert to __int64</td>
<td>MOVQ</td>
</tr>
<tr>
<td>_mm_packs_pi16</td>
<td>Pack</td>
<td>PACKSSWB</td>
</tr>
<tr>
<td>_mm_packs_pi32</td>
<td>Pack</td>
<td>PACKSSDW</td>
</tr>
<tr>
<td>_mm_packs_pu16</td>
<td>Pack</td>
<td>PACKUSWB</td>
</tr>
<tr>
<td>_mm_unpackhi_pi8</td>
<td>Interleave</td>
<td>PUNPCKHBW</td>
</tr>
<tr>
<td>_mm_unpackhi_pi16</td>
<td>Interleave</td>
<td>PUNPCKHWD</td>
</tr>
<tr>
<td>_mm_unpackhi_pi32</td>
<td>Interleave</td>
<td>PUNPCKHDQ</td>
</tr>
<tr>
<td>_mm_unpacklo_pi8</td>
<td>Interleave</td>
<td>PUNPCKLBW</td>
</tr>
<tr>
<td>_mm_unpacklo_pi16</td>
<td>Interleave</td>
<td>PUNPCKLWD</td>
</tr>
<tr>
<td>_mm_unpacklo_pi32</td>
<td>Interleave</td>
<td>PUNPCKLDQ</td>
</tr>
</tbody>
</table>

```c
void _mm_empty(void)

Empties the multimedia state.

__m64 _mm_cvtsi32_si64(int i)

Converts the integer object i to a 64-bit __m64 object. The integer value is zero-extended to 64 bits.

int _mm_cvtsi64_si32(__m64 m)

Converts the lower 32 bits of the __m64 object m to an integer.

__m64 _mm_cvtsi64_m64(__int64 i)
```
Moves the 64-bit integer object i to a __m64 object

__m64 _mm_cvtm64_si64(__m64 m)

Moves the __m64 object m to a 64-bit integer

__m64 _mm_packs_pi16(__m64 m1, __m64 m2)

Packs the four 16-bit values from m1 into the lower four 8-bit values of the result with signed saturation, and pack the four 16-bit values from m2 into the upper four 8-bit values of the result with signed saturation.

__m64 _mm_packs_pi32(__m64 m1, __m64 m2)

Packs the two 32-bit values from m1 into the lower two 16-bit values of the result with signed saturation, and pack the two 32-bit values from m2 into the upper two 16-bit values of the result with signed saturation.

__m64 _mm_packs_pu16(__m64 m1, __m64 m2)

Packs the four 16-bit values from m1 into the lower four 8-bit values of the result with unsigned saturation, and pack the four 16-bit values from m2 into the upper four 8-bit values of the result with unsigned saturation.

__m64 _mm_unpackhi_pi8(__m64 m1, __m64 m2)

Interleaves the four 8-bit values from the high half of m1 with the four values from the high half of m2. The interleaving begins with the data from m1.

__m64 _mm_unpackhi_pi16(__m64 m1, __m64 m2)

Interleaves the two 16-bit values from the high half of m1 with the two values from the high half of m2. The interleaving begins with the data from m1.

__m64 _mm_unpackhi_pi32(__m64 m1, __m64 m2)

Interleaves the 32-bit value from the high half of m1 with the 32-bit value from the high half of m2. The interleaving begins with the data from m1.

__m64 _mm_unpacklo_pi8(__m64 m1, __m64 m2)

Interleaves the four 8-bit values from the low half of m1 with the four values from the low half of m2. The interleaving begins with the data from m1.

__m64 _mm_unpacklo_pi16(__m64 m1, __m64 m2)

Interleaves the two 16-bit values from the low half of m1 with the two values from the low half of m2. The interleaving begins with the data from m1.
```
__m64 _mm_unpacklo_pi32(__m64 m1, __m64 m2)
Interleaves the 32-bit value from the low half of m1 with the 32-bit value from the low half of m2. The interleaving begins with the data from m1.
```

### MMX™ Technology Packed Arithmetic Intrinsics

The prototypes for MMX™ technology packed arithmetic intrinsics are in the `mmintrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_pi8</td>
<td>Addition</td>
<td>PADDB</td>
</tr>
<tr>
<td>_mm_add_pi16</td>
<td>Addition</td>
<td>PADDW</td>
</tr>
<tr>
<td>_mm_add_pi32</td>
<td>Addition</td>
<td>PADDW</td>
</tr>
<tr>
<td>_mm_adds_pi8</td>
<td>Addition</td>
<td>PADDUSB</td>
</tr>
<tr>
<td>_mm_adds_pi16</td>
<td>Addition</td>
<td>PADDUSW</td>
</tr>
<tr>
<td>_mm_adds_pi32</td>
<td>Addition</td>
<td>PADDUSW</td>
</tr>
<tr>
<td>_mm_sub_pi8</td>
<td>Subtraction</td>
<td>PSUBB</td>
</tr>
<tr>
<td>_mm_sub_pi16</td>
<td>Subtraction</td>
<td>PSUBW</td>
</tr>
<tr>
<td>_mm_sub_pi32</td>
<td>Subtraction</td>
<td>PSUBD</td>
</tr>
<tr>
<td>_mm_subs_pi8</td>
<td>Subtraction</td>
<td>PSUBSB</td>
</tr>
<tr>
<td>_mm_subs_pi16</td>
<td>Subtraction</td>
<td>PSUBSW</td>
</tr>
<tr>
<td>_mm_subs_pi32</td>
<td>Subtraction</td>
<td>PSUBUSB</td>
</tr>
<tr>
<td>_mm_subs_pi32</td>
<td>Subtraction</td>
<td>PSUBUSW</td>
</tr>
</tbody>
</table>
### Intrinsic Name
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_madd_pi16</td>
<td>Multiply and add</td>
<td>PMADDWD</td>
</tr>
<tr>
<td>_mm_mulhi_pi16</td>
<td>Multiplication</td>
<td>PMULHW</td>
</tr>
<tr>
<td>_mm_mullo_pi16</td>
<td>Multiplication</td>
<td>PMULLW</td>
</tr>
</tbody>
</table>

__m64 _mm_add_pi8(__m64 m1, __m64 m2)  
Add the eight 8-bit values in m1 to the eight 8-bit values in m2.

__m64 _mm_add_pi16(__m64 m1, __m64 m2)  
Add the four 16-bit values in m1 to the four 16-bit values in m2.

__m64 _mm_add_pi32(__m64 m1, __m64 m2)  
Add the two 32-bit values in m1 to the two 32-bit values in m2.

__m64 _mm_adds_pi8(__m64 m1, __m64 m2)  
Add the eight signed 8-bit values in m1 to the eight signed 8-bit values in m2 using saturating arithmetic.

__m64 _mm_adds_pi16(__m64 m1, __m64 m2)  
Add the four signed 16-bit values in m1 to the four signed 16-bit values in m2 using saturating arithmetic.

__m64 _mm_adds_pu8(__m64 m1, __m64 m2)  
Add the eight unsigned 8-bit values in m1 to the eight unsigned 8-bit values in m2 and using saturating arithmetic.

__m64 _mm_adds_pu16(__m64 m1, __m64 m2)  
Add the four unsigned 16-bit values in m1 to the four unsigned 16-bit values in m2 using saturating arithmetic.

__m64 _mm_sub_pi8(__m64 m1, __m64 m2)  
Subtract the eight 8-bit values in m2 from the eight 8-bit values in m1.

__m64 _mm_sub_pi16(__m64 m1, __m64 m2)  
Subtract the four 16-bit values in m2 from the four 16-bit values in m1.
__m64 _mm_sub_pi32(__m64 m1, __m64 m2)
Subtract the two 32-bit values in m2 from the two 32-bit values in m1.

__m64 _mm_subs_pi8(__m64 m1, __m64 m2)
Subtract the eight signed 8-bit values in m2 from the eight signed 8-bit values in m1 using saturating arithmetic.

__m64 _mm_subs_pi16(__m64 m1, __m64 m2)
Subtract the four signed 16-bit values in m2 from the four signed 16-bit values in m1 using saturating arithmetic.

__m64 _mm_subsPu8(__m64 m1, __m64 m2)
Subtract the eight unsigned 8-bit values in m2 from the eight unsigned 8-bit values in m1 using saturating arithmetic.

__m64 _mm_subsPu16(__m64 m1, __m64 m2)
Subtract the four unsigned 16-bit values in m2 from the four unsigned 16-bit values in m1 using saturating arithmetic.

__m64 _mm_madd_pi16(__m64 m1, __m64 m2)
Multiply four 16-bit values in m1 by four 16-bit values in m2 producing four 32-bit intermediate results, which are then summed by pairs to produce two 32-bit results.

__m64 _mm_mulhi_pi16(__m64 m1, __m64 m2)
Multiply four signed 16-bit values in m1 by four signed 16-bit values in m2 and produce the high 16 bits of the four results.

__m64 _mm_mullo_pi16(__m64 m1, __m64 m2)
Multiply four 16-bit values in m1 by four 16-bit values in m2 and produce the low 16 bits of the four results.

**MMX™ Technology Shift Intrinsics**

The prototypes for MMX™ technology shift intrinsics are in the mmintrin.h header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_sll_pi16</td>
<td>Logical shift left</td>
<td>PSLLW</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding MMX Instruction</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>_mm_slli_pi16</td>
<td>Logical shift left</td>
<td>PSLLWI</td>
</tr>
<tr>
<td>_mm_sll_pi32</td>
<td>Logical shift left</td>
<td>PSLLD</td>
</tr>
<tr>
<td>_mm_slli_pi32</td>
<td>Logical shift left</td>
<td>PSLLDI</td>
</tr>
<tr>
<td>_mm_sll_pi64</td>
<td>Logical shift left</td>
<td>PSLLQ</td>
</tr>
<tr>
<td>_mm_slli_pi64</td>
<td>Logical shift left</td>
<td>PSLLQI</td>
</tr>
<tr>
<td>_mm_sra_pi16</td>
<td>Arithmetic shift right</td>
<td>PSRAW</td>
</tr>
<tr>
<td>_mm_sra_pi16</td>
<td>Arithmetic shift right</td>
<td>PSRAWI</td>
</tr>
<tr>
<td>_mm_sra_pi32</td>
<td>Arithmetic shift right</td>
<td>PSRAD</td>
</tr>
<tr>
<td>_mm_sra_pi32</td>
<td>Arithmetic shift right</td>
<td>PSRADI</td>
</tr>
<tr>
<td>_mm_srl_pi16</td>
<td>Logical shift right</td>
<td>PSRLW</td>
</tr>
<tr>
<td>_mm_srl_pi16</td>
<td>Logical shift right</td>
<td>PSRLWI</td>
</tr>
<tr>
<td>_mm_srl_pi32</td>
<td>Logical shift right</td>
<td>PSRLD</td>
</tr>
<tr>
<td>_mm_srl_pi32</td>
<td>Logical shift right</td>
<td>PSRLDI</td>
</tr>
<tr>
<td>_mm_srl_pi64</td>
<td>Logical shift right</td>
<td>PSRLQ</td>
</tr>
<tr>
<td>_mm_srl_pi64</td>
<td>Logical shift right</td>
<td>PSRLQI</td>
</tr>
</tbody>
</table>

__m64 _mm_sll_pi16(__m64 m, __m64 count)  
Shifts four 16-bit values in m left the amount specified by count while shifting in zeros.

__m64 _mm_slli_pi16(__m64 m, int count)  
Shifts four 16-bit values in m left the amount specified by count while shifting in zeros. For the best performance, count should be a constant.

__m64 _mm_sll_pi32(__m64 m, __m64 count)
Shifts two 32-bit values in \( m \) left the amount specified by \( \text{count} \) while shifting in zeros.

\[
\_m64\ _mm\_slli\_pi32(\_m64\ m,\ \text{int}\ \text{count})
\]

Shifts two 32-bit values in \( m \) left the amount specified by \( \text{count} \) while shifting in zeros. For the best performance, \( \text{count} \) should be a constant.

\[
\_m64\ _mm\_slli\_pi64(\_m64\ m,\ \_m64\ \text{count})
\]

Shifts the 64-bit value in \( m \) left the amount specified by \( \text{count} \) while shifting in zeros.

\[
\_m64\ _mm\_sll\_pi64(\_m64\ m,\ \text{int}\ \text{count})
\]

Shifts the 64-bit value in \( m \) left the amount specified by \( \text{count} \) while shifting in zeros. For the best performance, \( \text{count} \) should be a constant.

\[
\_m64\ _mm\_sra\_pi16(\_m64\ m,\ \_m64\ \text{count})
\]

Shifts four 16-bit values in \( m \) right the amount specified by \( \text{count} \) while shifting in the sign bit.

\[
\_m64\ _mm\_srai\_pi16(\_m64\ m,\ \text{int}\ \text{count})
\]

Shifts four 16-bit values in \( m \) right the amount specified by \( \text{count} \) while shifting in the sign bit. For the best performance, \( \text{count} \) should be a constant.

\[
\_m64\ _mm\_sra\_pi32(\_m64\ m,\ \_m64\ \text{count})
\]

Shifts two 32-bit values in \( m \) right the amount specified by \( \text{count} \) while shifting in the sign bit.

\[
\_m64\ _mm\_sra\_pi32(\_m64\ m,\ \text{int}\ \text{count})
\]

Shifts two 32-bit values in \( m \) right the amount specified by \( \text{count} \) while shifting in the sign bit. For the best performance, \( \text{count} \) should be a constant.

\[
\_m64\ _mm\_sr\_pi16(\_m64\ m,\ \_m64\ \text{count})
\]

Shifts four 16-bit values in \( m \) right the amount specified by \( \text{count} \) while shifting in zeros.

\[
\_m64\ _mm\_sr\_pi16(\_m64\ m,\ \text{int}\ \text{count})
\]

Shifts four 16-bit values in \( m \) right the amount specified by \( \text{count} \) while shifting in zeros. For the best performance, \( \text{count} \) should be a constant.

\[
\_m64\ _mm\_sr\_pi32(\_m64\ m,\ \_m64\ \text{count})
\]

Shifts two 32-bit values in \( m \) right the amount specified by \( \text{count} \) while shifting in zeros.

\[
\_m64\ _mm\_sr\_pi32(\_m64\ m,\ \text{int}\ \text{count})
\]

Shifts two 32-bit values in \( m \) right the amount specified by \( \text{count} \) while shifting in zeros. For the best performance, \( \text{count} \) should be a constant.
__m64 _mm_srl_pi64(__m64 m, __m64 count)
Shifts the 64-bit value in m right the amount specified by count while shifting in zeros.

__m64 _mm_srli_pi64(__m64 m, int count)
Shifts the 64-bit value in m right the amount specified by count while shifting in zeros. For the best performance, count should be a constant.

**MMX™ Technology Logical Intrinsics**

The prototypes for MMX™ technology logical intrinsics are in the mmintrin.h header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_and_si64</td>
<td>Bitwise AND</td>
<td>PAND</td>
</tr>
<tr>
<td>_mm_andnot_si64</td>
<td>Bitwise ANDNOT</td>
<td>PANDN</td>
</tr>
<tr>
<td>_mm_or_si64</td>
<td>Bitwise OR</td>
<td>POR</td>
</tr>
<tr>
<td>_mm_xor_si64</td>
<td>Bitwise Exclusive OR</td>
<td>FXOR</td>
</tr>
</tbody>
</table>

__m64 _mm_and_si64(__m64 m1, __m64 m2)
Perform a bitwise AND of the 64-bit value in m1 with the 64-bit value in m2.

__m64 _mm_andnot_si64(__m64 m1, __m64 m2)
Perform a bitwise NOT on the 64-bit value in m1 and use the result in a bitwise AND with the 64-bit value in m2.

__m64 _mm_or_si64(__m64 m1, __m64 m2)
Perform a bitwise OR of the 64-bit value in m1 with the 64-bit value in m2.

__m64 _mm_xor_si64(__m64 m1, __m64 m2)
Perform a bitwise XOR of the 64-bit value in m1 with the 64-bit value in m2.

**MMX™ Technology Compare Intrinsics**

The prototypes for MMX™ technology compare intrinsics are in the mmintrin.h header file.
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpeq_pi8</td>
<td>Equal</td>
<td>PCMPEQB</td>
</tr>
<tr>
<td>_mm_cmpeq_pi16</td>
<td>Equal</td>
<td>PCMPEQW</td>
</tr>
<tr>
<td>_mm_cmpeq_pi32</td>
<td>Equal</td>
<td>PCMPEQD</td>
</tr>
<tr>
<td>_mm_cmpgt_pi8</td>
<td>Greater Than</td>
<td>PCMPGTB</td>
</tr>
<tr>
<td>_mm_cmpgt_pi16</td>
<td>Greater Than</td>
<td>PCMPGTW</td>
</tr>
<tr>
<td>_mm_cmpgt_pi32</td>
<td>Greater Than</td>
<td>PCMPGTD</td>
</tr>
</tbody>
</table>

__m64 _mm_cmpeq_pi8(__m64 m1, __m64 m2)

Sets the corresponding 8-bit resulting values to all ones if the 8-bit values in m1 are equal to the corresponding 8-bit values in m2; otherwise, sets them to all zeros.

__m64 _mm_cmpeq_pi16(__m64 m1, __m64 m2)

Sets the corresponding 16-bit resulting values to all ones if the 16-bit values in m1 are equal to the corresponding 16-bit values in m2; otherwise, set them to all zeros.

__m64 _mm_cmpeq_pi32(__m64 m1, __m64 m2)

Sets the corresponding 32-bit resulting values to all ones if the 32-bit values in m1 are equal to the corresponding 32-bit values in m2; otherwise, set them to all zeros.

__m64 _mm_cmpgt_pi8(__m64 m1, __m64 m2)

Sets the corresponding 8-bit resulting values to all ones if the 8-bit signed values in m1 are greater than the corresponding 8-bit signed values in m2; otherwise, set them to all zeros.

__m64 _mm_cmpgt_pi16(__m64 m1, __m64 m2)

Sets the corresponding 16-bit resulting values to all ones if the 16-bit signed values in m1 are greater than the corresponding 16-bit signed values in m2; otherwise, set them to all zeros.

__m64 _mm_cmpgt_pi32(__m64 m1, __m64 m2)

Sets the corresponding 32-bit resulting values to all ones, if the 32-bit signed values in m1 are greater than the corresponding 32-bit signed values in m2; otherwise, set them all to zeros.
### MMX™ Technology Set Intrinsics

The prototypes for MMX™ technology intrinsics are in the `mmintrin.h` header file.

**NOTE.** In the descriptions regarding the bits of the MMX register, bit 0 is the least significant and bit 63 is the most significant.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_mm_setzero_si64</code></td>
<td>set to zero</td>
<td>PXOR</td>
</tr>
<tr>
<td><code>_mm_set_pi32</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_set_pi16</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_set_pi8</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_set1_pi32</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_set1_pi16</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_set1_pi8</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_setr_pi32</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_setr_pi16</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_setr_pi8</code></td>
<td>set integer values</td>
<td>Composite</td>
</tr>
</tbody>
</table>

```c
__m64 _mm_setzero_si64()
```

Sets the 64-bit value to zero.
__m64 _mm_set_pi32(int i1, int i0)
Sets the 2 signed 32-bit integer values.

\[
\begin{array}{ll}
  \text{R0} & \text{R1} \\
i0 & i1 \\
\end{array}
\]

__m64 _mm_set_pi16(short s3, short s2, short s1, short s0)
Sets the 4 signed 16-bit integer values.

\[
\begin{array}{llll}
  \text{R0} & \text{R1} & \text{R2} & \text{R3} \\
w0 & w1 & w2 & w3 \\
\end{array}
\]

__m64 _mm_set_pi8(char b7, char b6, char b5, char b4, char b3, char b2, char b1, char b0)
Sets the 8 signed 8-bit integer values.

\[
\begin{array}{llll}
  \text{R0} & \text{R1} & \ldots & \text{R7} \\
b0 & b1 & \ldots & b7 \\
\end{array}
\]

__m64 _mm_set1_pi32(int i)
Sets the 2 signed 32-bit integer values to i.

\[
\begin{array}{ll}
  \text{R0} & \text{R1} \\
i & i \\
\end{array}
\]

__m64 _mm_set1_pi16(short s)
Sets the 4 signed 16-bit integer values to w.

\[
\begin{array}{llll}
  \text{R0} & \text{R1} & \text{R2} & \text{R3} \\
w & w & w & w \\
\end{array}
\]

__m64 _mm_set1_pi8(char b)
Sets the 8 signed 8-bit integer values to b.
__m64 _mm_setr_pi32(int i1, int i0)
Sets the 2 signed 32-bit integer values in reverse order.

__m64 _mm_setr_pi16(short s3, short s2, short s1, short s0)
Sets the 4 signed 16-bit integer values in reverse order.

__m64 _mm_setr_pi8(char b7, char b6, char b5, char b4, char b3, char b2, char b1, char b0)
Sets the 8 signed 8-bit integer values in reverse order.

**MMX™ Technology Intrinsics for IA-64 Architecture**

MMX™ technology intrinsics provide access to the MMX technology instruction set on systems based on IA-64 architecture. To provide source compatibility with the IA-32 architecture, these intrinsics are equivalent both in name and functionality to the set of IA-32 architecture-based MMX intrinsics.

The prototypes for MMX technology intrinsics are in the `mmintrin.h` header file.
Data Types

The C data type __m64 is used when using MMX technology intrinsics. It can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

The __m64 data type is not a basic ANSI C data type. Therefore, observe the following usage restrictions:

- Use the new data type only on the left-hand side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (" + ", " - ", and so on).
- Use the new data type as objects in aggregates, such as unions, to access the byte elements and structures; the address of an __m64 object may be taken.
- Use new data types only with the respective intrinsics described in this documentation.

Overview: Intel(R) Streaming SIMD Extensions

This section describes the C++ language-level features supporting the Intel® Streaming SIMD Extensions (Intel® SSE) in the Intel® C++ Compiler. These topics explain the following features of the intrinsics:

- Floating Point Intrinsics
- Arithmetic Operation Intrinsics
- Logical Operation Intrinsics
- Comparison Intrinsics
- Conversion Intrinsics
- Load Operations
- Set Operations
- Store Operations
- Cacheability Support
- Integer Intrinsics
- Intrinsics to Read and Write Registers
- Miscellaneous Intrinsics
- Using Streaming SIMD Extensions on IA-64 Architecture

The prototypes for SSE intrinsics are in the `xmmintrin.h` header file.

**NOTE.** You can also use the single `ia32intrin.h` header file for any IA-32 architecture-based intrinsics.

Details about Intel(R) Streaming SIMD Extension Intrinsics

The Intel® Streaming SIMD Extension (Intel® SSE) instructions use the following features:
Registers

Intel processors provide special register sets. The Streaming SIMD Extensions use eight 128-bit registers (xmm0 to xmm7).

Because each of these registers can hold more than one data element, the processor can process more than one data element simultaneously. This processing capability is also known as single-instruction multiple data processing (SIMD).

For each computational and data manipulation instruction in the new extension sets, there is a corresponding C intrinsic that implements that instruction directly. This frees you from managing registers and assembly programming. Further, the compiler optimizes the instruction scheduling so that your executable runs faster.

Data Types

Intrinsic functions use four new C data types as operands, representing the new registers that are used as the operands to these intrinsic functions.

New Data Types

The following table details for which instructions each of the new data types are available.

<table>
<thead>
<tr>
<th>New Data Type</th>
<th>Intel® Streaming SIMD Extensions</th>
<th>Intel® Streaming SIMD Extensions 2</th>
<th>Intel® Streaming SIMD Extensions 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m64</td>
<td>Available</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>__m128</td>
<td>Available</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>__m128d</td>
<td>Not available</td>
<td>Available</td>
<td>Available</td>
</tr>
</tbody>
</table>
**New Data Type**

<table>
<thead>
<tr>
<th></th>
<th>Intel® Streaming SIMD Extensions</th>
<th>Intel® Streaming SIMD Extensions 2 Intrinsics</th>
<th>Intel® Streaming SIMD Extensions 3 Intrinsics</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128i</td>
<td>Not available</td>
<td>Available</td>
<td>Available</td>
</tr>
</tbody>
</table>

**__m64 Data Type**

On the IA-64 architecture, the __m64 data type is used to represent the contents of a 64-bit general registers. The __m64 data type can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

**__m128 Data Types**

The __m128 data type is used to represent the contents of a Intel® SSE register used by the Intel® SSE intrinsics. The __m128 data type can hold four 32-bit floating-point values.

The __m128d data type can hold two 64-bit floating-point values.

The __m128i data type can hold sixteen 8-bit, eight 16-bit, four 32-bit, or two 64-bit integer values.

The compiler aligns __m128d and __m128i local and global data to 16-byte boundaries on the stack. To align integer, float, or double arrays, you can use the declspec align statement.

**Data Types Usage Guidelines**

These data types are not basic ANSI C data types. You must observe the following usage restrictions:

- Use data types only on either side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (+, -, etc).
- Use data types as objects in aggregates, such as unions, to access the byte elements and structures.
- Use data types only with the respective intrinsics described in this documentation.
Accessing __m128i Data

To access 8-bit data:
#define _mm_extract_epi8(x, imm) 
  (((imm) & 0x1) == 0) ? \ 
  _mm_extract_epi16((x), (imm) >> 1) & 0xff : \ 
  _mm_extract_epi16(_mm_srli_epi16((x), 8), (imm) >> 1))

For 16-bit data, use the following intrinsic:
int _mm_extract_epi16(__m128i a, int imm)

To access 32-bit data:
#define _mm_extract_epi32(x, imm) \ 
  _mm_cvtsi128_si32(_mm_srli_si128((x), 4 * (imm)))

To access 64-bit data (Intel® 64 architecture only):
#define _mm_extract_epi64(x, imm) \ 
  _mm_cvtsi128_si64(_mm_srli_si128((x), 8 * (imm)))

Writing Programs with Intel(R) Streaming SIMD Extensions Intrinsics

You should be familiar with the hardware features provided by the Intel® Streaming SIMD Extensions (Intel® SSE) when writing programs with the intrinsics. The following are four important issues to keep in mind:

- Certain intrinsics, such as _mm_loadr_ps and _mm_cmpgt_ss, are not directly supported by the instruction set. While these intrinsics are convenient programming aids, be mindful that they may consist of more than one machine-language instruction.
- Floating-point data loaded or stored as __m128 objects must be generally 16-byte-aligned.
- Some intrinsics require that their argument be immediates, that is, constant integers (literals), due to the nature of the instruction.
- The result of arithmetic operations acting on two NaN (Not a Number) arguments is undefined. Therefore, FP operations using NaN arguments will not match the expected behavior of the corresponding assembly instructions.
Arithmetic Intrinsics

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for arithmetic operations are in the xmmintrin.h header file.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with R0-R3. R0, R1, R2 and R3 each represent one of the 4 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_ss</td>
<td>Addition</td>
<td>ADDSS</td>
</tr>
<tr>
<td>_mm_add_ps</td>
<td>Addition</td>
<td>ADDPS</td>
</tr>
<tr>
<td>_mm_sub_ss</td>
<td>Subtraction</td>
<td>SUBSS</td>
</tr>
<tr>
<td>_mm_sub_ps</td>
<td>Subtraction</td>
<td>SUBPS</td>
</tr>
<tr>
<td>_mm_mul_ss</td>
<td>Multiplication</td>
<td>MULSS</td>
</tr>
<tr>
<td>_mm_mul_ps</td>
<td>Multiplication</td>
<td>MULPS</td>
</tr>
<tr>
<td>_mm_div_ss</td>
<td>Division</td>
<td>DIVSS</td>
</tr>
<tr>
<td>_mm_div_ps</td>
<td>Division</td>
<td>DIVPS</td>
</tr>
<tr>
<td>_mm_sqrt_ss</td>
<td>Squared Root</td>
<td>SQRTSS</td>
</tr>
<tr>
<td>_mm_sqrt_ps</td>
<td>Squared Root</td>
<td>SQRTPS</td>
</tr>
<tr>
<td>_mm_rcp_ss</td>
<td>Reciprocal</td>
<td>RCPSS</td>
</tr>
<tr>
<td>_mm_rcp_ps</td>
<td>Reciprocal</td>
<td>RCPPS</td>
</tr>
<tr>
<td>_mm_rsqrt_ss</td>
<td>Reciprocal Squared Root</td>
<td>RSQRTSS</td>
</tr>
<tr>
<td>_mm_rsqrt_ps</td>
<td>Reciprocal Squared Root</td>
<td>RSQRTPS</td>
</tr>
</tbody>
</table>
## Intel® SSE Instruction Operation and Corresponding Intel® SSE Instruction

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_min_ss</td>
<td>Computes Minimum</td>
<td>MINSS</td>
</tr>
<tr>
<td>_mm_min_ps</td>
<td>Computes Minimum</td>
<td>MINPS</td>
</tr>
<tr>
<td>_mm_max_ss</td>
<td>Computes Maximum</td>
<td>MAXSS</td>
</tr>
<tr>
<td>_mm_max_ps</td>
<td>Computes Maximum</td>
<td>MAXPS</td>
</tr>
</tbody>
</table>

**__m128 __mm_add_ss(__m128 a, __m128 b)**

Adds the lower single-precision, floating-point (SP FP) values of `a` and `b`; the upper 3 SP FP values are passed through from `a`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
</tbody>
</table>

**__m128 __mm_add_ps(__m128 a, __m128 b)**

Adds the four SP FP values of `a` and `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1 + b1</td>
<td>a2 + b2</td>
<td>a3 + b3</td>
</tr>
</tbody>
</table>

**__m128 __mm_sub_ss(__m128 a, __m128 b)**

Subtracts the lower SP FP values of `a` and `b`. The upper 3 SP FP values are passed through from `a`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 - b0</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
</tbody>
</table>

**__m128 __mm_sub_ps(__m128 a, __m128 b)**

Subtracts the four SP FP values of `a` and `b`.
__m128 _mm_mul_ss(__m128 a, __m128 b)
Multiplies the lower SP FP values of $a$ and $b$; the upper 3 SP FP values are passed through from $a$.

__m128 _mm_mul_ps(__m128 a, __m128 b)
Multiplies the four SP FP values of $a$ and $b$.

__m128 _mm_div_ss(__m128 a, __m128 b)
Divides the lower SP FP values of $a$ and $b$; the upper 3 SP FP values are passed through from $a$.

__m128 _mm_div_ps(__m128 a, __m128 b)
Divides the four SP FP values of $a$ and $b$.

__m128 _mm_sqrt_ss(__m128 a)
Computes the square root of the lower SP FP value of $a$; the upper 3 SP FP values are passed through.
Computes the square roots of the four SP FP values of `a`.

Computes the approximations of the reciprocals of the four SP FP values of `a`.

Computes the approximation of the reciprocal of the square root of the lower SP FP value of `a`; the upper 3 SP FP values are passed through.

Computes the approximations of the reciprocals of the square roots of the four SP FP values of `a`.

---

1510
Reciprocals of square roots:

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>recip(sqrt(a0))</td>
<td>recip(sqrt(a1))</td>
<td>recip(sqrt(a2))</td>
<td>recip(sqrt(a3))</td>
</tr>
</tbody>
</table>

```c
__m128 _mm_min_ss(__m128 a, __m128 b)
```

Computes the minimum of the lower SP FP values of \(a\) and \(b\); the upper 3 SP FP values are passed through from \(a\).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>min(a0, b0)</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
</tbody>
</table>

```c
__m128 _mm_min_ps(__m128 a, __m128 b)
```

Computes the minimum of the four SP FP values of \(a\) and \(b\).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>min(a0, b0)</td>
<td>min(a1, b1)</td>
<td>min(a2, b2)</td>
<td>min(a3, b3)</td>
</tr>
</tbody>
</table>

```c
__m128 _mm_max_ss(__m128 a, __m128 b)
```

Computes the maximum of the lower SP FP values of \(a\) and \(b\); the upper 3 SP FP values are passed through from \(a\).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>max(a0, b0)</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
</tbody>
</table>

```c
__m128 _mm_max_ps(__m128 a, __m128 b)
```

Computes the maximum of the four SP FP values of \(a\) and \(b\).
Logical Intrinsics

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for logical operations are in the `xmmintrin.h` header file.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with R0-R3. R0, R1, R2 and R3 each represent one of the four 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_and_ps</td>
<td>Bitwise AND</td>
<td>ANDFS</td>
</tr>
<tr>
<td>_mm_andnot_ps</td>
<td>Bitwise ANDNOT</td>
<td>ANDNPS</td>
</tr>
<tr>
<td>_mm_or_ps</td>
<td>Bitwise OR</td>
<td>ORPS</td>
</tr>
<tr>
<td>_mm_xor_ps</td>
<td>Bitwise Exclusive OR</td>
<td>XORPS</td>
</tr>
</tbody>
</table>

\[
\text{__m128 } \_\text{mm\_and\_ps}(\text{__m128 } a, \text{__m128 } b)
\]
Computes the bitwise AND of the four SP FP values of \(a\) and \(b\).

\[
\begin{array}{cccc}
\text{R0} & \text{R1} & \text{R2} & \text{R3} \\
\hline
a0 & b0 & a1 & b1 & a2 & b2 & a3 & b3 \\
\end{array}
\]

\[
\text{__m128 } \_\text{mm\_andnot\_ps}(\text{__m128 } a, \text{__m128 } b)
\]
Computes the bitwise AND-NOT of the four SP FP values of \(a\) and \(b\).

\[
\begin{array}{cccc}
\text{R0} & \text{R1} & \text{R2} & \text{R3} \\
\hline
\sim a0 & b0 & \sim a1 & b1 & \sim a2 & b2 & \sim a3 & b3 \\
\end{array}
\]

\[
\text{__m128 } \_\text{mm\_or\_ps}(\text{__m128 } a, \text{__m128 } b)
\]
Computes the bitwise OR of the four SP FP values of \(a\) and \(b\).
Computes bitwise XOR (exclusive-or) of the four SP FP values of \( a \) and \( b \).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_0 \mid b_0 )</td>
<td>( a_1 \mid b_1 )</td>
<td>( a_2 \mid b_2 )</td>
<td>( a_3 \mid b_3 )</td>
</tr>
</tbody>
</table>

\[
\_\text{m128} \_\text{mm\_xor\_ps}(\_\text{m128} \ a, \_\text{m128} \ b)
\]

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for comparison operations are in the `xmmintrin.h` header file.

Each comparison intrinsic performs a comparison of \( a \) and \( b \). For the packed form, the four SP FP values of \( a \) and \( b \) are compared, and a 128-bit mask is returned. For the scalar form, the lower SP FP values of \( a \) and \( b \) are compared, and a 32-bit mask is returned; the upper three SP FP values are passed through from \( a \). The mask is set to \( 0xffffffff \) for each element where the comparison is true and \( 0x0 \) where the comparison is false.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with \( R \) or \( R0-R3 \). \( R0, R1, R2 \) and \( R3 \) each represent one of the 4 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_\text{mm_cmpeq_ss}</td>
<td>Equal</td>
<td>CMPEQSS</td>
</tr>
<tr>
<td>_\text{mm_cmpeq_ps}</td>
<td>Equal</td>
<td>CMPEQPS</td>
</tr>
<tr>
<td>_\text{mm_cmplt_ss}</td>
<td>Less Than</td>
<td>CMPLTSS</td>
</tr>
<tr>
<td>_\text{mm_cmplt_ps}</td>
<td>Less Than</td>
<td>CMPLTPS</td>
</tr>
<tr>
<td>_\text{mm_cmple_ss}</td>
<td>Less Than or Equal</td>
<td>CMPLESS</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE Instruction</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>_mm_cmple_ps</td>
<td>Less Than or Equal</td>
<td>CMPLTSS</td>
</tr>
<tr>
<td>_mm_cmpgt_ss</td>
<td>Greater Than</td>
<td>CMPLTSS</td>
</tr>
<tr>
<td>_mm_cmpgt_ps</td>
<td>Greater Than</td>
<td>CMPLTSS</td>
</tr>
<tr>
<td>_mm_cmpge_ss</td>
<td>Greater Than or Equal</td>
<td>CMPLTSS</td>
</tr>
<tr>
<td>_mm_cmpge_ps</td>
<td>Greater Than or Equal</td>
<td>CMPLTSS</td>
</tr>
<tr>
<td>_mm_cmpneq_ss</td>
<td>Not Equal</td>
<td>CMPNEQSS</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>Not Equal</td>
<td>CMPNEQPS</td>
</tr>
<tr>
<td>_mm_cmpnlt_ss</td>
<td>Not Less Than</td>
<td>CMPNLTSS</td>
</tr>
<tr>
<td>_mm_cmpnlt_ps</td>
<td>Not Less Than</td>
<td>CMPNLTSS</td>
</tr>
<tr>
<td>_mm_cmpnle_ss</td>
<td>Not Less Than or Equal</td>
<td>CMPNLLESS</td>
</tr>
<tr>
<td>_mm_cmpnle_ps</td>
<td>Not Less Than or Equal</td>
<td>CMPNLLEPS</td>
</tr>
<tr>
<td>_mm_cmpngt_ss</td>
<td>Not Greater Than</td>
<td>CMPNLTSS</td>
</tr>
<tr>
<td>_mm_cmpngt_ps</td>
<td>Not Greater Than</td>
<td>CMPNLTPS</td>
</tr>
<tr>
<td>_mm_cmpnge_ss</td>
<td>Not Greater Than or Equal</td>
<td>CMPNLLESS</td>
</tr>
<tr>
<td>_mm_cmpnge_ps</td>
<td>Not Greater Than or Equal</td>
<td>CMPNLLEPS</td>
</tr>
<tr>
<td>_mm_cmpord_ss</td>
<td>Ordered</td>
<td>CMPORDSS</td>
</tr>
<tr>
<td>_mm_cmpord_ps</td>
<td>Ordered</td>
<td>CMPORDPS</td>
</tr>
<tr>
<td>_mm_cmpunord_ss</td>
<td>Unordered</td>
<td>CMPUNORDSS</td>
</tr>
<tr>
<td>_mm_cmpunord_ps</td>
<td>Unordered</td>
<td>CMPUNORDPS</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE Instruction</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_comieq_ss</td>
<td>Equal</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comilt_ss</td>
<td>Less Than</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comile_ss</td>
<td>Less Than or Equal</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comigt_ss</td>
<td>Greater Than</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comige_ss</td>
<td>Greater Than or Equal</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comineq_ss</td>
<td>Not Equal</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_ucomieq_ss</td>
<td>Equal</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomilt_ss</td>
<td>Less Than</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomile_ss</td>
<td>Less Than or Equal</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomigt_ss</td>
<td>Greater Than</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomige_ss</td>
<td>Greater Than or Equal</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomineq_ss</td>
<td>Not Equal</td>
<td>UCOMISS</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpeq_ss(__m128 a, __m128 b)

Compares for equality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff</td>
<td>0x0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpeq_ps(__m128 a, __m128 b)

Compares for equality.
# Comparisons

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ?</td>
<td>(a1 == b1) ?</td>
<td>(a2 == b2) ?</td>
<td>(a3 == b3) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128 __m_cmplt_ss(__m128 a, __m128 b)

Compares for less-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt; b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 __m_cmplt_ps(__m128 a, __m128 b)

Compares for less-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt; b0) ?</td>
<td>(a1 &lt; b1) ?</td>
<td>(a2 &lt; b2) ?</td>
<td>(a3 &lt; b3) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128 __m_cmple_ss(__m128 a, __m128 b)

Compares for less-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt;= b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 __m_cmple_ps(__m128 a, __m128 b)

Compares for less-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt;= b0) ?</td>
<td>(a1 &lt;= b1) ?</td>
<td>(a2 &lt;= b2) ?</td>
<td>(a3 &lt;= b3) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128 __m_cmpgt_ss(__m128 a, __m128 b)
Compares for greater-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a_0 &gt; b_0)$ ?</td>
<td>$a_1$</td>
<td>$a_2$</td>
<td>$a_3$</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpgt_ps(__m128 a, __m128 b)

Compares for greater-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a_0 &gt; b_0)$ ?</td>
<td>$(a_1 &gt; b_1)$ ?</td>
<td>$(a_2 &gt; b_2)$ ?</td>
<td>$(a_3 &gt; b_3)$ ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpge_ss(__m128 a, __m128 b)

Compares for greater-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a_0 &gt;= b_0)$ ?</td>
<td>$a_1$</td>
<td>$a_2$</td>
<td>$a_3$</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpge_ps(__m128 a, __m128 b)

Compares for greater-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a_0 &gt;= b_0)$ ?</td>
<td>$(a_1 &gt;= b_1)$ ?</td>
<td>$(a_2 &gt;= b_2)$ ?</td>
<td>$(a_3 &gt;= b_3)$ ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpneq_ss(__m128 a, __m128 b)

Compares for inequality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a_0 != b_0)$ ?</td>
<td>$a_1$</td>
<td>$a_2$</td>
<td>$a_3$</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
__m128 _mm_cmpneq_ps(__m128 a, __m128 b)

Compares for inequality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 != b0) ?</td>
<td>(a1 != b1) ?</td>
<td>(a2 != b2) ?</td>
<td>(a3 != b3) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnlt_ss(__m128 a, __m128 b)

Compares for not-less-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt; b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnlt_ps(__m128 a, __m128 b)

Compares for not-less-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt; b0) ?</td>
<td>!(a1 &lt; b1) ?</td>
<td>!(a2 &lt; b2) ?</td>
<td>!(a3 &lt; b3) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnle_ss(__m128 a, __m128 b)

Compares for not-less-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt;= b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnle_ps(__m128 a, __m128 b)

Compares for not-less-than-or-equal.
```
__m128 _mm_cmpngt_ss(__m128 a, __m128 b)
Compares for not-greater-than.

__m128 _mm_cmpngt_ps(__m128 a, __m128 b)
Compares for not-greater-than.

__m128 _mm_cmpnge_ss(__m128 a, __m128 b)
Compares for not-greater-than-or-equal.

__m128 _mm_cmpnge_ps(__m128 a, __m128 b)
Compares for not-greater-than-or-equal.

__m128 _mm_cmpord_ss(__m128 a, __m128 b)

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt;= b0) ?</td>
<td>!(a1 &lt;= b1) ?</td>
<td>!(a2 &lt;= b2) ?</td>
<td>!(a3 &lt;= b3) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>
```

Compares for ordered.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(\text{a0 ord? b0}) & ? & a1 & a2 & a3 \\
0xffffffff : 0x0 & & & \\
\text{___m128 ___mm_cmpord_ps(__m128 a, __m128 b)} & \\
\end{array}
\]

Compares for ordered.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(\text{a0 ord? b0}) & ? & (\text{a1 ord? b1}) & ? & (\text{a2 ord? b2}) & ? & (\text{a3 ord? b3}) & ? \\
0xffffffff : 0x0 & 0xffffffff : 0x0 & 0xffffffff : 0x0 & 0xffffffff : 0x0 & 0xffffffff : 0x0 \\
\text{___m128 ___mm_cmpunord_ss(__m128 a, __m128 b)} & \\
\end{array}
\]

Compares for unordered.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(\text{a0 unord? b0}) & ? & a1 & a2 & a3 \\
0xffffffff : 0x0 & & & \\
\text{___m128 ___mm_cmpunord_ps(__m128 a, __m128 b)} & \\
\end{array}
\]

Compares for unordered.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(\text{a0 unord? b0}) & ? & (\text{a1 unord? b1}) & ? & (\text{a2 unord? b2}) & ? & (\text{a3 unord? b3}) & ? \\
0xffffffff : 0x0 & 0xffffffff : 0x0 & 0xffffffff : 0x0 & 0xffffffff : 0x0 & 0xffffffff : 0x0 \\
\text{int ___mm_comieq_ss(__m128 a, __m128 b)} & \\
\end{array}
\]

Compares the lower SP FP value of \(a\) and \(b\) for \(a\) equal to \(b\). If \(a\) and \(b\) are equal, 1 is returned. Otherwise 0 is returned.

\[
\begin{array}{c}
R \\
(a0 == b0) & ? & 0x1 & : 0x0 \\
\end{array}
\]

1520
int _mm_comilt_ss(__m128 a, __m128 b)
Compares the lower SP FP value of a and b for a less than b. If a is less than b, 1 is returned. Otherwise 0 is returned.

R

(a0 < b0) ? 0x1 : 0x0

int _mm_comile_ss(__m128 a, __m128 b)
Compares the lower SP FP value of a and b for a less than or equal to b. If a is less than or equal to b, 1 is returned. Otherwise 0 is returned.

R

(a0 <= b0) ? 0x1 : 0x0

int _mm_comigt_ss(__m128 a, __m128 b)
Compares the lower SP FP value of a and b for a greater than b. If a is greater than b are equal, 1 is returned. Otherwise 0 is returned.

R

(a0 > b0) ? 0x1 : 0x0

int _mm_comige_ss(__m128 a, __m128 b)
Compares the lower SP FP value of a and b for a greater than or equal to b. If a is greater than or equal to b, 1 is returned. Otherwise 0 is returned.

R

(a0 >= b0) ? 0x1 : 0x0

int _mm_comineq_ss(__m128 a, __m128 b)
Compares the lower SP FP value of a and b for a not equal to b. If a and b are not equal, 1 is returned. Otherwise 0 is returned.
<table>
<thead>
<tr>
<th>Equation</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(a0 != b0) ? 0x1 : 0x0</code></td>
<td><code>int _mm_ucomieq_ss(__m128 a, __m128 b)</code> Compares the lower SP FP value of ( a ) and ( b ) for ( a ) equal to ( b ). If ( a ) and ( b ) are equal, 1 is returned. Otherwise 0 is returned.</td>
</tr>
<tr>
<td><code>(a0 == b0) ? 0x1 : 0x0</code></td>
<td><code>int _mm_ucomilt_ss(__m128 a, __m128 b)</code> Compares the lower SP FP value of ( a ) and ( b ) for ( a ) less than ( b ). If ( a ) is less than ( b ), 1 is returned. Otherwise 0 is returned.</td>
</tr>
<tr>
<td><code>(a0 &lt; b0) ? 0x1 : 0x0</code></td>
<td><code>int _mm_ucomile_ss(__m128 a, __m128 b)</code> Compares the lower SP FP value of ( a ) and ( b ) for ( a ) less than or equal to ( b ). If ( a ) is less than or equal to ( b ), 1 is returned. Otherwise 0 is returned.</td>
</tr>
<tr>
<td><code>(a0 &lt;= b0) ? 0x1 : 0x0</code></td>
<td><code>int _mm_ucomigt_ss(__m128 a, __m128 b)</code> Compares the lower SP FP value of ( a ) and ( b ) for ( a ) greater than ( b ). If ( a ) is greater than or equal to ( b ), 1 is returned. Otherwise 0 is returned.</td>
</tr>
<tr>
<td><code>(a0 &gt; b0) ? 0x1 : 0x0</code></td>
<td><code>int _mm_ucomige_ss(__m128 a, __m128 b)</code></td>
</tr>
</tbody>
</table>
Compares the lower SP FP value of \(a\) and \(b\) for \(a\) greater than or equal to \(b\). If \(a\) is greater than or equal to \(b\), 1 is returned. Otherwise 0 is returned.

\[
R
(a0 >= b0) ? 0x1 : 0x0
\]

\[
\text{int } \_\_m\_u\_comineq\_ss(\_\_m\_128\ a, \_\_m\_128\ b)
\]

Compares the lower SP FP value of \(a\) and \(b\) for \(a\) not equal to \(b\). If \(a\) and \(b\) are not equal, 1 is returned. Otherwise 0 is returned.

\[
R
r := (a0 != b0) ? 0x1 : 0x0
\]

Conversion Intrinsics

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for conversion operations are in the \texttt{xmmintrin.h} header file.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with \(R\) or \(R0\)-\(R3\). \(R0\), \(R1\), \(R2\) and \(R3\) each represent one of the 4 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m_cv__t_ss_si32</td>
<td>Convert to 32-bit integer</td>
<td>CVTSS2SI</td>
</tr>
<tr>
<td>__m_cv__t_ss_si64</td>
<td>Convert to 64-bit integer</td>
<td>CVTSS2SI</td>
</tr>
<tr>
<td>__m_cv__t_ps_pi32</td>
<td>Convert to two 32-bit integers</td>
<td>CVTPS2PI</td>
</tr>
<tr>
<td>__m_cv__t_ss_si32</td>
<td>Convert to 32-bit integer</td>
<td>CVTTSS2SI</td>
</tr>
<tr>
<td>__m_cv__t_ss_si64</td>
<td>Convert to 64-bit integer</td>
<td>CVTTSS2SI</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE Instruction</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_cvttps_pi32</td>
<td>Convert to two 32-bit integers</td>
<td>CVTTPS2PI</td>
</tr>
<tr>
<td>_mm_cvtsi32_ss</td>
<td>Convert from 32-bit integer</td>
<td>CVTSI2SS</td>
</tr>
<tr>
<td>_mm_cvtsi64_ss</td>
<td>Convert from 64-bit integer</td>
<td>CVTSI2SS</td>
</tr>
<tr>
<td>_mm_cvtpi32_ps</td>
<td>Convert from two 32-bit integers</td>
<td>CVTTPI2PS</td>
</tr>
<tr>
<td>_mm_cvtpi16_ps</td>
<td>Convert from four 16-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtpu16_ps</td>
<td>Convert from four 16-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtpi8_ps</td>
<td>Convert from four 8-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtpu8_ps</td>
<td>Convert from four 8-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtpi32x2_ps</td>
<td>Convert from four 32-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtps_pi16</td>
<td>Convert to four 16-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtps_pi8</td>
<td>Convert to four 8-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvts_f32</td>
<td>Extract</td>
<td>composite</td>
</tr>
</tbody>
</table>

int _mm_cvtsi32(__m128 a)

Converts the lower SP FP value of a to a 32-bit integer according to the current rounding mode.
_int64 _mm_cvtss_si64(__m128 a)
Converts the lower SP FP value of a to a 64-bit signed integer according to the current rounding mode.

__m64 _mm_cvtps_pi32(__m128 a)
Converts the two lower SP FP values of a to two 32-bit integers according to the current rounding mode, returning the integers in packed form.

int _mm_cvttss_si32(__m128 a)
Converts the lower SP FP value of a to a 32-bit integer with truncation.

__int64 _mm_cvtss_si64(__m128 a)
Converts the lower SP FP value of a to a 64-bit signed integer with truncation.

__m64 _mm_cvttps_pi32(__m128 a)
Converts the two lower SP FP values of a to two 32-bit integer with truncation, returning the integers in packed form.
### __m128 _mm_cvtsi32_ss(__m128 a, int b)\[R0\] __m128 __m128

Converts the 32-bit integer value \(b\) to an SP FP value; the upper three SP FP values are passed through from \(a\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(int)a0 & (int)a1
\end{array}
\]

\[
\_m128 \_mm_cvtsi32\_ss(_m128 a, \text{int} b)
\]

### __m128 _mm_cvtsi64_ss(__m128 a, __int64 b)\[R0\] __m128

Converts the signed 64-bit integer value \(b\) to an SP FP value; the upper three SP FP values are passed through from \(a\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(float)b & a1 & a2 & a3
\end{array}
\]

\[
\_m128 \_mm_cvtsi64\_ss(_m128 a, _\text{int64} b)
\]

### __m128 _mm_cvtpi32_ps(__m64 a, __m64 b)\[R0\] __m128

Converts the two 32-bit integer values in packed form in \(b\) to two SP FP values; the upper two SP FP values are passed through from \(a\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(float)b0 & (float)b1 & a2 & a3
\end{array}
\]

\[
\_m128 \_mm_cvtpi32\_ps(_m64 a, _m64 b)
\]

### __m128 _mm_cvtpi16_ps(__m64 a)\[R0\] __m128

Converts the four 16-bit signed integer values in \(a\) to four single precision FP values.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(float)a0 & (float)a1 & (float)a2 & (float)a3
\end{array}
\]

\[
\_m128 \_mm_cvtpi16\_ps(_m64 a)
\]

### __m128 _mm_cvtpu16_ps(__m64 a)\[R0\] __m128

Converts the four 16-bit unsigned integer values in \(a\) to four single precision FP values.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(float)a0 & (float)a1 & (float)a2 & (float)a3
\end{array}
\]

\[
\_m128 \_mm_cvtpu16\_ps(_m64 a)
\]
<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(float)a0</td>
<td>(float)a1</td>
<td>(float)a2</td>
<td>(float)a3</td>
</tr>
</tbody>
</table>

___m128 _mm_cvtpi8_ps(__m64 a)

Converts the lower four 8-bit signed integer values in a to four single precision FP values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(float)a0</td>
<td>(float)a1</td>
<td>(float)a2</td>
<td>(float)a3</td>
</tr>
</tbody>
</table>

___m128 _mm_cvtpu8_ps(__m64 a)

Converts the lower four 8-bit unsigned integer values in a to four single precision FP values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(float)a0</td>
<td>(float)a1</td>
<td>(float)a2</td>
<td>(float)a3</td>
</tr>
</tbody>
</table>

___m128 _mm_cvtpi32x2_ps(__m64 a, __m64 b)

Converts the two 32-bit signed integer values in a and the two 32-bit signed integer values in b to four single precision FP values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(float)a0</td>
<td>(float)a1</td>
<td>(float)b0</td>
<td>(float)b1</td>
</tr>
</tbody>
</table>

___m64 _mm_cvtps_pi16(__m128 a)

Converts the four single precision FP values in a to four signed 16-bit integer values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(short)a0</td>
<td>(short)a1</td>
<td>(short)a2</td>
<td>(short)a3</td>
</tr>
</tbody>
</table>

___m64 _mm_cvtps_pi8(__m128 a)

Converts the four single precision FP values in a to the lower four signed 8-bit integer values of the result.
float _mm_cvtss_f32(__m128 a)

Extracts a single precision floating point value from the first vector element of an __m128. It does so in the most efficient manner possible in the context used.

### Load Intrinsics

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for load operations are in the xmmintrin.h header file.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with R0-R3. R0, R1, R2 and R3 each represent one of the 4 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_loadh_pi</td>
<td>Load high</td>
<td>MOVHPS reg, mem</td>
</tr>
<tr>
<td>_mm_loadl_pi</td>
<td>Load low</td>
<td>MOVLPS reg, mem</td>
</tr>
<tr>
<td>_mm_load_ss</td>
<td>Load the low value and clear the three high values</td>
<td>MOVSS</td>
</tr>
<tr>
<td>_mm_load1_ps</td>
<td>Load one value into all four words</td>
<td>MOVSS + Shuffling</td>
</tr>
<tr>
<td>_mm_load_ps</td>
<td>Load four values, address aligned</td>
<td>MOVAPS</td>
</tr>
<tr>
<td>_mm_loadu_ps</td>
<td>Load four values, address unaligned</td>
<td>MOVUPS</td>
</tr>
<tr>
<td>_mm_loadr_ps</td>
<td>Load four values in reverse</td>
<td>MOVAPS + Shuffling</td>
</tr>
</tbody>
</table>

__m128 _mm_loadh_pi(__m128 a, __m64 const *p)
Sets the upper two SP FP values with 64 bits of data loaded from the address \( p \).

\[ \begin{array}{cccc}
R0 & R1 & R2 & R3 \\
a0 & a1 & *p0 & *p1 \\
\end{array} \]

\_m128 \_mm_loadl_pi(__m128 a, __m64 const *p)

Sets the lower two SP FP values with 64 bits of data loaded from the address \( p \); the upper two values are passed through from \( a \).

\[ \begin{array}{cccc}
R0 & R1 & R2 & R3 \\
*p0 & *p1 & a2 & a3 \\
\end{array} \]

\_m128 \_mm_load_ss(float * p )

Loads an SP FP value into the low word and clears the upper three words.

\[ \begin{array}{cccc}
R0 & R1 & R2 & R3 \\
*p & 0.0 & 0.0 & 0.0 \\
\end{array} \]

\_m128 \_mm_load1_ps(float * p)

Loads a single SP FP value, copying it into all four words.

\[ \begin{array}{cccc}
R0 & R1 & R2 & R3 \\
*p & *p & *p & *p \\
\end{array} \]

\_m128 \_mm_load_ps(float * p)

Loads four SP FP values. The address must be 16-byte-aligned.

\[ \begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\end{array} \]

\_m128 \_mm_loadu_ps(float * p)

 Loads four SP FP values. The address need not be 16-byte-aligned.
__m128 _mm_loadr_ps(float * p)

Loads four SP FP values in reverse order. The address must be 16-byte-aligned.

---

**Set Intrinsics**

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for set operations are in the *xmmintrin.h* header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R0, R1, R2 and R3 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_set_ss</td>
<td>Set the low value and clear the three high values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set1_ps</td>
<td>Set all four words with the same value</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set_ps</td>
<td>Set four values, address aligned</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_ps</td>
<td>Set four values, in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setzero_ps</td>
<td>Clear all four values</td>
<td>Composite</td>
</tr>
</tbody>
</table>

_**__m128 _mm_set_ss(float w )**_

Sets the low word of an SP FP value to \( w \) and clears the upper three words.
**Store Intrinsics**

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for store operations are in the `xmmintrin.h` header file.
The description for each intrinsic contains a table detailing the returns. In these tables, \( p[n] \) is an access to the \( n \) element of the result.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_storeh_pi</td>
<td>Store high</td>
<td>MOVHPS mem, reg</td>
</tr>
<tr>
<td>_mm_storel_pi</td>
<td>Store low</td>
<td>MOVLPS mem, reg</td>
</tr>
<tr>
<td>_mm_store_ss</td>
<td>Store the low value</td>
<td>MOVSS</td>
</tr>
<tr>
<td>_mm_storel_ps</td>
<td>Store the low value across all four words, address aligned</td>
<td>Shuffling + MOVSS</td>
</tr>
<tr>
<td>_mm_store_ps</td>
<td>Store four values, address aligned</td>
<td>MOVAPS</td>
</tr>
<tr>
<td>_mm_storeu_ps</td>
<td>Store four values, address unaligned</td>
<td>MOVUAPS</td>
</tr>
<tr>
<td>_mm_storer_ps</td>
<td>Store four values, in reverse order</td>
<td>MOVAPS + Shuffling</td>
</tr>
</tbody>
</table>

void _mm_storeh_pi(__m64 *p, __m128 a)
Stores the upper two SP FP values to the address \( p \).

\[
\begin{array}{ccc}
  *p0 & & *p1 \\
  \text{a2} & & \text{a3}
\end{array}
\]

void _mm_storel_pi(__m64 *p, __m128 a)
Stores the lower two SP FP values of \( a \) to the address \( p \).

\[
\begin{array}{ccc}
  *p0 & & *p1 \\
  \text{a0} & & \text{a1}
\end{array}
\]

void _mm_store_ss(float * p, __m128 a)
Stores the lower SP FP value.

\*p

\a0

void _mm_store1_ps(float * p, __m128 a )
Stores the lower SP FP value across four words.

\begin{tabular}{cccc}
 a0 & a0 & a0 & a0 \\
\end{tabular}

void _mm_store_ps(float *p, __m128 a )
Stores four SP FP values. The address must be 16-byte-aligned.

\begin{tabular}{cccc}
 a0 & a1 & a2 & a3 \\
\end{tabular}

void _mm_storeu_ps(float *p, __m128 a )
Stores four SP FP values. The address need not be 16-byte-aligned.

\begin{tabular}{cccc}
 a0 & a1 & a2 & a3 \\
\end{tabular}

void _mm_storer_ps(float * p, __m128 a )
Stores four SP FP values in reverse order. The address must be 16-byte-aligned.

\begin{tabular}{cccc}
 a3 & a2 & a1 & a0 \\
\end{tabular}
Cacheability Support Intrinsics

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for cacheability support are in the `xmmintrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_prefetch</td>
<td>Load</td>
<td>PREFETCH</td>
</tr>
<tr>
<td>_mm_stream_pi</td>
<td>Store</td>
<td>MOVNTQ</td>
</tr>
<tr>
<td>_mm_stream_ps</td>
<td>Store</td>
<td>MOVNTPS</td>
</tr>
<tr>
<td>_mm256_stream_ps</td>
<td>Store</td>
<td>VMOVNTPS</td>
</tr>
<tr>
<td>_mm_sfence</td>
<td>Store fence</td>
<td>SFENCE</td>
</tr>
</tbody>
</table>

void _mm_prefetch(char const*a, int sel)

Loads one cache line of data from address a to a location "closer" to the processor. The value sel specifies the type of prefetch operation: the constants _MM_HINT_T0, _MM_HINT_T1, _MM_HINT_T2, and _MM_HINT_NTA should be used for systems based on IA-32 architecture, and correspond to the type of prefetch instruction. The constants _MM_HINT_T1, _MM_HINT_NT1, _MM_HINT_NT2, and _MM_HINT_NTA should be used for systems based on IA-64 architecture.

void _mm_stream_pi(__m64 *p, __m64 a)

Stores the data in a to the address p without polluting the caches. This intrinsic requires you to empty the multimedia state for the MMX™ register. See the topic The EMMS Instruction: Why You Need It.

void _mm_stream_ps(float *p, __m128 a)

Stores the data in a to the address p without polluting the caches. The address must be 16-byte-aligned.

void _mm256_stream_ps(float *p, __m256 a)

Stores the data in a to the address p without polluting the caches. The address must be 32-byte (VEX.256 encoded version) aligned.
void _mm_sfence(void)

Guarantees that every preceding store is globally visible before any subsequent store.

**Integer Intrinsics**

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for integer operations are in the `xmmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0, R1...R7 represent the registers in which results are placed.

Before using these intrinsics, you must empty the multimedia state for the MMX™ technology register. See The EMMS Instruction: Why You Need It for more details.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_extract_pi16</td>
<td>Extract one of four words</td>
<td>PEXTRW</td>
</tr>
<tr>
<td>_mm_insert_pi16</td>
<td>Insert word</td>
<td>PINSRW</td>
</tr>
<tr>
<td>_mm_max_pi16</td>
<td>Compute maximum</td>
<td>PMAXSW</td>
</tr>
<tr>
<td>_mm_max_pu8</td>
<td>Compute maximum, unsigned</td>
<td>PMAXUB</td>
</tr>
<tr>
<td>_mm_min_pi16</td>
<td>Compute minimum</td>
<td>PMINSW</td>
</tr>
<tr>
<td>_mm_min_pu8</td>
<td>Compute minimum, unsigned</td>
<td>PMINUB</td>
</tr>
<tr>
<td>_mm_movemask_pi8</td>
<td>Create eight-bit mask</td>
<td>PMOVMSKB</td>
</tr>
<tr>
<td>_mm_mulhi_pu16</td>
<td>Multiply, return high bits</td>
<td>PMULHUW</td>
</tr>
<tr>
<td>_mm_shuffle_pi16</td>
<td>Return a combination of four words</td>
<td>PSHUFW</td>
</tr>
<tr>
<td>_mm_maskmove_si64</td>
<td>Conditional Store</td>
<td>MASKMOVQ</td>
</tr>
<tr>
<td>_mm_avg_pu8</td>
<td>Compute rounded average</td>
<td>PAVGB</td>
</tr>
</tbody>
</table>
### Intrinsic Name | Operation | Corresponding Intel® SSE Instruction
--- | --- | ---
_mm_avg_pu16 | Compute rounded average | PAVGW
_mm_sad_pu8 | Compute sum of absolute differences | FSADBW

```c
int _mm_extract_pi16(__m64 a, int n)
```

Extracts one of the four words of a. The selector n must be an immediate.

```c
R
(n==0) ? a0 : ( (n==1) ? a1 : ( (n==2) ? a2 : a3 ) )
```

```c
__m64 _mm_insert_pi16(__m64 a, int d, int n)
```

Inserts word d into one of four words of a. The selector n must be an immediate.

```c
R0    R1    R2    R3
(n==0) ? d : a0; (n==1) ? d : a1; (n==2) ? d : a2; (n==3) ? d : a3;
```

```c
__m64 _mm_max_pi16(__m64 a, __m64 b)
```

Computes the element-wise maximum of the words in a and b.

```c
R0    R1    R2    R3
min(a0, b0) min(a1, b1) min(a2, b2) min(a3, b3)
```

```c
__m64 _mm_max_pu8(__m64 a, __m64 b)
```

Computes the element-wise maximum of the unsigned bytes in a and b.

```c
R0    R1    R2    R7
min(a0, b0) min(a1, b1) ... min(a7, b7)
```

```c
__m64 _mm_min_pi16(__m64 a, __m64 b)
```

Computes the element-wise minimum of the words in a and b.
__m64 _mm_min_pu8(__m64 a, __m64 b)

Computes the element-wise minimum of the unsigned bytes in a and b.

__m64 _mm_movemask_pi8(__m64 b)

Creates an 8-bit mask from the most significant bits of the bytes in a.

__m64 _mm_mulhi_pu16(__m64 a, __m64 b)

Multiplies the unsigned words in a and b, returning the upper 16 bits of the 32-bit intermediate results.

__m64 _mm_shuffle_pi16(__m64 a, int n)

Returns a combination of the four words of a. The selector n must be an immediate.

void _mm_maskmove_si64(__m64 d, __m64 n, char *p)

Conditionally stores byte elements of d to address p. The high bit of each byte in the selector n determines whether the corresponding byte in d will be stored.
if (sign(n0))
if (sign(n1))
...  
if (sign(n7))

\[
p[0] := d0  \quad p[1] := d1  \quad \ldots  \quad p[7] := d7
\]

\texttt{\_m64 \_mm_avg\_pu8(\_m64 a, \_m64 b)}

Computes the (rounded) averages of the unsigned bytes in \(a\) and \(b\).

\[
\begin{array}{llll}
R0 & R1 & \ldots & R7 \\
(t >> 1) | (t & 0x01), where t = (unsigned char)a0 & (t >> 1) | (t & 0x01), where t = (unsigned char)a1 & \ldots & ((t >> 1) | (t & 0x01), where t = (unsigned char)a7 \\
(\text{unsigned char})b0 & (\text{unsigned char})b1
\end{array}
\]

\texttt{\_m64 \_mm_avg\_pu16(\_m64 a, \_m64 b)}

Computes the (rounded) averages of the unsigned short in \(a\) and \(b\).

\[
\begin{array}{llll}
R0 & R1 & \ldots & R7 \\
(t >> 1) | (t & 0x01), where t = (unsigned int)a0 & (t >> 1) | (t & 0x01), where t = (unsigned int)a1 & \ldots & ((t >> 1) | (t & 0x01), where t = (unsigned int)a7 \\
(\text{unsigned int})b0 & (\text{unsigned int})b1
\end{array}
\]

\texttt{\_m64 \_mm_sad\_pu8(\_m64 a, \_m64 b)}

Computes the sum of the absolute differences of the unsigned bytes in \(a\) and \(b\), returning the value in the lower word. The upper three words are cleared.

\[
\begin{array}{llll}
R0 & R1 & R2 & R3 \\
abs(a0-b0) + \ldots + 0 & 0 & 0
\end{array}
\]
Intrinsics to Read and Write Registers

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics to read from and write to registers are in the `xmmintrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_getcsr</td>
<td>Return control register</td>
<td>STMXCSR</td>
</tr>
<tr>
<td>_mm_setcsr</td>
<td>Set control register</td>
<td>LDMXCSR</td>
</tr>
</tbody>
</table>

`unsigned int _mm_getcsr(void)`

Returns the contents of the control register.

`void _mm_setcsr(unsigned int i)`

Sets the control register to the value specified.

Miscellaneous Intrinsics

The prototypes for Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics for miscellaneous operations are in the `xmmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0, R1, R2 and R3 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_shuffle_ps</td>
<td>Shuffle</td>
<td>SHUFFPS</td>
</tr>
<tr>
<td>_mm_unpackhi_ps</td>
<td>Unpack High</td>
<td>UNPCKHPS</td>
</tr>
<tr>
<td>_mm_unpacklo_ps</td>
<td>Unpack Low</td>
<td>UNPCKLPS</td>
</tr>
<tr>
<td>_mm_move_ss</td>
<td>Set low word, pass in three high values</td>
<td>MOVSS</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE Instruction</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>__m128 __m_shuffle_ps(__m128 a, __m128 b, unsigned int imm8)</td>
<td>Selects four specific SP FP values from a and b, based on the mask imm8. The mask must be an immediate. See Macro Function for Shuffle Using Streaming SIMD Extensions for a description of the shuffle semantics.</td>
<td></td>
</tr>
<tr>
<td>__m128 __m_unpackhi_ps(__m128 a, __m128 b)</td>
<td>Selects and interleaves the upper two SP FP values from a and b.</td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>a3</td>
</tr>
<tr>
<td>__m128 __m_unpacklo_ps(__m128 a, __m128 b)</td>
<td>Selects and interleaves the lower two SP FP values from a and b.</td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>a0</td>
<td>b0</td>
<td>a1</td>
</tr>
<tr>
<td>__m128 __m_move_ss(__m128 a, __m128 b)</td>
<td>Sets the low word to the SP FP value of b. The upper 3 SP FP values are passed through from a.</td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>b0</td>
<td>a1</td>
<td>a2</td>
</tr>
<tr>
<td>__m128 __m_movehl_ps(__m128 a, __m128 b)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Moves the upper 2 SP FP values of $b$ to the lower 2 SP FP values of the result. The upper 2 SP FP values of $a$ are passed through to the result.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
b2 & b3 & a2 & a3 \\
\end{array}
\]

\_m128 \_mm_movelh_ps(\_m128 a, \_m128 b)

Moves the lower 2 SP FP values of $b$ to the upper 2 SP FP values of the result. The lower 2 SP FP values of $a$ are passed through to the result.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
a0 & a1 & b0 & b1 \\
\end{array}
\]

int \_mm_movemask_ps(\_m128 a)

Creates a 4-bit mask from the most significant bits of the four SP FP values.

\[
R
\begin{align*}
& \text{sign}(a3)\ll 3 &|& \text{sign}(a2)\ll 2 &|& \text{sign}(a1)\ll 1 &|& \text{sign}(a0)
\end{align*}
\]

Using Intel(R) Streaming SIMD Extensions on IA-64 Architecture

The Intel\textsuperscript{®} Streaming SIMD Extensions (Intel\textsuperscript{®} SSE) intrinsics provide access to IA-64 instructions for streaming SIMD extensions. To provide source compatibility with the IA-32 architecture, these intrinsics are equivalent both in name and functionality to the set of IA-32 architecture-based Intel\textsuperscript{®} SSE intrinsics.

To write programs with the intrinsics, you should be familiar with the hardware features provided by Intel\textsuperscript{®} SSE. Keep the following issues in mind:

- Certain intrinsics are provided only for compatibility with previously-defined IA-32 architecture-based intrinsics. Using them on systems based on IA-64 architecture probably leads to performance degradation.
- Floating-point (FP) data loaded stored as \_m128 objects must be 16-byte-aligned.
- Some intrinsics require that their arguments be immediates -- that is, constant integers (literals), due to the nature of the instruction.
Data Types

The new data type __m128 is used with the SSE intrinsics. It represents a 128-bit quantity composed of four single-precision FP values. This corresponds to the 128-bit IA-32 architecture-based Intel® Streaming SIMD Extensions register.

The compiler aligns __m128 local data to 16-byte boundaries on the stack. Global data of these types is also 16 byte-aligned. To align integer, float, or double arrays, you can use thedeclspec alignment.

Because IA-64 instructions treat the SSE registers in the same way whether you are using packed or scalar data, there is no __m32 data type to represent scalar data. For scalar operations, use the __m128 objects and the "scalar" forms of the intrinsics; the compiler and the processor implement these operations with 32-bit memory references. But, for better performance the packed form should be substituting for the scalar form whenever possible.

The address of a __m128 object may be taken.


Implementation on Systems based on IA-64 architecture

Intel® SSE intrinsics are defined for the __m128 data type, a 128-bit quantity consisting of four single-precision FP values. Single-instruction multiple data (SIMD) instructions for systems based on IA-64 architecture operate on 64-bit FP register quantities containing two single-precision floating-point values. Thus, each __m128 operand is actually a pair of FP registers and therefore each intrinsic corresponds to at least one pair of IA-64 instructions operating on the pair of FP register operands.

Compatibility versus Performance

Many of the Intel® SSE intrinsics for systems based on IA-64 architecture were created for compatibility with existing IA-32 architecture-based intrinsics and not for performance. In some situations, intrinsic usage that improved performance on IA-32 architecture will not do so on systems based on IA-64 architecture. One reason for this is that some intrinsics map nicely into the IA-32 instruction set but not into the IA-64 instruction set. Thus, it is important to differentiate between intrinsics which were implemented for a performance advantage on systems based on IA-64 architecture, and those implemented simply to provide compatibility with existing IA-32 architecture-based code.

The following intrinsics are likely to reduce performance and should only be used to initially port legacy code or in non-critical code sections:
Any Intel® SSE scalar intrinsic (_ss variety) - use packed (_ps) version if possible

comi and ucomi Intel® SSE comparisons - these correspond to IA-32 architecture-based COMISS and UCOMISS instructions only. A sequence of IA-64 instructions are required to implement these.

Conversions in general are multi-instruction operations. These are particularly expensive:

- _mm_cvtpi16_ps, _mm_cvtpu16_ps, _mm_cvtpi8_ps, _mm_cvtpu8_ps, _mm_cvtpi32x2_ps,
- _mm_cvtps_pi16, _mm_cvtps_pi8

Intel® SSE utility intrinsic _mm_movemask_ps

If the inaccuracy is acceptable, the SIMD reciprocal and reciprocal square root approximation intrinsics (rcp and rsqrt) are much faster than the true div and sqrt intrinsics.

Macro Functions

Macro Function for Shuffle Operations

The Intel® Streaming SIMD Extensions (Intel® SSE) provide a macro function to help create constants that describe shuffle operations. The macro takes four small integers (in the range of 0 to 3) and combines them into an 8-bit immediate value used by the SHUFPS instruction.

Shuffle Function Macro

```
_MM_SHUFFLE2(x, y)
```

expands to the value of

```
(x<<1) | y
```

You can view the four integers as selectors for choosing which two words from the first input operand and which two words from the second are to be put into the result word.

View of Original and Result Words with Shuffle Function Macro
Macro Functions to Read and Write Control Registers

The following macro functions enable you to read and write bits to and from the control register. For Itanium®-based systems, these macros do not allow you to access all of the bits of the FPSR. Check the See Also section below for links to related topics.

<table>
<thead>
<tr>
<th>Exception State Macros</th>
<th>Macro Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_SET_EXCEPTION_STATE(x)</td>
<td>_MM_EXCEPT_INVALID</td>
</tr>
<tr>
<td>_MM_GET_EXCEPTION_STATE()</td>
<td>_MM_EXCEPT_DIV_ZERO</td>
</tr>
<tr>
<td></td>
<td>_MM_EXCEPT_DENORM</td>
</tr>
</tbody>
</table>

**Macro Definitions**

Write to and read from the six least significant control register bits, respectively.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_EXCEPT_OVERFLOW</td>
<td></td>
</tr>
<tr>
<td>_MM_EXCEPT_UNDERFLOW</td>
<td></td>
</tr>
<tr>
<td>_MM_EXCEPT_INEXACT</td>
<td></td>
</tr>
</tbody>
</table>

The following example tests for a divide-by-zero exception.

**Exception State Macros with _MM_EXCEPT_DIV_ZERO**
Exception Mask Macros

- **_MM_SET_EXCEPTION_MASK(x)**
  - **_MM_MASK_INVALID**
- **_MM_GET_EXCEPTION_MASK()**
  - **_MM_MASK_DIV_ZERO**
  - **_MM_MASK_DENORM**
  - **_MM_MASK_OVERFLOW**

**Macro Definitions** Write to and read from the seventh through twelfth control register bits, respectively.

**NOTE.** All six exception mask bits are always affected. Bits not set explicitly are cleared.

- **_MM_MASK_UNDERFLOW**
- **_MM_MASK_INEXACT**

To mask the overflow and underflow exceptions and unmask all other exceptions, use the macros as follows:

- **_MM_SET_EXCEPTION_MASK(_MM_MASK_OVERFLOW | _MM_MASK_UNDERFLOW)**

The following table lists the macros to set and get rounding modes, and the macro arguments that can be passed with the macros.

<table>
<thead>
<tr>
<th>Rounding Mode</th>
<th>Macro Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>_MM_SET_ROUNDING_MODE(x)</strong></td>
<td><strong>_MM_ROUND_NEAREST</strong></td>
</tr>
</tbody>
</table>
**Rounding Mode**

- `_MM_GET_ROUNDING_MODE()`
- `_MM_ROUND_DOWN`
- `_MM_ROUND_UP`
- `_MM_ROUND_TOWARD_ZERO`

**Macro Definition** Write to and read from bits thirteen and fourteen of the control register.

To test the rounding mode for round toward zero, use the `_MM_ROUND_TOWARD_ZERO` macro as follows.

```c
if (_MM_GET_ROUNDING_MODE() == _MM_ROUND_TOWARD_ZERO) {
    /* Rounding mode is round toward zero */
}
```

The following table lists the macros to set and get the flush-to-zero mode and the macro arguments that can be used.

<table>
<thead>
<tr>
<th>Flush-to-Zero Mode</th>
<th>Macro Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_MM_SET_FLUSH_ZERO_MODE(x)</code></td>
<td><code>_MM_FLUSH_ZERO_ON</code></td>
</tr>
<tr>
<td><code>_MM_GET_FLUSH_ZERO_MODE()</code></td>
<td><code>_MM_FLUSH_ZERO_OFF</code></td>
</tr>
</tbody>
</table>

**Macro Definition** Write to and read from bit fifteen of the control register.

To disable the flush-to-zero mode, use the `_MM_FLUSH_ZERO_OFF` macro.

```c
_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_OFF)
```

**See Also**

- Macro Functions
- Intrinsics to Read and Write Registers
- Native Intrinsics for IA-64 Instructions
Macro Function for Matrix Transposition

The Intel® Streaming SIMD Extensions (Intel® SSE) provide the following macro function to transpose a 4 by 4 matrix of single precision floating point values.

_MM_TRANSPOSE4_PS(row0, row1, row2, row3)

The arguments row0, row1, row2, and row3 are _m128 values whose elements form the corresponding rows of a 4 by 4 matrix. The matrix transposition is returned in arguments row0, row1, row2, and row3 where row0 now holds column 0 of the original matrix, row1 now holds column 1 of the original matrix, and so on.

The transposition function of this macro is illustrated in the "Matrix Transposition Using the _MM_TRANSPOSE4_PS" figure.

Matrix Transposition Using _MM_TRANSPOSE4_PS Macro
Intrinsics for Intel(R) Streaming SIMD Extensions 2

Overview: Intel(R) Streaming SIMD Extensions 2

This section describes the C++ language-level features supporting the Intel® Streaming SIMD Extensions 2 (Intel® SSE2) in the Intel® C++ Compiler. The features are divided into two categories:

• Floating-Point Intrinsics -- describes the arithmetic, logical, compare, conversion, memory, and initialization intrinsics for the double-precision floating-point data type (__m128d).
• Integer Intrinsics -- describes the arithmetic, logical, compare, conversion, memory, and initialization intrinsics for the extended-precision integer data type (__m128i).

1. There are no intrinsics for floating-point move operations. To move data from one register to another, a simple assignment, \( A = B \), suffices, where \( A \) and \( B \) are the source and target registers for the move operation.

2. On processors that do not support Intel® SSE2 instructions but do support MMX™ Technology, you can use the sse2mmx.h emulation pack to enable support for Intel® SSE2 instructions.

NOTE. Use the sse2mmx.h header file for the following processors:

• Itanium® processor
• Pentium® III processor
• Pentium® II processor
• Pentium® processor with MMX™ technology

Some intrinsics are "composites" - they require more than one instruction to implement them. Intrinsics that require one instruction to implement them are referred to as "simple".

You should be familiar with the hardware features provided by the Intel® SSE2 when writing programs with the intrinsics. The following are three important issues to keep in mind:

• Certain intrinsics, such as __mm_loadr_pd and __mm_cmpgt_sd, are not directly supported by the instruction set. While these intrinsics are convenient programming aids, be mindful of their implementation cost.
Data loaded or stored as \_m128d objects must be generally 16-byte-aligned.

Some intrinsics require that their argument be immediates, that is, constant integers (literals), due to the nature of the instruction.

The prototypes for Intel® SSE intrinsics are in the emmintrin.h header file.

**NOTE.** You can also use the single ia32intrin.h header file for any IA-32 architecture-based intrinsics.

### Floating-point Intrinsics

#### Arithmetic Intrinsics

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for floating-point arithmetic operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the emmintrin.h header file.

The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by R0 and R1, where R0 and R1 each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_sd</td>
<td>Addition</td>
<td>ADDSD</td>
</tr>
<tr>
<td>_mm_add_pd</td>
<td>Addition</td>
<td>ADDPD</td>
</tr>
<tr>
<td>_mm_sub_sd</td>
<td>Subtraction</td>
<td>SUBSD</td>
</tr>
<tr>
<td>_mm_sub_pd</td>
<td>Subtraction</td>
<td>SUBPD</td>
</tr>
<tr>
<td>_mm_mul_sd</td>
<td>Multiplication</td>
<td>MULSD</td>
</tr>
<tr>
<td>_mm_mul_pd</td>
<td>Multiplication</td>
<td>MULPD</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_div_sd</td>
<td>Division</td>
<td>DIVSD</td>
</tr>
<tr>
<td>_mm_div_pd</td>
<td>Division</td>
<td>DIVPD</td>
</tr>
<tr>
<td>_mm_sqrt_sd</td>
<td>Computes Square Root</td>
<td>SQRTSD</td>
</tr>
<tr>
<td>_mm_sqrt_pd</td>
<td>Computes Square Root</td>
<td>SQRTPD</td>
</tr>
<tr>
<td>_mm_min_sd</td>
<td>Computes Minimum</td>
<td>MINSD</td>
</tr>
<tr>
<td>_mm_min_pd</td>
<td>Computes Minimum</td>
<td>MINPD</td>
</tr>
<tr>
<td>_mm_max_sd</td>
<td>Computes Maximum</td>
<td>MAXSD</td>
</tr>
<tr>
<td>_mm_max_pd</td>
<td>Computes Maximum</td>
<td>MAXPD</td>
</tr>
</tbody>
</table>

__m128d _mm_add_sd(__m128d a, __m128d b)

Adds the lower DP FP (double-precision, floating-point) values of a and b; the upper DP FP value is passed through from a.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1</td>
</tr>
</tbody>
</table>

__m128d _mm_add_pd(__m128d a, __m128d b)

Adds the two DP FP values of a and b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1 + b1</td>
</tr>
</tbody>
</table>

__m128d _mm_sub_sd(__m128d a, __m128d b)

Subtracts the lower DP FP value of b from a. The upper DP FP value is passed through from a.
__m128d _mm_sub_pd(__m128d a, __m128d b)
Subtracts the two DP FP values of b from a.

__m128d _mm_mul_sd(__m128d a, __m128d b)
Multiplies the lower DP FP values of a and b. The upper DP FP is passed through from a.

__m128d _mm_mul_pd(__m128d a, __m128d b)
Multiplies the two DP FP values of a and b.

__m128d _mm_div_sd(__m128d a, __m128d b)
Divides the lower DP FP values of a and b. The upper DP FP value is passed through from a.

__m128d _mm_div_pd(__m128d a, __m128d b)
Divides the two DP FP values of a and b.
\[ \text{\_m128d \_mm_sqrt_sd(__m128d } a, \text{__m128d } b) \]

Computes the square root of the lower DP FP value of \( b \). The upper DP FP value is passed through from \( a \).

\[ \text{\_m128d \_mm_sqrt_pd(__m128d } a) \]

Computes the square roots of the two DP FP values of \( a \).

\[ \text{\_m128d \_mm_min_sd(__m128d } a, \text{__m128d } b) \]

Computes the minimum of the lower DP FP values of \( a \) and \( b \). The upper DP FP value is passed through from \( a \).

\[ \text{\_m128d \_mm_min_pd(__m128d } a, \text{__m128d } b) \]

Computes the minima of the two DP FP values of \( a \) and \( b \).

\[ \text{\_m128d \_mm_max_sd(__m128d } a, \text{__m128d } b) \]

Computes the maximum of the lower DP FP values of \( a \) and \( b \). The upper DP FP value is passed through from \( a \).
Computes the maxima of the two DP FP values of \( a \) and \( b \).

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_and_pd</td>
<td>Computes AND</td>
<td>ANDPD</td>
</tr>
<tr>
<td>_mm_andnot_pd</td>
<td>Computes AND and NOT</td>
<td>ANDNPD</td>
</tr>
<tr>
<td>_mm_or_pd</td>
<td>Computes OR</td>
<td>ORPD</td>
</tr>
<tr>
<td>_mm_xor_pd</td>
<td>Computes XOR</td>
<td>XORPD</td>
</tr>
<tr>
<td>__m128d _mm_max_pd(__m128d a, __m128d b)</td>
<td>Computes the bitwise AND of the two DP FP values of ( a ) and ( b ).</td>
<td></td>
</tr>
</tbody>
</table>
__m128d _mm_andnot_pd(__m128d a, __m128d b)

Computes the bitwise AND of the 128-bit value in \( b \) and the bitwise NOT of the 128-bit value in \( a \).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(~a0) &amp; b0</td>
<td>(~a1) &amp; b1</td>
</tr>
</tbody>
</table>

__m128d _mm_or_pd(__m128d a, __m128d b)

Computes the bitwise OR of the two DP FP values of \( a \) and \( b \).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>b0</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
</tr>
</tbody>
</table>

__m128d _mm_xor_pd(__m128d a, __m128d b)

Computes the bitwise XOR of the two DP FP values of \( a \) and \( b \).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 ^ b0</td>
<td>a1 ^ b1</td>
</tr>
</tbody>
</table>

**Compare Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for floating-point comparison operations are listed in the following table. The prototypes for the Intel® SSE2 intrinsics are in the emmintrin.h header file.

Each comparison intrinsic performs a comparison of \( a \) and \( b \). For the packed form, the two DP FP values of \( a \) and \( b \) are compared, and a 128-bit mask is returned. For the scalar form, the lower DP FP values of \( a \) and \( b \) are compared, and a 64-bit mask is returned; the upper DP FP value is passed through from \( a \).

The mask is set to \( 0xffffffffffffffff \) for each element where the comparison is true, and set to \( 0x0 \) where the comparison is false. The \( r \) following the instruction name indicates that the operands to the instruction are reversed in the actual implementation.
The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by $R$, $R0$ and $R1$, where $R$, $R0$ and $R1$ each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpeq_pd</td>
<td>Equality</td>
<td>CMPEQPD</td>
</tr>
<tr>
<td>_mm_cmplt_pd</td>
<td>Less Than</td>
<td>CMPLTPD</td>
</tr>
<tr>
<td>_mm_cmple_pd</td>
<td>Less Than or Equal</td>
<td>CMPLEPD</td>
</tr>
<tr>
<td>_mm_cmpgt_pd</td>
<td>Greater Than</td>
<td>CMPLTPDr</td>
</tr>
<tr>
<td>_mm_cmpge_pd</td>
<td>Greater Than or Equal</td>
<td>CMPLEPDr</td>
</tr>
<tr>
<td>_mm_cmpord_pd</td>
<td>Ordered</td>
<td>CMPORDPD</td>
</tr>
<tr>
<td>_mm_cmpunord_pd</td>
<td>Unordered</td>
<td>CMPUNORDPD</td>
</tr>
<tr>
<td>_mm_cmpneq_pd</td>
<td>Inequality</td>
<td>CMFNEQPD</td>
</tr>
<tr>
<td>_mm_cmplnt_pd</td>
<td>Not Less Than</td>
<td>CMPLNTPD</td>
</tr>
<tr>
<td>_mm_cmplne_pd</td>
<td>Not Less Than or Equal</td>
<td>CMPLNEPD</td>
</tr>
<tr>
<td>_mm_cmpngt_pd</td>
<td>Not Greater Than</td>
<td>CMPLNTPD</td>
</tr>
<tr>
<td>_mm_cmplnge_pd</td>
<td>Not Greater Than or Equal</td>
<td>CMPLNEPDr</td>
</tr>
<tr>
<td>_mm_cmpeq_sd</td>
<td>Equality</td>
<td>CMPEQSD</td>
</tr>
<tr>
<td>_mm_cmplt_sd</td>
<td>Less Than</td>
<td>CMPLTSD</td>
</tr>
<tr>
<td>_mm_cmple_sd</td>
<td>Less Than or Equal</td>
<td>CMPLESD</td>
</tr>
<tr>
<td>_mm_cmpgt_sd</td>
<td>Greater Than</td>
<td>CMPLTSDr</td>
</tr>
<tr>
<td>_mm_cmpge_sd</td>
<td>Greater Than or Equal</td>
<td>CMPLESDr</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>_mm_cmpord_sd</td>
<td>Ordered</td>
<td>CMPORDSD</td>
</tr>
<tr>
<td>_mm_cmpunord_sd</td>
<td>Unordered</td>
<td>CMFUNORDSD</td>
</tr>
<tr>
<td>_mm_cmpneq_sd</td>
<td>Inequality</td>
<td>CMFNEQSD</td>
</tr>
<tr>
<td>_mm_cmpnlt_sd</td>
<td>Not Less Than</td>
<td>CMFNLTSDD</td>
</tr>
<tr>
<td>_mm_cmpnle_sd</td>
<td>Not Less Than or Equal</td>
<td>CMFNLTSDD</td>
</tr>
<tr>
<td>_mm_cmpngt_sd</td>
<td>Not Greater Than</td>
<td>CMFNLTSDDr</td>
</tr>
<tr>
<td>_mm_cmpnge_sd</td>
<td>Not Greater Than or Equal</td>
<td>CMFNLTSDDr</td>
</tr>
<tr>
<td>_mm_comieq_sd</td>
<td>Equality</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comilt_sd</td>
<td>Less Than</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comile_sd</td>
<td>Less Than or Equal</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comigt_sd</td>
<td>Greater Than</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comige_sd</td>
<td>Greater Than or Equal</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comineq_sd</td>
<td>Not Equal</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_ucomieq_sd</td>
<td>Equality</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomilt_sd</td>
<td>Less Than</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomile_sd</td>
<td>Less Than or Equal</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomigt_sd</td>
<td>Greater Than</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomige_sd</td>
<td>Greater Than or Equal</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomineq_sd</td>
<td>Not Equal</td>
<td>UCOMISD</td>
</tr>
</tbody>
</table>
**_m128d _mm_cmpeq_pd(_m128d a, _m128d b)_**

Compares the two DP FP values of `a` and `b` for equality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(a0 == b0) ? 0xffffffffffffffff : 0x0</code></td>
<td><code>(a1 == b1) ? 0xffffffffffffffff : 0x0</code></td>
</tr>
</tbody>
</table>

**_m128d _mm_cmplt_pd(_m128d a, _m128d b)_**

Compares the two DP FP values of `a` and `b` for `a` less than `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(a0 &lt; b0) ? 0xffffffffffffffff : 0x0</code></td>
<td><code>(a1 &lt; b1) ? 0xffffffffffffffff : 0x0</code></td>
</tr>
</tbody>
</table>

**_m128d _mm_cmple_pd(_m128d a, _m128d b)_**

Compares the two DP FP values of `a` and `b` for `a` less than or equal to `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(a0 &lt;= b0) ? 0xffffffffffffffff : 0x0</code></td>
<td><code>(a1 &lt;= b1) ? 0xffffffffffffffff : 0x0</code></td>
</tr>
</tbody>
</table>

**_m128d _mm_cmpgt_pd(_m128d a, _m128d b)_**

Compares the two DP FP values of `a` and `b` for `a` greater than `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(a0 &gt; b0) ? 0xffffffffffffffff : 0x0</code></td>
<td><code>(a1 &gt; b1) ? 0xffffffffffffffff : 0x0</code></td>
</tr>
</tbody>
</table>

**_m128d _mm_cmpge_pd(_m128d a, _m128d b)_**

Compares the two DP FP values of `a` and `b` for `a` greater than or equal to `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(a0 &gt;= b0) ? 0xffffffffffffffff : 0x0</code></td>
<td><code>(a1 &gt;= b1) ? 0xffffffffffffffff : 0x0</code></td>
</tr>
</tbody>
</table>
\_m128d \_mm\_cmpord\_pd(\_m128d a, \_m128d b)

Compares the two DP FP values of a and b for ordered.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 ord b0) ? 0xfffffffffffffffffffffff :</td>
<td>(a1 ord b1) ? 0xfffffffffffffffffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

\_m128d \_mm\_cmpunord\_pd(\_m128d a, \_m128d b)

Compares the two DP FP values of a and b for unordered.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 unord b0) ? 0xfffffffffffffffffffffff :</td>
<td>(a1 unord b1) ? 0xfffffffffffffffffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

\_m128d \_mm\_cmpneq\_pd(\_m128d a, \_m128d b)

Compares the two DP FP values of a and b for inequality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 != b0) ? 0xfffffffffffffffffffffff :</td>
<td>(a1 != b1) ? 0xfffffffffffffffffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

\_m128d \_mm\_cmpnlt\_pd(\_m128d a, \_m128d b)

Compares the two DP FP values of a and b for a not less than b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt; b0) ? 0xfffffffffffffffffffffff :</td>
<td>!(a1 &lt; b1) ? 0xfffffffffffffffffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

\_m128d \_mm\_cmpnle\_pd(\_m128d a, \_m128d b)

Compares the two DP FP values of a and b for a not less than or equal to b.
__m128d _mm_cmpngt_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for a not greater than b.

__m128d _mm_cmpnge_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for a not greater than or equal to b.

__m128d _mm_cmpeq_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for equality. The upper DP FP value is passed through from a.

__m128d _mm_cmplt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a less than b. The upper DP FP value is passed through from a.
__m128d _mm_cmple_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \( a \) and \( b \) for \( a \) less than or equal to \( b \). The upper DP FP value is passed through from \( a \).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
(a0 <= b0) ? 0xffffffffffffffff : 0x0 & a1 \\
\end{array}
\]

__m128d _mm_cmpgt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \( a \) and \( b \) for \( a \) greater than \( b \). The upper DP FP value is passed through from \( a \).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
(a0 > b0) ? 0xffffffffffffffff : 0x0 & a1 \\
\end{array}
\]

__m128d _mm_cmpge_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \( a \) and \( b \) for \( a \) greater than or equal to \( b \). The upper DP FP value is passed through from \( a \).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
(a0 >= b0) ? 0xffffffffffffffff : 0x0 & a1 \\
\end{array}
\]

__m128d _mm_cmpord_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \( a \) and \( b \) for ordered. The upper DP FP value is passed through from \( a \).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
(a0 ord b0) ? 0xffffffffffffffff : 0x0 & a1 \\
\end{array}
\]

__m128d _mm_cmpunord_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \( a \) and \( b \) for unordered. The upper DP FP value is passed through from \( a \).
(a0 unord b0) ? 0xffffffffffffffff : 0x0

__m128d _mm_cmpneq_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for inequality. The upper DP FP value is passed through from a.

(a0 != b0) ? 0xffffffffffffffff : 0x0

__m128d _mm_cmpnlt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a not less than b. The upper DP FP value is passed through from a.

!(a0 < b0) ? 0xffffffffffffffff : 0x0

__m128d _mm_cmpnle_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a not less than or equal to b. The upper DP FP value is passed through from a.

!(a0 <= b0) ? 0xffffffffffffffff : 0x0

__m128d _mm_cmpngt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a not greater than b. The upper DP FP value is passed through from a.
<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &gt; b0) ? 0xfffffffffffffffffffffff : a1</td>
<td>0x0</td>
</tr>
<tr>
<td>__m128d _mm_cmpnge_sd(__m128d a, __m128d b)</td>
<td></td>
</tr>
<tr>
<td><strong>Compares the lower DP FP value of</strong> a and b <strong>for</strong> a <strong>not</strong> greater than or equal to b. The upper DP FP value is passed through from a.</td>
<td></td>
</tr>
<tr>
<td>!(a0 &gt;= b0) ? 0xfffffffffffffffffffffff : a1</td>
<td>0x0</td>
</tr>
<tr>
<td>int _mm_comieq_sd(__m128d a, __m128d b)</td>
<td></td>
</tr>
<tr>
<td><strong>Compares the lower DP FP value of</strong> a and b <strong>for</strong> a <strong>equal to</strong> b. If a and b are equal, 1 is returned. Otherwise 0 is returned.</td>
<td></td>
</tr>
<tr>
<td>(a0 == b0) ? 0x1 : 0x0</td>
<td></td>
</tr>
<tr>
<td>int _mm_comilt_sd(__m128d a, __m128d b)</td>
<td></td>
</tr>
<tr>
<td><strong>Compares the lower DP FP value of</strong> a and b <strong>for</strong> a <strong>less than</strong> b. If a is less than b, 1 is returned. Otherwise 0 is returned.</td>
<td></td>
</tr>
<tr>
<td>(a0 &lt; b0) ? 0x1 : 0x0</td>
<td></td>
</tr>
<tr>
<td>int _mm_comile_sd(__m128d a, __m128d b)</td>
<td></td>
</tr>
<tr>
<td><strong>Compares the lower DP FP value of</strong> a and b <strong>for</strong> a <strong>less than or equal to</strong> b. If a is less than or equal to b, 1 is returned. Otherwise 0 is returned.</td>
<td></td>
</tr>
<tr>
<td>(a0 &lt;= b0) ? 0x1 : 0x0</td>
<td></td>
</tr>
</tbody>
</table>
int _mm_comigt_sd(__m128d a, __m128d b)
Comparisons the lower DP FP value of a and b for a greater than b. If a is greater than b are equal, 1 is returned. Otherwise 0 is returned.

R
(a0 > b0) ? 0x1 : 0x0

int _mm_comige_sd(__m128d a, __m128d b)
Comparisons the lower DP FP value of a and b for a greater than or equal to b. If a is greater than or equal to b, 1 is returned. Otherwise 0 is returned.

R
(a0 >= b0) ? 0x1 : 0x0

int _mm_comineq_sd(__m128d a, __m128d b)
Comparisons the lower DP FP value of a and b for a not equal to b. If a and b are not equal, 1 is returned. Otherwise 0 is returned.

R
(a0 != b0) ? 0x1 : 0x0

int _mm_ucomieq_sd(__m128d a, __m128d b)
Comparisons the lower DP FP value of a and b for a equal to b. If a and b are equal, 1 is returned. Otherwise 0 is returned.

R
(a0 == b0) ? 0x1 : 0x0

int _mm_ucomilt_sd(__m128d a, __m128d b)
Comparisons the lower DP FP value of a and b for a less than b. If a is less than b, 1 is returned. Otherwise 0 is returned.
R

(a0 < b0) ? 0x1 : 0x0

int _mm_ucomile_sd(__m128d a, __m128d b)
Comparates the lower DP FP value of a and b for a less than or equal to b. If a is less than or equal to b, 1 is returned. Otherwise 0 is returned.

R

(a0 <= b0) ? 0x1 : 0x0

int _mm_ucomigt_sd(__m128d a, __m128d b)
Comparates the lower DP FP value of a and b for a greater than b. If a is greater than b are equal, 1 is returned. Otherwise 0 is returned.

R

(a0 > b0) ? 0x1 : 0x0

int _mm_ucomige_sd(__m128d a, __m128d b)
Comparates the lower DP FP value of a and b for a greater than or equal to b. If a is greater than or equal to b, 1 is returned. Otherwise 0 is returned.

R

(a0 >= b0) ? 0x1 : 0x0

int _mm_ucomineq_sd(__m128d a, __m128d b)
Comparates the lower DP FP value of a and b for a not equal to b. If a and b are not equal, 1 is returned. Otherwise 0 is returned.

R

(a0 != b0) ? 0x1 : 0x0
Conversion Intrinsics

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for floating-point conversion operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the `emmintrin.h` header file.

Each conversion intrinsic takes one data type and performs a conversion to a different type. Some conversions, such as those performed by the `__mm_cvtpd_ps` intrinsic, result in a loss of precision. The rounding mode used in such cases is determined by the value in the MXCSR register. The default rounding mode is round-to-nearest.

NOTE. The rounding mode used by the C and C++ languages when performing a type conversion is to truncate. The `__mm_cvttpd_epi32` and `__mm_cvttsd_si32` intrinsics use the truncate rounding mode regardless of the mode specified by the MXCSR register.

The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by `R`, `R0`, `R1`, `R2`, and `R3`, where each represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__mm_cvtpd_ps</code></td>
<td>Convert DP FP to SP FP</td>
<td>CVTPD2PS</td>
</tr>
<tr>
<td><code>__mm_cvtps_pd</code></td>
<td>Convert from SP FP to DP FP</td>
<td>CVTPS2PD</td>
</tr>
<tr>
<td><code>__mm_cvtepi32_pd</code></td>
<td>Convert lower integer values to DP FP</td>
<td>CVTDQ2PD</td>
</tr>
<tr>
<td><code>__mm_cvtpd_epi32</code></td>
<td>Convert DP FP values to integer values</td>
<td>CVTPD2DQ</td>
</tr>
<tr>
<td><code>__mm_cvtsd_si32</code></td>
<td>Convert lower DP FP value to integer value</td>
<td>CVTSD2SI</td>
</tr>
<tr>
<td><code>__mm_cvtsd_ss</code></td>
<td>Convert lower DP FP value to SP FP</td>
<td>CVTSD2SS</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_cvtsi32_sd</td>
<td>Convert signed integer value to DP FP</td>
<td>CVTSI2SD</td>
</tr>
<tr>
<td>_mm_cvttss_sd</td>
<td>Convert lower SP FP value to DP FP</td>
<td>CVTSS2SD</td>
</tr>
<tr>
<td>_mm_cvttpd_epi32</td>
<td>Convert DP FP values to signed integers</td>
<td>CVTTPD2DQ</td>
</tr>
<tr>
<td>_mm_cvttsd_si32</td>
<td>Convert lower DP FP to signed integer</td>
<td>CVTTSD2SI</td>
</tr>
<tr>
<td>_mm_cvtpd_pi32</td>
<td>Convert two DP FP values to signed integer values</td>
<td>CVTPD2PI</td>
</tr>
<tr>
<td>_mm_cvtpd_pi32</td>
<td>Convert two DP FP values to signed integer values using truncate</td>
<td>CVTPD2PI</td>
</tr>
<tr>
<td>_mm_cvtpi32_pd</td>
<td>Convert two signed integer values to DP FP</td>
<td>CVTPI2PD</td>
</tr>
<tr>
<td>_mm_cvtsd_f64</td>
<td>Extract DP FP value from first vector element</td>
<td>None</td>
</tr>
</tbody>
</table>

```c
__m128 _mm_cvtpd_ps(__m128d a)
```

Converts the two DP FP values of `a` to SP FP values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(float) <code>a</code></td>
<td>(float) <code>a1</code></td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

```c
__m128d _mm_cvtps_pd(__m128 a)
```

Converts the lower two SP FP values of `a` to DP FP values.
__m128d _mm_cvtepi32_pd(__m128i a)

Converts the lower two signed 32-bit integer values of \( a \) to DP FP values.

__m128i _mm_cvtpd_epi32(__m128d a)

Converts the two DP FP values of \( a \) to 32-bit signed integer values.

int _mm_cvtsd_si32(__m128d a)

Converts the lower DP FP value of \( a \) to a 32-bit signed integer value.

__m128 _mm_cvtsd_ss(__m128 a, __m128d b)

Converts the lower DP FP value of \( b \) to an SP FP value. The upper SP FP values in \( a \) are passed through.

__m128d _mm_cvtsi32_sd(__m128d a, int b)

Converts the signed integer value in \( b \) to a DP FP value. The upper DP FP value in \( a \) is passed through.
Converts the lower SP FP value of b to a DP FP value. The upper value DP FP value in a is passed through.

Converts the two DP FP values of a to 32-bit signed integers using truncate.

Converts the lower DP FP value of a to a 32-bit signed integer using truncate.

Converts the two DP FP values of a to 32-bit signed integer values.

Converts the two DP FP values of a to 32-bit signed integer values using truncate.
Converts the two 32-bit signed integer values of a to DP FP values.

This intrinsic extracts a double precision floating point value from the first vector element of an __m128d. It does so in the most efficient manner possible in the context used. This intrinsic does not map to any specific SSE2 instruction.

**Load Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for floating-point load operations are listed in this topic. The prototypes for Intel® SSE2 intrinsics are in the emmintrin.h header file.

The load and set operations are similar in that both initialize __m128d data. However, the set operations take a double argument and are intended for initialization with constants, while the load operations take a double pointer argument and are intended to mimic the instructions for loading data from memory.

The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by R0 and R1, where R0 and R1 each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_load_pd</td>
<td>Loads two DP FP values</td>
<td>MOVAPD</td>
</tr>
<tr>
<td>_mm_load1_pd</td>
<td>Loads a single DP FP value, copying to both elements</td>
<td>MOVSD + shuffling</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_loadr_pd</td>
<td>Loads two DP FP values in reverse order</td>
<td>MOVAPD + shuffling</td>
</tr>
<tr>
<td>_mm_loadu_pd</td>
<td>Loads two DP FP values</td>
<td>MOVUPD</td>
</tr>
<tr>
<td>_mm_load_sd</td>
<td>Loads a DP FP value, sets upper DP FP to zero</td>
<td>MOVSD</td>
</tr>
<tr>
<td>_mm_loadh_pd</td>
<td>Loads a DP FP value as the upper DP FP value of the result</td>
<td>MOVHPD</td>
</tr>
<tr>
<td>_mm_loadl_pd</td>
<td>Loads a DP FP value as the lower DP FP value of the result</td>
<td>MOVLPD</td>
</tr>
</tbody>
</table>

__m128d _mm_load_pd(double const*dp)

Loads two DP FP values. The address p must be 16-byte aligned.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p[0]</td>
<td>p[1]</td>
</tr>
</tbody>
</table>

__m128d _mm_load1_pd(double const*dp)

Loads a single DP FP value, copying to both elements. The address p need not be 16-byte aligned.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>*p</td>
<td>*p</td>
</tr>
</tbody>
</table>

__m128d _mm_loadr_pd(double const*dp)

Loads two DP FP values in reverse order. The address p must be 16-byte aligned.
__m128d _mm_loadu_pd(double const*dp)

Loads two DP FP values. The address p need not be 16-byte aligned.

__m128d _mm_load_sd(double const*dp)

Loads a DP FP value. The upper DP FP is set to zero. The address p need not be 16-byte aligned.

__m128d _mm_loadh_pd(__m128d a, double const*dp)

Loads a DP FP value as the upper DP FP value of the result. The lower DP FP value is passed through from a. The address p need not be 16-byte aligned.

__m128d _mm_load1_pd(__m128d a, double const*dp)

Loads a DP FP value as the lower DP FP value of the result. The upper DP FP value is passed through from a. The address p need not be 16-byte aligned.
Set Intrinsics

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for floating-point set operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the emmintrin.h header file.

The load and set operations are similar in that both initialize __m128d data. However, the set operations take a double argument and are intended for initialization with constants, while the load operations take a double pointer argument and are intended to mimic the instructions for loading data from memory.

Some of the these intrinsics are composite intrinsics because they require more than one instruction to implement them.

The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by R0 and R1, where R0 and R1 each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_set_sd</td>
<td>Sets lower DP FP value to w and upper to zero</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set1_pd</td>
<td>Sets two DP FP values to w</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set_pd</td>
<td>Sets lower DP FP to x and upper to w</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_pd</td>
<td>Sets lower DP FP to w and upper to x</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setzero_pd</td>
<td>Sets two DP FP values to zero</td>
<td>XORPD</td>
</tr>
<tr>
<td>_mm_move_sd</td>
<td>Sets lower DP FP value to the lower DP FP value of b</td>
<td>MOVSD</td>
</tr>
</tbody>
</table>

__m128d _mm_set_sd(double w)

Sets the lower DP FP value to w and sets the upper DP FP value to zero.
### _mm_set1_pd(double w)_

Sets the 2 DP FP values to \( w \).

### _mm_set_pd(double w, double x)_

Sets the lower DP FP value to \( x \) and sets the upper DP FP value to \( w \).

### _mm_setr_pd(double w, double x)_

Sets the lower DP FP value to \( w \) and sets the upper DP FP value to \( x \). \( r0 := w \ r1 := x \)

### _mm_setzero_pd(void)_

Sets the 2 DP FP values to zero.

### _mm_move_sd(_mm128d a, _mm128d b)_

Sets the lower DP FP value to the lower DP FP value of \( b \). The upper DP FP value is passed through from \( a \).
**Store Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for floating-point store operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the `emmintrin.h` header file.

The store operations assign the initialized data to the address.

The detailed description of each intrinsic contains a table detailing the returns. In these tables, `dp[n]` is an access to the `n` element of the result.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_stream_pd</td>
<td>Store</td>
<td>MOVNTPD</td>
</tr>
<tr>
<td>_mm_store_sd</td>
<td>Stores lower DP FP value of <code>a</code></td>
<td>MOVSD</td>
</tr>
<tr>
<td>_mm_store1_pd</td>
<td>Stores lower DP FP value of <code>a</code> twice</td>
<td>MOVAPD + shuffling</td>
</tr>
<tr>
<td>_mm_store_pd</td>
<td>Stores two DP FP values</td>
<td>MOVAPD</td>
</tr>
<tr>
<td>_mm_storeu_pd</td>
<td>Stores two DP FP values</td>
<td>MOVUPD</td>
</tr>
<tr>
<td>_mm_storer_pd</td>
<td>Stores two DP FP values in reverse order</td>
<td>MOVAPD + shuffling</td>
</tr>
<tr>
<td>_mm_storeh_pd</td>
<td>Stores upper DP FP value of <code>a</code></td>
<td>MOVHPD</td>
</tr>
<tr>
<td>_mm_store1_pd</td>
<td>Stores lower DP FP value of <code>a</code></td>
<td>MOVLPD</td>
</tr>
</tbody>
</table>

```c
void _mm_store_sd(double *dp, __m128d a)
```
Stores the lower DP FP value of \( a \). The address \( dp \) need not be 16-byte aligned.

\[
*dp
\]

\begin{align*}
\text{a0}
\end{align*}

*dp

\begin{align*}
\text{void \_mm\_store1\_pd(double *dp, __m128d a)}
\end{align*}

Stores the lower DP FP value of \( a \) twice. The address \( dp \) must be 16-byte aligned.

\begin{align*}
 \text{dp[0]} & \quad \text{dp[1]} \\
\text{a0} & \quad \text{a0}
\end{align*}

*dp

\begin{align*}
\text{void \_mm\_store\_pd(double *dp, __m128d a)}
\end{align*}

Stores two DP FP values. The address \( dp \) must be 16-byte aligned.

\begin{align*}
 \text{dp[0]} & \quad \text{dp[1]} \\
\text{a0} & \quad \text{a1}
\end{align*}

*dp

\begin{align*}
\text{void \_mm\_storeu\_pd(double *dp, __m128d a)}
\end{align*}

Stores two DP FP values. The address \( dp \) need not be 16-byte aligned.

\begin{align*}
 \text{dp[0]} & \quad \text{dp[1]} \\
\text{a0} & \quad \text{a1}
\end{align*}

*dp

\begin{align*}
\text{void \_mm\_storer\_pd(double *dp, __m128d a)}
\end{align*}

Stores two DP FP values in reverse order. The address \( dp \) must be 16-byte aligned.

\begin{align*}
 \text{dp[0]} & \quad \text{dp[1]} \\
\text{a1} & \quad \text{a0}
\end{align*}

*dp

\begin{align*}
\text{void \_mm\_storeh\_pd(double *dp, __m128d a)}
\end{align*}

Stores the upper DP FP value of \( a \).
void _mm_storel_pd(double *dp, __m128d a)
Stores the lower DP FP value of a.

### Integer Intrinsics

#### Arithmetic Intrinsics

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer arithmetic operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0, R1...R15 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_epi8</td>
<td>Addition</td>
<td>PADDB</td>
</tr>
<tr>
<td>_mm_add_epi16</td>
<td>Addition</td>
<td>PADDW</td>
</tr>
<tr>
<td>_mm_add_epi32</td>
<td>Addition</td>
<td>PADDDD</td>
</tr>
<tr>
<td>_mm_add_epi64</td>
<td>Addition</td>
<td>PADDQ</td>
</tr>
<tr>
<td>_mm_add_epi64</td>
<td>Addition</td>
<td>PADDQ</td>
</tr>
<tr>
<td>_mm_add_epi8</td>
<td>Addition</td>
<td>PADDDB</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_adds_epi16</td>
<td>Addition</td>
<td>PADDSW</td>
</tr>
<tr>
<td>_mm_adds_epu8</td>
<td>Addition</td>
<td>PADDUSB</td>
</tr>
<tr>
<td>_mm_adds_epu16</td>
<td>Addition</td>
<td>PADDUSW</td>
</tr>
<tr>
<td>_mm_avg_epu8</td>
<td>Computes Average</td>
<td>PAVGB</td>
</tr>
<tr>
<td>_mm_avg_epu16</td>
<td>Computes Average</td>
<td>PAVGN</td>
</tr>
<tr>
<td>_mm_madd_epi16</td>
<td>Multiplication and Addition</td>
<td>PMADDWD</td>
</tr>
<tr>
<td>_mm_max_epi16</td>
<td>Computes Maxima</td>
<td>PMAXSW</td>
</tr>
<tr>
<td>_mm_max_epu8</td>
<td>Computes Maxima</td>
<td>PMAXUB</td>
</tr>
<tr>
<td>_mm_min_epi16</td>
<td>Computes Minima</td>
<td>PMINSW</td>
</tr>
<tr>
<td>_mm_min_epu8</td>
<td>Computes Minima</td>
<td>PMINUB</td>
</tr>
<tr>
<td>_mm_mulhi_epi16</td>
<td>Multiplication</td>
<td>PMULHW</td>
</tr>
<tr>
<td>_mm_mulhi_epu16</td>
<td>Multiplication</td>
<td>PMULHUW</td>
</tr>
<tr>
<td>_mm_mullo_epi16</td>
<td>Multiplication</td>
<td>PMULLW</td>
</tr>
<tr>
<td>_mm_mul_su32</td>
<td>Multiplication</td>
<td>PMULUDQ</td>
</tr>
<tr>
<td>_mm_mul_epu32</td>
<td>Multiplication</td>
<td>PMULUDQ</td>
</tr>
<tr>
<td>_mm_sad_epu8</td>
<td>Computes Difference/Add</td>
<td>FSADDBW</td>
</tr>
<tr>
<td>_mm_sub_epi8</td>
<td>Subtraction</td>
<td>FSUBBB</td>
</tr>
<tr>
<td>_mm_sub_epi16</td>
<td>Subtraction</td>
<td>FSUBBW</td>
</tr>
<tr>
<td>_mm_sub_epi32</td>
<td>Subtraction</td>
<td>FSUBBD</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_sub_si64</td>
<td>Subtraction</td>
<td>PSUBQ</td>
</tr>
<tr>
<td>_mm_sub_epi64</td>
<td>Subtraction</td>
<td>PSUBQ</td>
</tr>
<tr>
<td>_mm_subs_epi8</td>
<td>Subtraction</td>
<td>PSUBSB</td>
</tr>
<tr>
<td>_mm_subs_epi16</td>
<td>Subtraction</td>
<td>PSUBSW</td>
</tr>
<tr>
<td>_mm_subs_epi16</td>
<td>Subtraction</td>
<td>PSUBUSW</td>
</tr>
</tbody>
</table>

__m128i _mm_add_epi8(__m128i a, __m128i b)  
Adds the 16 signed or unsigned 8-bit integers in a to the 16 signed or unsigned 8-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1 + b1;</td>
<td>...</td>
<td>a15 + b15</td>
</tr>
</tbody>
</table>

__m128i _mm_add_epi16(__m128i a, __m128i b)  
Adds the 8 signed or unsigned 16-bit integers in a to the 8 signed or unsigned 16-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1 + b1</td>
<td>...</td>
<td>a7 + b7</td>
</tr>
</tbody>
</table>

__m128i _mm_add_epi32(__m128i a, __m128i b)  
Adds the 4 signed or unsigned 32-bit integers in a to the 4 signed or unsigned 32-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1 + b1</td>
<td>a2 + b2</td>
<td>a3 + b3</td>
</tr>
</tbody>
</table>
### _m64 _mm_add_si64(_m64 a, _m64 b)_

Adds the signed or unsigned 64-bit integer a to the signed or unsigned 64-bit integer b.

<table>
<thead>
<tr>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
</tr>
</tbody>
</table>

### _m128i _mm_add_epi64(_m128i a, _m128i b)_

Adds the 2 signed or unsigned 64-bit integers in a to the 2 signed or unsigned 64-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1 + b1</td>
</tr>
</tbody>
</table>

### _m128i _mm_adds_epi8(_m128i a, _m128i b)_

Adds the 16 signed 8-bit integers in a to the 16 signed 8-bit integers in b using saturating arithmetic.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignedSaturate</td>
<td>SignedSaturate</td>
<td>...</td>
<td>SignedSaturate</td>
</tr>
<tr>
<td>(a0 + b0)</td>
<td>(a1 + b1)</td>
<td></td>
<td>(a15 + b15)</td>
</tr>
</tbody>
</table>

### _m128i _mm_adds_epi16(_m128i a, _m128i b)_

Adds the 8 signed 16-bit integers in a to the 8 signed 16-bit integers in b using saturating arithmetic.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignedSaturate</td>
<td>SignedSaturate</td>
<td>...</td>
<td>SignedSaturate</td>
</tr>
<tr>
<td>(a0 + b0)</td>
<td>(a1 + b1)</td>
<td></td>
<td>(a7 + b7)</td>
</tr>
</tbody>
</table>

### _m128i _mm_adds_epu8(_m128i a, _m128i b)_

Adds the 16 unsigned 8-bit integers in a to the 16 unsigned 8-bit integers in b using saturating arithmetic.
<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnsignedSaturate</td>
<td>UnsignedSaturate</td>
<td>...</td>
<td>UnsignedSaturate</td>
</tr>
<tr>
<td>(a0 + b0)</td>
<td>(a1 + b1)</td>
<td>...</td>
<td>(a15 + b15)</td>
</tr>
</tbody>
</table>

__m128i _mm_adds_epu16(__m128i a, __m128i b)

Adds the 8 unsigned 16-bit integers in a to the 8 unsigned 16-bit integers in b using saturating arithmetic.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnsignedSaturate</td>
<td>UnsignedSaturate</td>
<td>...</td>
<td>UnsignedSaturate</td>
</tr>
<tr>
<td>(a0 + b0)</td>
<td>(a1 + b1)</td>
<td>...</td>
<td>(a7 + b7)</td>
</tr>
</tbody>
</table>

__m128i _mm_avg_epu8(__m128i a, __m128i b)

Computes the average of the 16 unsigned 8-bit integers in a and the 16 unsigned 8-bit integers in b and rounds.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 + b0) / 2</td>
<td>(a1 + b1) / 2</td>
<td>...</td>
<td>(a15 + b15) / 2</td>
</tr>
</tbody>
</table>

__m128i _mm_avg_epu16(__m128i a, __m128i b)

Computes the average of the 8 unsigned 16-bit integers in a and the 8 unsigned 16-bit integers in b and rounds.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 + b0) / 2</td>
<td>(a1 + b1) / 2</td>
<td>...</td>
<td>(a7 + b7) / 2</td>
</tr>
</tbody>
</table>

__m128i _mm_madd_epi16(__m128i a, __m128i b)

Multiplies the 8 signed 16-bit integers from a by the 8 signed 16-bit integers from b. Adds the signed 32-bit integer results pairwise and packs the 4 signed 32-bit integer results.


```c
__m128i _mm_max_epi16(__m128i a, __m128i b)
```

Computes the pairwise maxima of the 8 signed 16-bit integers from `a` and the 8 signed 16-bit integers from `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>max(a0, b0)</td>
<td>max(a1, b1)</td>
<td>...</td>
<td>max(a7, b7)</td>
</tr>
</tbody>
</table>

```c
__m128i _mm_max_epu8(__m128i a, __m128i b)
```

Computes the pairwise maxima of the 16 unsigned 8-bit integers from `a` and the 16 unsigned 8-bit integers from `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>max(a0, b0)</td>
<td>max(a1, b1)</td>
<td>...</td>
<td>max(a15, b15)</td>
</tr>
</tbody>
</table>

```c
__m128i _mm_min_epi16(__m128i a, __m128i b)
```

Computes the pairwise minima of the 8 signed 16-bit integers from `a` and the 8 signed 16-bit integers from `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>min(a0, b0)</td>
<td>min(a1, b1)</td>
<td>...</td>
<td>min(a7, b7)</td>
</tr>
</tbody>
</table>

```c
__m128i _mm_min_epu8(__m128i a, __m128i b)
```

Computes the pairwise minima of the 16 unsigned 8-bit integers from `a` and the 16 unsigned 8-bit integers from `b`.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>min(a0, b0)</td>
<td>min(a1, b1)</td>
<td>...</td>
<td>min(a15, b15)</td>
</tr>
</tbody>
</table>

```c
__m128i _mm_mulhi_epi16(__m128i a, __m128i b)
```

Multiplies the 8 signed 16-bit integers from `a` by the 8 signed 16-bit integers from `b`. Packs the upper 16-bits of the 8 signed 32-bit results.
<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 * b0)[31:16]</td>
<td>(a1 * b1)[31:16]</td>
<td>...</td>
<td>(a7 * b7)[31:16]</td>
</tr>
</tbody>
</table>

__m128i _mm_mulhi_epu16(__m128i a, __m128i b)

Multiplies the 8 unsigned 16-bit integers from \( a \) by the 8 unsigned 16-bit integers from \( b \). Packs the upper 16-bits of the 8 unsigned 32-bit results.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 * b0)[31:16]</td>
<td>(a1 * b1)[31:16]</td>
<td>...</td>
<td>(a7 * b7)[31:16]</td>
</tr>
</tbody>
</table>

__m128i _mm_mullo_epi16(__m128i a, __m128i b)

Multiplies the 8 signed or unsigned 16-bit integers from \( a \) by the 8 signed or unsigned 16-bit integers from \( b \). Packs the lower 16-bits of the 8 signed or unsigned 32-bit results.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 * b0)[15:0]</td>
<td>(a1 * b1)[15:0]</td>
<td>...</td>
<td>(a7 * b7)[15:0]</td>
</tr>
</tbody>
</table>

__m64 _mm_mul_su32(__m64 a, __m64 b)

Multiplies the lower 32-bit integer from \( a \) by the lower 32-bit integer from \( b \), and returns the 64-bit integer result.

<table>
<thead>
<tr>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 * b0</td>
</tr>
</tbody>
</table>

__m128i _mm_mul_epu32(__m128i a, __m128i b)

Multiplies 2 unsigned 32-bit integers from \( a \) by 2 unsigned 32-bit integers from \( b \). Packs the 2 unsigned 64-bit integer results.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 * b0</td>
<td>a2 * b2</td>
</tr>
</tbody>
</table>

__m128i _mm_sad_epu8(__m128i a, __m128i b)
Computes the absolute difference of the 16 unsigned 8-bit integers from \( a \) and the 16 unsigned 8-bit integers from \( b \). Sums the upper 8 differences and lower 8 differences, and packs the resulting 2 unsigned 16-bit integers into the upper and lower 64-bit elements.

\[
\begin{array}{c|c|c|c}
\text{R0} & \text{R1 to R3} & \text{R4} & \text{R5 to R7} \\
\hline
\text{abs}(a0 - b0) + 0x0 & \text{abs}(a8 - b8) + 0x0 \\
\text{abs}(a1 - b1) & \text{abs}(a9 - b9) \\
\cdots + \text{abs}(a7 - b7) & \cdots + \text{abs}(a15 - b15) \\
\hline
\end{array}
\]

__m128i _mm_sub_epi8(__m128i a, __m128i b)

Subtracts the 16 signed or unsigned 8-bit integers of \( b \) from the 16 signed or unsigned 8-bit integers of \( a \).

\[
\begin{array}{c|c|c|c}
\text{R0} & \text{R1} & \cdots & \text{R15} \\
\hline
a0 - b0 & a1 - b1 & \cdots & a15 - b15 \\
\hline
\end{array}
\]

__m128i _mm_sub_epi16(__m128i a, __m128i b)

Subtracts the 8 signed or unsigned 16-bit integers of \( b \) from the 8 signed or unsigned 16-bit integers of \( a \).

\[
\begin{array}{c|c|c|c}
\text{R0} & \text{R1} & \cdots & \text{R7} \\
\hline
a0 - b0 & a1 - b1 & \cdots & a7 - b7 \\
\hline
\end{array}
\]

__m128i _mm_sub_epi32(__m128i a, __m128i b)

Subtracts the 4 signed or unsigned 32-bit integers of \( b \) from the 4 signed or unsigned 32-bit integers of \( a \).

\[
\begin{array}{c|c|c|c}
\text{R0} & \text{R1} & \text{R2} & \text{R3} \\
\hline
a0 - b0 & a1 - b1 & a2 - b2 & a3 - b3 \\
\hline
\end{array}
\]

__m64 _mm_sub_epi64 (__m64 a, __m64 b)

Subtracts the signed or unsigned 64-bit integer \( b \) from the signed or unsigned 64-bit integer \( a \).
\[ a - b \]

\[
\text{\_m128i } \text{\_mm_sub_epi64(\_m128i a, \_m128i b)}
\]

Subtracts the 2 signed or unsigned 64-bit integers in \( b \) from the 2 signed or unsigned 64-bit integers in \( a \).

| R0 | R1 | ...
| --- | --- | ---
| a0 - b0 | a1 - b1 | 

\[
\text{\_m128i } \text{\_mm_subs_epi8(\_m128i a, \_m128i b)}
\]

Subtracts the 16 signed 8-bit integers of \( b \) from the 16 signed 8-bit integers of \( a \) using saturating arithmetic.

| R0 | R1 | ...
| --- | --- | ---
| SignedSaturate (a0 - b0) | SignedSaturate (a1 - b1) | SignedSaturate (a15 - b15) |

\[
\text{\_m128i } \text{\_mm_subs_epi16(\_m128i a, \_m128i b)}
\]

Subtracts the 8 signed 16-bit integers of \( b \) from the 8 signed 16-bit integers of \( a \) using saturating arithmetic.

| R0 | R1 | ...
| --- | --- | ---
| SignedSaturate (a0 - b0) | SignedSaturate (a1 - b1) | SignedSaturate (a7 - b7) |

\[
\text{\_m128i } \text{\_mm_subs_epu8 (\_m128i a, \_m128i b)}
\]

Subtracts the 16 unsigned 8-bit integers of \( b \) from the 16 unsigned 8-bit integers of \( a \) using saturating arithmetic.

| R0 | R1 | ...
| --- | --- | ---
| UnsignedSaturate (a0 - b0) | UnsignedSaturate (a1 - b1) | UnsignedSaturate (a15 - b15) |
Subtracts the 8 unsigned 16-bit integers of \( \mathbf{b} \) from the 8 unsigned 16-bit integers of \( \mathbf{a} \) using saturating arithmetic.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnsignedSaturate ((a_0 - b_0))</td>
<td>UnsignedSaturate ((a_1 - b_1))</td>
<td>...</td>
<td>UnsignedSaturate ((a_7 - b_7))</td>
</tr>
</tbody>
</table>

**Logical Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer logical operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in register \( R \). The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_and_si128</td>
<td>Computes AND</td>
<td>PAND</td>
</tr>
<tr>
<td>_mm_andnot_si128</td>
<td>Computes AND and NOT</td>
<td>PANDN</td>
</tr>
<tr>
<td>_mm_or_si128</td>
<td>Computes OR</td>
<td>POR</td>
</tr>
<tr>
<td>_mm_xor_si128</td>
<td>Computes XOR</td>
<td>PXOR</td>
</tr>
</tbody>
</table>

\_m128i \_mm_and_si128\((\_m128i \text{a, } \_m128i \text{b})\)

Computes the bitwise AND of the 128-bit value in \( \text{a} \) and the 128-bit value in \( \text{b} \).

\_m128i \_mm_or_si128\((\_m128i \text{a, } \_m128i \text{b})\)

Computes the bitwise AND of the 128-bit value in \( \text{a} \) and the bitwise NOT of the 128-bit value in \( \text{b} \).
R0

\((\neg a) \& b\)

\_m128i _mm_or_si128(_m128i a, _m128i b)

Computes the bitwise OR of the 128-bit value in \(a\) and the 128-bit value in \(b\).

R0

\(a \mid b\)

\_m128i _mm_xor_si128(_m128i a, _m128i b)

Computes the bitwise XOR of the 128-bit value in \(a\) and the 128-bit value in \(b\).

R0

\(a ^ b\)

Shift Intrinsics

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer shift operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. \(R, R0, R1...R7\) represent the registers in which results are placed.

**NOTE.** The `count` argument is one shift count that applies to all elements of the operand being shifted. It is not a vector shift count that shifts each element by a different amount.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Shift Type</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_slli_si128</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLDQ</td>
</tr>
</tbody>
</table>

1587
<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Shift Type</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_slli_epi16</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLW</td>
</tr>
<tr>
<td>_mm_sll_epi16</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLW</td>
</tr>
<tr>
<td>_mm_slli_epi32</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLLD</td>
</tr>
<tr>
<td>_mm_sll_epi32</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLLD</td>
</tr>
<tr>
<td>_mm_slli_epi64</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLQ</td>
</tr>
<tr>
<td>_mm_sll_epi64</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLQ</td>
</tr>
<tr>
<td>_mm_srai_epi16</td>
<td>Shift right</td>
<td>Arithmetic</td>
<td>PSRAW</td>
</tr>
<tr>
<td>_mm_sra_epi16</td>
<td>Shift right</td>
<td>Arithmetic</td>
<td>PSRAW</td>
</tr>
<tr>
<td>_mm_srai_epi32</td>
<td>Shift right</td>
<td>Arithmetic</td>
<td>PSRAD</td>
</tr>
<tr>
<td>_mm_sra_epi32</td>
<td>Shift right</td>
<td>Arithmetic</td>
<td>PSRAD</td>
</tr>
<tr>
<td>_mm_srl_epi16</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLDQ</td>
</tr>
<tr>
<td>_mm_srl_epi16</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLDQ</td>
</tr>
<tr>
<td>_mm_srl_epi32</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLDQ</td>
</tr>
<tr>
<td>_mm_srl_epi32</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLDQ</td>
</tr>
<tr>
<td>_mm_srl_epi64</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLQ</td>
</tr>
<tr>
<td>_mm_srl_epi64</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLQ</td>
</tr>
</tbody>
</table>

__m128i _mm_slli_si128(__m128i a, int imm)

Shifts the 128-bit value in a left by imm bytes while shifting in zeros. imm must be an immediate.
\( a << (\text{imm} \times 8) \)

\[
__m128i \ _mm_slli_epi16(__m128i \ a, \ \text{int} \ \text{count})
\]

Shifts the 8 signed or unsigned 16-bit integers in \( a \) left by \( \text{count} \) bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
\hline
a0 \ll \text{count} & a1 \ll \text{count} & \ldots & a7 \ll \text{count} \\
\hline
\end{array}
\]

\[
__m128i \ _mm_slli_epi16(__m128i \ a, \ __m128i \ \text{count})
\]

Shifts the 8 signed or unsigned 16-bit integers in \( a \) left by \( \text{count} \) bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
\hline
a0 \ll \text{count} & a1 \ll \text{count} & \ldots & a7 \ll \text{count} \\
\hline
\end{array}
\]

\[
__m128i \ _mm_sll_epi32(__m128i \ a, \ \text{int} \ \text{count})
\]

Shifts the 4 signed or unsigned 32-bit integers in \( a \) left by \( \text{count} \) bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R3 \\
\hline
a0 \ll \text{count} & a1 \ll \text{count} & a2 \ll \text{count} & a3 \ll \text{count} \\
\hline
\end{array}
\]

\[
__m128i \ _mm_sll_epi32(__m128i \ a, \ __m128i \ \text{count})
\]

Shifts the 4 signed or unsigned 32-bit integers in \( a \) left by \( \text{count} \) bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R3 \\
\hline
a0 \ll \text{count} & a1 \ll \text{count} & a2 \ll \text{count} & a3 \ll \text{count} \\
\hline
\end{array}
\]

\[
__m128i \ _mm_sll_epi64(__m128i \ a, \ \text{int} \ \text{count})
\]

Shifts the 2 signed or unsigned 64-bit integers in \( a \) left by \( \text{count} \) bits while shifting in zeros.
__m128i _mm_sll_epi64(__m128i a, __m128i count)

Shifts the 2 signed or unsigned 64-bit integers in `a` left by `count` bits while shifting in zeros.

__m128i _mm_srai_epi16(__m128i a, int count)

Shifts the 8 signed 16-bit integers in `a` right by `count` bits while shifting in the sign bit.

__m128i _mm_sra_epi16(__m128i a, __m128i count)

Shifts the 8 signed 16-bit integers in `a` right by `count` bits while shifting in the sign bit.

__m128i _mm_sra_epi32(__m128i a, int count)

Shifts the 4 signed 32-bit integers in `a` right by `count` bits while shifting in the sign bit.

__m128i _mm_sra_epi32(__m128i a, __m128i count)

Shifts the 4 signed 32-bit integers in `a` right by `count` bits while shifting in the sign bit.
\texttt{\_mm\_srl\_si128(__m128i a, int imm)}

Shifts the 128-bit value in \texttt{a} right by \texttt{imm} bytes while shifting in zeros. \texttt{imm} must be an immediate.

\texttt{srl(a, imm*8)}

\texttt{\_mm\_srl\_epi16(__m128i a, int count)}

Shifts the 8 signed or unsigned 16-bit integers in \texttt{a} right by \texttt{count} bits while shifting in zeros.

\texttt{\_mm\_srl\_epi32(__m128i a, __m128i count)}

Shifts the 4 signed or unsigned 32-bit integers in \texttt{a} right by \texttt{count} bits while shifting in zeros.
Shifts the 2 signed or unsigned 64-bit integers in a right by count bits while shifting in zeros.

**Compare Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer comparison operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the emmintrin.h header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0, R1...R15 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpeq_epi8</td>
<td>Equality</td>
<td>PCMPEQQB</td>
</tr>
<tr>
<td>_mm_cmpeq_epi16</td>
<td>Equality</td>
<td>PCMPEQW</td>
</tr>
<tr>
<td>_mm_cmpeq_epi32</td>
<td>Equality</td>
<td>PCMPEQD</td>
</tr>
<tr>
<td>_mm_cmpgt_epi8</td>
<td>Greater Than</td>
<td>PCMPGTB</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_cmpgt_epi16</td>
<td>Greater Than</td>
<td>PCMPGTW</td>
</tr>
<tr>
<td>_mm_cmpgt_epi32</td>
<td>Greater Than</td>
<td>PCMPGTD</td>
</tr>
<tr>
<td>_mm_cmplt_epi8</td>
<td>Less Than</td>
<td>PCMPGTBr</td>
</tr>
<tr>
<td>_mm_cmplt_epi16</td>
<td>Less Than</td>
<td>PCMPGTWr</td>
</tr>
<tr>
<td>_mm_cmplt_epi32</td>
<td>Less Than</td>
<td>PCMPGTDr</td>
</tr>
</tbody>
</table>

__m128i _mm_cmpeq_epi8(__m128i a, __m128i b)\n
Compares the 16 signed or unsigned 8-bit integers in a and the 16 signed or unsigned 8-bit integers in b for equality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ? 0xff</td>
<td>(a1 == b1) ? 0xff</td>
<td>...</td>
<td>(a15 == b15) ?</td>
</tr>
<tr>
<td>: 0x0</td>
<td>: 0x0</td>
<td></td>
<td>: 0xff : 0x0</td>
</tr>
</tbody>
</table>

__m128i _mm_cmpeq_epi16(__m128i a, __m128i b)\n
Compares the 8 signed or unsigned 16-bit integers in a and the 8 signed or unsigned 16-bit integers in b for equality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ?</td>
<td>(a1 == b1) ?</td>
<td>...</td>
<td>(a7 == b7) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td></td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128i _mm_cmpeq_epi32(__m128i a, __m128i b)\n
Compares the 4 signed or unsigned 32-bit integers in a and the 4 signed or unsigned 32-bit integers in b for equality.
<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ?</td>
<td>(a1 == b1) ?</td>
<td>(a2 == b2) ?</td>
<td>(a3 == b3) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128i _mm_cmpgt_epi8(__m128i a, __m128i b)

Compares the 16 signed 8-bit integers in a and the 16 signed 8-bit integers in b for greater than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &gt; b0) ? 0xff</td>
<td>(a1 &gt; b1) ? 0xff</td>
<td>...</td>
<td>(a15 &gt; b15) ?</td>
</tr>
<tr>
<td>: 0x0</td>
<td>: 0x0</td>
<td></td>
<td>: 0xff : 0x0</td>
</tr>
</tbody>
</table>

__m128i _mm_cmpgt_epi16(__m128i a, __m128i b)

Compares the 8 signed 16-bit integers in a and the 8 signed 16-bit integers in b for greater than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &gt; b0) ?</td>
<td>(a1 &gt; b1) ?</td>
<td>...</td>
<td>(a7 &gt; b7) ?</td>
</tr>
<tr>
<td>0xffff : 0x0</td>
<td>0xffff : 0x0</td>
<td></td>
<td>0xffff : 0x0</td>
</tr>
</tbody>
</table>

__m128i _mm_cmpgt_epi32(__m128i a, __m128i b)

Compares the 4 signed 32-bit integers in a and the 4 signed 32-bit integers in b for greater than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &gt; b0) ?</td>
<td>(a1 &gt; b1) ?</td>
<td>(a2 &gt; b2) ?</td>
<td>(a3 &gt; b3) ?</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128i _mm_cmplt_epi8(__m128i a, __m128i b)

Compares the 16 signed 8-bit integers in a and the 16 signed 8-bit integers in b for less than.
### __m128i __mm_cmplt_epi16( __m128i a, __m128i b)###

Compares the 8 signed 16-bit integers in a and the 8 signed 16-bit integers in b for less than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt; b0) ? 0xff</td>
<td>(a1 &lt; b1) ? 0xff</td>
<td>...</td>
<td>(a15 &lt; b15) ? 0xff</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>...</td>
<td>0x0</td>
</tr>
</tbody>
</table>

### __m128i __mm_cmplt_epi32( __m128i a, __m128i b)###

Compares the 4 signed 32-bit integers in a and the 4 signed 32-bit integers in b for less than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt; b0) ? 0xffffffff</td>
<td>(a1 &lt; b1) ? 0xffffffff</td>
<td>...</td>
<td>(a7 &lt; b7) ? 0xffffffff</td>
</tr>
<tr>
<td>0xffffffff : 0x0</td>
<td>0xffffffff : 0x0</td>
<td>...</td>
<td>0xffffffff : 0x0</td>
</tr>
</tbody>
</table>

### Conversion Intrinsics###

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer conversion operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0, R1, R2 and R3 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__mm_cvtsi64_sd</td>
<td>Convert and pass through</td>
<td>CVTSI2SD</td>
</tr>
<tr>
<td>__mm_cvtss_sd</td>
<td>Convert according to rounding</td>
<td>CVTSD2SI</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_cvttsd_si64</td>
<td>Convert using truncation</td>
<td>CVTTSD2SI</td>
</tr>
<tr>
<td>_mm_cvtepi32_ps</td>
<td>Convert to SP FP</td>
<td>None</td>
</tr>
<tr>
<td>_mm_cvtps_epi32</td>
<td>Convert from SP FP</td>
<td>None</td>
</tr>
<tr>
<td>_mm_cvttps_epi32</td>
<td>Convert from SP FP using</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>truncate</td>
<td></td>
</tr>
</tbody>
</table>

__m128d _mm_cvtsi64_sd(__m128d a, __int64 b)\n
Converts the signed 64-bit integer value in b to a DP FP value. The upper DP FP value in a is passed through.

**R0**

| (double) b | a1 |

__int64 _mm_cvtsd_si64(__m128d a)\n
Converts the lower DP FP value of a to a 64-bit signed integer value according to the current rounding mode.

**R**

| (__int64) a0 |

__int64 _mm_cvtttsd_si64(__m128d a)\n
Converts the lower DP FP value of a to a 64-bit signed integer value using truncation.

**R**

| (__int64) a0 |

__m128 _mm_cvtepi32_ps(__m128i a)\n
Converts the 4 signed 32-bit integer values of a to SP FP values.
### _m128i _mm_cvtps_epi32(__m128 a)

Converts the 4 SP FP values of a to signed 32-bit integer values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(float) a0</td>
<td>(float) a1</td>
<td>(float) a2</td>
<td>(float) a3</td>
</tr>
</tbody>
</table>

### _m128i _mm_cvttps_epi32(__m128 a)

Converts the 4 SP FP values of a to signed 32 bit integer values using truncate.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(int) a0</td>
<td>(int) a1</td>
<td>(int) a2</td>
<td>(int) a3</td>
</tr>
</tbody>
</table>

### Move Intrinsics

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer move operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0, R1, R2 and R3 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cvtsi32_si128</td>
<td>Move and zero</td>
<td>MOVVD</td>
</tr>
<tr>
<td>_mm_cvtsi64_si128</td>
<td>Move and zero</td>
<td>MOVQ</td>
</tr>
<tr>
<td>_mm_cvtss2si32</td>
<td>Move lowest 32 bits</td>
<td>MOVDD</td>
</tr>
<tr>
<td>_mm_cvtsi128_si64</td>
<td>Move lowest 64 bits</td>
<td>MOVQ</td>
</tr>
</tbody>
</table>
__m128i _mm_cvtsi32_si128(int a)

Moves 32-bit integer a to the least significant 32 bits of an __m128i object. zeroes the upper 96 bits of the __m128i object.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

__m128i _mm_cvtsi64_si128(__int64 a)

Moves 64-bit integer a to the lower 64 bits of an __m128i object, zeroing the upper bits.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0x0</td>
</tr>
</tbody>
</table>

_int_mm_cvtsi128_si32(__m128i a)

Moves the least significant 32 bits of a to a 32-bit integer.

<table>
<thead>
<tr>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
</tr>
</tbody>
</table>

__int64 _mm_cvtsi128_si64(__m128i a)

Moves the lower 64 bits of a to a 64-bit integer.

<table>
<thead>
<tr>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
</tr>
</tbody>
</table>

**Load Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer load operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the emmintrin.h header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0 and R1 represent the registers in which results are placed.
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_load_si128</td>
<td>Load</td>
<td>MOVDQA</td>
</tr>
<tr>
<td>_mm_loadu_si128</td>
<td>Load</td>
<td>MOVDQU</td>
</tr>
<tr>
<td>_mm_loadl_epi64</td>
<td>Load and zero</td>
<td>MOVQ</td>
</tr>
</tbody>
</table>

---

__m128i _mm_load_si128(__m128i const*p)

Loads 128-bit value. Address p must be 16-byte aligned.

```
R
*p
```

__m128i _mm_loadu_si128(__m128i const*p)

Loads 128-bit value. Address p not need be 16-byte aligned.

```
R
*p
```

__m128i _mm_loadl_epi64(__m128i const*p)

Load the lower 64 bits of the value pointed to by p into the lower 64 bits of the result, zeroing the upper 64 bits of the result.

```
R0
*p[63:0]
R1
0x0
```

**Set Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer set operations are listed in this topic. These intrinsics are composite intrinsics because they require more than one instruction to implement them. The prototypes for the Intel® SSE2 intrinsics are in the emmintrin.h header file.
The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. \( R, R_0, R_1...R_{15} \) represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128i_mm_set_epi64</td>
<td>Set two integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_set_epi32</td>
<td>Set four integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_set_epi16</td>
<td>Set eight integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_set_epi8</td>
<td>Set sixteen integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_set1_epi64</td>
<td>Set two integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_set1_epi32</td>
<td>Set four integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_set1_epi16</td>
<td>Set eight integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_set1_epi8</td>
<td>Set sixteen integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_setr_epi64</td>
<td>Set two integer values in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_setr_epi32</td>
<td>Set four integer values in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_setr_epi16</td>
<td>Set eight integer values in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_setr_epi8</td>
<td>Set sixteen integer values in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>__m128i_mm_setzero_si128</td>
<td>Set to zero</td>
<td>Composite</td>
</tr>
</tbody>
</table>

_sets two 64-bit integer values.

_sets four 32-bit integer values.

_sets eight 16-bit integer values.

_sets sixteen 8-bit integer values.

_sets two integer values.

_sets four integer values.

_sets eight integer values.

_sets sixteen integer values.

_sets two integer values in reverse order.

_sets four integer values in reverse order.

_sets eight integer values in reverse order.

_sets sixteen integer values in reverse order.

_sets to zero.
<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>q1</td>
</tr>
</tbody>
</table>

__m128i _mm_set_epi32(int i3, int i2, int i1, int i0)

Sets the 4 signed 32-bit integer values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i0</td>
<td>i1</td>
<td>i2</td>
<td>i3</td>
</tr>
</tbody>
</table>

__m128i _mm_set_epi16(short w7, short w6, short w5, short w4, short w3, short w2, short w1, short w0)

Sets the 8 signed 16-bit integer values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>w0</td>
<td>w1</td>
<td>...</td>
<td>w7</td>
</tr>
</tbody>
</table>

__m128i _mm_set_epi8(char b15, char b14, char b13, char b12, char b11, char b10, char b9, char b8, char b7, char b6, char b5, char b4, char b3, char b2, char b1, char b0)

Sets the 16 signed 8-bit integer values.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>b0</td>
<td>b1</td>
<td>...</td>
<td>b15</td>
</tr>
</tbody>
</table>

__m128i _mm_set1_epi64(__m64 q)

Sets the 2 64-bit integer values to q.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>q</td>
</tr>
</tbody>
</table>

__m128i _mm_set1_epi32(int i)

Sets the 4 signed 32-bit integer values to i.
Sets the 8 signed 16-bit integer values to \( w \).

Sets the 16 signed 8-bit integer values to \( b \).

Sets the 2 64-bit integer values in reverse order.

Sets the 4 signed 32-bit integer values in reverse order.

Sets the 8 signed 16-bit integer values in reverse order.
Sets the 16 signed 8-bit integer values in reverse order.

Sets the 128-bit value to zero.

**Store Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for integer store operations are listed in this topic. The prototypes for the Intel® SSE2 intrinsics are in the `emmintrin.h` header file.

The detailed description of each intrinsic contains a table detailing the returns. In these tables, \( p \) is an access to the result.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_mm_stream_si128</code></td>
<td>Store</td>
<td>MOVNTDQ</td>
</tr>
<tr>
<td><code>_mm_stream_si32</code></td>
<td>Store</td>
<td>MOVNTI</td>
</tr>
<tr>
<td><code>_mm_store_si128</code></td>
<td>Store</td>
<td>MOVDQA</td>
</tr>
<tr>
<td><code>_mm_storeu_si128</code></td>
<td>Store</td>
<td>MOVDQU</td>
</tr>
</tbody>
</table>
### Corresponding Intel® SSE2 Instruction

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_maskmoveu_si128</td>
<td>Conditional store</td>
<td>MASKMOVQDQU</td>
</tr>
<tr>
<td>_mm_storel_epi64</td>
<td>Store lowest</td>
<td>MOVQ</td>
</tr>
</tbody>
</table>

void 

\[
\text{mm\_stream\_si128}(_m128i \ p, _m128i \ a) \\
\]

Stores the data in \( a \) to the address \( p \) without polluting the caches. If the cache line containing address \( p \) is already in the cache, the cache will be updated. Address \( p \) must be 16 byte aligned.

\[
* p \\
\]

\[
a \\
\]

void 

\[
\text{mm\_stream\_si32}(\text{int} \ p, \text{int} \ a) \\
\]

Stores the data in \( a \) to the address \( p \) without polluting the caches. If the cache line containing address \( p \) is already in the cache, the cache will be updated.

\[
* p \\
\]

\[
a \\
\]

void 

\[
\text{mm\_store\_si128}(_m128i \ p, _m128i \ b) \\
\]

Stores 128-bit value. Address \( p \) must be 16 byte aligned.

\[
* p \\
\]

\[
a \\
\]

void 

\[
\text{mm\_storeu\_si128}(_m128i \ p, _m128i \ b) \\
\]

Stores 128-bit value. Address \( p \) need not be 16-byte aligned.

\[
* p \\
\]

\[
a \\
\]

void 

\[
\text{mm\_maskmoveu\_si128}(_m128i \ d, _m128i \ n, \text{char} \ * p) \\
\]
Conditionally store byte elements of \( d \) to address \( p \). The high bit of each byte in the selector \( n \) determines whether the corresponding byte in \( d \) will be stored. Address \( p \) need not be 16-byte aligned.

\[
\begin{array}{llll}
\text{if} \ (n0[7]) & \text{if} \ (n1[7]) & \ldots & \text{if} \ (n15[7]) \\
\end{array}
\]

\begin{verbatim}
void _mm_storel_epi64(__m128i *p, __m128i a)
\end{verbatim}

Stores the lower 64 bits of the value pointed to by \( p \).

\[
*p[63:0] = a0
\]

Miscellaneous Functions and Intrinsics

Cacheability Support Intrinsics

The prototypes for Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for cacheability support are in the `emmintrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_stream_pd</td>
<td>Store</td>
<td>MOVNTPD</td>
</tr>
<tr>
<td>_mm256_stream_pd</td>
<td>Store</td>
<td>VMOVNTPD</td>
</tr>
<tr>
<td>_mm_stream_si128</td>
<td>Store</td>
<td>MOVNTDQ</td>
</tr>
<tr>
<td>_mm256_stream_si256</td>
<td>Store</td>
<td>VMOVNTDQ</td>
</tr>
<tr>
<td>_mm_stream_si32</td>
<td>Store</td>
<td>MOVNTI</td>
</tr>
<tr>
<td>_mm_stream_si64</td>
<td>Store</td>
<td>MOVNTI</td>
</tr>
<tr>
<td>_mm_clflush</td>
<td>Flush</td>
<td>CLFLUSH</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE2 Instruction</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_lfence</td>
<td>Guarantee visibility</td>
<td>LFENCE</td>
</tr>
<tr>
<td>_mm_mfence</td>
<td>Guarantee visibility</td>
<td>MFENCE</td>
</tr>
</tbody>
</table>

void _mm_stream_pd(double *p, __m128d a)
Stores the data in a to the address p without polluting caches. The address p must be 16-byte (128-bit version) aligned. If the cache line containing address p is already in the cache, the cache will be updated. p[0] := a0 p[1] := a1

<table>
<thead>
<tr>
<th>p[0]</th>
<th>p[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a1</td>
</tr>
</tbody>
</table>

void _mm256_stream_pd(double *p, __m256d a)
Stores the data in a to the address p without polluting caches. The address p must be 32-byte (VEX.256 encoded version) aligned. If the cache line containing address p is already in the cache, the cache will be updated. p[0] := a0 p[1] := a1

<table>
<thead>
<tr>
<th>p[0]</th>
<th>p[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a1</td>
</tr>
</tbody>
</table>

void _mm_stream_si128(__m128i *p, __m128i a)
Stores the data in a to the address p without polluting the caches. If the cache line containing address p is already in the cache, the cache will be updated. Address p must be 16-byte (128-bit version) aligned.

<table>
<thead>
<tr>
<th>*p</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

void _mm256_stream_si256(__m256i *p, __m256i a)
Stores the data in a to the address p without polluting the caches. If the cache line containing address p is already in the cache, the cache will be updated. Address p must be 32-byte (VEX.256 encoded version) aligned.
void _mm_stream_si32(int *p, int a)
Stores the 32-bit integer data in a to the address p without polluting the caches. If the cache line containing address p is already in the cache, the cache will be updated.

void _mm_stream_si64(__int64 *p, __int64 a)
Stores the 64-bit integer data in a to the address p without polluting the caches. If the cache line containing address p is already in the cache, the cache is updated.

void _mm_clflush(void const*p)
Cache line containing p is flushed and invalidated from all caches in the coherency domain.

void _mm_lfence(void)
Guarantees that every load instruction that precedes, in program order, the load fence instruction is globally visible before any load instruction which follows the fence in program order.

void _mm_mfence(void)
Guarantees that every memory access that precedes, in program order, the memory fence instruction is globally visible before any memory instruction which follows the fence in program order.
**Miscellaneous Intrinsics**

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) intrinsics for miscellaneous operations are listed in the following table followed by descriptions.

The prototypes for Intel® SSE2 intrinsics are in the *emmintrin.h* header file.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding Intel® SSE 2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_packs_epi16</td>
<td>Packed Saturation</td>
<td>PACKSSWB</td>
</tr>
<tr>
<td>_mm_packs_epi32</td>
<td>Packed Saturation</td>
<td>PACKSSDW</td>
</tr>
<tr>
<td>_mm_packus_epi16</td>
<td>Packed Saturation</td>
<td>PACKUSWB</td>
</tr>
<tr>
<td>_mm_extract_epi16</td>
<td>Extraction</td>
<td>PEXTRW</td>
</tr>
<tr>
<td>_mm_insert_epi16</td>
<td>Insertion</td>
<td>PINSRW</td>
</tr>
<tr>
<td>_mm_movemask_epi8</td>
<td>Mask Creation</td>
<td>PMOVMSKB</td>
</tr>
<tr>
<td>_mm_shuffle_epi32</td>
<td>Shuffle</td>
<td>FSHUFD</td>
</tr>
<tr>
<td>_mm_shufflehi_epi16</td>
<td>Shuffle</td>
<td>FSHUFW</td>
</tr>
<tr>
<td>_mm_shufflelo_epi16</td>
<td>Shuffle</td>
<td>FSHUFLW</td>
</tr>
<tr>
<td>_mm_unpackhi_epi8</td>
<td>Interleave</td>
<td>PUNPCKHBW</td>
</tr>
<tr>
<td>_mm_unpackhi_epi16</td>
<td>Interleave</td>
<td>PUNPCKHWD</td>
</tr>
<tr>
<td>_mm_unpackhi_epi32</td>
<td>Interleave</td>
<td>PUNPCKHDQ</td>
</tr>
<tr>
<td>_mm_unpackhi_epi64</td>
<td>Interleave</td>
<td>PUNPCKHQDQ</td>
</tr>
<tr>
<td>_mm_unpacklo_epi8</td>
<td>Interleave</td>
<td>PUNPCKLBW</td>
</tr>
<tr>
<td>_mm_unpacklo_epi16</td>
<td>Interleave</td>
<td>PUNPCKLWD</td>
</tr>
<tr>
<td>_mm_unpacklo_epi32</td>
<td>Interleave</td>
<td>PUNPCKLDQ</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding Intel® SSE 2 Instruction</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>__mm_unpacklo_epi64</td>
<td>Interleave</td>
<td>FUNPCKLQDQ</td>
</tr>
<tr>
<td>__mm_movepi64_epi64</td>
<td>Move</td>
<td>MOVDQ2Q</td>
</tr>
<tr>
<td>__mm_movpi64_epi64</td>
<td>Move</td>
<td>MOVDQ2Q</td>
</tr>
<tr>
<td>__mm_move_epi64</td>
<td>Move</td>
<td>MOVQ</td>
</tr>
<tr>
<td>__mm_unpackhi_pd</td>
<td>Interleave</td>
<td>UNPCKHPD</td>
</tr>
<tr>
<td>__mm_unpacklo_pd</td>
<td>Interleave</td>
<td>UNPCKLPD</td>
</tr>
<tr>
<td>__mm_movemask_pd</td>
<td>Create mask</td>
<td>MOVMSKPD</td>
</tr>
<tr>
<td>__mm_shuffle_pd</td>
<td>Select values</td>
<td>SHUFPD</td>
</tr>
</tbody>
</table>

__m128i __mm Packs_epi16(__m128i a, __m128i b)

Packs the 16 signed 16-bit integers from a and b into 8-bit integers and saturates.

<table>
<thead>
<tr>
<th>R0</th>
<th>...</th>
<th>R7</th>
<th>R8</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed</td>
<td>...</td>
<td>Signed</td>
<td>Signed</td>
<td>...</td>
<td>Signed</td>
</tr>
<tr>
<td>Saturate(a0)</td>
<td>Saturate(a7)</td>
<td>Saturate(b0)</td>
<td>Saturate(b7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128i __mm Packs_epi32(__m128i a, __m128i b)

Packs the 8 signed 32-bit integers from a and b into signed 16-bit integers and saturates.

<table>
<thead>
<tr>
<th>R0</th>
<th>...</th>
<th>R3</th>
<th>R4</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed</td>
<td>...</td>
<td>Signed</td>
<td>Signed</td>
<td>...</td>
<td>Signed</td>
</tr>
<tr>
<td>Saturate(a0)</td>
<td>Saturate(a3)</td>
<td>Saturate(b0)</td>
<td>Saturate(b3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128i __mm Packus_epi16(__m128i a, __m128i b)

Packs the 16 signed 16-bit integers from a and b into 8-bit unsigned integers and saturates.
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>...</td>
<td>R7</td>
<td>R8</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsigned</td>
<td>...</td>
<td>Unsigned</td>
<td>Unsigned</td>
<td>...</td>
</tr>
<tr>
<td>Saturate(a0)</td>
<td></td>
<td>Saturate(a7)</td>
<td>Saturate(b0)</td>
<td>Saturate(b15)</td>
</tr>
</tbody>
</table>

\[
\text{int } _\text{mmExtract_epi16}(\text{__m128i } a, \text{ int imm})
\]

Extracts the selected signed or unsigned 16-bit integer from \(a\) and zero extends. The selector \(imm\) must be an immediate.

\[
\text{R0}
\]

\[
(imm == 0) \ ? a0: \ (imm == 1) \ ? a1: \ ... \ (imm==7) \ ? a7)
\]

\[
\text{__m128i } _\text{mmInsert_epi16}(\text{__m128i } a, \text{ int b, int imm})
\]

Inserts the least significant 16 bits of \(b\) into the selected 16-bit integer of \(a\). The selector \(imm\) must be an immediate.

\[
\text{R0} \quad \text{R1} \quad \text{...} \quad \text{R7}
\]

\[
(imm == 0) \ ? b : \ (imm == 1) \ ? b : \ ... \ (imm == 7) \ ? b : \\
\text{a0}; \quad \text{a1}; \quad \text{...} \quad \text{a7};
\]

\[
\text{int } _\text{mmMoveMask_epi8}(\text{__m128i } a)
\]

Creates a 16-bit mask from the most significant bits of the 16 signed or unsigned 8-bit integers in \(a\) and zero extends the upper bits.

\[
\text{R0}
\]

\[
\]

\[
\text{__m128i } _\text{mmShuffle_epi32}(\text{__m128i } a, \text{ int imm})
\]

Shuffles the 4 signed or unsigned 32-bit integers in \(a\) as specified by \(imm\). The shuffle value, \(imm\), must be an immediate. See Macro Function for Shuffle for a description of shuffle semantics.

\[
\text{__m128i } _\text{mmShufflehi_epi16}(\text{__m128i } a, \text{ int imm})
\]

Shuffles the upper 4 signed or unsigned 16-bit integers in \(a\) as specified by \(imm\). The shuffle value, \(imm\), must be an immediate. See Macro Function for Shuffle for a description of shuffle semantics.
Shuffles the lower 4 signed or unsigned 16-bit integers in \textit{a} as specified by \textit{imm}. The shuffle value, \textit{imm}, must be an immediate. See Macro Function for Shuffle for a description of shuffle semantics.

Interleaves the upper 8 signed or unsigned 8-bit integers in \textit{a} with the upper 8 signed or unsigned 8-bit integers in \textit{b}.

\begin{verbatim}
    R0   R1   R2   R3   ...   R14  R15
a8    b8    a9    b9    ...    a15  b15
\end{verbatim}

Interleaves the upper 4 signed or unsigned 16-bit integers in \textit{a} with the upper 4 signed or unsigned 16-bit integers in \textit{b}.

\begin{verbatim}
    R0   R1   R2   R3   R4   R5   R6   7R
a4    b4    a5    b5    a6    b6    a7    7b
\end{verbatim}

Interleaves the upper 2 signed or unsigned 32-bit integers in \textit{a} with the upper 2 signed or unsigned 32-bit integers in \textit{b}.

\begin{verbatim}
    R0   R1   R2   R3
a2    b2    a3    b3
\end{verbatim}

Interleaves the upper signed or unsigned 64-bit integer in \textit{a} with the upper signed or unsigned 64-bit integer in \textit{b}.

\begin{verbatim}
    R0   R1
a1    b1
\end{verbatim}
Interleaves the lower 8 signed or unsigned 8-bit integers in a with the lower 8 signed or unsigned 8-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>...</th>
<th>R14</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>b0</td>
<td>a1</td>
<td>b1</td>
<td>...</td>
<td>a7</td>
<td>b7</td>
</tr>
</tbody>
</table>

`__m128i __m128i_mm_unpacklo_epi16(__m128i a, __m128i b)`

Interleaves the lower 4 signed or unsigned 16-bit integers in a with the lower 4 signed or unsigned 16-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>b0</td>
<td>a1</td>
<td>b1</td>
<td>a2</td>
<td>b2</td>
<td>a3</td>
<td>b3</td>
</tr>
</tbody>
</table>

`__m128i __m128i_mm_unpacklo_epi32(__m128i a, __m128i b)`

Interleaves the lower 2 signed or unsigned 32-bit integers in a with the lower 2 signed or unsigned 32-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>b0</td>
<td>a1</td>
<td>b1</td>
</tr>
</tbody>
</table>

`__m128i __m128i_mm_unpacklo_epi64(__m128i a, __m128i b)`

Interleaves the lower signed or unsigned 64-bit integer in a with the lower signed or unsigned 64-bit integer in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>b0</td>
</tr>
</tbody>
</table>

`__m64 __m64_mm_movepi64_pi64(__m128i a)`

Returns the lower 64 bits of a as an __m64 type.
__m128i _mm_movpi64_pi64(__m64 a)
Moves the 64 bits of a to the lower 64 bits of the result, zeroing the upper bits.

R0       R1
a0       0X0

__m128i _mm_move_epi64(__m128i a)
Moves the lower 64 bits of a to the lower 64 bits of the result, zeroing the upper bits.

R0       R1
a0       0X0

__m128d _mm_unpackhi_pd(__m128d a, __m128d b)
Interleaves the upper DP FP values of a and b.

R0       R1
a1       b1

__m128d _mm_unpacklo_pd(__m128d a, __m128d b)
Interleaves the lower DP FP values of a and b.

R0       R1
a0       b0

int _mm_movemask_pd(__m128d a)
Creates a two-bit mask from the sign bits of the two DP FP values of a.

R

sign(a1) << 1 | sign(a0)

__m128d _mm_shuffle_pd(__m128d a, __m128d b, int i)
Selects two specific DP FP values from \( a \) and \( b \), based on the mask \( i \). The mask must be an immediate. See Macro Function for Shuffle for a description of the shuffle semantics.

**Casting Support Intrinsics**

The Intel\textsuperscript{®} C++ Compiler supports casting between various single-precision, double-precision, and integer vector types. These intrinsics do not convert values; they change one data type to another without changing the value.

The intrinsics for casting support do not have any corresponding Intel\textsuperscript{®} Streaming SIMD Extensions 2 (Intel\textsuperscript{®} SSE2) instructions.

\[
\begin{align*}
__m128 & \_mm\_castpd\_ps(__m128d \text{ in}); \\
__m128i & \_mm\_castpd\_si128(__m128d \text{ in}); \\
__m128d & \_mm\_castps\_pd(__m128 \text{ in}); \\
__m128i & \_mm\_castps\_si128(__m128 \text{ in}); \\
__m128 & \_mm\_castsi128\_ps(__m128i \text{ in}); \\
__m128d & \_mm\_castsi128\_pd(__m128i \text{ in});
\end{align*}
\]

**Pause Intrinsic**

The prototype for this Intel\textsuperscript{®} Streaming SIMD Extensions 2 (Intel\textsuperscript{®} SSE2) intrinsic is in the \texttt{xmmintrin.h} header file.

**\texttt{PAUSE} Intrinsic**

\begin{verbatim}
void _mm_pause(void)
\end{verbatim}

The \texttt{pause} intrinsic is used in spin-wait loops with the processors implementing dynamic execution (especially out-of-order execution). In the spin-wait loop, the \texttt{pause} intrinsic improves the speed at which the code detects the release of the lock and provides especially significant performance gain.

The execution of the next instruction is delayed for an implementation-specific amount of time. The \texttt{PAUSE} instruction does not modify the architectural state. For dynamic scheduling, the \texttt{PAUSE} instruction reduces the penalty of exiting from the spin-loop.

**Example of loop with the \texttt{PAUSE} instruction:**
In this example, the program spins until memory location A matches the value in register eax.
The code sequence that follows shows a test-and-test-and-set.

```
spin_loop: pause
cmp eax, A
jne spin_loop
```

In this example, the spin occurs only after the attempt to get a lock has failed.

```
get_lock: mov eax, 1
xchg eax, A ; Try to get lock
cmp eax, 0 ; Test if successful
jne spin_loop
```

**Critical Section**
```
// critical_section code
mov A, 0 ; Release lock
jmp continue
spin_loop: pause;
// spin-loop hint
cmp 0, A ;
// check lock availability
jne spin_loop
jmp get_lock
// continue: other code
```

**NOTE.** The first branch is predicted to fall-through to the critical section in anticipation of successfully gaining access to the lock. It is highly recommended that all spin-wait loops include the PAUSE instruction. Since PAUSE is backwards compatible to all existing IA-32 architecture-based processor generations, a test for processor type (a CPUID test) is not needed. All legacy processors execute PAUSE instruction as a NOP, but in processors that use the PAUSE instruction as a hint there can be significant performance benefit.
Macro Function for Shuffle

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) provide a macro function to help create constants that describe shuffle operations. The macro takes two small integers (in the range of 0 to 1) and combines them into an 2-bit immediate value used by the SHUFPD instruction. See the following example.

Shuffle Function Macro

\[
\_\_\text{SHUFFLE\_2}(x, y)
\]

expands to the value of

\[(x<<1) \mid y\]

You can view the two integers as selectors for choosing which two words from the first input operand and which two words from the second are to be put into the result word.

View of Original and Result Words with Shuffle Function Macro

; m1 = 127 a b 0
; m2 = 127 c d 0
m3 = \_\_\text{SHUFFLE\_PD}(m1, m2, \_\_\text{SHUFFLE\_2}(1, 0))
; m3 = 127 c b 0
Overview: Intel(R) Streaming SIMD Extensions 3

The Intel® C++ intrinsics listed in this section are designed for the Intel® Pentium® 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3). They will not function correctly on other IA-32 architecture-based processors. The new Intel® SSE3 intrinsics include:

- Floating-point Vector Intrinsics
- Integer Vector Intrinsics
- Miscellaneous Intrinsics
- Macro Functions

The prototypes for these intrinsics are in the pmmintrin.h header file.

NOTE. You can also use the single ia32intrin.h header file for any IA-32 architecture-based intrinsics.

Integer Vector Intrinsics

The integer vector intrinsic listed here is designed for the Intel® Pentium® 4 processor with Streaming SIMD Extensions 3 (Intel® SSE3).

The prototype for this intrinsic is in the pmmintrin.h header file.

R represents the register into which the returns are placed.

```
__m128i _mm_lddqu_si128(__m128i const *p);
```

Loads an unaligned 128-bit value. This differs from movdqu in that it can provide higher performance in some cases. However, it also may provide lower performance than movdqu if the memory value being read was just previously written.
Single-precision Floating-point Vector Intrinsics

The single-precision floating-point vector intrinsics listed here are designed for the Intel® Pentium® 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3).

The results of each intrinsic operation are placed in the registers R0, R1, R2, and R3.

The prototypes for these intrinsics are in the pmmintrin.h header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE3 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_addsub_ps</td>
<td>Subtract and add</td>
<td>ADDSUBPS</td>
</tr>
<tr>
<td>_mm_hadd_ps</td>
<td>Add</td>
<td>HADDPS</td>
</tr>
<tr>
<td>_mm_hsub_ps</td>
<td>Subtracts</td>
<td>HSUBPS</td>
</tr>
<tr>
<td>_mm_movehdup_ps</td>
<td>Duplicates</td>
<td>MOVSHDUP</td>
</tr>
<tr>
<td>_mm_moveldup_ps</td>
<td>Duplicates</td>
<td>MOVSLDUP</td>
</tr>
</tbody>
</table>

extern __m128 _mm_addsub_ps(__m128 a, __m128 b);

Subtracts even vector elements while adding odd vector elements.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 - b0;</td>
<td>a1 + b1;</td>
<td>a2 - b2;</td>
<td>a3 + b3;</td>
</tr>
</tbody>
</table>

extern __m128 _mm_hadd_ps(__m128 a, __m128 b);

 Adds adjacent vector elements.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + a1;</td>
<td>a2 + a3;</td>
<td>b0 + b1;</td>
<td>b2 + b3;</td>
</tr>
</tbody>
</table>
extern __m128 _mm_hsub_ps(__m128 a, __m128 b);

Subtracts adjacent vector elements.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 - a1;</td>
<td>a2 - a3;</td>
<td>b0 - b1;</td>
<td>b2 - b3;</td>
</tr>
</tbody>
</table>

extern __m128 _mm_movehdup_ps(__m128 a);

Duplicates odd vector elements into even vector elements.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1;</td>
<td>a1;</td>
<td>a3;</td>
<td>a3;</td>
</tr>
</tbody>
</table>

extern __m128 _mm_moveldup_ps(__m128 a);

Duplicates even vector elements into odd vector elements.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0;</td>
<td>a0;</td>
<td>a2;</td>
<td>a2;</td>
</tr>
</tbody>
</table>

**Double-precision Floating-point Vector Intrinsics**

The double-precision floating-point intrinsics listed here are designed for the Intel® Pentium® 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3).

The results of each intrinsic operation are placed in the registers R0 and R1.

The prototypes for these intrinsics are in the pmmintrin.h header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE3 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_addsub_pd</td>
<td>Subtract and add</td>
<td>ADDSUBPD</td>
</tr>
<tr>
<td>_mm_hadd_pd</td>
<td>Add</td>
<td>HADDPD</td>
</tr>
<tr>
<td>_mm_hsub_pd</td>
<td>Subtract</td>
<td>HSUBPD</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding Intel® SSE3 Instruction</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>_mm_loadupsd_pd</td>
<td>Duplicate</td>
<td>MOVDDUP</td>
</tr>
<tr>
<td>_mm_movedupsd_pd</td>
<td>Duplicate</td>
<td>MOVDDUP</td>
</tr>
</tbody>
</table>

extern __m128d _mm_addsub_pd(__m128d a, __m128d b);

Adds upper vector element while subtracting lower vector element.

\[
\begin{array}{ll}
R0 & a0 - b0; \\
R1 & a1 + b1;
\end{array}
\]

extern __m128d _mm_hadd_pd(__m128d a, __m128d b);

Adds adjacent vector elements.

\[
\begin{array}{ll}
R0 & a0 + a1; \\
R1 & b0 + b1;
\end{array}
\]

extern __m128d _mm_hsub_pd(__m128d a, __m128d b);

Subtracts adjacent vector elements.

\[
\begin{array}{ll}
R0 & a0 - a1; \\
R1 & b0 - b1;
\end{array}
\]

extern __m128d _mm_loadupsd_pd(double const * dp);

Duplicates a double value into upper and lower vector elements.

\[
\begin{array}{ll}
R0 & *dp; \\
R1 & *dp;
\end{array}
\]

extern __m128d _mm_movedupsd_pd(__m128d a);

Duplicates lower vector element into upper vector element.
The miscellaneous intrinsics listed here are designed for the Intel® Pentium® 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3).

The prototypes for these intrinsics are in the `pmmintrin.h` header file.

```c
extern void _mm_monitor(void const *p, unsigned extensions, unsigned hints);
```

Generates the `MONITOR` instruction. This sets up an address range for the monitor hardware using `p` to provide the logical address, and will be passed to the monitor instruction in register `eax`. The extensions parameter contains optional extensions to the monitor hardware which will be passed in `ecx`. The hints parameter will contain hints to the monitor hardware, which will be passed in `edx`. A non-zero value for extensions will cause a general protection fault.

```c
extern void _mm_mwait(unsigned extensions, unsigned hints);
```

Generates the `MWAIT` instruction. This instruction is a hint that allows the processor to stop execution and enter an implementation-dependent optimized state until occurrence of a class of events. In future processor designs extensions and hints parameters may be used to convey additional information to the processor. All non-zero values of extensions and hints are reserved. A non-zero value for extensions will cause a general protection fault.

### Macro Functions

The macro function intrinsics listed here are designed for the Intel® Pentium® 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3).

The prototypes for these intrinsics are in the `pmmintrin.h` header file.

```c
_MM_SET_DENORMALS_ZERO_MODE(x)
```

Macro arguments: one of `__MM_DENORMALS_ZERO_ON`, `__MM_DENORMALS_ZERO_OFF` This causes "denormals are zero" mode to be turned on or off by setting the appropriate bit of the control register.

```c
_MM_GET_DENORMALS_ZERO_MODE()
```
No arguments. This returns the current value of the denormals are zero mode bit of the control register.
Overview: Intel(R) Supplemental Streaming SIMD Extensions 3

Intel's C++ intrinsics listed in this section correspond to the Intel(R) Supplemental Streaming SIMD Extensions 3 instructions. The prototypes for these intrinsics are in tmmintrin.h. You can also use the ia32intrin.h header file for these intrinsics.

- Addition Intrinsics
- Subtraction Intrinsics
- Multiplication Intrinsics
- Absolute Value Intrinsics
- Shuffle Intrinsics
- Concatenate Intrinsics
- Negation Intrinsics

Addition Intrinsics

These Intel(R) Supplemental Streaming SIMD Extensions 3 (Intel(R) SSSE3) intrinsics are used for horizontal addition. The prototypes for these intrinsics are in tmmintrin.h. You can also use the ia32intrin.h header file for these intrinsics.

extern __m128i _mm_hadd_epi16 (__m128i a, __m128i b);

Add horizontally packed signed words.
Interpreting a, b, and r as arrays of 16-bit signed integers:
for (i = 0; i < 4; i++) {
    r[i] = a[2*i] + a[2*i+1];
    r[i+4] = b[2*i] + b[2*i+1];
}
extern __m128i _mm_hadd_epi32 (__m128i a, __m128i b);

Add horizontally packed signed dwords.

Interpreting \( a \), \( b \), and \( r \) as arrays of 32-bit signed integers:

for (i = 0; i < 2; i++) {
    r[i] = a[2*i] + a[2*i+1];
    r[i+2] = b[2*i] + b[2*i+1];
}

extern __m128i _mm_hadds_epi16 (__m128i a, __m128i b);

Add horizontally packed signed words with signed saturation.

Interpreting \( a \), \( b \), and \( r \) as arrays of 16-bit signed integers:

for (i = 0; i < 4; i++) {
    r[i] = signed_saturate_to_word(a[2*i] + a[2*i+1]);
    r[i+4] = signed_saturate_to_word(b[2*i] + b[2*i+1]);
}

extern __m64 _mm_hadd_pi16 (__m64 a, __m64 b);

Add horizontally packed signed words.

Interpreting \( a \), \( b \), and \( r \) as arrays of 16-bit signed integers:

for (i = 0; i < 2; i++) {
    r[i] = a[2*i] + a[2*i+1];
    r[i+2] = b[2*i] + b[2*i+1];
}

extern __m64 _mm_hadd_pi32 (__m64 a, __m64 b);

Add horizontally packed signed dwords.

Interpreting \( a \), \( b \), and \( r \) as arrays of 32-bit signed integers:

\[ r[0] = a[1] + a[0]; \]
\[ r[1] = b[1] + b[0]; \]

extern __m64 _mm_hadds_pi16 (__m64 a, __m64 b);

Add horizontally packed signed words with signed saturation.
Interpreting \(a\), \(b\), and \(r\) as arrays of 16-bit signed integers:
for (i = 0; i < 2; i++) {
    r[i] = signed_saturate_to_word(a[2*i] + a[2*i+1]);
    r[i+2] = signed_saturate_to_word(b[2*i] + b[2*i+1]);
}

Subtraction Intrinsics

These Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) intrinsics are used for horizontal subtraction. The prototypes for these intrinsics are in tmmintrin.h. You can also use the ia32intrin.h header file for these intrinsics.
	exttt{extern \_m128i \_mm\_hsub\_epi16 (\_m128i a, \_m128i b);} Subtract horizontally packed signed words.

Interpreting \(a\), \(b\), and \(r\) as arrays of 16-bit signed integers:
for (i = 0; i < 4; i++) {
    r[i] = a[2*i] - a[2*i+1];
    r[i+4] = b[2*i] - b[2*i+1];
}
	exttt{extern \_m128i \_mm\_hsub\_epi32 (\_m128i a, \_m128i b);} Subtract horizontally packed signed dwords.

Interpreting \(a\), \(b\), and \(r\) as arrays of 32-bit signed integers:
for (i = 0; i < 2; i++) {
    r[i] = a[2*i] - a[2*i+1];
    r[i+2] = b[2*i] - b[2*i+1];
}
	exttt{extern \_m128i \_mm\_hsubs\_epi16 (\_m128i a, \_m128i b);} Subtract horizontally packed signed words with signed saturation.
Interpreting $a$, $b$, and $r$ as arrays of 16-bit signed integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = signed_saturate_to_word(a[2*i] - a[2*i+1]);
    r[i+4] = signed_saturate_to_word(b[2*i] - b[2*i+1]);
}
```

`extern __m64 _mm_hsub_pi16 (__m64 a, __m64 b);`

Subtract horizontally packed signed words.

Interpreting $a$, $b$, and $r$ as arrays of 16-bit signed integers:

```c
for (i = 0; i < 2; i++) {
    r[i] = a[2*i] - a[2*i+1];
    r[i+2] = b[2*i] - b[2*i+1];
}
```

`extern __m64 _mm_hsub_pi32 (__m64 a, __m64 b);`

Subtract horizontally packed signed dwords.

Interpreting $a$, $b$, and $r$ as arrays of 32-bit signed integers:

```c
r[0] = a[0] - a[1];
r[1] = b[0] - b[1];
```

`extern __m64 __m64 __m_hsub_pi16 (__m64 a, __m64 b);`

Subtract horizontally packed signed words with signed saturation.

Interpreting $a$, $b$, and $r$ as arrays of 16-bit signed integers:

```c
for (i = 0; i < 2; i++) {
    r[i] = signed_saturate_to_word(a[2*i] - a[2*i+1]);
    r[i+2] = signed_saturate_to_word(b[2*i] - b[2*i+1]);
}
```
**Multiplication Intrinsics**

These Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) intrinsics are used for multiplication. The prototypes for these intrinsics are in `tmmintrin.h`. You can also use the `ia32intrin.h` header file for these intrinsics.

```c
extern __m128i _mm_maddubs_epi16 (__m128i a, __m128i b);
```

Multiply signed and unsigned bytes, add horizontal pair of signed words, pack saturated signed words. Interpreting `a` as array of unsigned 8-bit integers, `b` as arrays of signed 8-bit integers, and `r` as arrays of 16-bit signed integers:

```c
for (i = 0; i < 8; i++) {
    r[i] = signed_saturate_to_word(a[2*i+1] * b[2*i+1] + a[2*i]*b[2*i]);
}
```

```c
extern __m64 _mm_maddubs_pi16 (__m64 a, __m64 b);
```

Multiply signed and unsigned bytes, add horizontal pair of signed words, pack saturated signed words. Interpreting `a` as array of unsigned 8-bit integers, `b` as arrays of signed 8-bit integers, and `r` as arrays of 16-bit signed integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = signed_saturate_to_word(a[2*i+1] * b[2*i+1] + a[2*i]*b[2*i]);
}
```

```c
extern __m128i _mm_mulhrs_epi16 (__m128i a, __m128i b);
```

Multiply signed words, scale and round signed dwords, pack high 16-bits. Interpreting `a`, `b`, and `r` as arrays of signed 16-bit integers:

```c
for (i = 0; i < 8; i++) {
    r[i] = (( (int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
}
```

```c
extern __m64 _mm_mulhrs_pi16 (__m64 a, __m64 b);
```

Multiply signed words, scale and round signed dwords, pack high 16-bits.
Interpreting $a, b,$ and $r$ as arrays of signed 16-bit integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = ((int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
}
```

### Absolute Value Intrinsics

These Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) intrinsics are used to compute absolute values. The prototypes for these intrinsics are in `tmmintrin.h`. You can also use the `ia32intrin.h` header file for these intrinsics.

```c
extern __m128i _mm_abs_epi8 (__m128i a);
```

Compute absolute value of signed bytes.

Interpreting $a$ and $r$ as arrays of signed 8-bit integers:

```c
for (i = 0; i < 16; i++) {
    r[i] = abs(a[i]);
}
```

```c
extern __m128i _mm_abs_epi16 (__m128i a);
```

Compute absolute value of signed words.

Interpreting $a$ and $r$ as arrays of signed 16-bit integers:

```c
for (i = 0; i < 8; i++) {
    r[i] = abs(a[i]);
}
```

```c
extern __m128i _mm_abs_epi32 (__m128i a);
```

Compute absolute value of signed dwords.

Interpreting $a$ and $r$ as arrays of signed 32-bit integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = abs(a[i]);
}
```

```c
extern __m64 _mm_abs_pi8 (__m64 a);
```
Compute absolute value of signed bytes.
Interpreting $a$ and $r$ as arrays of signed 8-bit integers:
for ($i = 0; i < 8; i++$) {
    $r[i] = \text{abs}(a[i])$;
}

extern __m64 _mm_abs_pi16 (__m64 a);

Compute absolute value of signed words.
Interpreting $a$ and $r$ as arrays of signed 16-bit integers:
for ($i = 0; i < 4; i++$) {
    $r[i] = \text{abs}(a[i])$;
}

extern __m64 _mm_abs_pi32 (__m64 a);

Compute absolute value of signed dwords.
Interpreting $a$ and $r$ as arrays of signed 32-bit integers:
for ($i = 0; i < 2; i++$) {
    $r[i] = \text{abs}(a[i])$;
}

Shuffle Intrinsics

These Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) intrinsics are used to perform shuffle operations. The prototypes for these intrinsics are in tmmintrin.h. You can also use the ia32intrin.h header file for these intrinsics.

extern __m128i _mm_shuffle_epi8 (__m128i a, __m128i b);
Shuffle bytes from $a$ according to contents of $b$. 
Interpreting $a$, $b$, and $r$ as arrays of unsigned 8-bit integers:
for ($i = 0; i < 16; i++){  
    if ($b[i] \& 0x80$){  
        $r[i] = 0;$  
    }  
    else  
    {  
        $r[i] = a[b[i] \& 0x0F]$;  
    }  
}

extern __m64 _mm_shuffle_pi8 (__m64 a, __m64 b);
Shuffle bytes from $a$ according to contents of $b$.

Interpreting $a$, $b$, and $r$ as arrays of unsigned 8-bit integers:
for ($i = 0; i < 8; i++){  
    if ($b[i] \& 0x80$){  
        $r[i] = 0;$  
    }  
    else  
    {  
        $r[i] = a[b[i] \& 0x07]$;  
    }  
}

Concatenate Intrinsics

These Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) intrinsics are used for concatenation. The prototypes for these intrinsics are in tmmintrin.h. You can also use the ia32intrin.h header file for these intrinsics.

extern __m128i _mm_alignr_epi8 (__m128i a, __m128i b, int n);
Concatenate \( a \) and \( b \), extract byte-aligned result shifted to the right by \( n \).

Interpreting \( t_1 \) as 256-bit unsigned integer, \( a \), \( b \), and \( r \) as 128-bit unsigned integers:
\[
t_1[255:128] = a;
t_1[127:0] = b;
t_1[255:0] = t_1[255:0] >> (8 \times n); // unsigned shift
\]
\[
r[127:0] = t_1[127:0];
\]

extern \_m64 \_mm_alignr\_pi8 (_m64 a, _m64 b, int n);

Concatenate \( a \) and \( b \), extract byte-aligned result shifted to the right by \( n \).

Interpreting \( t_1 \) as 127-bit unsigned integer, \( a \), \( b \) and \( r \) as 64-bit unsigned integers:
\[
t_1[127:64] = a;
t_1[63:0] = b;
t_1[127:0] = t_1[127:0] >> (8 \times n); // unsigned shift
\]
\[
r[63:0] = t_1[63:0];
\]

**Negation Intrinsics**

These Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) intrinsics are used for negation. The prototypes for these intrinsics are in tmmintrin.h. You can also use the ia32intrin.h header file for these intrinsics.

extern \_m128i \_mm\_sign\_epi8 (_m128i a, _m128i b);

Negate packed bytes in \( a \) if corresponding sign in \( b \) is less than zero.
Interpreting \( a, b, \) and \( r \) as arrays of signed 8-bit integers:

```c
for (i = 0; i < 16; i++){
    if (b[i] < 0){
        r[i] = -a[i];
    }
    else
    if (b[i] == 0){
        r[i] = 0;
    }
    else
    {
        r[i] = a[i];
    }
}
```

```c
extern __m128i __mm_sign_epi16 (__m128i a, __m128i b);
```

Negate packed words in \( a \) if corresponding sign in \( b \) is less than zero.
Interpreting $a$, $b$, and $c$ as arrays of signed 16-bit integers:

```c
for (i = 0; i < 8; i++){
    if (b[i] < 0){
        r[i] = -a[i];
    } else
    if (b[i] == 0){
        r[i] = 0;
    } else
    {
        r[i] = a[i];
    }
}

extern __m128i _mm_sign_epi32 (__m128i a, __m128i b);
```

Negate packed dwords in $a$ if corresponding sign in $b$ is less than zero.
Interpreting $a$, $b$, and $r$ as arrays of signed 32-bit integers:

```c
for (i = 0; i < 4; i++){
    if (b[i] < 0){
        r[i] = -a[i];
    } else
        if (b[i] == 0){
            r[i] = 0;
        } else{
            r[i] = a[i];
        }
}
```

```
extern __m64 __m64_mm_sign_pi8 (__m64 a, __m64 b);
```

Negate packed bytes in $a$ if corresponding sign in $b$ is less than zero.
Interpreting $a$, $b$, and $r$ as arrays of signed 8-bit integers:
for ($i = 0; i < 16; i++$){
    if ($b[i] < 0$){
        $r[i] = -a[i]$;
    }
    else
    if ($b[i] == 0$){
        $r[i] = 0$;
    }
    else{
        $r[i] = a[i]$;
    }
}
extern __m64 _mm_sign_pi16 (__m64 a, __m64 b);

Negate packed words in $a$ if corresponding sign in $b$ is less than zero.
Interpreting \(a\), \(b\), and \(r\) as arrays of signed 16-bit integers:

```c
for (i = 0; i < 8; i++)
    if (b[i] < 0)
        r[i] = -a[i];
    else
        if (b[i] == 0)
            r[i] = 0;
        else
            r[i] = a[i];

extern __m64 _mm_sign_pi32 (__m64 a, __m64 b);
```

Negate packed dwords in \(a\) if corresponding sign in \(b\) is less than zero.
Interpreting $a$, $b$, and $r$ as arrays of signed 32-bit integers:

```c
for (i = 0; i < 2; i++){
    if (b[i] < 0){
        r[i] = -a[i];
    } else if (b[i] == 0){
        r[i] = 0;
    } else{
        r[i] = a[i];
    }
}
```
Intrinsics for Intel(R) Streaming SIMD Extensions 4

Overview: Intel(R) Streaming SIMD Extensions 4

The intrinsics in this section correspond to Intel® Streaming SIMD Extensions 4 (Intel® SSE4) instructions. Intel® SSE4 includes the following categories:

- Vectorizing Compiler and Media Accelerators. The prototypes for these intrinsics are in the smmintrin.h file.
- Efficient Accelerated String and Text Processing. The prototypes for these intrinsics are in the nmmintrin.h file.

Vectorizing Compiler and Media Accelerators

Overview: Vectorizing Compiler and Media Accelerators

The intrinsics in this section correspond to Intel® Streaming SIMD Extensions 4 (Intel® SSE4) Vectorizing Compiler and Media Accelerators instructions.

- Packed Blending Intrinsics for Streaming SIMD Extensions 4
- Floating Point Dot Product Intrinsics for Streaming SIMD Extensions 4
- Packed Format Conversion Intrinsics for Streaming SIMD Extensions 4
- Packed Integer Min/Max Intrinsics for Streaming SIMD Extensions 4
- Floating Point Rounding Intrinsics for Streaming SIMD Extensions 4
- DWORD Multiply Intrinsics for Streaming SIMD Extensions 4
- Register Insertion/Extraction Intrinsics for Streaming SIMD Extensions 4
- Test Intrinsics for Streaming SIMD Extensions 4
- Packed DWORD to Unsigned WORD Intrinsic for Streaming SIMD Extensions 4
- Packed Compare for Equal Intrinsics for Streaming SIMD Extensions 4
- Cacheability Support Intrinsic for Streaming SIMD Extension 4

The prototypes for these intrinsics are in the smmintrin.h file.
### Packed Blending Intrinsics

These Intel® Streaming SIMD Extension 4 (Intel® SSE4) intrinsics pack multiple operations in a single instruction. Blending conditionally copies one field in the source onto the corresponding field in the destination. The prototypes for these intrinsics are in the `smmintrin.h` file.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128 _mm_blend_ps(__m128 v1, __m128 v2, const int mask)</td>
<td>Selects float single precision data from 2 sources using constant mask</td>
<td>BLENDPS</td>
</tr>
<tr>
<td>__m128d _mm_blend_pd(__m128d v1, __m128d v2, const int mask)</td>
<td>Selects float double precision data from 2 sources using constant mask</td>
<td>BLENDPD</td>
</tr>
<tr>
<td>__m128 _mm_blendv_ps(__m128 v1, __m128 v2, __m128 v3)</td>
<td>Selects float single precision data from 2 sources using variable mask</td>
<td>BLENDVPS</td>
</tr>
<tr>
<td>__m128d _mm_blendv_pd(__m128d v1, __m128d v2, __m128d v3)</td>
<td>Selects float double precision data from 2 sources using variable mask</td>
<td>BLENDVPD</td>
</tr>
<tr>
<td>__m128i _mm_blendv_epi8(__m128i v1, __m128i v2, __m128i mask)</td>
<td>Selects integer bytes from 2 sources using variable mask</td>
<td>PBLENDVB</td>
</tr>
<tr>
<td>__m128i _mm_blendv_epi16(__m128i v1, __m128i v2, const int mask)</td>
<td>Selects integer words from 2 sources using constant mask</td>
<td>PBLENDW</td>
</tr>
</tbody>
</table>
Floating Point Dot Product Intrinsics

These Intel® Streaming SIMD Extension (Intel® SSE4) intrinsics enable floating point single precision and double precision dot products. The prototypes for these intrinsics are in the smmintrin.h file.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_dp_pd</td>
<td>Double precision dot product</td>
<td>DPPD</td>
</tr>
<tr>
<td>_mm_dp_ps</td>
<td>Single precision dot product</td>
<td>DPPS</td>
</tr>
</tbody>
</table>

__m128d _mm_dp_pd (__m128d a, __m128d b, const int mask)

This intrinsic calculates the dot product of double precision packed values with mask-defined summing and zeroing of the parts of the result.

__m128 _mm_dp_ps (__m128 a, __m128 b, const int mask)

This intrinsic calculates the dot product of single precision packed values with mask-defined summing and zeroing of the parts of the result.

Packed Format Conversion Intrinsics

These Intel® Streaming SIMD Extension 4 (Intel® SSE4) intrinsics convert a packed integer to a zero-extended or sign-extended integer with wider type. The prototypes for these intrinsics are in the smmintrin.h file.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_mm_cvtepi8_epi32(__m128i a)</td>
<td>Sign extend 4 bytes into 4 double words</td>
<td>FMOVSXBD</td>
</tr>
<tr>
<td>__m128i _mm_cvtepi8_epi64 (__m128i a)</td>
<td>Sign extend 2 bytes into 2 quad words</td>
<td>FMOVXSBQ</td>
</tr>
<tr>
<td>__m128i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_mm_cvtepi8_epi16(__m128i a)</td>
<td>Sign extend 8 bytes into 8 words</td>
<td>FMOVXSBW</td>
</tr>
</tbody>
</table>
### Packed Integer Min/Max Intrinsics

These Intel® Streaming SIMD Extensions 4 (Intel® SSE4) intrinsics compare packed integers in the destination operand and the source operand, and return the minimum or maximum for each packed operand in the destination operand. The prototypes for these intrinsics are in the `smmintrin.h` file.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__m128i</code></td>
<td>Sign extend 2 double words into 2 quad words</td>
<td>PMOVSXDQ</td>
</tr>
<tr>
<td><code>__m128i</code></td>
<td>Sign extend 4 words into 4 double words</td>
<td>PMOVSXWD</td>
</tr>
<tr>
<td><code>__m128i</code></td>
<td>Sign extend 2 words into 2 quad words</td>
<td>PMOVSXWQ</td>
</tr>
<tr>
<td><code>__m128i</code></td>
<td>Zero extend 4 bytes into 4 double words</td>
<td>PMOVZXBD</td>
</tr>
<tr>
<td><code>__m128i</code></td>
<td>Zero extend 2 bytes into 2 quad words</td>
<td>PMOVZXBQ</td>
</tr>
<tr>
<td><code>__m128i</code></td>
<td>Zero extend 8 bytes into 8 word</td>
<td>PMOVZXBW</td>
</tr>
<tr>
<td><code>__m128i</code></td>
<td>Zero extend 2 double words into 2 quad words</td>
<td>PMOVSXDQ</td>
</tr>
<tr>
<td><code>__m128i</code></td>
<td>Zero extend 4 words into 4 double words</td>
<td>PMOVSXWD</td>
</tr>
<tr>
<td><code>__m128i</code></td>
<td>Zero extend 2 words into 2 quad words</td>
<td>PMOVSXWQ</td>
</tr>
<tr>
<td>Intrinsic Syntax</td>
<td>Operation</td>
<td>Corresponding Intel® SSE4 Instruction</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>__m128i _mm_max_epi8( __m128i a, __m128i b)</td>
<td>Calculates maximum of signed packed integer bytes</td>
<td>PMAXSB</td>
</tr>
<tr>
<td>__m128i _mm_max_epi32( __m128i a, __m128i b)</td>
<td>Calculates maximum of signed packed integer double words</td>
<td>PMAXSD</td>
</tr>
<tr>
<td>__m128i _mm_max_epu32( __m128i a, __m128i b)</td>
<td>Calculates maximum of unsigned packed integer double words</td>
<td>PMAXUD</td>
</tr>
<tr>
<td>__m128i _mm_max_epu16( __m128i a, __m128i b)</td>
<td>Calculates maximum of unsigned packed integer words</td>
<td>PMAXUW</td>
</tr>
<tr>
<td>__m128i _mm_min_epi8( __m128i a, __m128i b)</td>
<td>Calculates minimum of signed packed integer bytes</td>
<td>PMINSB</td>
</tr>
<tr>
<td>__m128i _mm_min_epi32( __m128i a, __m128i b)</td>
<td>Calculates minimum of signed packed integer double words</td>
<td>PMINSD</td>
</tr>
<tr>
<td>__m128i _mm_min_epu32( __m128i a, __m128i b)</td>
<td>Calculates minimum of unsigned packed integer double words</td>
<td>PMINUD</td>
</tr>
<tr>
<td>__m128i _mm_min_epu16( __m128i a, __m128i b)</td>
<td>Calculates minimum of unsigned packed integer words</td>
<td>PMINUW</td>
</tr>
</tbody>
</table>
Floating Point Rounding Intrinsics

These Intel® Streaming SIMD Extension 4 (Intel® SSE4) rounding intrinsics cover scalar and packed single-precision and double precision floating-point operands. The prototypes for these intrinsics are in the _smmintrin.h_ file.

The _floor_ and _ceil_ intrinsics correspond to the definitions of _floor_ and _ceil_ in the ISO 9899:1999 standard for the C programming language.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128d _mm_round_pd(__m128d s1, int iRoundMode)</td>
<td>Packed float double precision rounding</td>
<td>ROUNDPD</td>
</tr>
<tr>
<td>__m128d _mm_floor_pd(__m128d s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128d _mm_ceil_pd(__m128d s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128 _mm_round_ps(__m128 s1, int iRoundMode)</td>
<td>Packed float single precision rounding</td>
<td>ROUNDFS</td>
</tr>
<tr>
<td>__m128 _mm_floor_ps(__m128 s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128 _mm_ceil_ps(__m128 s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128d _mm_round_sd(__m128d dst, __m128d s1, int iRoundMode)</td>
<td>Single float double precision rounding</td>
<td>ROUNDSD</td>
</tr>
<tr>
<td>__m128d _mm_floor_sd(__m128d dst, __m128d s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128d _mm_ceil_sd(__m128d dst, __m128d s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128 _mm_round_ss(__m128 dst, __m128d s1, int iRoundMode)</td>
<td>Single float single precision rounding</td>
<td>ROUNDSS</td>
</tr>
<tr>
<td>__m128 _mm_floor_ss(__m128 dst, __m128 s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128 _mm_ceil_ss(__m128 dst, __m128 s1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DWORD Multiply Intrinsics

These Intel® Streaming SIMD Extensions (Intel® SSE4) DWORD multiply intrinsics are designed to aid vectorization. They enable four simultaneous 32-bit by 32-bit multiplies. The prototypes for these intrinsics are in the `smmintrin.h` file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128i _mm_mul_epi32( __m128i a, __m128i b)</td>
<td>Packed integer 32-bit multiplication of 2 low pairs of operands producing two 64-bit results</td>
<td>PMULDQ</td>
</tr>
<tr>
<td>__m128i _mm_mullo_epi32( __m128i a, __m128i b)</td>
<td>Packed integer 32-bit multiplication with truncation of upper halves of results</td>
<td>PMULLD</td>
</tr>
</tbody>
</table>

Register Insertion/Extraction Intrinsics

These Intel® Streaming SIMD Extensions (Intel® SSE4) intrinsics enable data insertion and extraction between general purpose registers and XMM registers. The prototypes for these intrinsics are in the `smmintrin.h` file.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128 _mm_insert_ps(__m128 dst, __m128 src, const int ndx)</td>
<td>Insert single precision float into packed single precision array element selected by index</td>
<td>INSERTPS</td>
</tr>
<tr>
<td>int _mm_extract_ps(__m128 src, const int ndx)</td>
<td>Extract single precision float from packed single precision array element selected by index</td>
<td>EXTRACTPS</td>
</tr>
</tbody>
</table>

1645
### Intrinsic Syntax

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>int __mm_extract_epi8(__m128i src, const int ndx)</td>
<td>Extract integer byte from packed integer array element selected by index</td>
<td>PEXTRB</td>
</tr>
<tr>
<td>int __mm_extract_epi32(__m128i src, const int ndx)</td>
<td>Extract integer double word from packed integer array element selected by index</td>
<td>PEXTRD</td>
</tr>
<tr>
<td>__int64 __mm_extract_epi64(__m128i src, const int ndx)</td>
<td>Extract integer quad word from packed integer array element selected by index</td>
<td>PEXTRQ</td>
</tr>
<tr>
<td>int __mm_extract_epi16(__m128i src, int ndx)</td>
<td>Extract integer word from packed integer array element selected by index</td>
<td>PEXTRW</td>
</tr>
<tr>
<td>__m128i __mm_insert_epi8(__m128i s1, int s2, const int ndx)</td>
<td>Insert integer byte into packed integer array element selected by index</td>
<td>PINSRB</td>
</tr>
<tr>
<td>__m128i __mm_insert_epi32(__m128i s1, int s2, const int ndx)</td>
<td>Insert integer double word into packed integer array element selected by index</td>
<td>PINSRD</td>
</tr>
<tr>
<td>__m128i __mm_insert_epi64(__m128i s2, int s, const int ndx)</td>
<td>Insert integer quad word into packed integer array element selected by index</td>
<td>PINSRQ</td>
</tr>
</tbody>
</table>

### Test Intrinsics

These Intel® Streaming SIMD Extensions (Intel® SSE4) intrinsics perform packed integer 128-bit comparisons. The prototypes for these intrinsics are in the _smmintrin.h_ file.
Corresponding Intel® SSE4 Instruction

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_testz_si128</td>
<td>Check for all zeros in specified bits of a 128-bit value</td>
<td>PTEST</td>
</tr>
<tr>
<td>_mm_testc_si128</td>
<td>Check for all ones in specified bits of a 128-bit value</td>
<td>PTEST</td>
</tr>
<tr>
<td>_mm_testnzc_si128</td>
<td>Check for at least one zero and at least one one in the specified bits of a 128-bit value</td>
<td>PTEST</td>
</tr>
</tbody>
</table>

**int _mm_testz_si128 (__m128i s1, __m128i s2)**

Returns 1 if the bitwise AND operation on s1 and s2 results in all zeros, else returns 0. That is,

\[
_{\text{mm\_testz\_si128}} := ( (s1 \& s2) == 0 ? 1 : 0 )
\]

This intrinsic checks if the ZF flag equals 1 as a result of the instruction PTEST s1, s2. For example, it allows you to check if all set bits in s2 (mask) are zeros in s1.

Corresponding instruction: PTEST

**int _mm_testc_si128 (__m128i s1, __m128i s2)**

Returns 1 if the bitwise AND operation on s2 and logical NOT s1 results in all zeros, else returns 0. That is,

\[
_{\text{mm\_testc\_si128}} := ( (~s1 \& s2) == 0 ? 1 : 0 )
\]

This intrinsic checks if the CF flag equals 1 as a result of the instruction PTEST s1, s2. For example it allows you to check if all set bits in s2 (mask) are also set in s1.

Corresponding instruction: PTEST

**int _mm_testnzc_si128 (__m128i s1, __m128i s2)**

Returns 1 if the following conditions are true: bitwise operation of s1 AND s2 does not equal all zeros and bitwise operation of NOT s1 AND s2 does not equal all zeros, otherwise returns 0. That is,

\[
_{\text{mm\_testnzc\_si128}} := ( ( (s1 \& s2) != 0 \&\& (~s1 \& s2) != 0 ) ? 1 : 0 )
\]

This intrinsic checks if both the CF and ZF flags are not 1 as a result of the instruction PTEST s1, s2. For example, it allows you to check that the result has both zeros and ones in s1 on positions specified as set bits in s2 (mask).

Corresponding instruction: PTEST
Packed DWORD to Unsigned WORD Intrinsic

The prototype for this Intel® Streaming SIMD Extensions (Intel® SSE4) intrinsic is in the smmintrin.h file.

```c
__m128i _mm_packus_epi32(__m128i m1, __m128i m2);
```

Converts 8 packed signed DWORDs into 8 packed unsigned WORDs, using unsigned saturation to handle overflow condition.

Corresponding instruction: PACKUSDW

Packed Compare for Equal Intrinsic

The prototype for this Intel® Streaming SIMD Extensions (Intel® SSE4) intrinsic is in the smmintrin.h file.

```c
__m128i _mm_cmpeq_epi64(__m128i a, __m128i b)
```

Performs a packed integer 64-bit comparison for equality. The intrinsic zeroes or fills with ones the corresponding parts of the result.

Corresponding instruction: PCMPEQQ

Cacheability Support Intrinsic

The prototype for this Intel® Streaming SIMD Extensions (Intel® SSE4) intrinsic is in the smmintrin.h file.

```c
extern __m128i _mm_stream_load_si128(__m128i* v1);
```

Loads _m128 data from a 16-byte aligned address (v1) to the destination operand (m128i) without polluting the caches.

Corresponding instruction: MOVNTDQA

Efficient Accelerated String and Text Processing

Overview: Efficient Accelerated String and Text Processing

The intrinsics in this section correspond to Intel® Streaming SIMD Extensions 4 (Intel® SSE4) Efficient Accelerated String and Text Processing instructions. These intrinsics include:
Packed Comparison Intrinsics for Streaming SIMD Extensions 4
Application Targeted Accelerators Intrinsics

The prototypes for these intrinsics are in the nmmintrin.h file.

Packed Compare Intrinsics

These Intel® Streaming SIMD Extensions (Intel® SSE4) intrinsics perform packed comparisons. Some of these intrinsics could map to more than one instruction; the Intel® C++ Compiler selects the instruction to generate.

The prototypes for these intrinsics are in the nmmintrin.h file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpestri</td>
<td>Packed comparison, generates index</td>
<td>PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestrm</td>
<td>Packed comparison, generates mask</td>
<td>PCMPESTRM</td>
</tr>
<tr>
<td>_mm_cmpistri</td>
<td>Packed comparison, generates index</td>
<td>PCMPISTRI</td>
</tr>
<tr>
<td>_mm_cmpistrm</td>
<td>Packed comparison, generates mask</td>
<td>PCMPISTRM</td>
</tr>
<tr>
<td>_mm_cmpestrz</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestrc</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestrs</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestro</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestra</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpistrz</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
<tr>
<td>_mm_cmpistrc</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpistrs</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
<tr>
<td>_mm_cmpistro</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
<tr>
<td>_mm_cmpistra</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
</tbody>
</table>


```cpp
int _mm_cmpestrm(__m128i src1, int len1, __m128i src2, int len2, const int mode)
```

This intrinsic performs a packed comparison of string data with explicit lengths, generating an index and storing the result in ECX.

```cpp
__m128i _mm_cmpestrm(__m128i src1, int len1, __m128i src2, int len2, const int mode)
```

This intrinsic performs a packed comparison of string data with explicit lengths, generating a mask and storing the result in XMM0.

```cpp
int _mm_cmpestri(__m128i src1, int len1, __m128i src2, int len2, const int mode)
```

This intrinsic performs a packed comparison of string data with implicit lengths, generating an index and storing the result in ECX.

```cpp
__m128i _mm_cmpestri(__m128i src1, __m128i src2, const int mode)
```

This intrinsic performs a packed comparison of string data with implicit lengths, generating a mask and storing the result in XMM0.

```cpp
int _mm_cmpestrz(__m128i src1, int len1, __m128i src2, int len2, const int mode);
```

This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if ZFlag == 1, otherwise 0.

```cpp
int _mm_cmpestrc(__m128i src1, int len1, __m128i src2, int len2, const int mode);
```

This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if CFlag == 1, otherwise 0.

```cpp
int _mm_cmpestrs(__m128i src1, int len1, __m128i src2, int len2, const int mode);
```
This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if SFlag == 1, otherwise 0.

int _mm_cmpestro(__m128i src1, int len1, __m128i src2, int len2, const int mode);

This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if OFlag == 1, otherwise 0.

int _mm_cmpestra(__m128i src1, int len1, __m128i src2, int len2, const int mode);

This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if CFlag == 0 and ZFlag == 0, otherwise 0.

int _mm_cmpistrz(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (ZFlag == 1), otherwise 0.

int _mm_cmpistrc(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (CFlag == 1), otherwise 0.

int _mm_cmpistrs(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (SFlag == 1), otherwise 0.

int _mm_cmpistro(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (OFlag == 1), otherwise 0.

int _mm_cmpistra(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (ZFlag == 0 and CFlag == 0), otherwise 0.

Application Targeted Accelerators Intrinsic

These Intel® Streaming SIMD Extensions (Intel® SSE4) intrinsics extend the capabilities of Intel architecture by adding performance-optimized, low-latency, lower power fixed-function accelerators on the processor die to benefit specific applications.

The prototypes for application targeted accelerator intrinsics are in the file nmmintrin.h.
<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding Intel® SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int __mm_popcnt_u32(unsigned int v)</code></td>
<td>Counts number of set bits in a data operation</td>
<td>POPCNT</td>
</tr>
<tr>
<td><code>int __mm_popcnt_u64(unsigned __int64 v)</code></td>
<td>Counts number of set bits in a data operation</td>
<td>POPCNT</td>
</tr>
<tr>
<td><code>unsigned int __mm_crc32_u8(unsigned int crc, unsigned char v)</code></td>
<td>Accumulates cyclic redundancy check</td>
<td>CRC32</td>
</tr>
<tr>
<td><code>unsigned int __mm_crc32_u16(unsigned int crc, unsigned short v)</code></td>
<td>Performs cyclic redundancy check</td>
<td>CRC32</td>
</tr>
<tr>
<td><code>unsigned int __mm_crc32_u32(unsigned int crc, unsigned int v)</code></td>
<td>Performs cyclic redundancy check</td>
<td>CRC32</td>
</tr>
<tr>
<td><code>unsigned __int64 __mm_crc32_u64(unsigned __int64 crc, unsigned __int64 v)</code></td>
<td>Performs cyclic redundancy check</td>
<td>CRC32</td>
</tr>
</tbody>
</table>

Starting with an initial value in the first operand, accumulates a CRC32 value for the second operand and stores the result in the destination operand. Accumulates CRC32 on r/m8.

```
unsigned int __mm_crc32_u8 (unsigned int crc, unsigned char v);
```

Starting with an initial value in the first operand, accumulates a CRC32 value for the second operand and stores the result in the destination operand. Accumulates CRC32 on r/m16.

```
unsigned int __mm_crc32_u16(unsigned int crc, unsigned short v);
```

Starting with an initial value in the first operand, accumulates a CRC32 value for the second operand and stores the result in the destination operand. Accumulates CRC32 on r/m32.

```
unsigned __int64 __mm_crc32_u64(unsigned __int64 crc, unsigned __int64 v);
```
Starting with an initial value in the first operand, accumulates a CRC32 value for the second operand and stores the result in the destination operand. Accumulates CRC32 on r/m64.
Overview: Intrinsics Performance Across Intel Architectures

This section provides a series of tables that compare intrinsics performance across architectures. Before implementing intrinsics across architectures, please note the following.

- Intrinsics may generate code that does not run on all Intel architecture processors. You should therefore use CPUID to detect the processor and generate the appropriate code.
- Implement intrinsics by processor family, not by specific processor. The guiding principle for which processor family (IA-32 architecture-based processor or Itanium® processor) the intrinsic is implemented on is performance, not compatibility. Where there is added performance on both families, the intrinsic is identical.

Intrinsics that Increase Performance Across All IA

The following intrinsics provide significant performance gain over a non-intrinsic-based code equivalent.

```
int abs(int)
long labs(long)
unsigned long _lrotl(unsigned long value, int shift)
unsigned long _lrotr(unsigned long value, int shift)
unsigned int _rotl(unsigned int value, int shift)
unsigned int _rotr(unsigned int value, int shift)
__int64 __i64_rotl(__int64 value, int shift)
__int64 __i64_rotr(__int64 value, int shift)
double fabs(double)
```
double log(double)
float logf(float)
double log10(double)
float log10f(float)
double exp(double)
float expf(float)
double pow(double, double)
float powf(float, float)
double sin(double)
float sinf(float)
double cos(double)
float cosf(float)
double tan(double)
float tanf(float)
double acos(double)
float acosf(float)
double acosh(double)
float acoshf(float)
double asin(double)
float asinf(float)
double asinh(double)
float asinhf(float)
double atan(double)
float atanf(float)
double atanh(double)
float atanhf(float)
float cabs(double)*
double ceil(double)
float ceilf(float)
double cosh(double)
float coshf(float)
float fabsf(float)
double floor(double)
float floorf(float)
double fmod(double)
float fmodf(float)
double hypot(double, double)
float hypotf(float)
double rint(double)
float rintf(float)
double sinh(double)
float sinhf(float)
float sqrtf(float)
double tanh(double)
float tanhf(float)
char * _strset(char *, _int32)
void * memcmp(const void * cs, const void * ct, size_t n)
void * memcpy(void * s, const void * ct, size_t n)
void * memset(void * s, int c, size_t n)
char * strcat(char * s, const char * ct)
int * strcmp(const char *, const char *)
char * strcpy(char *, const char *)
size_t strlen(const char * cs)
int strncpy(char *, char *, int)
int strncmp(char *, char *, int)
void * __alloca(int)
int _setjmp(jmp_buf)
_exception_code(void)
_exception_info(void)
_abnormal_termination(void)
void _enable()
void _disable()
int _bswap(int)
int _in_byte(int)
int _in_dword(int)
int _in_word(int)
int _inp(int)
int _inpd(int)
int _inpw(int)
int _out_byte(int, int)
int _out_dword(int, int)
int _out_word(int, int)
int _outp(int, int)
int _outpd(int, int)
int _outpw(int, int)
unsigned short _rotwl(unsigned short val, int count)
unsigned short _rotwr(unsigned short val, int count)

Performance of MMX(TM) Technology Intrinsics

Key to the table entries

- A = Expected to give significant performance gain over non-intrinsic-based code equivalent.
- B = Non-intrinsic-based source code would be better; the intrinsic's implementation may map directly to native instructions, but they offer no significant performance gain.
- C = Requires contorted implementation for particular microarchitecture. Will result in very poor performance if used.
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>MMX™ Technology, Intel® SSE, Intel® SSE2</th>
<th>IA-64 Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_empty</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cvtsi32_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cvtsi64_si32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_packs_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_packs_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_packs_ps16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpackhi_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpackhi_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpackhi_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpacklo_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpacklo_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpacklo_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_add_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_add_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_add_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_ps8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_ps16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_ps32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_ps64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX™ Technology, Intel® SSE, Intel® SSE2</td>
<td>IA-64 Architecture</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>_mm_sub_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_subs_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_subs_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_subs_pu8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_subs_pu16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_madd_pi16</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_mulhi_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_mullo_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sll_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_slli_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sll_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_slli_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sll_pi64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_slli_pi64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sra_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srai_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sra_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX™ Technology, Intel® SSE, Intel® SSE2</td>
<td>IA-64 Architecture</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>_mm_srai_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srl_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srl_i16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srl_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srli_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srl_s164</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srli_s164</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_and_s164</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_andnot_s164</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_or_s164</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_xor_s164</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpeq_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpeq_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpeq_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpgt_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpgt_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpgt_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setzero_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX™ Technology, Intel® SSE, Intel® SSE2</td>
<td>IA-64 Architecture</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>_mm_set_pi16</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_set_pi8</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_set1_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set1_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set1_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setr_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setr_pi16</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_setr_pi8</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

_mm_empty is implemented in IA-64 instructions as a NOP for source compatibility only.

Performance of Intel(R) Streaming SIMD Extensions Intrinsics

Regular Intel® Streaming SIMD Extensions (Intel® SSE) intrinsics work on four 32-bit single precision values. On IA-64 architecture-based systems, basic operations like add and compare require two SIMD instructions. All these operations can be executed in the same cycle so the throughput is one basic SSE operation per cycle or four 32-bit single precision operations per cycle.

Key to the table entries

- A = Expected to give significant performance gain over non-intrinsic-based code equivalent.
- B = Non-intrinsic-based source code would be better; the intrinsic's implementation may map directly to native instructions but they offer no significant performance gain.
- C = Requires contorted implementation for particular microarchitecture. Will result in very poor performance if used.
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>MMX(TM) Technology</th>
<th>Intel® SSE</th>
<th>Intel® SSE2</th>
<th>IA-64 Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_add_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_sub_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_sub_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_mul_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_mul_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_div_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_div_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_sqrt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_sqrt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_rcp_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_rcp_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_rsqrt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_rsqrt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_min_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_min_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_max_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_max_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_and_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>Intel® SSE</td>
<td>IA-64 Architecture</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
<td>------------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>_mm_andnot_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_or_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_xor_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpeq_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpeq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmplt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmplt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmple_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmple_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpgt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpgt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpge_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpge_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpneq_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpnlt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpnlt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpnle_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpnle_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

1665
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>MMX(TM) Technology</th>
<th>Intel® SSE</th>
<th>Intel® SSE2</th>
<th>IA-64 Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpngt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpngt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpnge_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpnge_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpord_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpord_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpunord_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cmpunord_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_comieg_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_comil_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_comile_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_comigt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_comige_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_comineq_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_ucomieg_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_ucomil_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_ucomile_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_ucomigt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_ucomige_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

1666
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>MMX(TM) Technology</th>
<th>Intel® SSE</th>
<th>Intel® SSE2</th>
<th>IA-64 Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_ucomineq_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtss_si32</td>
<td>N/A</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtps_pi32</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cvttss_si32</td>
<td>N/A</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cvttps_pi32</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtsi32_ss</td>
<td>N/A</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtipi32_ps</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtipi16_ps</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtpi16_ps</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_cvt8_pi32</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_cvt8_pi8</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtsi32x2_ps</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtss_pi16</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_cvtss_pi8</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_move_ss</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_shuffle_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_unpackhi_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_unpacklo_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_movehl_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>Intel® SSE Technology</td>
<td>IA-64 Architecture</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------</td>
<td>-----------------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>_mm_movelh_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_movemask_ps</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_getcsr</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_setcsr</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_loadh_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_loadl_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_load_ss</td>
<td>N/A</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>_mm_load1_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_load_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_loadu_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_loadr_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_storeh_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_storel_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_store_ss</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_store_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_store1_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_storeu_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_storer_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_set_ss</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>Intel® SSE Technology</td>
<td>IA-64 Architecture</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------</td>
<td>-----------------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>_mm_set1_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_set_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_setr_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_setzero_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_prefetch</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_stream_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_stream_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_sfence</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_extract_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_insert_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_max_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_max_pu8</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_min_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_min_pu8</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_movemask_pi8</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_mulhi_pu16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_shuffle_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>_mm_maskmove_si64</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>_mm_avg_pu8</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>Intel® SSE</td>
<td>Intel® SSE2</td>
<td>IA-64 Architecture</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------</td>
<td>------------</td>
<td>-------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>_mm_avg_pu16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sad_pu8</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>
Intrinsics for Advanced Encryption Standard Implementation

Overview: Intrinsics for Carry-less Multiplication Instruction and Advanced Encryption Standard Instructions

The Intel® C++ Compiler provides intrinsics to enable carry-less multiplication and encryption based on Advanced Encryption Standard (AES) specifications. The carry-less multiplication intrinsic corresponds to a single new instruction, \texttt{PCLMULQDQ}. The AES extension intrinsics correspond to AES extension intructions.

The AES extension instructions and the PCLMULQDQ instruction follow the same system software requirements for XMM state support and single-instruction multiple data (SIMD) floating-point exception support as Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® SSE3, Intel® Supplemental SSE3, and Intel® SSE4 extensions.

Intel®64 processors using 32nm processing technology support the AES extension instructions as well as the \texttt{PCLMULQDQ} instruction.

AES Encryption and Cryptographic Processing

AES encryption involves processing 128-bit input data (plaintext) through a finite number of iterative operation, referred to as "AES round", into a 128-bit encrypted block (ciphertext). Decryption follows the reverse direction of iterative operation using the "equivalent inverse cipher" instead of the "inverse cipher".

The cryptographic processing at each round involves two input data, one is the "state", the other is the "round key". Each round uses a different "round key". The round keys are derived from the cipher key using a "key schedule" algorithm. The "key schedule" algorithm is independent of the data processing of encryption/decryption, and can be carried out independently from the encryption/decryption phase.

The AES standard supports cipher key of sizes 128, 192, and 256 bits. The respective cipher key sizes corresponds to 10, 12, and 14 rounds of iteration.
Carry-less Multiplication Instruction and AES Extension Instructions

A single instruction, `PCLMULQDQ`, performs carry-less multiplication for two binary numbers that are up to 64-bit wide.

The AES extensions provide:

- two instructions to accelerate AES rounds on encryption (`AESENC` and `AESENCLAST`)
- two instructions for AES rounds on decryption using the equivalent inverse cipher (`AESDEC` and `AESDECLAST`)
- instructions for the generation of key schedules (`AESIMC` and `AESKEYGENASSIST`)

Detecting Support for Using Instructions

Before any application attempts to use the `PCLMULQDQ` or the AES extension instructions, it must first detect if the instructions are supported by the processor.

To detect support for the `PCLMULQDQ` instruction, your application must check the following:

```
CPUID.01H:ECX.PCLMULQDQ[bit 1] = 1.
```

To detect support for the AES extension instructions, your application must check the following:

```
```

Operating systems that support handling of the SSE state also support applications that use AES extension instruction and the `PCLMULQDQ` instruction.

Intrinsics for Carry-less Multiplication Instruction and Advanced Encryption Standard Instructions

The prototypes for the Carry-less multiplication intrinsic and the AES intrinsics are defined in the `wmmintrin.h` file.

Carry-less Multiplication Intrinsic

The single general purpose block encryption intrinsic description is provided below.

```
__m128i _mm_clmul_epi64_sil28(__m128i v1, __m128i v2, const int imm8);
```

Performs a carry-less multiplication of one quadword of `v1` by one quadword of `v2`, and returns the result. The `imm8` value is used to determine which quadwords of `v1` and `v2` should be used.

Corresponding Instruction: `PCLMULQDQ`
Advanced Encryption Standard Intrinsics

The AES intrinsics are described below.

`__m128i _mm_aesdec_si128(__m128i v, __m128i rkey);`
Performs one round of an AES decryption flow using the Equivalent Inverse Cipher operating on a 128-bit data (state) from `v` with a 128-bit round key from `rkey`.

Corresponding Instruction: AESDEC

`__m128i _mm_aesdeclast_si128(__m128i v, __m128i rkey);`
Performs the last round of an AES decryption flow using the Equivalent Inverse Cipher operating on a 128-bit data (state) from `v` with a 128-bit round key from `rkey`.

Corresponding Instruction: AESDECLAST

`__m128i _mm_aesenc_si128(__m128i v, __m128i rkey);`
Performs one round of an AES encryption flow operating on a 128-bit data (state) from `v` with a 128-bit round key from `rkey`.

Corresponding Instruction: AESENC

`__m128i _mm_aesenclast_si128(__m128i v, __m128i rkey);`
Performs the last round of an AES encryption flow operating on a 128-bit data (state) from `v` with a 128-bit round key from `rkey`.

Corresponding Instruction: AESENCLAST

`__m128i _mm_aesimc_si128(__m128i v);`
Performs the InvMixColumn transformation on a 128-bit round key from `v` and returns the result.

Corresponding Instruction: AESIMC

`__m128i _mm_aeskeygenassist_si128(__m128i ckey, const int rcon);`
Assists in AES round key generation using an 8-bit Round Constant (RCON) specified in `rcon` operating on 128 bits of data specified in `ckey` and returns the result.

Corresponding Instruction: AESKEYGENASSIST

See Also
- Intrinsics for Advanced Encryption Standard Implementation
Overview: Intrinsics for Carry-less Multiplication Instruction and Advanced Encryption Standard Instructions
Intrinsics for Converting Half Floats

Overview: Intrinsics to Convert Half Float Types

The half-float or 16-bit float is a popular type in some application domains. The half-float type is regarded as a storage type because although data is often stored as a half-float, computation is never done on values in these type. Usually values are converted to regular 32-bit floats before any computation.

Support for half-float type is restricted to just conversions to/from 32-bit floats. The main benefits of using half float type are:

- reduced storage requirements
- less consumption of memory bandwidth and cache
- accuracy and precision adequate for many applications

Half Float Intrinsics

The half-float intrinsics are provided to convert half-float values to 32-bit floats for computation purposes and conversely, 32-bit float values to half-float values for data storage purposes.

The intrinsics are translated into library calls that do the actual conversions.

The half-float intrinsics are supported on IA-32 and Intel(R) 64 architectures running on Windows*/Linux*/Mac OS* X operating systems. The minimum processor requirement is an Intel(R) Pentium 4 processor and an operating system supporting Streaming SIMD2 Extensions (SSE2) instructions.

Role of Immediate Byte in Half Float Intrinsic Operations

For all half-float intrinsics an immediate byte controls rounding mode, flush to zero, and other non-volatile set values. The format of the imm8 byte is as shown in the diagram below.

The imm8 value is used for special MXCSR overrides.
In the diagram,

- MBZ = Most significant Bit is Zero; used for error checking
- MS1 = 1 : use MXCSR RC, else use imm8.RC
- SAE = 1 : all exceptions are suppressed
- MS2 = 1 : use MXCSR FTZ/DAZ control, else use imm8.FTZ/DAZ.

The compiler passes the bits to the library function, with error checking - the most significant bit must be zero.

**Intrinsics for Converting Half Floats**

There are four intrinsics for converting half-floats to 32-bit floats and 32-bit floats to half-floats. The prototypes for these half-float conversion intrinsics are in the [emmintrin.h](#) file.

```c
float _cvtsh_ss(unsigned short x, int imm);
```
This intrinsic takes a half-float value, \( x \), and converts it to a 32-bit float value, which is returned.

```c
unsigned short _cvtss_sh(float x, int imm);
```
This intrinsic takes a 32-bit float value, \( x \), and converts it to a half-float value, which is returned.
This intrinsic takes four packed half-float values and converts them to four 32-bit float values, which are returned. The upper 64-bits of \( x \) are ignored. The lower 64-bits are taken as four 16-bit float values for conversion.

\[
\text{__m128i } \_\text{mm_cvtph}(\_\text{m128 } x, \text{ int imm});
\]

This intrinsic takes four packed 32-bit float values and converts them to four half-float values, which are returned. The upper 64-bits in the returned result are all zeros. The lower 64-bits contain the four packed 16-bit float values.

**See Also**
- Intrinsic for Converting Half Floats
- Overview: Intrinsics to convert half-float types
Part VI

Compiler Reference

Topics:

- Intel C++ Compiler Pragmas
- Intel Math Library
- Intel C++ Class Libraries
- Intel's C/C++ Language Extensions
Overview: Intel® C++ Compiler Pragmas

Pragmas are directives that provide instructions to the compiler for use in specific cases. For example, you can use the `novector` pragma to specify that a loop should never be vectorized. The keyword `#pragma` is standard in the C++ language, but individual pragmas are machine-specific or operating system-specific, and vary by compiler.

Some pragmas provide the same functionality as compiler options. Pragmas override behavior specified by compiler options.

The Intel® C++ Compiler pragmas are categorized as follows:

- **Intel-Specific Pragmas** - pragmas developed or modified by Intel to work specifically with the Intel® C++ Compiler
- **Intel Supported Pragmas** - pragmas developed by external sources that are supported by the Intel® C++ Compiler for compatibility reasons

Using Pragmas

You enter pragmas into your C++ source code using the following syntax:

```
#pragma <pragma name>
```

Individual Pragma Descriptions

Each pragma description has the following details:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Description</td>
<td>Contains a brief description of what the pragma does</td>
</tr>
<tr>
<td>Syntax</td>
<td>Contains the pragma syntax</td>
</tr>
<tr>
<td>Arguments</td>
<td>Contains a list of the arguments with descriptions</td>
</tr>
<tr>
<td>Description</td>
<td>Contains a detailed description of what the pragma does</td>
</tr>
<tr>
<td>Example</td>
<td>Contains typical usage example/s</td>
</tr>
</tbody>
</table>
Intel-Specific Pragmas

The Intel-specific C++ compiler pragmas described in the Intel-Specific Pragma reference are listed below. Click on the pragmas for a more detailed description.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_section</td>
<td>allocates variable in specified section</td>
</tr>
<tr>
<td>distribute_point</td>
<td>instructs the compiler to prefer loop distribution at the location indicated</td>
</tr>
<tr>
<td>intel_omp_task</td>
<td>specifies a unit of work, potentially executed by a different thread</td>
</tr>
<tr>
<td>intel_omp_taskq</td>
<td>specifies a unit of work, potentially executed by a different thread</td>
</tr>
<tr>
<td>ivdep</td>
<td>instructs the compiler to ignore assumed vector dependencies</td>
</tr>
<tr>
<td>loop_count</td>
<td>indicates the loop count is likely to be an integer</td>
</tr>
<tr>
<td>memref_control</td>
<td>provides a method for controlling load latency at the variable level</td>
</tr>
<tr>
<td>novector</td>
<td>specifies that the loop should never be vectorized</td>
</tr>
<tr>
<td>optimize</td>
<td>enables or disables optimizations for specific functions; provides some degree of compatibility with Microsoft's implementation of optimize pragma</td>
</tr>
<tr>
<td>optimization_level</td>
<td>enables control of optimization for a specific function</td>
</tr>
<tr>
<td>parallel/noparallel</td>
<td>facilitates auto-parallelization of an immediately following DO loop; using keyword [always] forces the compiler to auto-parallelize; noparallel pragma prevents auto-parallelization</td>
</tr>
<tr>
<td>prefetch/noprefetch</td>
<td>asserts that the data prefetches are generated or not generated for some memory references</td>
</tr>
</tbody>
</table>
Pragma swp indicates preference for loops to be software pipelined; noswp indicates the loops not to be software pipelined.

Pragma unroll/nounroll instructs the compiler the number of times to unroll/not to unroll a loop.

Pragma unroll_and_jam/nounroll_and_jam instructs the compiler to partially unroll higher loops and jam the resulting loops back together. Specifying the nounroll_and_jam pragma prevents unrolling and jamming of loops.

Pragma unused describes variables that are unused (warnings not generated).

Pragma vector indicates to the compiler that the loop should be vectorized according to the arguments: always/aligned/unaligned/nontemporal/temporal.

### Intel-specificPragma Reference

**alloc_section**

Allocates variable in specified section. Controls section attribute specification for variables.

**Syntax**

```
#pragma alloc_section(var1, var2,..., "r;attribute-list")
```

**Arguments**

- `var` variable that can be used to define a symbol in the section
- `"r;attribute-list"` a comma-separated list of attributes; defined values are: 'short' and 'long'
Description

The alloc_section pragma places the listed variables, var1, var2, etc., in the specified section. This pragma controls section attribute specification for variables. The compiler decides whether the variable, as defined by var1/var2/etc, should go to a "data", "bss", or "rdata" section.

The section name is enclosed in double quotation marks. It should be previously introduced into the program using #pragma section. The list of comma-separated variable names follows the section name after a separating comma.

All listed variables are necessarily defined before this pragma, in the same translation unit and in the same scope. The variables have static storage; their linkage does not matter in C modules, but in C++ modules they are defined with the extern "C" linkage specification.

Example

The following example illustrates how to use the pragma.

```c
#pragma alloc_section(var1, "r;short")
int var1 = 20;
#pragma alloc_section(var2, "r;short")
extern int var2;
```

distribute_point

Instructs the compiler to prefer loop distribution at the location indicated.

Syntax

: 

#pragma distribute_point

Arguments

None

Description

The distribute_point pragma is used to suggest to the compiler to split large loops into smaller ones; this is particularly useful in cases where optimizations like software-pipelining (SWP) or vectorization cannot take place due to excessive register usage.
Using `distribute_point` pragma for a loop distribution strategy enables software pipelining for the new, smaller loops in the IA-64 architecture. By splitting a loop into smaller segments, it is possible to get each smaller loop or at least one of the smaller loops to SWP or vectorize.

- When the pragma is placed inside a loop, the compiler distributes the loop at that point. All loop-carried dependencies are ignored.
- When inside the loop, pragmas cannot be placed within an `if` statement.
- When the pragma is placed outside the loop, the compiler distributes the loop based on an internal heuristic. The compiler determines where to distribute the loops and observes data dependency. If the pragmas are placed inside the loop, the compiler supports multiple instances of the pragma.

**Example**

**Example 1: Using `distribute_point` pragma outside the loop**

The following example uses the `distribute_point` pragma outside the loop.

```c
#define NUM 1024

void loop_distribution_pragma1(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] )
{
    int i;
    // Before distribution or splitting the loop
    #pragma distribute_point
    for (i=0; i< NUM; i++) {
        a[i] = a[i] + i;
        b[i] = b[i] + i;
        c[i] = c[i] + i;
        x[i] = x[i] + i;
        y[i] = y[i] + i;
        z[i] = z[i] + i;
    }
}
```
Example 2: Using distribute_point pragma inside the loop

The following example uses the `distribute_point` pragma inside the loop.

```c
#define NUM 1024

void loop_distributionPragma2(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] )
{
    int i;

    // After distribution or splitting the loop.
    for (i=0; i< NUM; i++) {
        a[i] = a[i] +i;
        b[i] = b[i] +i;
        c[i] = c[i] +i;
        #pragma distribute_point
        x[i] = x[i] +i;
        y[i] = y[i] +i;
        z[i] = z[i] +i;
    }
}
```

Example 3: Using distribute_point pragma inside and outside the loop
The following example shows how to use the distribute_point pragma, first outside the loop and then inside the loop.

```c
void dist1(int a[], int b[], int c[], int d[])
{
    #pragma distribute_point
    // Compiler will automatically decide where to distribute. Data dependency is observed.
    for (int i=1; i<1000; i++) {
        b[i] = a[i] + 1;
        c[i] = a[i] + b[i];
        d[i] = c[i] + 1;
    }
}

void dist2(int a[], int b[], int c[], int d[])
{
    for (int i=1; i<1000; i++) {
        b[i] = a[i] + 1;
        #pragma distribute_point
        // Distribution will start here, ignoring all loop-carried dependency.
        c[i] = a[i] + b[i];
        d[i] = c[i] + 1;
    }
}
```
**intel_omp_task**

*Specifies a unit of work, potentially executed by a different thread.*

**Syntax**

```
#pragma intel_omp_task
: [clause[,clause]...] :
structured-block
```

*where clause can be any of the following:*

- `private( variable-list )`
- `captureprivate( variable-list )`

**Arguments**

- **private( variable-list )** The `private` clause creates a private, default-constructed version for each object in `variable-list` for the task. The original object referenced by the variable has an indeterminate value upon entry to the construct, must not be modified within the dynamic extent of the construct, and has an indeterminate value upon exit from the construct.

- **captureprivate( variable-list )** The `captureprivate` clause creates a private, copy-constructed version for each object in `variable-list` for the task at the time the task is enqueued. The original object referenced by each variable retains its value but must not be modified within the dynamic extent of the task construct.

**Description**

The `intel_omp_task` pragma specifies a unit of work, potentially executed by a different thread.
Example

For an example on how to use `intel_omp_task pragma` see topic Workqueuing Example Function in Optimizing Applications>Using Parallelism: OpenMP* Support>Intel(R) Workqueuing Model.

See Also
• Intel-specific Pragma Reference
• Workqueing Constructs

`intel_omp_taskq`

Specifies an environment for the while loop in which to enqueue the units of work specified by the enclosed task pragma.

Syntax

```c
#pragma intel_omp_taskq
```

where clause can be any of the following:

- private (variable-list)
- firstprivate (variable-list)
- lastprivate (variable-list)
- reduction (operator : variable-list)
- ordered
- nowait
Arguments

`private(variable-list)` The `private` clause creates a private, default-constructed version for each object in `variable-list` for the `taskq`. It also implies `captureprivate` on each enclosed task. The original object referenced by each variable has an indeterminate value upon entry to the construct, must not be modified within the dynamic extent of the construct, and has an indeterminate value upon exit from the construct.

`firstprivate` The `firstprivate` clause creates a private, copy-constructed version for each object in `variable-list` for the `taskq`. It also implies `captureprivate` on each enclosed task. The original object referenced by each variable must not be modified within the dynamic extent of the construct and has an indeterminate value upon exit from the construct.

`lastprivate` The `lastprivate` clause creates a private, default-constructed version for each object in `variable-list` for the `taskq`. It also implies `captureprivate` on each enclosed task. The original object referenced by each variable has an indeterminate value upon entry to the construct, must not be modified within the dynamic extent of the construct, and is copy-assigned the value of the object from the last enclosed task after that task completes execution.

`reduction(operator : variable-list)` The `reduction` clause performs a reduction operation with the given operator in enclosed task constructs for each object in `variable-list`. `operator` and `variable-list` are defined the same as in the OpenMP Specifications.

`ordered` The `ordered` clause performs ordered constructs in enclosed task constructs in original sequential execution order. The `taskq` directive, to which the `ordered` is bound, must have an `ordered` clause present.

`nowait` The `nowait` clause removes the implied barrier at the end of the `taskq`. Threads may exit the `taskq` construct before completing all the `task` constructs queued within it.
Description

The `intel_omp_taskq` pragma specifies the environment within which the enclosed units of work (tasks) are to be executed. From among all the threads that encounter a `taskq` pragma, one is chosen to execute it initially.

Conceptually, the `taskq` pragma causes an empty queue to be created by the chosen thread, and then the code inside the `taskq` block is executed single-threaded. All the other threads wait for work to be enqueued on the conceptual queue.

The `task` pragma specifies a unit of work, potentially executed by a different thread. When a `task` pragma is encountered lexically within a `taskq` block, the code inside the `task` block is conceptually enqueued on the queue associated with the `taskq`. The conceptual queue is disbanded when all work enqueued on it finishes, and when the end of the `taskq` block is reached.

Example

For an example on how to use `taskq` pragma see topic Workqueuing Example Function in Optimizing Applications>Using Parallelism: OpenMP* Support>Intel(R) Workqueing Model.

See Also

- Intel-specific Pragma Reference
- Workqueing Constructs

`ivdep`

*Instructs the compiler to ignore assumed vector dependencies.*

Syntax

```
#pragma ivdep
```

Arguments

None
Description

The `ivdep` pragma instructs the compiler to ignore assumed vector dependencies. To ensure correct code, the compiler treats an assumed dependence as a proven dependence, which prevents vectorization. This pragma overrides that decision. Use this pragma only when you know that the assumed loop dependencies are safe to ignore.

**NOTE.** The proven dependencies that prevent vectorization are not ignored, only assumed dependencies are ignored.

When an `ivdep` pragma is specified for applications designed to run on IA-64 architectures, the `-ivdep-parallel` (Linux* systems) or `/Qivdep-parallel` (Windows* systems) option indicates there is no loop-carried memory dependency in the loop. This technique is useful for some sparse matrix applications. See Example 2.

Example

Example 1

The loop in this example will not vectorize without the `ivdep` pragma, since the value of \( k \) is not known; vectorization would be illegal if \( k < 0 \).

```c
void ignore_vec_dep(int *a, int k, int c, int m)
{
    #pragma ivdep
    for (int i = 0; i < m; i++)
        a[i] = a[i + k] * c;
}
```

The pragma binds only the `for` loop contained in current function. This includes a `for` loop contained in a sub-function called by the current function.

Example 2
The following loop requires the parallel option in addition to the ivdep pragma to indicate there is no loop-carried dependencies:

```c
#pragma ivdep
for (i=1; i<n; i++)
{
    e[ix[2][i]] = e[ix[2][i]]+1.0;
    e[ix[3][i]] = e[ix[3][i]]+2.0;
}
```

**Example 3**

The following loop requires the parallel option in addition to the ivdep pragma to ensure there is no loop-carried dependency for the store into a().

```c
#pragma ivdep
for (j=0; j<n; j++)
{
    a[b[j]] = a[b[j]] + 1;
}
```

**See Also**

- Intel-specific Pragma Reference
- `vector`
- `novector`

In addition to the ivdep pragma, the vector pragma can be used to override the efficiency heuristics of the vectorizer.

**loop_count**

*Specifies the iterations for the for loop.*

**Syntax**

```
#pragma loop_count (n)
#pragma loop_count = n
```
or
:
#pragma loop_count (n1[, n2]...)
#pragma loop_count = n1[, n2]...
or
:
#pragma loop_count min(n), max(n), avg(n)
#pragma loop_count min=n, max=n, avg=n

Arguments

(n) or =n

Non-negative integer value. The compiler will attempt to iterate the next loop the number of times specified in \( n \); however, the number of iterations is not guaranteed.

(n1[,n2]...) or =
n1[,n2]...

Non-negative integer values. The compiler will attempt to iterate the next loop the number of times specified by \( n1 \) or \( n2 \), or some other unspecified number of times. This behavior allows the compiler some flexibility in attempting to unroll the loop. The number of iterations is not guaranteed.

min(n), max(n),
avg(n) or min=n,
max=n, avg=n

Non-negative integer values. Specify one or more in any order without duplication. The compiler insures the next loop iterates for the specified maximum, minimum, or average number (\( n1 \)) of times. The specified number of iterations is guaranteed for min and max.

Description

The \texttt{loop_count} pragma affects heuristics in vectorization, loop-transformations, and software pipelining (IA-64 architecture). The pragma specifies the minimum, maximum, or average number of iterations for a \texttt{for} loop. In addition, a list of commonly occurring values can be specified to help the compiler generate multiple versions and perform complete unrolling.

You can specify more than one pragma for a single loop; however, do not duplicate the pragma.

Example

Example 1: Using \texttt{#pragma loop_count (n)}
The following example illustrates how to use the pragma to iterate through the loop and enable software pipelining on IA-64 architectures.

```c
void loop_count(int a[], int b[])
{
    #pragma loop_count (10000)
    for (int i=0; i<1000; i++)
        a[i] = b[i] + 1.2;
}
```

**Example 2: Using `#pragma loop_count min(n), max(n), avg(n)`**

The following example illustrates how to use the pragma to iterate through the loop a minimum of three, a maximum of ten, and average of five times.

```c
#include <stdio.h>
int i;
int main()
{
    #pragma loop_count min(3), max(10), avg(5)
    for (i=1;i<=15;i++)
        printf("i=%d\n",i);
}
```

**memref_control**

*Provides a method to control load latency and temporal locality at the variable level.*

**Syntax**

```
#pragma memref_control [name1[:<locality>][:<latency>]],[name2...]
```
Arguments

name1, name2

Specifies the name of array or pointer. You must specify at least one name; however, you can specify names with associated locality and latency values.

locality

An optional integer value that indicates the desired cache level to store data for future access. This will determine the load/store hint (or prefetch hint) to be used for this reference. The value can be one of the following:

- |1 = 0
- |2 = 1
- |3 = 2
- mem = 3

To use this argument, you must also specify name.

latency

An optional integer value that indicates the load (or the latency that has to be overlapped if a prefetch is issued for this address). The value can be one of the following:

- 1_latency = 0
- 2_latency = 1
- 3_latency = 2
- mem_latency = 3

To use this argument, you must also specify name and locality.

Description

The memref_control pragma is supported on Itanium® processors only. This pragma provides a method for controlling load latency and temporal locality at the variable level. The memref_control pragma allows you to specify locality and latency at the array level. For example, using this pragma allows you to control the following:

- The location (cache level) to store data for future access.
- The most appropriate latency value to be used for a load, or the latency that has to be overlapped if a prefetch is issued for this reference.
When you specify source-level and the data locality information at a high level for a particular data access, the compiler decides how best to use this information. If the compiler can prefetch profitably for the reference, then it issues a `prefetch` with a distance that covers the specified latency specified and then schedules the corresponding load with a smaller latency. It also uses the hints on the prefetch and load appropriately to keep the data in the specified cache level.

If the compiler cannot compute the address in advance, or decides that the overheads for prefetching are too high, it uses the specified latency to separate the load and its use (in a pipelined loop or a Global Code Scheduler loop). The hint on the load/store will correspond to the cache level passed with the locality argument.

You can use this with the `prefetch` and `noprefetch` to further tune the hints and prefetch strategies. When using the `memref_control` with `noprefetch`, keep the following guidelines in mind:

- Specifying `noprefetch` along with the `memref_control` causes the compiler to not issue prefetches; instead the latency values specified in the `memref_control` are used to schedule the load.
- There is no ordering requirements for using the two pragmas together. Specify the two pragmas in either order as long as both are specified consecutively just before the loop where it is to be applied. Issuing a `prefetch` with one hint and loading it later using a different hint can provide greater control over the hints used for specific architectures.
- `memref_control` is handled differently from the `prefetch` or `noprefetch`. Even if the load cannot be prefetched, the reference can still be loaded using a non-default load latency passed to the `latency` argument.

**Example**

**Example 1: Using `#pragma memref_control when prefetching is not possible`**
The following example illustrates a case where the address is not known in advance, so prefetching is not possible. The compiler, in this case, schedules the loads of the tab array with an L3 load latency of 15 cycles (inside a software pipelined loop or GCS loop).

```c
#pragma memref_control tab : l2 : l3_latency
for (i=0; i<n; i++)
{
    x = <generate 64 random bits inline>;
    dum += tab[x&mask]; x>>=6;
    dum += tab[x&mask]; x>>=6;
    dum += tab[x&mask]; x>>=6;
}
```

Example 2: Using #pragma memref_control with prefetch and noprefetch pragmas [sparse matrix]

The following example illustrates one way of using memref_control, prefetch, and noprefetch together.

```c
if( size <= 1000 ) {
  v#pragma noprefetch cp, vp
  #pragma memref_control x:12:13_latency

  #pragma noprefetch yp, bp, rp
  #pragma noprefetch xp
  for (iii=0; iii<rag1m0; iii++) {
```
if( ip < rag2 ) {
    sum -= vp[ip]*x[cp[ip]];
    ip++;
} else {
    xp[i] = sum*yp[i];
    i++;
    sum = bp[i];
    rag2 = rp[i+1];
}

xp[i] = sum*yp[i];
} else {

#pragma prefetch cp, vp
#pragma memref_control x:l2:mem_latency

#pragma prefetch yp, bp, rp
#pragma noprefetch xp

    for (iii=0; iii<rag1m0; iii++) {

if( ip < rag2 ) {
    sum -= vp[ip]*x[cp[ip]];
    ip++;
} else {
    xp[i] = sum*yp[i];
    i++;
    sum = bp[i];
    rag2 = rp[i+1];
}

xp[i] = sum*yp[i];

**novector**

Specifies that the loop should never be vectorized.

**Syntax**

```
#pragma novector
```

**Arguments**

None

**Description**

The novector pragma specifies that a particular loop should never be vectorized, even if it is legal to do so. When avoiding vectorization of a loop is desirable (when vectorization results in a performance regression rather than improvement), the novector pragma can be used in the source text to disable vectorization of a loop. This behavior is in contrast to the vector always pragma.

**Example**

**Example: Using the novector pragma**
When you know the trip count ($ub - lb$) is too low to make vectorization worthwhile, you can use novector to tell the compiler not to vectorize, even if the loop is considered vectorizable.

```c
void foo(int lb, int ub)
{
    #pragma novector
    for(j=lb; j<ub; j++)
    {
        a[j]=a[j]+b[j];
    }
}
```

**See Also**
- Intel-specific Pragma Reference
- vector pragma

**optimize**

*Enables or disables optimizations for specific functions.*

**Syntax**

```
#pragma optimize("", on|off)
```

**Arguments**

The compiler ignores first argument values. Valid second arguments for `optimize` are given below.

- `off`  
  disables optimization
- `on`   
  enables optimization
**Description**

The `optimize` pragma is used to enable or disable optimizations for specific functions. Specifying `#pragma optimize("", off)` disables optimization until either the compiler finds a matching `#pragma optimize("", on)` statement or until the compiler reaches the end of the translation unit.

**Example**

**Example 1: Disabling optimization for a single function using the `#pragma optimize`**

In the following example, optimizations are disabled for the `alpha()` function but not for `omega()`.

```c++
#pragma optimize("", off)
alpha() {
...
}
#pragma optimize("", on)
omega() {
...
}
```

**Example 2: Disabling optimization for all functions using the `#pragma optimize`**

In the following example, optimizations are disabled for both the `alpha()` and `omega()` functions.

```c++
#pragma optimize("", off)
alpha() {
...
}
omega() {
...
}
**optimization_level**

*Controls optimization for one function or all functions after its first occurrence.*

**Syntax**

```c
#pragma [intel|GCC] optimization_level n
```

**Arguments**

- `intel|GCC` indicates the interpretation to use
- `n` an integer value specifying an optimization level; valid values are:
  - 0 same optimizations as `-O0`
  - 1 same optimizations as `-O1`
  - 2 same optimizations as `-O2`
  - 3 same optimizations as `-O3`

**NOTE.** For more information on the optimizations levels, see Enabling Automatic Optimizations in Optimizing Applications>Using Compiler Optimizations

**Description**

The `optimization_level` pragma is used to restrict optimization for a specific function while optimizing the remaining application using a different, higher optimization level. For example, if you specify `-O3` (Linux* and Mac OS* X systems) for the application and specify `#pragma optimization_level 1`, the marked function will be optimized at the `-O1` option level, while the remaining application will be optimized at the higher level.

In general, the pragma optimizes the function at the level specified as `n`; however, certain compiler optimizations, like Inter-procedural Optimization (IPO), are not enabled or disabled during translation unit compilation. For example, if you enable IPO and a specific optimization level, IPO is enabled even for the function targeted by this pragma; however, IPO might not be fully implemented regardless of the optimization level specified at the command line. The reverse is also true.
Scope of optimization restriction

On Linux* and Mac OS* X systems, the scope of the optimization restriction can be affected by arguments passed to the -pragma-optimization-level compiler option as explained in the following table.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma intel optimization_level n</td>
<td>Applies pragma only to the next function, using the specified optimization level, regardless of the argument passed to the -pragma-optimization-level option.</td>
</tr>
<tr>
<td>#pragma GCC optimization_level n</td>
<td>Applies pragma to all subsequent functions, using the specified optimization level, regardless of the argument passed to the -pragma-optimization-level option.</td>
</tr>
<tr>
<td>or #pragma GCC optimization_level reset</td>
<td>Specifying reset reverses the effect of the most recent #pragma GCC optimization_level statement, by returning to the optimization level previously specified.</td>
</tr>
<tr>
<td>#pragma optimization_level n</td>
<td>Applies either the intel or GCC interpretation. Interpretation depends on argument passed to the -pragma-optimization-level option.</td>
</tr>
</tbody>
</table>

NOTE. On Windows* systems, the pragma always uses the intel interpretation; the pragma is applied only to the next function.

Using the intel interpretation of #pragma optimization_level

Place the pragma immediately before the function being affected.

Example:

```cpp
#pragma intel optimization_level 1

gamma() {
  ...
}
```

Using the GCC* interpretation of #pragma optimization_level

Place the pragma in any location prior to the functions being affected.
**parallel/noparallel**

The **parallel** pragma helps the compiler to resolve dependencies thereby facilitating auto-parallelization of an immediately following FOR loop. The **noparallel** pragma prevents auto-parallelization of an immediately following FOR loop.

**Syntax**

: 

#pragma parallel
#pragma parallel {always}
#pragma noparallel

**Arguments**

*always*  

Overrides compiler heuristics that estimate the likelihood that parallelization of a loop would increase performance. Using this keyword allows a loop to be parallelized even when the compiler estimates that parallelization might not improve performance.

**Description**

The **parallel** pragma instructs the compiler to ignore potential dependencies that it assumes could exist and which would prevent correct parallelization in the immediately following loop. However, if dependencies are proven, they are not ignored.

Using **#pragma parallel always** overrides the compiler heuristics that estimate the likelihood that parallelization of a loop would increase performance. It allows a loop to be parallelized even when the compiler estimates that parallelization might not improve performance.

The **#pragma noparallel** prevents autoparallelization of immediately following DO loops.

These pragmas take effect only if autoparallelization is enabled by the switch /Qparallel (Windows* operating system) or -parallel (Linux* or Mac OS* X operating systems).
CAUTION. The #pragma parallel always should be used with care. Overriding the heuristics of the compiler should only be done if you are absolutely sure the parallelization will improve performance.

Example
The following example illustrates how to use the #pragma parallel.

```c++
void add(int k, float *a, float *b)
{
    #pragma parallel
    for (int i = 0; i < 10000; i++)
    {
        a[i] = a[i+k] + b[i];
    }
}
```

See Also
- Intel-specific Pragma Reference
- Programming for Multithread Platform Consistency in Optimizing Applications>Using Parallelism: Automatic Parallelization

prefetch/noprefetch
Invites the compiler to issue data prefetches from memory (prefetch) or disables data prefetching (noprefetch).

Syntax:
```c++
#pragma prefetch
#pragma prefetch [var1 [: hint1 [: distance1]] [, var2 [: hint2 [: distance2]]]...]
#pragma noprefetch [var1 [, var2]...]
```

Arguments
- `var` Optional memory reference (data to be prefetched)
**hint**

Optional hint to the compiler to specify the type of prefetch. Possible values are the constants defined in the header `xmmintrin.h`:

- `_MM_HINT_T0` - for integer data that will be reused
- `_MM_HINT_NT1` - for integer and floating point data that will be reused from L2 cache
- `_MM_HINT_NT2` - for data that will be reused from L3 cache
- `_MM_HINT_NTA` - for data that will not be reused

To use this argument, you must also specify `var`.

**distance**

Optional integer argument with a value greater than 0. It indicates the number of loop iterations ahead of which a prefetch is issued, before the corresponding load or store instruction. To use this argument, you must also specify `var` and `hint`.

**Description**

The `prefetch` pragma is supported by Intel® Itanium® processors only. The `prefetch` pragma hints to the compiler to generate data prefetches for some memory references. This affects the heuristics used in the compiler. Prefetching data can minimize the effects of memory latency.

If you specify `#pragma prefetch` with no arguments, all arrays accessed in the immediately following loop are prefetched.

If the loop includes the expression `A(j)`, placing `#pragma prefetch A` in front of the loop asks the compiler to insert prefetches for `A(j + d)` within the loop. Here, `d` is the number of iterations ahead of which to prefetch the data, and is determined by the compiler.

The `prefetch` pragma affects the immediately following loop provided that the compiler general optimization level is `-O1` (Linux* operating systems) or `/O1` (Windows* operating systems) or higher. Remember that `-O2` or `/O2` is the default optimization level.

The `noprefetch` pragma is also supported by Intel® Itanium® processors only. This pragma hints to the compiler not to generate data prefetches for some memory references. This affects the heuristics used in the compiler.

**Example**

**Example 1: Using noprefetch and prefetch pragmas**
The following example demonstrates how to use the \texttt{noprefetch} and \texttt{prefetch} pragmas together:
\begin{verbatim}
#pragma noprefetch b

#pragma prefetch a
for(i=0; i<m; i++)
{
  a[i]=b[i]+1;
}
\end{verbatim}

\textbf{Example 2: Using \texttt{noprefetch} and \texttt{prefetch} pragmas}

The following is yet another example of how to use the \texttt{noprefetch} and \texttt{prefetch} pragmas:
\begin{verbatim}
for (i=i0; i!=i1; i+=is) {

  float sum = b[i];
  int ip = srow[i];
  int c = col[ip];

  #pragma noprefetch col
  #pragma prefetch value:1:80
  #pragma prefetch x:1:40

  for(; ip<srow[i+1]; c=col[++ip])
    sum -= value[ip] * x[c];
  y[i] = sum;
}
\end{verbatim}

\textbf{Example 3: Using \texttt{noprefetch}, \texttt{prefetch}, \texttt{memref\_control} pragmas}
The following example, which is for IA-64 architecture only, demonstrates how to use the `prefetch`, `noprefetch`, and `memref_control` pragmas together:

```c
#define SIZE 10000
int prefetch(int *a, int *b)
{
    int i, sum = 0;
    #pragma memref_control a:l2
    #pragma noprefetch a
    #pragma prefetch b
    for (i = 0; i<SIZE; i++)
        sum += a[i] * b[i];
    return sum;
}
#include <stdio.h>
int main()
{
    int i, arr1[SIZE], arr2[SIZE];
    for (i = 0; i<SIZE; i++) {
        arr1[i] = i;
        arr2[i] = i;
    }
    printf("Demonstrating the use of prefetch, noprefetch,\n"        "and memref_control pragma together.\n")
        prefetch(arr1, arr2);
    return 0;
}
```

**See Also**
- Intel-specific Pragma Reference
- `memref_control pragma`
swp/noswp

Indicates a preference for loops to be software pipelined or not pipelined.

Syntax:

```c
#pragma swp
#pragma noswp
```

Arguments

None

Description

The `swp` pragma indicates a preference for loops to be software pipelined. The pragma does not help data dependency, but overrides heuristics based on profile counts or unequal control flow.

The software pipelining optimization triggered by the `swp` pragma applies instruction scheduling to certain innermost loops, allowing instructions within a loop to be split into different stages, allowing increased instruction-level parallelism.

This strategy can reduce the impact of long-latency operations, resulting in faster loop execution. Loops chosen for software pipelining are always the innermost loops that do not contain procedure calls that are not inlined. Because the optimizer no longer considers fully unrolled loops as innermost loops, fully unrolling loops can allow an additional loop to become the innermost loop.

The `noswp` pragma is used to instruct the compiler not to software pipeline that loop. This may be advantageous for loops that iterate few times, as pipelining introduces overhead in executing the prolog and epilog code sequences.

Example

Example: Using swp pragma
The following example demonstrates one way of using the pragma to instruct the compiler to attempt software pipelining.

```c
void swp(int a[], int b[])
{
    #pragma swp
    for (int i = 0; i < 100; i++)
        if (a[i] == 0)
            b[i] = a[i] + 1;
        else
            b[i] = a[i] * 2;
}
```

**See Also**
- Intel-specific Pragma Reference
- Loop Unrolling Options
- Optimizer Report Generation

**unroll/nounroll**

*Indicates to the compiler to unroll or not to unroll a counted loop.*

**Syntax**

```c
#pragma unroll
#pragma unroll(n)
#pragma nounroll
```

**Arguments**

`n` is the unrolling factor representing the number of times to unroll a loop; it is an integer constant from 0 through 255.
Description

The `unroll[n]` pragma tells the compiler how many times to unroll a counted loop.

The `unroll` pragma must precede the `FOR` statement for each `FOR` loop it affects. If `n` is specified, the optimizer unrolls the loop `n` times. If `n` is omitted or if it is outside the allowed range, the optimizer assigns the number of times to unroll the loop.

This pragma is supported only when option O3 is set. The `unroll` pragma overrides any setting of loop unrolling from the command line.

The pragma can be applied for the innermost loop nest as well as for the outer loop nest. If applied to outer loop nests, the current implementation supports complete outer loop unrolling. The loops inside the loop nest are either not unrolled at all or completely unrolled. The compiler generates correct code by comparing `n` and the loop count.

When unrolling a loop increases register pressure and code size it may be necessary to prevent unrolling of a loop. In such cases, use the `nounroll` pragma. The `nounroll` pragma instructs the compiler not to unroll a specified loop.

Example

Example 1: Using unroll pragma for innermost loop unrolling

```cpp
void unroll(int a[], int b[], int c[], int d[])
{
    #pragma unroll(4)
    for (int i = 1; i < 100; i++) {
        b[i] = a[i] + 1;
        d[i] = c[i] + 1;
    }
}
```

Example 2: Using unroll pragma for outer loop unrolling
In Example 2, placing the `#pragma unroll` before the first `for` loop causes the compiler to unroll the outer loop completely. If a `#pragma unroll` is placed before the inner `for` loop as well as before the outer `for` loop, the compiler ignores the inner `for` loop unroll pragma. If the `#pragma unroll` is placed only for the innermost loop, the compiler unrolls the innermost loop according to some factor.

```c
int m = 0;
int dir[4]= {1,2,3,4};
int data[10];
#pragma unroll (4) // outer loop unrolling
for (int i = 0; i < 4; i++)
{
    for (int j = dir[i]; data[j]==N ; j+=dir[i])
        m++;
}
```

**See Also**
- Intel-specific Pragma Reference
- Loop Unrolling Options
- Optimizer Report Generation
- Applying Optimization Strategies

### unroll_and_jam/nounroll_and_jam

*Hints to the compiler to enable or disable loop unrolling and jamming. These pragmas can only be applied to iterative `FOR` loops.*

**Syntax**

```c

#pragma unroll_and_jam
#pragma unroll_and_jam (n)
#pragma nounroll_and_jam
```
Arguments

\( n \)

is the unrolling factor representing the number of times to unroll a loop; it is an integer constant from 0 through 255

Description

The `unroll_and_jam` pragma partially unrolls one or more loops higher in the nest than the innermost loop and fuses/jams the resulting loops back together. This transformation allows more reuses in the loop.

This pragma is not effective on innermost loops. Ensure that the immediately following loop is not the innermost loop after compiler-initiated interchanges are completed.

Specifying this pragma is a hint to the compiler that the unroll and jam sequence is legal and profitable. The compiler enables this transformation whenever possible.

The `unroll_and_jam` pragma must precede the `FOR` statement for each `FOR` loop it affects. If \( n \) is specified, the optimizer unrolls the loop \( n \) times. If \( n \) is omitted or if it is outside the allowed range, the optimizer assigns the number of times to unroll the loop. The compiler generates correct code by comparing \( n \) and the loop count.

This pragma is supported only when compiler option `O3` is set. The `unroll_and_jam` pragma overrides any setting of loop unrolling from the command line.

When unrolling a loop increases register pressure and code size it may be necessary to prevent unrolling of a nested/imperfect nested loop. In such cases, use the `nounroll_and_jam` pragma. The `nounroll_and_jam` pragma hints to the compiler not to unroll a specified loop.
Example

Example: Using unroll_and_jam pragma

```c
int a[10][10];
int b[10][10];
int c[10][10];
int d[10][10];
void unroll(int n)
{
    int i, j, k;
    #pragma unroll_and_jam (6)
    for (i = 1; i < n; i++) {
        #pragma unroll_and_jam (6)
        for (j = 1; j < n; j++) {
            for (k = 1; k < n; k++) {
                a[i][j] += b[i][k]*c[k][j];
            }
        }
    }
}
```

See Also
- Intel-specific Pragma Reference
- Loop Unrolling Options
- Optimizer Report Generation
- Applying Optimization Strategies
unused

Describes variables that are unused (warnings not generated).

Syntax

#:pragma unused

Arguments
None

Description
The unused pragma is implemented for compatibility with Apple* implementation of GCC.

vector {always|aligned|unaligned|nontemporal|temporal}

Indicates to the compiler that the loop should be vectorized according to the argument keywords always/aligned/unaligned/nontemporal/temporal.

Syntax

#:pragma vector {always|aligned|unaligned|nontemporal|temporal}

#:pragma vector nontemporal[(var1[, var2, ...])]

Arguments

always

instructs the compiler to override any efficiency heuristic during the decision to vectorize or not, and vectorize non-unit strides or very unaligned memory accesses; controls the vectorization of the subsequent loop in the program.

aligned

instructs compiler to use aligned data movement instructions for all array references when vectorizing.
unaligned instructs compiler to use unaligned data movement instructions for all array references when vectorizing

nontemporal directs the compiler to use non-temporal (that is, streaming) stores on systems based on IA-32 and Intel(R) 64 architectures; optionally takes a comma separated list of variables

temporal directs the compiler to use temporal (that is, non-streaming) stores on systems based on IA-32 and Intel(R) 64 architectures

Description

The vector pragma indicates that the loop should be vectorized, if it is legal to do so, ignoring normal heuristic decisions about profitability. The vector pragma takes several argument keywords to specify the kind of loop vectorization required. These keywords are aligned, unaligned, always, and nontemporal.

Using aligned/unaligned keywords

When the aligned/unaligned argument keyword is used with this pragma, it indicates that the loop should be vectorized using aligned/unaligned data movement instructions for all array references. Specify only one argument keyword: aligned or unaligned.

CAUTION. If you specify aligned as an argument, you must be sure that the loop is vectorizable using this pragma. Otherwise, the compiler generates incorrect code.

Using always keyword

When the always argument keyword is used, the pragma controls the vectorization of the subsequent loop in the program. As the compiler does not apply the vector pragma to nested loops, each nested loop needs a preceding pragma statement. Place the pragma before the loop control statement.

NOTE. The pragma vector{always|aligned|unaligned} should be used with care. Overriding the efficiency heuristics of the compiler should only be done if the programmer is absolutely sure that vectorization will improve performance. Furthermore, instructing the compiler to implement all array references with aligned data movement instructions will cause a run-time exception in case some of the access patterns are actually unaligned.
Using nontemporal/temporal keywords

The nontemporal and temporal argument keywords are used to control how the "stores" of
register contents to storage are performed (streaming versus non-streaming) on systems based
on IA-32 and Intel® 64 architectures.

By default, the compiler automatically determines whether a streaming store should be used
for each variable.

Streaming stores may cause significant performance improvements over non-streaming stores
for large numbers on certain processors. However, the misuse of streaming stores can
significantly degrade performance.

Example

Example 1: Using pragma vector aligned

The loop in the following example uses the aligned argument keyword to request that the loop
be vectorized with aligned instructions, as the arrays are declared in such a way that the
compiler could not normally prove this would be safe to do so.

```cpp
void vec_aligned(float *a, int m, int c)
{
    int i;
    // Instruct compiler to ignore assumed vector dependencies.
    #pragma vector aligned
    for (i = 0; i < m; i++)
        a[i] = a[i] * c;
    // Alignment unknown but compiler can still align.
    for (i = 0; i < 100; i++)
        a[i] = a[i] + 1.0f;
}
```

Example 2: Using pragma vector always
The following example illustrates how to use the vector always pragma.

```c
void vec_always(int *a, int *b, int m)
{
    #pragma vector always
    for(int i = 0; i <= m; i++)
        a[32*i] = b[99*i];
}
```

**Example 3a: Using pragma vector nontemporal**

A float-type loop together with the generated assembly are shown in the following example. For large $N$, significant performance improvements result on Pentium 4 systems over a non-streaming implementation.

```c
float a[1000];
void foo(int N){
    int i;
    #pragma vector nontemporal
    for (i = 0; i < N; i++) {
        a[i] = 1;
    }
}
```

**Example 3b: Using ASM code for the loop body**

```
    .B1.2:
    movntps XMMWORD PTR _a[eax], xmm0

    movntps XMMWORD PTR _a[eax+16], xmm0

    add eax, 32

    cmp eax, 4096

    jl .B1.2
```

**Example 4: Using pragma vector nontemporal with variables**
The following example illustrates how to use the `#pragma vector nontemporal` with variables for implementing streaming stores.

declare A[1000];
declare B[1000];

void foo(int n){
    int i;
    #pragma vector nontemporal (A, B)
    for (i=0; i<n; i++){
        A[i] = 0;
        B[i] = i;
    }
}

### Intel Supported Pragmas

The Intel® C++ Compiler supports the following pragmas:

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_text</td>
<td>names the code section where the specified function definitions are to reside</td>
</tr>
<tr>
<td>auto_inline</td>
<td>excludes any function defined within the range where off is specified from being considered as candidates for automatic inline expansion</td>
</tr>
<tr>
<td>bss_seg</td>
<td>indicates to the compiler the segment where uninitialized variables are stored in the .obj file</td>
</tr>
<tr>
<td>check_stack</td>
<td>on argument indicates that stack checking should be enabled for functions that follow and off argument indicates that stack checking should be disabled for functions that follow.</td>
</tr>
<tr>
<td>code_seg</td>
<td>specifies a code section where functions are to be allocated</td>
</tr>
<tr>
<td>comment</td>
<td>places a comment record into an object file or executable file</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>component</td>
<td>controls collecting of browse information or dependency information from within source files</td>
</tr>
<tr>
<td>conform</td>
<td>specifies the run-time behavior of the /Zc:forScope compiler option</td>
</tr>
<tr>
<td>const_seg</td>
<td>specifies the segment where functions are stored in the .obj file</td>
</tr>
<tr>
<td>data_seg</td>
<td>specifies the default section for initialized data</td>
</tr>
<tr>
<td>deprecated</td>
<td>indicates that a function, type, or any other identifier may not be supported in a future release or indicates that a function, type, or any other identifier should not be used any more</td>
</tr>
<tr>
<td>poison</td>
<td>labels the identifiers you want removed from your program; an error results when compiling a &quot;poisoned&quot; identifier; #pragma POISON is also supported.</td>
</tr>
<tr>
<td>float_control</td>
<td>specifies floating-point behavior for a function</td>
</tr>
<tr>
<td>fp_contract</td>
<td>allows or disallows the implementation to contract expressions</td>
</tr>
<tr>
<td>include_directory</td>
<td>appends the string argument to the list of places to search for #include files; HP compatible pragma</td>
</tr>
<tr>
<td>init_seg</td>
<td>specifies the section to contain C++ initialization code for the translation unit</td>
</tr>
<tr>
<td>message</td>
<td>displays the specified string literal to the standard output device</td>
</tr>
<tr>
<td>optimize</td>
<td>specifies optimizations to be performed on a function-by-function basis; implemented to partly support Microsoft's implementation of same pragma; click here for Intel's implementation</td>
</tr>
<tr>
<td>options</td>
<td>GCC-compatible (MacOS) pragma; sets the alignment of fields in structures</td>
</tr>
<tr>
<td>pointers_to_members</td>
<td>specifies whether a pointer to a class member can be declared before its associated class definition and is used to control the pointer size and the code required to interpret the pointer</td>
</tr>
</tbody>
</table>
### Pragma

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pop_macro</td>
<td>sets the value of the macro_name macro to the value on the top of the stack for this macro</td>
</tr>
<tr>
<td>push_macro</td>
<td>saves the value of the macro_name macro on the top of the stack for this macro</td>
</tr>
<tr>
<td>region/endregion</td>
<td>specifies a code segment in the Microsoft Visual Studio* 2005 Code Editor that expands and contracts by using the outlining feature</td>
</tr>
<tr>
<td>section</td>
<td>creates a section in an .obj file. Once a section is defined, it remains valid for the remainder of the compilation</td>
</tr>
<tr>
<td>start_map_region</td>
<td>used in conjunction with the stop_map_region pragma</td>
</tr>
<tr>
<td>stop_map_region</td>
<td>used in conjunction with the start_map_region pragma</td>
</tr>
<tr>
<td>fenv_access</td>
<td>informs an implementation that a program may test status flags or run under a non-default control mode</td>
</tr>
<tr>
<td>vtordisp</td>
<td>on argument enables the generation of hidden vtordisp members and off disables them</td>
</tr>
<tr>
<td>warning</td>
<td>allows selective modification of the behavior of compiler warning messages</td>
</tr>
<tr>
<td>weak</td>
<td>declares symbol you enter to be weak</td>
</tr>
</tbody>
</table>

### See Also
- Intel C++ Compiler Pragmas
- Intel-specific Pragmas
Overview: Intel® Math Library

The Intel® C++ Compiler includes a mathematical software library containing highly optimized and very accurate mathematical functions. These functions are commonly used in scientific or graphic applications, as well as other programs that rely heavily on floating-point computations. To include support for C99 _Complex data types, use the -std=c99 compiler option.

The mathimf.h header file includes prototypes for the library functions. For a complete list of the functions available, refer to the Function List.

Math Libraries

The math library linked to an application depends on the compilation or linkage options specified.

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libimf.a</td>
<td>Default static math library.</td>
</tr>
<tr>
<td>libimf.so</td>
<td>Default shared math library.</td>
</tr>
</tbody>
</table>

See Also

- Intel Math Library
- Using Intel® Math Library

Using the Intel Math Library

To use the Intel math library, include the header file, mathimf.h, in your program. Here are two example programs that illustrate the use of the math library on Linux* operating systems.
Example Using Real Functions

// real_math.c
#include <stdio.h>
#include <mathimf.h>

int main() {
    float fp32bits;
    double fp64bits;
    long double fp80bits;

    long double pi_by_four = 3.141592653589793238/4.0;
    // pi/4 radians is about 45 degrees
    fp32bits = (float) pi_by_four; // float approximation to pi/4
    fp64bits = (double) pi_by_four; // double approximation to pi/4
    fp80bits = pi_by_four; // long double (extended) approximation to pi/4

    printf("When x = %8.8f, sinf(x) = %8.8f \n", fp32bits, sinf(fp32bits));
    printf("When x = %16.16f, sin(x) = %16.16f \n", fp64bits, sin(fp64bits));
    printf("When x = %20.20Lf, sinl(x) = %20.20f \n", fp80bits, sinl(fp80bits));
    return 0;
}

The command for compiling real_math.c is:
icc real_math.c

The output of a.out will look like this:
When x = 0.78539816, sinf(x) = 0.70710678
When x = 0.7853981633974483, sin(x) = 0.7071067811865475
When x = 0.78539816339744827900, sinl(x) = 0.70710678118654750275
**Example Using Complex Functions**

// complex_math.c

#include <stdio.h>
#include <complex.h>

int main()
{
    float _Complex c32in, c32out;
    double _Complex c64in, c64out;
    double pi_by_four = 3.141592653589793238/4.0;
    c64in = 1.0 + I* pi_by_four;
    // Create the double precision complex number 1 + (pi/4) * i

    // where I is the imaginary unit.
    c32in = (float _Complex) c64in;
    // Create the float complex value from the double complex value.
    c64out = cexp(c64in);
    c32out = cexpf(c32in);
    // Call the complex exponential,
    // cexp(z) = cexp(x+iy) = e^(x + i y) = e^x * (cos(y) + i sin(y))
    printf("When z = %7.7f + %7.7f i, cexpf(z) = %7.7f + %7.7f i\n", crealf(c32in), cimagf(c32in), crealf(c32out), cimagf(c32out));
    printf("When z = %12.12f + %12.12f i, cexp(z) = %12.12f + %12.12f i\n", creal(c64in), cimag(c64in), creal(c64out), cimagf(c64out));
    return 0;
}

The command to compile *complex_math.c* is:

icc -std=c99 complex_math.c

The output of *a.out* will look like this:
When $z = 1.0000000 + 0.7853982 \, i$, $\text{cexpf}(z) = 1.9221154 + 1.9221156 \, i$

When $z = 1.000000000000 + 0.785398163397 \, i$, $\text{cexp}(z) = 1.922115514080 + 1.922115514080 \, i$

NOTE. _Complex data types are supported in C but not in C++ programs. It is necessary to include the `-std=c99` compiler option when compiling programs that require support for _Complex data types.

**Exception Conditions**

If you call a math function using argument(s) that may produce undefined results, an error number is assigned to the system variable `errno`. Math function errors are usually domain errors or range errors.

**Domain errors** result from arguments that are outside the domain of the function. For example, `acos` is defined only for arguments between -1 and +1 inclusive. Attempting to evaluate `acos(-2)` or `acos(3)` results in a domain error, where the return value is QNaN.

**Range errors** occur when a mathematically valid argument results in a function value that exceeds the range of representable values for the floating-point data type. Attempting to evaluate `exp(1000)` results in a range error, where the return value is INF.

When domain or range error occurs, the following values are assigned to `errno`:

- domain error (EDOM): `errno = 33`
- range error (ERANGE): `errno = 34`

The following example shows how to read the `errno` value for an EDOM and ERANGE error.

```c
#include <errno.h>
#include <mathimf.h>
#include <stdio.h>

int main(void)
{
    // ...
}
```
double neg_one=-1.0;
double zero=0.0;

// The natural log of a negative number is considered a domain error - EDOM
printf("log(\%e) = \%e and errno(EDOM) = \%d \n",neg_one,log(neg_one),errno);

// The natural log of zero is considered a range error - ERANGE
printf("log(\%e) = \%e and errno(ERANGE) = \%d \n",zero,log(zero),errno);
}

The output of *errno.c* will look like this:
log(-1.000000e+00) = nan and errno(EDOM) = 33
log(0.000000e+00) = -inf and errno(ERANGE) = 34

For the math functions in this section, a corresponding value for *errno* is listed when applicable.

**Other Considerations**

Some math functions are inlined automatically by the compiler. The functions actually inlined may vary and may depend on any vectorization or processor-specific compilation options used. For more information, see *Inline Function Expansion* in *Optimizing Applications>*Using Interprocedural Optimizations>*Inline Expansion of Functions.*

A change of the default precision control or rounding mode may affect the results returned by some of the mathematical functions. See *Overview: Tuning Performance* in *Floating-point Operations>*Tuning Performance.*

**Math Functions**

**Function List**

The Intel Math Library functions are listed here by function type.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trigonometric Functions</strong></td>
<td>acos</td>
</tr>
<tr>
<td></td>
<td>acosd</td>
</tr>
<tr>
<td></td>
<td>asin</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>asind</td>
</tr>
<tr>
<td></td>
<td>atan</td>
</tr>
<tr>
<td></td>
<td>atan2</td>
</tr>
<tr>
<td></td>
<td>atand</td>
</tr>
<tr>
<td></td>
<td>atand2</td>
</tr>
<tr>
<td></td>
<td>cos</td>
</tr>
<tr>
<td></td>
<td>cosd</td>
</tr>
<tr>
<td></td>
<td>cot</td>
</tr>
<tr>
<td></td>
<td>cotd</td>
</tr>
<tr>
<td></td>
<td>sin</td>
</tr>
<tr>
<td></td>
<td>sincos</td>
</tr>
<tr>
<td></td>
<td>sincosd</td>
</tr>
<tr>
<td></td>
<td>sind</td>
</tr>
<tr>
<td></td>
<td>tan</td>
</tr>
<tr>
<td></td>
<td>tand</td>
</tr>
<tr>
<td>Hyperbolic Functions</td>
<td>acosh</td>
</tr>
<tr>
<td></td>
<td>asinh</td>
</tr>
<tr>
<td></td>
<td>atanh</td>
</tr>
<tr>
<td></td>
<td>cosh</td>
</tr>
<tr>
<td></td>
<td>sinh</td>
</tr>
<tr>
<td></td>
<td>sinhcosh</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>tanh</td>
</tr>
<tr>
<td>Exponential Functions</td>
<td>cbrt</td>
</tr>
<tr>
<td></td>
<td>exp</td>
</tr>
<tr>
<td></td>
<td>exp10</td>
</tr>
<tr>
<td></td>
<td>exp2</td>
</tr>
<tr>
<td></td>
<td>expm1</td>
</tr>
<tr>
<td></td>
<td>frexp</td>
</tr>
<tr>
<td></td>
<td>hypot</td>
</tr>
<tr>
<td></td>
<td>invsqr</td>
</tr>
<tr>
<td></td>
<td>ilogb</td>
</tr>
<tr>
<td></td>
<td>ldexp</td>
</tr>
<tr>
<td></td>
<td>log</td>
</tr>
<tr>
<td></td>
<td>log10</td>
</tr>
<tr>
<td></td>
<td>log1p</td>
</tr>
<tr>
<td></td>
<td>log2</td>
</tr>
<tr>
<td></td>
<td>logb</td>
</tr>
<tr>
<td></td>
<td>pow</td>
</tr>
<tr>
<td></td>
<td>scalb</td>
</tr>
<tr>
<td></td>
<td>scalbln</td>
</tr>
<tr>
<td></td>
<td>scalbn</td>
</tr>
<tr>
<td></td>
<td>sqrt</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------</td>
</tr>
<tr>
<td><strong>Special Functions</strong></td>
<td>annuity</td>
</tr>
<tr>
<td></td>
<td>compound</td>
</tr>
<tr>
<td></td>
<td>erf</td>
</tr>
<tr>
<td></td>
<td>erfc</td>
</tr>
<tr>
<td></td>
<td>erfinv</td>
</tr>
<tr>
<td></td>
<td>gamma</td>
</tr>
<tr>
<td></td>
<td>gamma_r</td>
</tr>
<tr>
<td></td>
<td>j0</td>
</tr>
<tr>
<td></td>
<td>j1</td>
</tr>
<tr>
<td></td>
<td>jn</td>
</tr>
<tr>
<td></td>
<td>lgamma</td>
</tr>
<tr>
<td></td>
<td>lgamma_r</td>
</tr>
<tr>
<td></td>
<td>tgamma</td>
</tr>
<tr>
<td></td>
<td>y0</td>
</tr>
<tr>
<td></td>
<td>y1</td>
</tr>
<tr>
<td></td>
<td>yn</td>
</tr>
<tr>
<td><strong>Nearest Integer Functions</strong></td>
<td>ceil</td>
</tr>
<tr>
<td></td>
<td>floor</td>
</tr>
<tr>
<td></td>
<td>llrint</td>
</tr>
<tr>
<td></td>
<td>llround</td>
</tr>
<tr>
<td></td>
<td>lrint</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>lround</td>
</tr>
<tr>
<td></td>
<td>modf</td>
</tr>
<tr>
<td></td>
<td>nearbyint</td>
</tr>
<tr>
<td></td>
<td>rint</td>
</tr>
<tr>
<td></td>
<td>round</td>
</tr>
<tr>
<td></td>
<td>trunc</td>
</tr>
<tr>
<td>Remainder Functions</td>
<td>fmod</td>
</tr>
<tr>
<td></td>
<td>remainder</td>
</tr>
<tr>
<td></td>
<td>remquo</td>
</tr>
<tr>
<td>Miscellaneous Functions</td>
<td>copysign</td>
</tr>
<tr>
<td></td>
<td>fabs</td>
</tr>
<tr>
<td></td>
<td>fdim</td>
</tr>
<tr>
<td></td>
<td>finite</td>
</tr>
<tr>
<td></td>
<td>fma</td>
</tr>
<tr>
<td></td>
<td>fmax</td>
</tr>
<tr>
<td></td>
<td>fmin</td>
</tr>
<tr>
<td></td>
<td>fpclassify</td>
</tr>
<tr>
<td></td>
<td>isfinite</td>
</tr>
<tr>
<td></td>
<td>isgreater</td>
</tr>
<tr>
<td></td>
<td>isgreaterequal</td>
</tr>
<tr>
<td></td>
<td>isnan</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td></td>
<td>isless</td>
</tr>
<tr>
<td></td>
<td>islessequal</td>
</tr>
<tr>
<td></td>
<td>islessgreater</td>
</tr>
<tr>
<td></td>
<td>isnan</td>
</tr>
<tr>
<td></td>
<td>isnormal</td>
</tr>
<tr>
<td></td>
<td>isnunordered</td>
</tr>
<tr>
<td></td>
<td>nextafter</td>
</tr>
<tr>
<td></td>
<td>nexttoward</td>
</tr>
<tr>
<td></td>
<td>signbit</td>
</tr>
<tr>
<td></td>
<td>significand</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cabs</td>
</tr>
<tr>
<td></td>
<td>cacos</td>
</tr>
<tr>
<td></td>
<td>cacosh</td>
</tr>
<tr>
<td></td>
<td>carg</td>
</tr>
<tr>
<td></td>
<td>casin</td>
</tr>
<tr>
<td></td>
<td>casinh</td>
</tr>
<tr>
<td></td>
<td>catan</td>
</tr>
<tr>
<td></td>
<td>catanh</td>
</tr>
<tr>
<td></td>
<td>ccos</td>
</tr>
<tr>
<td></td>
<td>cexp</td>
</tr>
<tr>
<td></td>
<td>cexp2</td>
</tr>
</tbody>
</table>
**Trigonometric Functions**

The Intel Math Library supports the following trigonometric functions:

**acos**

**Description:** The $\text{acos}$ function returns the principal value of the inverse cosine of $x$ in the range $[0, \pi]$ radians for $x$ in the interval $[-1,1]$.

**errno:** EDOM, for $|x| > 1$

**Calling interface:**

```c
double acos(double x);
```
long double acosl(long double x);
float acosf(float x);

acosd

**Description:** The **acosd** function returns the principal value of the inverse cosine of x in the range \([0,180]\) degrees for x in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)

**Calling interface:**
- double acosd(double x);
- long double acosdl(long double x);
- float acosdf(float x);

asin

**Description:** The **asin** function returns the principal value of the inverse sine of x in the range \([-\pi/2, +\pi/2]\) radians for x in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)

**Calling interface:**
- double asin(double x);
- long double asinl(long double x);
- float asinf(float x);

asind

**Description:** The **asind** function returns the principal value of the inverse sine of x in the range \([-90,90]\) degrees for x in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)

**Calling interface:**
- double asind(double x);
- long double asindl(long double x);
- float asindf(float x);
atan

**Description:** The atan function returns the principal value of the inverse tangent of $x$ in the range $[-\pi/2, +\pi/2]$ radians.

**Calling interface:**

double atan(double x);
long double atanl(long double x);
float atanf(float x);

atan2

**Description:** The atan2 function returns the principal value of the inverse tangent of $y/x$ in the range $[-\pi, +\pi]$ radians.

**errno:** EDOM, for $x = 0$ and $y = 0$

**Calling interface:**

double atan2(double y, double x);
long double atan2l(long double y, long double x);
float atan2f(float y, float x);

atand

**Description:** The atand function returns the principal value of the inverse tangent of $x$ in the range $[-90,90]$ degrees.

**Calling interface:**

double atand(double x);
long double atandl(long double x);
float atandf(float x);

atan2d

**Description:** The atan2d function returns the principal value of the inverse tangent of $y/x$ in the range $[-180, +180]$ degrees.

**errno:** EDOM, for $x = 0$ and $y = 0$.

**Calling interface:**

double atan2d(double x, double y);
long double atan2dl(long double x, long double y);
float atan2df(float x, float y);
cos

**Description:** The `cos` function returns the cosine of \( x \) measured in radians. This function may be inlined by the Itanium® compiler.

**Calling interface:**
```cpp
double cos(double x);
long double cosl(long double x);
float cosf(float x);
```

cosd

**Description:** The `cosd` function returns the cosine of \( x \) measured in degrees.

**Calling interface:**
```cpp
double cosd(double x);
long double cosdl(long double x);
float cosdf(float x);
```

cot

**Description:** The `cot` function returns the cotangent of \( x \) measured in radians.

`errno`: `ERANGE`, for overflow conditions at \( x = 0 \).

**Calling interface:**
```cpp
double cot(double x);
long double cotl(long double x);
float cotf(float x);
```

cotd

**Description:** The `cotd` function returns the cotangent of \( x \) measured in degrees.

`errno`: `ERANGE`, for overflow conditions at \( x = 0 \).

**Calling interface:**
```cpp
double cotd(double x);
long double cotdl(long double x);
float cotdf(float x);
```
**sin**

**Description:** The sin function returns the sine of x measured in radians. This function may be inlined by the Itanium® compiler.

**Calling interface:**
```c
double sin(double x);
long double sinl(long double x);
float sinf(float x);
```

**sincos**

**Description:** The sincos function returns both the sine and cosine of x measured in radians. This function may be inlined by the Itanium® compiler.

**Calling interface:**
```c
void sincos(double x, double *sinval, double *cosval);
void sincosl(long double x, long double *sinval, long double *cosval);
void sincosf(float x, float *sinval, float *cosval);
```

**sincosd**

**Description:** The sincosd function returns both the sine and cosine of x measured in degrees.

**Calling interface:**
```c
void sincosd(double x, double *sinval, double *cosval);
void sincosdl(long double x, long double *sinval, long double *cosval);
void sincosdf(float x, float *sinval, float *cosval);
```

**sind**

**Description:** The sind function computes the sine of x measured in degrees.

**Calling interface:**
```c
double sind(double x);
long double sindl(long double x);
float sindf(float x);
```

**tan**

**Description:** The tan function returns the tangent of x measured in radians.
Calling interface:
double tan(double x);
long double tanl(long double x);
float tanf(float x);

tand

Description: The tand function returns the tangent of x measured in degrees.

errno: ERANGE, for overflow conditions

Calling interface:
double tand(double x);
long double tandl(long double x);
float tandf(float x);

Hyperbolic Functions

The Intel Math Library supports the following hyperbolic functions:

acosh

Description: The acosh function returns the inverse hyperbolic cosine of x.

errno: EDOM, for x < 1

Calling interface:
double acosh(double x);
long double acoshl(long double x);
float acoshf(float x);

asinh

Description: The asinh function returns the inverse hyperbolic sine of x.

Calling interface:
double asinh(double x);
long double asinhl(long double x);
float asinhf(float x);
**atanh**

**Description:** The `atanh` function returns the inverse hyperbolic tangent of `x`.

- **errno:** EDOM, for `x > 1`
- **errno:** ERANGE, for `x = 1`

**Calling interface:**

```c
double atanh(double x);
long double atanhl(long double x);
float atanhf(float x);
```

**cosh**

**Description:** The `cosh` function returns the hyperbolic cosine of `x`, `(e^x + e^{-x})/2`.

- **errno:** ERANGE, for overflow conditions

**Calling interface:**

```c
double cosh(double x);
long double coshl(long double x);
float coshf(float x);
```

**sinh**

**Description:** The `sinh` function returns the hyperbolic sine of `x`, `(e^x - e^{-x})/2`.

- **errno:** ERANGE, for overflow conditions

**Calling interface:**

```c
double sinh(double x);
long double sinhl(long double x);
float sinhf(float x);
```

**sinhcosh**

**Description:** The `sinhcosh` function returns both the hyperbolic sine and hyperbolic cosine of `x`.

- **errno:** ERANGE, for overflow conditions

**Calling interface:**

```c
void sinhcosh(double x, double *sinval, double *cosval);
void sinhcoshl(long double x, long double *sinval, long double *cosval);
```
void sinhcsf(float x, float *sinval, float *cosval);

tanh

**Description:** The `tanh` function returns the hyperbolic tangent of \( x \), \((e^x - e^{-x}) / (e^x + e^{-x})\).

**Calling interface:**
```c
double tanh(double x);
long double tanhl(long double x);
float tanhf(float x);
```

**Exponential Functions**

The Intel Math Library supports the following exponential functions:

cbrt

**Description:** The `cbrt` function returns the cube root of \( x \).

**Calling interface:**
```c
double cbrt(double x);
long double cbrtl(long double x);
float cbrtf(float x);
```

exp

**Description:** The `exp` function returns \( e \) raised to the \( x \) power, \( e^x \). This function may be inlined by the Itanium® compiler.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**
```c
double exp(double x);
long double expl(long double x);
float expf(float x);
```

exp10

**Description:** The `exp10` function returns 10 raised to the \( x \) power, \( 10^x \).

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**
double exp10(double x);
long double exp10l(long double x);
float exp10f(float x);

exp2
Description: The exp2 function returns 2 raised to the x power, $2^x$.
errno: ERANGE, for underflow and overflow conditions
Calling interface:
double exp2(double x);
long double exp2l(long double x);
float exp2f(float x);

expm1
Description: The expm1 function returns $e$ raised to the x power minus 1, $e^x - 1$.
errno: ERANGE, for overflow conditions
Calling interface:
double expm1(double x);
long double expm1l(long double x);
float expm1f(float x);

frexp
Description: The frexp function converts a floating-point number x into signed normalized fraction in $[1/2, 1)$ multiplied by an integral power of two. The signed normalized fraction is returned, and the integer exponent stored at location exp.
Calling interface:
double frexp(double x, int *exp);
long double frexpl(long double x, int *exp);
float frexpf(float x, int *exp);

hypot
Description: The hypot function returns the square root of $(x^2 + y^2)$.
errno: ERANGE, for overflow conditions
Calling interface:
double hypot(double x, double y);
long double hypotl(long double x, long double y);
float hypotf(float x, float y);

ilogb

Description: The ilogb function returns the exponent of x base two as a signed int value.
errno: ERANGE, for x = 0
Calling interface:
int ilogb(double x);
int ilogbl(long double x);
int ilogbf(float x);

invsqrt

Description: The invsqrt function returns the inverse square root. This function may be inlined by the Itanium® compiler.
Calling interface:
double invsqrt(double x);
long double invsqrtl(long double x);
float invsqrtf(float x);

ldexp

Description: The ldexp function returns $x \cdot 2^{\exp}$, where exp is an integer value.
errno: ERANGE, for underflow and overflow conditions
Calling interface:
double ldexp(double x, int exp);
long double ldexpl(long double x, int exp);
float ldexpf(float x, int exp);

log

Description: The log function returns the natural log of x, ln(x). This function may be inlined by the Itanium® compiler.
errno: EDOM, for x < 0
errno: ERANGE, for x = 0

Calling interface:
double log(double x);
long double logl(long double x);
float logf(float x);

log10
Description: The log10 function returns the base-10 log of x, \( \log_{10}(x) \). This function may be inlined by the Itanium\(^\circledast\) compiler.

errno: EDOM, for x < 0
errno: ERANGE, for x = 0

Calling interface:
double log10(double x);
long double log10l(long double x);
float log10f(float x);

log1p
Description: The log1p function returns the natural log of \( (x+1) \), \( \ln(x + 1) \).

errno: EDOM, for x < -1
errno: ERANGE, for x = -1

Calling interface:
double log1p(double x);
long double log1pl(long double x);
float log1pf(float x);

log2
Description: The log2 function returns the base-2 log of x, \( \log_{2}(x) \).

errno: EDOM, for x < 0
errno: ERANGE, for x = 0

Calling interface:
double log2(double x);
long double log2l(long double x);
float log2f(float x);
**logb**

**Description:** The logb function returns the signed exponent of x.

**errno:** EDOM, for x = 0

**Calling interface:**

double logb(double x);
long double logbl(long double x);
float logbf(float x);

**pow**

**Description:** The pow function returns x raised to the power of y, \( x^y \). This function may be inlined by the Itanium® compiler.

**Calling interface:**

double pow(double x, double y);
long double powl(double x, double y);
float powf(float x, float y);

**scalb**

**Description:** The scalb function returns \( x \times 2^y \), where y is a floating-point value.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**

double scalb(double x, double y);
long double scalbl(long double x, long double y);
float scalbf(float x, float y);

**scalbn**

**Description:** The scalbn function returns \( x \times 2^n \), where n is an integer value.

**errno:** ERANGE, for underflow and overflow conditions
**Calling interface:**
double scalbn(double x, int n);
long double scalbnl (long double x, int n);
float scalbnf(float x, int n);

**scalbn**

**Description:** The `scalbn` function returns \(x \times 2^n\), where \(n\) is a long integer value.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**
double scalbn(double x, long int n);
long double scalbnl (long double x, long int n);
float scalbnf(float x, long int n);

**sqrt**

**Description:** The `sqrt` function returns the correctly rounded square root.

**errno:** EDOM, for \(x < 0\)

**Calling interface:**
double sqrt(double x);
long double sqrtl(long double x);
float sqrtf(float x);

**Special Functions**

The Intel Math Library supports the following special functions:

**annuity**

**Description:** The `annuity` function computes the present value factor for an annuity, \((1 - (1+x)^{-y}) / x\), where \(x\) is a rate and \(y\) is a period.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**
double annuity(double x, double y);
long double annuityl(long double x, long double y);
float annuityf(float x, float y);
compound

Description: The `compound` function computes the compound interest factor, \((1+x)^y\), where \(x\) is a rate and \(y\) is a period.

errno: ERANGE, for underflow and overflow conditions

Calling interface:

double compound(double x, double y);
long double compoundl(long double x, long double y);
float compoundf(float x, float y);

erf

Description: The `erf` function returns the error function value.

Calling interface:

double erf(double x);
long double erfl(long double x);
float erff(float x);

erfc

Description: The `erfc` function returns the complementary error function value.

errno: EDOM, for finite or infinite \(|x| > 1\)

Calling interface:

double erfc(double x);
long double erfcl(long double x);
float erfcf(float x);

erfinv

Description: The `erfinv` function returns the value of the inverse error function of \(x\).

errno: EDOM, for finite or infinite \(|x| > 1\)

Calling interface:

double erfinv(double x);
long double erfinvl(long double x);
float erfinvf(float x);
**gamma**

**Description:** The `gamma` function returns the value of the logarithm of the absolute value of gamma.

**errno:** `ERANGE`, for overflow conditions when x is a negative integer.

**Calling interface:**
```c
double gamma(double x);
long double gammal(long double x);
float gammaf(float x);
```

**gamma_r**

**Description:** The `gamma_r` function returns the value of the logarithm of the absolute value of gamma. The sign of the `gamma` function is returned in the integer `signgam`.

**Calling interface:**
```c
double gamma_r(double x, int *signgam);
long double gammal_r(long double x, int *signgam);
float gammaf_r(float x, int *signgam);
```

**j0**

**Description:** Computes the Bessel function (of the first kind) of x with order 0.

**Calling interface:**
```c
double j0(double x);
long double j0l(long double x);
float j0f(float x);
```

**j1**

**Description:** Computes the Bessel function (of the first kind) of x with order 1.

**Calling interface:**
```c
double j1(double x);
long double j1l(long double x);
float j1f(float x);
```
jn

Description: Computes the Bessel function (of the first kind) of \( x \) with order \( n \).

Calling interface:

double jn(int n, double x);
long double jnl(int n, long double x);
float jnf(int n, float x);

lgamma

Description: The \( \text{lgamma} \) function returns the value of the logarithm of the absolute value of gamma.

errno: ERANGE, for overflow conditions, \( x=0 \) or negative integers.

Calling interface:

double lgamma(double x);
long double lgammal(long double x);
float lgammaf(float x);

lgamma_r

Description: The \( \text{lgamma_r} \) function returns the value of the logarithm of the absolute value of gamma. The sign of the \( \text{gamma} \) function is returned in the integer \( \text{signgam} \).

errno: ERANGE, for overflow conditions, \( x=0 \) or negative integers.

Calling interface:

double lgamma_r(double x, int *signgam);
long double lgammal_r(long double x, int *signgam);
float lgammaf_r(float x, int *signgam);

tgamma

Description: The \( \text{tgamma} \) function computes the gamma function of \( x \).

errno: EDOM, for \( x=0 \) or negative integers.

Calling interface:

double tgamma(double x);
long double tgammal(long double x);
float tgammaf(float x);
**y0**

**Description:** Computes the Bessel function (of the second kind) of $x$ with order 0.

**errno:** EDOM, for $x \leq 0$

**Calling interface:**

double y0(double x);
long double y0l(long double x);
float y0f(float x);

**y1**

**Description:** Computes the Bessel function (of the second kind) of $x$ with order 1.

**errno:** EDOM, for $x \leq 0$

**Calling interface:**

double y1(double x);
long double y1l(long double x);
float y1f(float x);

**yn**

**Description:** Computes the Bessel function (of the second kind) of $x$ with order $n$.

**errno:** EDOM, for $x \leq 0$

**Calling interface:**

double yn(int n, double x);
long double ynl(int n, long double x);
float ynf(int n, float x);

**Nearest Integer Functions**

The Intel Math Library supports the following nearest integer functions:

**ceil**

**Description:** The `ceil` function returns the smallest integral value not less than $x$ as a floating-point number. This function may be inlined by the Itanium® compiler.

**Calling interface:**

double ceil(double x);
long double ceill(long double x);
float ceilf(float x);

floor

**Description:** The `floor` function returns the largest integral value not greater than `x` as a floating-point value. This function may be inlined by the Itanium® compiler.

**Calling interface:**
- double floor(double x);
- long double floorl(long double x);
- float floorf(float x);

llrint

**Description:** The `llrint` function returns the rounded integer value (according to the current rounding direction) as a `long long int`.

**errno:** ERANGE, for values too large

**Calling interface:**
- long long int llrint(double x);
- long long int llrintl(long double x);
- long long int llrintf(float x);

llround

**Description:** The `llround` function returns the rounded integer value as a `long long int`.

**errno:** ERANGE, for values too large

**Calling interface:**
- long long int llround(double x);
- long long int llroundl(long double x);
- long long int llroundf(float x);

lrint

**Description:** The `lrint` function returns the rounded integer value (according to the current rounding direction) as a `long int`.

**errno:** ERANGE, for values too large
Calling interface:
long int lrint(double x);
long int lrintl(long double x);
long int lrintf(float x);

lround

Description: The lround function returns the rounded integer value as a long int. Halfway cases are rounded away from zero.
errno: ERANGE, for values too large

Calling interface:
long int lround(double x);
long int lroundl(long double x);
long int lroundf(float x);

modf

Description: The modf function returns the value of the signed fractional part of x and stores the integral part at *iptr as a floating-point number.

Calling interface:
double modf(double x, double *iptr);
long double modfl(long double x, long double *iptr);
float modff(float x, float *iptr);

nearbyint

Description: The nearbyint function returns the rounded integral value as a floating-point number, using the current rounding direction.

Calling interface:
double nearbyint(double x);
long double nearbyintl(long double x);
float nearbyintf(float x);

rint

Description: The rint function returns the rounded integral value as a floating-point number, using the current rounding direction.
Calling interface:
double rint(double x);
long double rintl(long double x);
float rintf(float x);

round

Description: The \texttt{round} function returns the nearest integral value as a floating-point number. Halfway cases are rounded away from zero.

Calling interface:
double round(double x);
long double roundl(long double x);
float roundf(float x);

trunc

Description: The \texttt{trunc} function returns the truncated integral value as a floating-point number.

Calling interface:
double trunc(double x);
long double truncl(long double x);
float truncf(float x);

Remainder Functions
The Intel Math Library supports the following remainder functions:

fmod

Description: The \texttt{fmod} function returns the value $x - n \times y$ for integer $n$ such that if $y$ is nonzero, the result has the same sign as $x$ and magnitude less than the magnitude of $y$.

errno: EDOM, for $y = 0$

Calling interface:
double fmod(double x, double y);
long double fmodl(long double x, long double y);
float fmodf(float x, float y);
remainder

Description: The remainder function returns the value of \( x \ REM y \) as required by the IEEE standard.

Calling interface:

double remainder(double x, double y);
long double remainderl(long double x, long double y);
float remainderf(float x, float y);

remquo

Description: The remquo function returns the value of \( x \ REM y \). In the object pointed to by quo the function stores a value whose sign is the sign of \( x/y \) and whose magnitude is congruent modulo \( 2^n \) of the integral quotient of \( x/y \). N is an implementation-defined integer. For systems based on IA-64 architecture, N is equal to 24. For all other systems, N is equal to 31.

Calling interface:

double remquo(double x, double y, int *quo);
long double remquol(long double x, long double y, int *quo);
float remquof(float x, float y, int *quo);

Miscellaneous Functions

The Intel Math Library supports the following miscellaneous functions:

copysign

Description: The copysign function returns the value with the magnitude of \( x \) and the sign of \( y \).

Calling interface:

double copysign(double x, double y);
long double copysignl(long double x, long double y);
float copysignf(float x, float y);

fabs

Description: The fabs function returns the absolute value of \( x \).

Calling interface:

double fabs(double x);
long double fabsl(long double x);
float fabsf(float x);

**fdim**

**Description:** The `fdim` function returns the positive difference value, \( x - y \) (for \( x > y \)) or \( +0 \) (for \( x \leq y \)).

**errno:** ERANGE, for values too large

**Calling interface:**
- `double fdim(double x, double y);`
- `long double fdiml(long double x, long double y);`
- `float fdimf(float x, float y);`

**finite**

**Description:** The `finite` function returns 1 if \( x \) is not a NaN or +/- infinity. Otherwise 0 is returned.

**Calling interface:**
- `int finite(double x);`
- `int finitel(long double x);`
- `int finitef(float x);`

**fma**

**Description:** The `fma` functions return \((x * y) + z\).

**Calling interface:**
- `double fma(double x, double y, double z);`
- `long double fmal(long double x, long double y, long double z);`
- `float fmaf(float x, float y, float double z);`

**fmax**

**Description:** The `fmax` function returns the maximum numeric value of its arguments.

**Calling interface:**
- `double fmax(double x, double y);`
- `long double fmaxl(long double x, long double y);`
- `float fmaxf(float x, float y);`
**fmin**

**Description:** The `fmin` function returns the minimum numeric value of its arguments.

**Calling interface:**

double fmin(double x, double y);
long double fminl(long double x, long double y);
float fminf(float x, float y);

**fpclassify**

**Description:** The `fpclassify` function returns the value of the number classification macro appropriate to the value of its argument.

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (NaN)</td>
</tr>
<tr>
<td>1 (Infinity)</td>
</tr>
<tr>
<td>2 (Zero)</td>
</tr>
<tr>
<td>3 (Subnormal)</td>
</tr>
<tr>
<td>4 (Finite)</td>
</tr>
</tbody>
</table>

**Calling interface:**

double fpclassify(double x);
long double fpclassifyl(long double x);
float fpclassifyf(float x);

**isfinite**

**Description:** The `isfinite` function returns 1 if `x` is not a NaN or +/- infinity. Otherwise 0 is returned.

**Calling interface:**

int isfinite(double x);
int isfinitei(long double x);
int isfinitef(float x);
**isgreater**

**Description:** The `isgreater` function returns 1 if `x` is greater than `y`. This function does not raise the invalid floating-point exception.

**Calling interface:**
```c
int isgreater(double x, double y);
int isgreaterl(long double x, long double y);
int isgreatererf(float x, float y);
```

**isgreaterequal**

**Description:** The `isgreaterequal` function returns 1 if `x` is greater than or equal to `y`. This function does not raise the invalid floating-point exception.

**Calling interface:**
```c
int isgreaterequal(double x, double y);
int isgreaterequall(long double x, long double y);
int isgreaterequalf(float x, float y);
```

**isinf**

**Description:** The `isinf` function returns a non-zero value if and only if its argument has an infinite value.

**Calling interface:**
```c
int isinf(double x);
int isnfl(long double x);
int isinff(float x);
```

**isless**

**Description:** The `isless` function returns 1 if `x` is less than `y`. This function does not raise the invalid floating-point exception.

**Calling interface:**
```c
int isless(double x, double y);
int islessl(long double x, long double y);
int islesssf(float x, float y);
```
**islessequall**

**Description:** The `islessequall` function returns 1 if `x` is less than or equal to `y`. This function does not raise the invalid floating-point exception.

**Calling interface:**
```c
int islessequall(double x, double y);
int islessequall(long double x, long double y);
int islessequallf(float x, float y);
```

**isnan**

**Description:** The `isnan` function returns a non-zero value if and only if `x` has a NaN value.

**Calling interface:**
```c
int isnan(double x);
int isnanl(long double x);
int isnanf(float x);
```

**isnormal**

**Description:** The `isnormal` function returns a non-zero value if and only if `x` is normal.

**Calling interface:**
```c
int isnormal(double x);
int isnormall(long double x);
int isnormalf(float x);
```

**isunordered**

**Description:** The `isunordered` function returns 1 if either `x` or `y` is a NaN. This function does not raise the invalid floating-point exception.
**Calling interface:**

int isunordered(double x, double y);
int isunorderedl(long double x, long double y);
int isunorderedf(float x, float y);

**nextafter**

**Description:** The `nextafter` function returns the next representable value in the specified format after `x` in the direction of `y`.

**errno:** ERANGE, for overflow and underflow conditions

**Calling interface:**

double nextafter(double x, double y);
long double nextafterl(long double x, long double y);
float nextafterf(float x, float y);

**nexttoward**

**Description:** The `nexttoward` function returns the next representable value in the specified format after `x` in the direction of `y`. If `x` equals `y`, then the function returns `y` converted to the type of the function. Use the `/Qlong-double` option on Windows* operating systems for accurate results.

**errno:** ERANGE, for overflow and underflow conditions

**Calling interface:**

double nexttoward(double x, long double y);
long double nexttowardl(long double x, long double y);
float nexttowardf(float x, long double y);

**signbit**

**Description:** The `signbit` function returns a non-zero value if and only if the sign of `x` is negative.

**Calling interface:**

int signbit(double x);
int signbitl(long double x);
int signbitf(float x);
**significand**

**Description:** The significand function returns the significand of \( x \) in the interval \([1,2)\). For \( x \) equal to zero, NaN, or +/- infinity, the original \( x \) is returned.

**Calling interface:**
- `double significand(double x);`
- `long double significandl(long double x);`
- `float significandf(float x);`

**Complex Functions**

The Intel Math Library supports the following complex functions:

**cabs**

**Description:** The `cabs` function returns the complex absolute value of \( z \).

**Calling interface:**
- `double cabs(double _Complex z);`
- `long double cabsl(long double _Complex z);`
- `float cabsf(float _Complex z);`

**cacos**

**Description:** The `cacos` function returns the complex inverse cosine of \( z \).

**Calling interface:**
- `double _Complex cacos(double _Complex z);`
- `long double _Complex cacosl(long double _Complex z);`
- `float _Complex cacosf(float _Complex z);`

**cacosh**

**Description:** The `cacosh` function returns the complex inverse hyperbolic cosine of \( z \).

**Calling interface:**
- `double _Complex cacosh(double _Complex z);`
- `long double _Complex cacoshl(long double _Complex z);`
- `float _Complex cacoshf(float _Complex z);`
**carg**

**Description:** The `carg` function returns the value of the argument in the interval \([-\pi, \pi]\).

**Calling interface:**

double carg(double _Complex z);
lng double cargl(long double _Complex z);
float cargf(float _Complex z);

**casin**

**Description:** The `casin` function returns the complex inverse sine of \(z\).

**Calling interface:**

double _Complex casin(double _Complex z);
lng double _Complex casinl(long double _Complex z);
float _Complex casinf(float _Complex z);

**casinh**

**Description:** The `casinh` function returns the complex inverse hyperbolic sine of \(z\).

**Calling interface:**

double _Complex casinh(double _Complex z);
lng double _Complex casinhl(long double _Complex z);
float _Complex casinhf(float _Complex z);

**catan**

**Description:** The `catan` function returns the complex inverse tangent of \(z\).

**Calling interface:**

double _Complex catan(double _Complex z);
lng double _Complex catanl(long double _Complex z);
float _Complex catanf(float _Complex z);

**catanh**

**Description:** The `catanh` function returns the complex inverse hyperbolic tangent of \(z\).

**Calling interface:**

double _Complex catanh(double _Complex z);
long double _Complex catanh1(long double _Complex z);
float _Complex catanhf(float _Complex z);

ccos

**Description:** The `ccos` function returns the complex cosine of `z`.

**Calling interface:**
double _Complex ccos(double _Complex z);
long double _Complex ccosl(long double _Complex z);
float _Complex ccosf(float _Complex z);

ccosh

**Description:** The `ccosh` function returns the complex hyperbolic cosine of `z`.

**Calling interface:**
double _Complex ccosh(double _Complex z);
long double _Complex ccoshl(long double _Complex z);
float _Complex ccoshf(float _Complex z);

cexp

**Description:** The `cexp` function returns \( e^z \) (\( e \) raised to the power `z`).

**Calling interface:**
double _Complex cexp(double _Complex z);
long double _Complex cexpl(long double _Complex z);
float _Complex cexpf(float _Complex z);

cexp2

**Description:** The `cexp` function returns \( 2^z \) (\( 2 \) raised to the power `z`).

**Calling interface:**
double _Complex cexp2(double _Complex z);
long double _Complex cexp2l(long double _Complex z);
float _Complex cexp2f(float _Complex z);
cexp10

**Description:** The `cexp10` function returns $10^z$ (10 raised to the power of $z$).

**Calling interface:**

```c
double _Complex cexp10(double _Complex z);
long double _Complex cexp10l(long double _Complex z);
float _Complex cexp10f(float _Complex z);
```

cimag

**Description:** The `cimag` function returns the imaginary part value of $z$.

**Calling interface:**

```c
double cimag(double _Complex z);
long double cimagl(long double _Complex z);
float cimagf(float _Complex z);
```

cis

**Description:** The `cis` function returns the cosine and sine (as a complex value) of $z$ measured in radians.

**Calling interface:**

```c
double _Complex cis(double x);
long double _Complex cisl(long double z);
float _Complex cisf(float z);
```

cisd

**Description:** The `cisd` function returns the cosine and sine (as a complex value) of $z$ measured in degrees.

**Calling interface:**

```c
double _Complex cisd(double x);
long double _Complex cisdl(long double z);
float _Complex cisdf(float z);
```

clog

**Description:** The `clog` function returns the complex natural logarithm of $z$. 
**Calling interface:**
double _Complex clog(double _Complex z);
long double _Complex clogl(long double _Complex z);
float _Complex clogf(float _Complex z);

clog2

**Description:** The clog2 function returns the complex logarithm base 2 of \( z \).

**Calling interface:**
double _Complex clog2(double _Complex z);
long double _Complex clog2l(long double _Complex z);
float _Complex clog2f(float _Complex z);

clog10

**Description:** The clog10 function returns the complex logarithm base 10 of \( z \).

**Calling interface:**
double _Complex clog10(double _Complex z);
long double _Complex clog10l(long double _Complex z);
float _Complex clog10f(float _Complex z);

**conj**

**Description:** The conj function returns the complex conjugate of \( z \) by reversing the sign of its imaginary part.

**Calling interface:**
double _Complex conj(double _Complex z);
long double _Complex conjl(long double _Complex z);
float _Complex conjf(float _Complex z);

cpow

**Description:** The cpow function returns the complex power function, \( x^y \).

**Calling interface:**
double _Complex cpow(double _Complex x, double _Complex y);
long double _Complex cpowl(long double _Complex x, long double _Complex y);
float _Complex cpowf(float _Complex x, float _Complex y);
cproj

**Description:** The `cproj` function returns a projection of `z` onto the Riemann sphere.

**Calling interface:**
```
double _Complex cproj(double _Complex z);
long double _Complex cprojl(long double _Complex z);
float _Complex cprojf(float _Complex z);
```

creal

**Description:** The `creal` function returns the real part of `z`.

**Calling interface:**
```
double creal(double _Complex z);
long double creall(long double _Complex z);
float crealf(float _Complex z);
```

csin

**Description:** The `csin` function returns the complex sine of `z`.

**Calling interface:**
```
double _Complex csin(double _Complex z);
long double _Complex csinl(long double _Complex z);
float _Complex csinf(float _Complex z);
```

csinh

**Description:** The `csinh` function returns the complex hyperbolic sine of `z`.

**Calling interface:**
```
double _Complex csinh(double _Complex z);
long double _Complex csinhl(long double _Complex z);
float _Complex csinhf(float _Complex z);
```

csqrt

**Description:** The `csqrt` function returns the complex square root of `z`.

**Calling interface:**
```
double _Complex csqrt(double _Complex z);
```
$$\text{long double } \_\text{Complex csqrtl}(\text{long double } \_\text{Complex } z);$$
$$\text{float } \_\text{Complex csqrtf}(\text{float } \_\text{Complex } z);$$

c**tan**

**Description:** The c**tan** function returns the complex tangent of \( z \).

**Calling interface:**
$$\text{double } \_\text{Complex ctan}(\text{double } \_\text{Complex } z);$$
$$\text{long double } \_\text{Complex ctanl}(\text{long double } \_\text{Complex } z);$$
$$\text{float } \_\text{Complex ctanf}(\text{float } \_\text{Complex } z);$$

c**tanh**

**Description:** The c**tanh** function returns the complex hyperbolic tangent of \( z \).

**Calling interface:**
$$\text{double } \_\text{Complex ctanh}(\text{double } \_\text{Complex } z);$$
$$\text{long double } \_\text{Complex ctanhl}(\text{long double } \_\text{Complex } z);$$
$$\text{float } \_\text{Complex ctanhf}(\text{float } \_\text{Complex } z);$$

**C99 Macros**
The Intel Math Library and mathimf.h header file support the following C99 macros:
$$\text{int fpclassify(x);}$$
$$\text{int isfinite(x);}$$
$$\text{int isgreater(x, y);}$$
$$\text{int isgreaterequal(x, y);}$$
$$\text{int isinf(x);}$$
$$\text{int isles(x, y);}$$
$$\text{int islessequal(x, y);}$$
$$\text{int islessgreater(x, y);}$$
$$\text{int isnan(x);}$$
$$\text{int isnormal(x);}$$
$$\text{int unordered(x, y);}$$
$$\text{int signbit(x);}$$
See Also
- Math Functions
- Miscellaneous Functions
Introduction to the Class Libraries

Overview: Intel C++ Class Libraries

The Intel C++ Class Libraries enable Single-Instruction, Multiple-Data (SIMD) operations. The principle of SIMD operations is to exploit microprocessor architecture through parallel processing. The effect of parallel processing is increased data throughput using fewer clock cycles. The objective is to improve application performance of complex and computation-intensive audio, video, and graphical data bit streams.

Hardware and Software Requirements

The Intel C++ Class Libraries are functions abstracted from the instruction extensions available on Intel processors as specified in the table that follows:

Processor Requirements for Use of Class Libraries

<table>
<thead>
<tr>
<th>Header File</th>
<th>Extension Set</th>
<th>Available on These Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivec.h</td>
<td>MMX™ technology</td>
<td>Intel® Pentium® processor with MMX™ technology, Intel® Pentium® II processor, Intel® Pentium® III processor, Intel® Pentium® 4 processor, Intel® Xeon® processor, and Intel® Itanium® processor</td>
</tr>
<tr>
<td>fvec.h</td>
<td>Streaming SIMD Extensions</td>
<td>Intel® Pentium® III processor, Intel® Pentium® 4 processor, Intel® Xeon® processor, and Intel® Itanium® processor</td>
</tr>
<tr>
<td>dvec.h</td>
<td>Streaming SIMD Extensions 2</td>
<td>Intel® Pentium® 4 processor and Intel® Xeon® processors</td>
</tr>
</tbody>
</table>

About the Classes

The Intel C++ Class Libraries for SIMD Operations include:

- Integer vector (Ivec) classes
- Floating-point vector (Fvec) classes
You can find the definitions for these operations in three header files: ivec.h, fvec.h, and dvec.h. The classes themselves are not partitioned like this. The classes are named according to the underlying type of operation. The header files are partitioned according to architecture:

- ivec.h is specific to architectures with MMX™ technology
- fvec.h is specific to architectures with Streaming SIMD Extensions
- dvec.h is specific to architectures with Streaming SIMD Extensions 2

Streaming SIMD Extensions 2 intrinsics cannot be used on IA-64 architecture based systems. The mmclass.h header file includes the classes that are usable on the IA-64 architecture.

This documentation is intended for programmers writing code for the Intel architecture, particularly code that would benefit from the use of SIMD instructions. You should be familiar with C++ and the use of C++ classes.

Details About the Libraries

The Intel C++ Class Libraries for SIMD Operations provide a convenient interface to access the underlying instructions for processors as specified in Processor Requirements for Use of Class Libraries. These processor-instruction extensions enable parallel processing using the single instruction-multiple data (SIMD) technique as illustrated in the following figure.

**SIMD Data Flow**

Performing four operations with a single instruction improves efficiency by a factor of four for that particular instruction.

These new processor instructions can be implemented using assembly inlining, intrinsics, or the C++ SIMD classes. Compare the coding required to add four 32-bit floating-point values, using each of the available interfaces:
Comparison Between Inlining, Intrinsics and Class Libraries

<table>
<thead>
<tr>
<th>Assembly Inlining</th>
<th>Intrinsics</th>
</tr>
</thead>
<tbody>
<tr>
<td>... __m128 a,b,c; __asm{ movaps xmm0,b movaps xmm1,c addps xmm0,xmm1 a,b,c; a = _mm_add_ps(b,c); ... movaps a, xmm0 } ...</td>
<td></td>
</tr>
</tbody>
</table>

This table shows an addition of two single-precision floating-point values using assembly inlining, intrinsics, and the libraries. You can see how much easier it is to code with the Intel C++ SIMD Class Libraries. Besides using fewer keystrokes and fewer lines of code, the notation is like the standard notation in C++, making it much easier to implement over other methods.

C++ Classes and SIMD Operations

The use of C++ classes for SIMD operations is based on the concept of operating on arrays, or vectors of data, in parallel. Consider the addition of two vectors, A and B, where each vector contains four elements. Using the integer vector (Ivec) class, the elements A[i] and B[i] from each array are summed as shown in the following example.

**Typical Method of Adding Elements Using a Loop**

```c
short a[4], b[4], c[4];
for (i=0; i<4; i++) /* needs four iterations */
c[i] = a[i] + b[i]; /* returns c[0], c[1], c[2], c[3] */
```

The following example shows the same results using one operation with Ivec Classes.
**SIMD Method of Adding Elements Using Ivec Classes**

```c
sIs16vec4 ivecA, ivecB, ivec C; /*needs one iteration*/
ivecC = ivecA + ivecB; /*returns ivecC0, ivecC1, ivecC2, ivecC3 */
```

**Available Classes**

The Intel C++ SIMD classes provide parallelism, which is not easily implemented using typical mechanisms of C++. The following table shows how the Intel C++ SIMD classes use the classes and libraries.

**SIMD Vector Classes**

<table>
<thead>
<tr>
<th>Instruction Set</th>
<th>Class</th>
<th>Signedness</th>
<th>Data Type</th>
<th>Size</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMX™ technology</td>
<td>I64vec1</td>
<td>unspecified</td>
<td>__m64</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>I32vec2</td>
<td>unspecified</td>
<td>int</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>Is32vec2</td>
<td>signed</td>
<td>int</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>Iu32vec2</td>
<td>unsigned</td>
<td>int</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>I16vec4</td>
<td>unspecified</td>
<td>short</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>Is16vec4</td>
<td>signed</td>
<td>short</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>Iu16vec4</td>
<td>unsigned</td>
<td>short</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>I8vec8</td>
<td>unspecified</td>
<td>char</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>Is8vec8</td>
<td>signed</td>
<td>char</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>Iu8vec8</td>
<td>unsigned</td>
<td>char</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Streaming SIMD Extensions</td>
<td>F32vec4</td>
<td>signed</td>
<td>float</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>Streaming SIMD Extensions</td>
<td>F32vec1</td>
<td>signed</td>
<td>float</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>Instruction Set</td>
<td>Class</td>
<td>Signedness</td>
<td>Data Type</td>
<td>Size</td>
<td>Elements</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
<td>------------</td>
<td>-----------</td>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>Streaming SIMD Extensions 2</td>
<td>F64vec2</td>
<td>signed</td>
<td>double</td>
<td>64</td>
<td>2</td>
</tr>
<tr>
<td>I128vec1</td>
<td>unspecified</td>
<td>_m128i</td>
<td>128</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>I64vec2</td>
<td>unspecified</td>
<td>long int</td>
<td>64</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Is64vec2</td>
<td>signed</td>
<td>long int</td>
<td>64</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Iu64vec2</td>
<td>unsigned</td>
<td>long int</td>
<td>32</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>I32vec4</td>
<td>unspecified</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Is32vec4</td>
<td>signed</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Iu32vec4</td>
<td>unsigned</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>I16vec8</td>
<td>unspecified</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Is16vec8</td>
<td>signed</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Iu16vec8</td>
<td>unsigned</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>I8vec16</td>
<td>unspecified</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Is8vec16</td>
<td>signed</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Iu8vec16</td>
<td>unsigned</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Most classes contain similar functionality for all data types and are represented by all available intrinsics. However, some capabilities do not translate from one data type to another without suffering from poor performance, and are therefore excluded from individual classes.
NOTE. Intrinsics that take immediate values and cannot be expressed easily in classes are not implemented. (For example, _mm_shuffle_ps, _mm_shuffle_pi16, _mm_shuffle_ps, _mm_extract_pi16, _mm_insert_pi16).

Access to Classes Using Header Files

The required class header files are installed in the include directory with the Intel® C++ Compiler. To enable the classes, use the #include directive in your program file as shown in the table that follows.

Include Directives for Enabling Classes

<table>
<thead>
<tr>
<th>Instruction Set Extension</th>
<th>Include Directive</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMX Technology</td>
<td>#include &lt;ivec.h&gt;</td>
</tr>
<tr>
<td>Streaming SIMD Extensions</td>
<td>#include &lt;fvec.h&gt;</td>
</tr>
<tr>
<td>Streaming SIMD Extensions 2</td>
<td>#include &lt;dvec.h&gt;</td>
</tr>
</tbody>
</table>

Each succeeding file from the top down includes the preceding class. You only need to include fvec.h if you want to use both the Ivec and Fvec classes. Similarly, to use all the classes including those for the Streaming SIMD Extensions 2, you need only to include the dvec.h file.

Usage Precautions

When using the C++ classes, you should follow some general guidelines. More detailed usage rules for each class are listed in Integer Vector Classes, and Floating-point Vector Classes.

Clear MMX Registers

If you use both the Ivec and Fvec classes at the same time, your program could mix MMX instructions, called by Ivec classes, with Intel x87 architecture floating-point instructions, called by Fvec classes. Floating-point instructions exist in the following Fvec functions:

- fvec constructors
- debug functions (cout and element access)
- rsqrt_nr
**NOTE.** MMX registers are aliased on the floating-point registers, so you should clear the MMX state with the EMMS instruction intrinsic before issuing an x87 floating-point instruction, as in the following example.

```c++
ivecA = ivecA & ivecB;  // Ivec logical operation that uses MMX instructions
empty ();  // clear state
cout << f32vec4a;  // F32vec4 operation that uses x87 floating-point instructions
```

**CAUTION.** Failure to clear the MMX registers can result in incorrect execution or poor performance due to an incorrect register state.

**Follow EMMS Instruction Guidelines**

Intel strongly recommends that you follow the guidelines for using the EMMS instruction. Refer to this topic before coding with the `Ivec` classes.

**Capabilities of C++ SIMD Classes**

The fundamental capabilities of each C++ SIMD class include:

- computation
- horizontal data motion
- branch compression/elimination
- caching hints

Understanding each of these capabilities and how they interact is crucial to achieving desired results.

**Computation**

The SIMD C++ classes contain vertical operator support for most arithmetic operations, including shifting and saturation.
Computation operations include: +, -, *, /, reciprocal (rcp and rcp_nr), square root (sqrt), reciprocal square root (rsqrt and rsqrt_nr).

Operations rcp and rsqrt are new approximating instructions with very short latencies that produce results with at least 12 bits of accuracy. Operations rcp_nr and rsqrt_nr use software refining techniques to enhance the accuracy of the approximations, with a minimal impact on performance. (The "nr" stands for Newton-Raphson, a mathematical technique for improving performance using an approximate result.)

**Horizontal Data Support**

The C++ SIMD classes provide horizontal support for some arithmetic operations. The term "horizontal" indicates computation across the elements of one vector, as opposed to the vertical, element-by-element operations on two different vectors.

The add_horizontal, unpack_low and pack_sat functions are examples of horizontal data support. This support enables certain algorithms that cannot exploit the full potential of SIMD instructions.

Shuffle intrinsics are another example of horizontal data flow. Shuffle intrinsics are not expressed in the C++ classes due to their immediate arguments. However, the C++ class implementation enables you to mix shuffle intrinsics with the other C++ functions. For example:

```cpp
F32vec4 fveca, fvecb, fvecd;
fveca += fvecb;
fvecd = _mm_shuffle_ps(fveca,fvecb,0);
```

Typically every instruction with horizontal data flow contains some inefficiency in the implementation. If possible, implement your algorithms without using the horizontal capabilities.

**Branch Compression/Elimination**

Branching in SIMD architectures can be complicated and expensive, possibly resulting in poor predictability and code expansion. The SIMD C++ classes provide functions to eliminate branches, using logical operations, max and min functions, conditional selects, and compares.

Consider the following example:

```cpp
short a[4], b[4], c[4];
for (i=0; i<4; i++)
c[i] = a[i] > b[i] ? a[i] : b[i];
```
This operation is independent of the value of $i$. For each $i$, the result could be either $A$ or $B$ depending on the actual values. A simple way of removing the branch altogether is to use the `select_gt` function, as follows:

```c
Is16vec4 a, b, c

c = select_gt(a, b, a, b)
```

**Caching Hints**

Streaming SIMD Extensions provide prefetching and streaming hints. Prefetching data can minimize the effects of memory latency. Streaming hints allow you to indicate that certain data should not be cached. This results in higher performance for data that should be cached.

**Integer Vector Classes**

**Overview: Integer Vector Classes**

The `Ivec` classes provide an interface to SIMD processing using integer vectors of various sizes. The class hierarchy is represented in the following figure.

**Ivec Class Hierarchy**
The M64 and M128 classes define the __m64 and __m128i data types from which the rest of the Ivec classes are derived. The first generation of child classes are derived based solely on bit sizes of 128, 64, 32, 16, and 8 respectively for the I128vec1, I64vec1, I64vec2, I32vec2, I32vec4, I16vec4, I16vec8, I8vec16, and I8vec8 classes. The latter seven of these classes require specification of signedness and saturation.

**CAUTION.** Do not intermix the M64 and M128 data types. You will get unexpected behavior if you do.

The signedness is indicated by the s and u in the class names:

- Is64vec2
- Iu64vec2
- Is32vec4
- Iu32vec4
- Is16vec8
- Iu16vec8
- Is8vec16
- Iu8vec16
- Is32vec2
- Iu32vec2
- Is16vec4
- Iu16vec4
- Is8vec8
- Iu8vec8

**Terms, Conventions, and Syntax Defined**

The following are special terms and syntax used in this chapter to describe functionality of the classes with respect to their associated operations.

**Ivec Class Syntax Conventions**

The name of each class denotes the data type, signedness, bit size, number of elements using the following generic format:

```
<type><signedness><bits>vec<elements>
```

```
{ F | I } { s | u } { 64 | 32 | 16 | 8 } vec { 8 | 4 | 2 | 1 }
```

where
Special Terms and Conventions

The following terms are used to define the functionality and characteristics of the classes and operations defined in this manual.

- **Nearest Common Ancestor** -- This is the intermediate or parent class of two classes of the same size. For example, the nearest common ancestor of `Iu8vec8` and `Is8vec8` is `I8vec8`. Also, the nearest common ancestor between `Iu8vec8` and `I16vec4` is `M64`.

- **Casting** -- Changes the data type from one class to another. When an operation uses different data types as operands, the return value of the operation must be assigned to a single data type. Therefore, one or more of the data types must be converted to a required data type. This conversion is known as a typecast. Sometimes, typecasting is automatic, other times you must use special syntax to explicitly typecast it yourself.

- **Operator Overloading** -- This is the ability to use various operators on the same user-defined data type of a given class. Once you declare a variable, you can add, subtract, multiply, and perform a range of operations. Each family of classes accepts a specified range of operators, and must comply by rules and restrictions regarding typecasting and operator overloading as defined in the header files. The following table shows the notation used in this documentation to address typecasting, operator overloading, and other rules.

### Class Syntax Notation Conventions

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`I[s</td>
<td>u][N]vec[N]`</td>
</tr>
<tr>
<td><code>I64vec1</code></td>
<td><code>__m64</code> data type</td>
</tr>
<tr>
<td>`I[s</td>
<td>u]64vec2`</td>
</tr>
</tbody>
</table>
### Class Name

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I[s</td>
<td>u]32vec4</td>
</tr>
<tr>
<td>I[s</td>
<td>u]8vec16</td>
</tr>
<tr>
<td>I[s</td>
<td>u]16vec8</td>
</tr>
<tr>
<td>I[s</td>
<td>u]32vec2</td>
</tr>
<tr>
<td>I[s</td>
<td>u]16vec4</td>
</tr>
<tr>
<td>I[s</td>
<td>u]8vec8</td>
</tr>
</tbody>
</table>

### Rules for Operators

To use operators with the Ivec classes you must use one of the following three syntax conventions:

\[
[Ivec\_Class] \ R = [Ivec\_Class] \ A \ [\ operator ] [Ivec\_Class] \ B
\]

**Example 1:**

\[
I64vec1 \ R = I64vec1 \ A \ & \ I64vec1 \ B;
\]

**Example 2:**

\[
I64vec1 \ R = \ \text{andnot}(I64vec1 \ A, \ I64vec1 \ B);
\]

**Example 3:**

\[
I64vec1 \ R \ &= \ I64vec1 \ A;
\]

*\[\ operator \]* an operator (for example, &, |, or ^)

*\[Ivec\_Class\]* an Ivec class

R, A, B variables declared using the pertinent Ivec classes

The table that follows shows automatic and explicit sign and size typecasting. "Explicit" means that it is illegal to mix different types without an explicit typecasting. "Automatic" means that you can mix types freely and the compiler will do the typecasting for you.

### Summary of Rules Major Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Sign Typecasting</th>
<th>Size Typecasting</th>
<th>Other Typecasting Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Operators</td>
<td>Sign Typecasting</td>
<td>Size Typecasting</td>
<td>Other Typecasting Requirements</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Logical</td>
<td>Automatic</td>
<td>Automatic (to left)</td>
<td>Explicit typecasting is required for different types used in non-logical expressions on the right side of the assignment.</td>
</tr>
<tr>
<td>Addition and Subtraction</td>
<td>Automatic</td>
<td>Explicit</td>
<td>N/A</td>
</tr>
<tr>
<td>Multiplication</td>
<td>Automatic</td>
<td>Explicit</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift</td>
<td>Automatic</td>
<td>Explicit</td>
<td>Casting Required to ensure arithmetic shift.</td>
</tr>
<tr>
<td>Compare</td>
<td>Automatic</td>
<td>Explicit</td>
<td>Explicit casting is required for signed classes for the less-than or greater-than operations.</td>
</tr>
<tr>
<td>Conditional Select</td>
<td>Automatic</td>
<td>Explicit</td>
<td>Explicit casting is required for signed classes for less-than or greater-than operations.</td>
</tr>
</tbody>
</table>

**Data Declaration and Initialization**

The following table shows literal examples of constructor declarations and data type initialization for all class sizes. All values are initialized with the most significant element on the left and the least significant to the right.

**Declaration and Initialization Data Types for Ivec Classes**
<table>
<thead>
<tr>
<th>Operation</th>
<th>Class</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>M128</td>
<td>I128vec1 A; Iu8vec16 A;</td>
</tr>
<tr>
<td>Declaration</td>
<td>M64</td>
<td>I64vec1 A; Iu8vec16 A;</td>
</tr>
<tr>
<td>__m128 Initialization</td>
<td>M128</td>
<td>I128vec1 A(__m128 m); Iu16vec8(__m128 m);</td>
</tr>
<tr>
<td>__m64 Initialization</td>
<td>M64</td>
<td>I64vec1 A(__m64 m); Iu8vec8 A(__m64 m);</td>
</tr>
<tr>
<td>__int64 Initialization</td>
<td>M64</td>
<td>I64vec1 A = __int64 m; Iu8vec8 A = __int64 m;</td>
</tr>
<tr>
<td>int i Initialization</td>
<td>M64</td>
<td>I64vec1 A = int i; Iu8vec8 A = int i;</td>
</tr>
<tr>
<td>int Initialization</td>
<td>I32vec2</td>
<td>I32vec2 A(int A1, int A0); I32vec2 A(signed int A1, signed int A0); I32vec2 A(unsigned int A1, unsigned int A0);</td>
</tr>
<tr>
<td>int Initialization</td>
<td>I32vec4</td>
<td>I32vec4 A(short A3, short A2, short A1, short A0); I32vec4 A(signed short A3, ..., signed short A0); I32vec4 A(unsigned short A3, ..., unsigned short A0);</td>
</tr>
<tr>
<td>Operation</td>
<td>Class</td>
<td>Syntax</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>short int</td>
<td>I16vec4</td>
<td>I16vec4 A(short A3, short A2, short A1, short A0);</td>
</tr>
<tr>
<td>Initialization</td>
<td></td>
<td>Is16vec4 A(signed short A3, ..., signed short A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iu16vec4 A(unsigned short A3, ..., unsigned short A0);</td>
</tr>
<tr>
<td>short int</td>
<td>I16vec8</td>
<td>I16vec8 A(short A7, short A6, ..., short A1, short A0);</td>
</tr>
<tr>
<td>Initialization</td>
<td></td>
<td>Is16vec8 A(signed A7, ..., signed short A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iu16vec8 A(unsigned short A7, ..., unsigned short A0);</td>
</tr>
<tr>
<td>char</td>
<td>I8vec8</td>
<td>I8vec8 A(char A7, char A6, ..., char A1, char A0);</td>
</tr>
<tr>
<td>Initialization</td>
<td></td>
<td>Is8vec8 A(signed char A7, ..., signed char A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iu8vec8 A(unsigned char A7, ..., unsigned char A0);</td>
</tr>
<tr>
<td>char</td>
<td>I8vec16</td>
<td>I8vec16 A(char A15, ..., char A0);</td>
</tr>
<tr>
<td>Initialization</td>
<td></td>
<td>Is8vec16 A(signed char A15, ..., signed char A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iu8vec16 A(unsigned char A15, ..., unsigned char A0);</td>
</tr>
</tbody>
</table>
Assignment Operator

Any Ivec object can be assigned to any other Ivec object; conversion on assignment from one Ivec object to another is automatic.

Assignment Operator Examples

Is16vec4 A;
Is8vec8 B;
I64vec1 C;
A = B; /* assign Is8vec8 to Is16vec4 */
B = C; /* assign I64vec1 to Is8vec8 */
B = A & C; /* assign M64 result of '&' to Is8vec8 */

Logical Operators

The logical operators use the symbols and intrinsics listed in the following table.

<table>
<thead>
<tr>
<th>Bitwise Operation</th>
<th>Operator Symbols</th>
<th>Syntax Usage</th>
<th>Corresponding Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>w/assign</td>
<td>Standard</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;</td>
<td>&amp;=</td>
<td>R = A &amp; B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
<td>R = A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>^</td>
<td>^=</td>
<td>R = A^B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANDNOT</td>
<td>andnot</td>
<td>N/A</td>
<td>R = A andnot B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Logical Operators and Miscellaneous Exceptions

A and B converted to M64. Result assigned to Iu8vec8.

I64vec1 A;
Is8vec8 B;
Iu8vec8 C;
C = A & B;

Same size and signedness operators return the nearest common ancestor.

I32vec2 R = Is32vec2 A ^ Iu32vec2 B;

A&B returns M64, which is cast to Iu8vec8.

C = Iu8vec8(A&B) + C;

When A and B are of the same class, they return the same type. When A and B are of different classes, the return value is the return type of the nearest common ancestor.

The logical operator returns values for combinations of classes, listed in the following tables, apply when A and B are of different classes.

**Ivec Logical Operator Overloading**

<table>
<thead>
<tr>
<th>Return (R)</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>NAND</th>
<th>A Operand</th>
<th>B Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>I64vec1</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td>I[s</td>
<td>u]64vec2 A</td>
</tr>
<tr>
<td>I64vec2</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td>I[s</td>
<td>u]64vec2 A</td>
</tr>
<tr>
<td>I32vec2</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td>I[s</td>
<td>u]32vec2 A</td>
</tr>
<tr>
<td>I32vec4</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td>I[s</td>
<td>u]32vec4 A</td>
</tr>
<tr>
<td>I16vec4</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td>I[s</td>
<td>u]16vec4 A</td>
</tr>
<tr>
<td>I16vec8</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td>I[s</td>
<td>u]16vec8 A</td>
</tr>
<tr>
<td>I8vec8</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td>I[s</td>
<td>u]8vec8 A</td>
</tr>
<tr>
<td>I8vec16</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td>I[s</td>
<td>u]8vec16 A</td>
</tr>
</tbody>
</table>
For logical operators with assignment, the return value of \( R \) is always the same data type as the pre-declared value of \( R \) as listed in the table that follows.

**Ivec Logical Operator Overloading with Assignment**

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Left Side (R)</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>Right Side (Any Ivec Type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I128vec1</td>
<td>I128vec1 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I64vec1</td>
<td>I64vec1 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I64vec2</td>
<td>I64vec2 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I[x]32vec4</td>
<td>I[x]32vec4 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I[x]32vec2</td>
<td>I[x]32vec2 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I[x]16vec8</td>
<td>I[x]16vec8 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I[x]16vec4</td>
<td>I[x]16vec4 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I[x]8vec16</td>
<td>I[x]8vec16 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I[x]8vec8</td>
<td>I[x]8vec8 R</td>
<td>&amp;=</td>
<td></td>
<td></td>
<td>A;</td>
</tr>
</tbody>
</table>

**Addition and Subtraction Operators**

The addition and subtraction operators return the class of the nearest common ancestor when the right-side operands are of different signs. The following code provides examples of usage and miscellaneous exceptions.

**Syntax Usage for Addition and Subtraction Operators**

Return nearest common ancestor type, I16vec4.
Is16vec4 A;
Iu16vec4 B;
I16vec4 C;
C = A + B;

Returns type left-hand operand type.

Is16vec4 A;
Iu16vec4 B;
A += B;
B -= A;

Explicitly convert B to Is16vec4.

Is16vec4 A,C;
Iu32vec24 B;
C = A + C;
C = A + (Is16vec4)B;

### Addition and Subtraction Operators with Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbols</th>
<th>Syntax</th>
<th>Corresponding Intrinsics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>R = A + B</td>
<td>_mm_add_epi64</td>
</tr>
<tr>
<td></td>
<td>+=</td>
<td>R += A</td>
<td>_mm_add_epi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_epi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_epi8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_pi8</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>R = A - B</td>
<td>_mm_sub_epi64</td>
</tr>
<tr>
<td></td>
<td>-=</td>
<td>R -= A</td>
<td>_mm_sub_epi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_epi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_epi8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_pi8</td>
</tr>
</tbody>
</table>
The following table lists addition and subtraction return values for combinations of classes when the right side operands are of different signedness. The two operands must be the same size, otherwise you must explicitly indicate the typecasting.

### Addition and Subtraction Operator Overloading

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Available Operators</th>
<th>Right Side Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Add</td>
<td>Sub</td>
</tr>
<tr>
<td>I64vec2 R</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I32vec4 R</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I32vec2 R</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I16vec8 R</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I16vec4 R</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I8vec8 R</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I8vec16 R</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

The following table shows the return data type values for operands of the addition and subtraction operators with assignment. The left side operand determines the size and signedness of the return value. The right side operand must be the same size as the left operand; otherwise, you must use an explicit typecast.

### Addition and Subtraction with Assignment

<table>
<thead>
<tr>
<th>Return Value (R)</th>
<th>Left Side (R)</th>
<th>Add</th>
<th>Sub</th>
<th>Right Side (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I[x]32vec4</td>
<td>I[x]32vec2 R</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]32vec2 R</td>
<td>I[x]32vec2 R</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>Return Value (R)</td>
<td>Left Side (R)</td>
<td>Add</td>
<td>Sub</td>
<td>Right Side (A)</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>-----</td>
<td>-----</td>
<td>----------------</td>
</tr>
<tr>
<td>I[x]16vec8</td>
<td>I[x]16vec8</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]16vec4</td>
<td>I[x]16vec4</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]8vec16</td>
<td>I[x]8vec16</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]8vec8</td>
<td>I[x]8vec8</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
</tbody>
</table>

**Multiplication Operators**

The multiplication operators can only accept and return data types from the I[s|u]16vec4 or I[s|u]16vec8 classes, as shown in the following example.

**Syntax Usage for Multiplication Operators**

Explicitly convert B to Is16vec4.

```plaintext
Is16vec4 A,C;
Iu32vec2 B;
C = A * C;
C = A * (Is16vec4)B;
```

Return nearest common ancestor type, I16vec4

```plaintext
Is16vec4 A;
Iu16vec4 B;
I16vec4 C;
C = A + B;
```

The `mul_high` and `mul_add` functions take Is16vec4 data only.

```plaintext
Is16vec4 A,B,C,D;
C = mul_high(A,B);
D = mul_add(A,B);
```
### Multiplication Operators with Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Syntax Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>R = A * B</td>
</tr>
<tr>
<td>*=</td>
<td>R *= A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mul_high</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R = mul_high(A, B)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mul_add</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R = mul_high(A, B)</td>
</tr>
</tbody>
</table>

The multiplication return operators always return the nearest common ancestor as listed in the table that follows. The two operands must be 16 bits in size, otherwise you must explicitly indicate typecasting.

### Multiplication Operator Overloading

<table>
<thead>
<tr>
<th>R</th>
<th>Mul</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>I16vec4 R</td>
<td>*</td>
<td>I[s</td>
</tr>
<tr>
<td>I16vec8 R</td>
<td>*</td>
<td>I[s</td>
</tr>
<tr>
<td>Is16vec4 R</td>
<td>mul_add</td>
<td>Is16vec4 A</td>
</tr>
<tr>
<td>Is16vec8</td>
<td>mul_add</td>
<td>Is16vec8 A</td>
</tr>
<tr>
<td>Is32vec2 R</td>
<td>mul_high</td>
<td>Is16vec4 A</td>
</tr>
<tr>
<td>Is32vec4 R</td>
<td>mul_high</td>
<td>s16vec8 A</td>
</tr>
</tbody>
</table>
The following table shows the return values and data type assignments for operands of the multiplication operators with assignment. All operands must be 16 bytes in size. If the operands are not the right size, you must use an explicit typecast.

**Multiplication with Assignment**

<table>
<thead>
<tr>
<th>Return Value (R)</th>
<th>Left Side (R)</th>
<th>Mul</th>
</tr>
</thead>
<tbody>
<tr>
<td>I[x]16vec8</td>
<td>I[x]16vec8</td>
<td>*=</td>
</tr>
<tr>
<td>I[x]16vec4</td>
<td>I[x]16vec4</td>
<td>*=</td>
</tr>
</tbody>
</table>

**Shift Operators**

The right shift argument can be any integer or Ivec value, and is implicitly converted to a M64 data type. The first or left operand of a << can be of any type except I[s|u]8vec[8|16].

**Example Syntax Usage for Shift Operators**

Automatic size and sign conversion.

Is16vec4 A, C;
Iu32vec2 B;
C = A;

A&B returns I16vec4, which must be cast to Iu16vec4 to ensure logical shift, not arithmetic shift.

Is16vec4 A, C;
Iu16vec4 B, R;
R = (Iu16vec4)(A & B) C;

A&B returns I16vec4, which must be cast to Is16vec4 to ensure arithmetic shift, not logical shift.

R = (Is16vec4)(A & B) C;

**Shift Operators with Corresponding Intrinsics**
Right shift operations with signed data types use arithmetic shifts. All unsigned and intermediate classes correspond to logical shifts. The following table shows how the return type is determined by the first argument type.

**Shift Operator Overloading**

<table>
<thead>
<tr>
<th>Option</th>
<th>R</th>
<th>Right Shift</th>
<th>Left Shift</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>I64vec1</td>
<td>&gt;&gt;</td>
<td>&lt;&lt;=</td>
<td>I64vec1</td>
<td>I64vec1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;&gt;&gt;=</td>
<td>&lt;=</td>
<td>A; B</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>I32vec2</td>
<td>&gt;&gt;</td>
<td>&lt;&lt;=</td>
<td>I32vec2</td>
<td>I32vec2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;&gt;&gt;=</td>
<td>&lt;=</td>
<td>A; B</td>
<td></td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Is32vec2</td>
<td>&gt;&gt;</td>
<td>&lt;&lt;=</td>
<td>Is32vec2</td>
<td>Is32vec2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;&gt;&gt;=</td>
<td>&lt;=</td>
<td>A; B</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>Iu32vec2</td>
<td>&gt;&gt;</td>
<td>&lt;&lt;=</td>
<td>Iu32vec2</td>
<td>Iu32vec2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;&gt;&gt;=</td>
<td>&lt;=</td>
<td>A; B</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>I16vec4</td>
<td>&gt;&gt;</td>
<td>&lt;&lt;=</td>
<td>I16vec4</td>
<td>I16vec4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;&gt;&gt;=</td>
<td>&lt;=</td>
<td>A; B</td>
<td></td>
</tr>
</tbody>
</table>
Comparison Operators

The equality and inequality comparison operands can have mixed signedness, but they must be of the same size. The comparison operators for less-than and greater-than must be of the same sign and size.

Example of Syntax Usage for Comparison Operator

The nearest common ancestor is returned for compare for equal/not-equal operations.

```c
Iu8vec8 A;
Is8vec8 B;
I8vec8 C;
C = cmpneq(A, B);
```

**Type cast needed for different-sized elements for equal/not-equal comparisons.**

```c
Iu8vec8 A, C;
Is16vec4 B;
C = cmpeq((Iu8vec8)A, B);
```

**Type cast needed for sign or size differences for less-than and greater-than comparisons.**

```c
Iu16vec4 A;
Is16vec4 B, C;
C = cmpge((Is16vec4)A, B);
C = cmptg(B, C);
```

Inequality Comparison Symbols and Corresponding Intrinsics
Comparison operators have the restriction that the operands must be the size and sign as listed in the Compare Operator Overloading table.

### Compare Operator Overloading

<table>
<thead>
<tr>
<th>R</th>
<th>Comparison</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>I32vec2 R</td>
<td>cmpeq</td>
<td>I[s</td>
<td>u]32vec2</td>
</tr>
<tr>
<td></td>
<td>cmpne</td>
<td>I[s</td>
<td>u]32vec2</td>
</tr>
<tr>
<td>I16vec4 R</td>
<td></td>
<td>I[s</td>
<td>u]16vec4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I[s</td>
<td>u]16vec4</td>
</tr>
<tr>
<td>I8vec8 R</td>
<td></td>
<td>I[s</td>
<td>u]8vec8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I[s</td>
<td>u]8vec8</td>
</tr>
<tr>
<td>R</td>
<td>Comparison</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>I32vec2 R</td>
<td>cmpgt</td>
<td>Is32vec2 B</td>
<td>Is32vec2 B</td>
</tr>
<tr>
<td></td>
<td>cmpge</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cmplt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cmple</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I16vec4 R</td>
<td></td>
<td>Is16vec4 B</td>
<td>Is16vec4 B</td>
</tr>
<tr>
<td>I8vec8 R</td>
<td></td>
<td>Is8vec8 B</td>
<td>Is8vec8 B</td>
</tr>
</tbody>
</table>

**Conditional Select Operators**

For conditional select operands, the third and fourth operands determine the type returned. Third and fourth operands with same size, but different signedness, return the nearest common ancestor data type.

**Conditional Select Syntax Usage**

Return the nearest common ancestor data type if third and fourth operands are of the same size, but different signs.

I16vec4 R = select_neq(Is16vec4, Is16vec4, Is16vec4, Iu16vec4);

**Conditional Select for Equality**

R0 := (A0 == B0) ? C0 : D0;
R1 := (A1 == B1) ? C1 : D1;
R2 := (A2 == B2) ? C2 : D2;
R3 := (A3 == B3) ? C3 : D3;

**Conditional Select for Inequality**

R0 := (A0 != B0) ? C0 : D0;
R1 := (A1 != B1) ? C1 : D1;
R2 := (A2 != B2) ? C2 : D2;
R3 := (A3 != B3) ? C3 : D3;

**Conditional Select Symbols and Corresponding Intrinsics**
<table>
<thead>
<tr>
<th>Conditional Select For:</th>
<th>Operators</th>
<th>Syntax</th>
<th>Corresponding Intrinsic</th>
<th>Additional Intrinsic (Applies to All)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>select_eq</td>
<td>R = select_eq(A, B, C, D)</td>
<td>_mm_cmpeq_pi32</td>
<td>_mm_and_si64 _mm_or_si64 _mm_and_not_si64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_cmpeq_pi16</td>
<td>_mm_cmpeq_pi8 _mm_cmpgt_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_cmplt_pi32</td>
<td>_mm_cmplt_pi8 _mm_cmpl_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_cmpge_pi32</td>
<td>_mm_cmpge_pi8 _mm_cmplt_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_cmplt_pi16</td>
<td>_mm_cmplt_pi8 _mm_cmplt_pi16</td>
</tr>
</tbody>
</table>

All conditional select operands must be of the same size. The return data type is the nearest common ancestor of operands C and D. For conditional select operations using greater-than or less-than operations, the first and second operands must be signed as listed in the table that follows.

**Conditional Select Operator Overloading**
The following table shows the mapping of return values from _R0_ to _R7_ for any number of elements. The same return value mappings also apply when there are fewer than four return values.

### Conditional Select Operator Return Value Mapping

<table>
<thead>
<tr>
<th>Return Value</th>
<th>A Operands</th>
<th>Available Operators</th>
<th>B Operands</th>
<th>C and D Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0 C0 : D0;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0 C1 : D1;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0 C2 : D2;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0 C3 : D3;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0 C4 : D4;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0 C5 : D5;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0 C6 : D6;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Debug Operations

The debug operations do not map to any compiler intrinsics for MMX™ instructions. They are provided for debugging programs only. Use of these operations may result in loss of performance, so you should not use them outside of debugging.

Output

The four 32-bit values of A are placed in the output buffer and printed in the following format (default in decimal):

```cpp
cout << Is32vec4 A;
cout << Iu32vec4 A;
cout << hex << Iu32vec4 A; /* print in hex format */
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"
```

Corresponding Intrinsics: none

The two 32-bit values of A are placed in the output buffer and printed in the following format (default in decimal):

```cpp
cout << Is32vec2 A;
cout << Iu32vec2 A;
cout << hex << Iu32vec2 A; /* print in hex format */
"[1]:A1 [0]:A0"
```

Corresponding Intrinsics: none

The eight 16-bit values of A are placed in the output buffer and printed in the following format (default in decimal):

```cpp
cout << Is16vec8 A;
cout << Iu16vec8 A;
cout << hex << Iu16vec8 A; /* print in hex format */
```
The four 16-bit values of \( A \) are placed in the output buffer and printed in the following format (default in decimal):

\[
\text{cout} \ll \text{Is16vec4} \ A; \\
\text{cout} \ll \text{Iu16vec4} \ A; \\
\text{cout} \ll \text{hex} \ll \text{Iu16vec4} \ A; /* \text{print in hex format} */
\]

Corresponding Intrinsics: none

The sixteen 8-bit values of \( A \) are placed in the output buffer and printed in the following format (default is decimal):

\[
\text{cout} \ll \text{Is8vec16} \ A; \text{cout} \ll \text{Iu8vec16} \ A; \text{cout} \ll \text{hex} \ll \text{Iu8vec8} \ A; \\
/* \text{print in hex format instead of decimal} */
\]

Corresponding Intrinsics: none

The eight 8-bit values of \( A \) are placed in the output buffer and printed in the following format (default is decimal):

\[
\text{cout} \ll \text{Is8vec8} \ A; \text{cout} \ll \text{Iu8vec8} \ A; \text{cout} \ll \text{hex} \ll \text{Iu8vec8} \ A; \\
/* \text{print in hex format instead of decimal} */
\]

Corresponding Intrinsics: none

**Element Access Operators**

\[
\text{int} \ R = \text{Is64vec2} \ A[i]; \\
\text{unsigned int} \ R = \text{Iu64vec2} \ A[i]; \\
\text{int} \ R = \text{Is32vec4} \ A[i]; \\
\text{unsigned int} \ R = \text{Iu32vec4} \ A[i]; \\
\text{int} \ R = \text{Is32vec2} \ A[i];
\]
unsigned int R = Iu32vec2 A[i];
short R = Is16vec8 A[i];
unsigned short R = Iu16vec8 A[i];
short R = Is16vec4 A[i];
unsigned short R = Iu16vec4 A[i];
signed char R = Is8vec16 A[i];
unsigned char R = Iu8vec16 A[i];
signed char R = Is8vec8 A[i];
unsigned char R = Iu8vec8 A[i];

Access and read element \(i\) of \(A\). If DEBUG is enabled and the user tries to access an element outside of \(A\), a diagnostic message is printed and the program aborts.

Corresponding Intrinsics: none

**Element Assignment Operators**

Is64vec2 A[i] = int R;
Is32vec4 A[i] = int R;
Iu32vec4 A[i] = unsigned int R;
Is32vec2 A[i] = int R;
Iu32vec2 A[i] = unsigned int R;
Is16vec8 A[i] = short R;
Iu16vec8 A[i] = unsigned short R;
Is16vec4 A[i] = short R;
Iu16vec4 A[i] = unsigned short R;
Is8vec16 A[i] = signed char R;
Iu8vec16 A[i] = unsigned char R;
Is8vec8 A[i] = signed char R;
Iu8vec8 A[i] = unsigned char R;
Assign $R$ to element $i$ of $A$. If `DEBUG` is enabled and the user tries to assign a value to an element outside of $A$, a diagnostic message is printed and the program aborts.

**Corresponding Intrinsics:** none

**Unpack Operators**

Interleave the 64-bit value from the high half of $A$ with the 64-bit value from the high half of $B$.

```plaintext
I364vec2 unpack_high(I64vec2 A, I64vec2 B);
Is64vec2 unpack_high(Is64vec2 A, Is64vec2 B);
Iu64vec2 unpack_high(Iu64vec2 A, Iu64vec2 B);
R0 = A1;
R1 = B1;
```

**Corresponding intrinsic:** `_mm_unpackhi_epi64`

Interleave the two 32-bit values from the high half of $A$ with the two 32-bit values from the high half of $B$.

```plaintext
I32vec4 unpack_high(I32vec4 A, I32vec4 B);
Is32vec4 unpack_high(Is32vec4 A, Is32vec4 B);
Iu32vec4 unpack_high(Iu32vec4 A, Iu32vec4 B);
R0 = A1;
R1 = B1;
R2 = A2;
R3 = B2;
```

**Corresponding intrinsic:** `_mm_unpackhi_epi32`

Interleave the 32-bit value from the high half of $A$ with the 32-bit value from the high half of $B$.

```plaintext
I32vec2 unpack_high(I32vec2 A, I32vec2 B);
Is32vec2 unpack_high(Is32vec2 A, Is32vec2 B);
Iu32vec2 unpack_high(Iu32vec2 A, Iu32vec2 B);
R0 = A1;
R1 = B1;
```

**Corresponding intrinsic:** `_mm_unpackhi_pi32`
Interleave the four 16-bit values from the high half of $A$ with the two 16-bit values from the high half of $B$.

$I_{16}\text{vec}8$ unpack_high($I_{16}\text{vec}8$ A, $I_{16}\text{vec}8$ B);

$I_{16}\text{vec}8$ unpack_high($I_{16}\text{vec}8$ A, $I_{16}\text{vec}8$ B);

$I_{16}\text{vec}8$ unpack_high($I_{16}\text{vec}8$ A, $I_{16}\text{vec}8$ B);

$I_{16}\text{vec}8$ unpack_high($I_{16}\text{vec}8$ A, $I_{16}\text{vec}8$ B);

$R_0 = A_2; R_1 = B_2; R_2 = A_3; R_3 = B_3$;

Corresponding intrinsic: ${\_}_m{\_}\text{unpackhi_epi16}$

Interleave the two 16-bit values from the high half of $A$ with the two 16-bit values from the high half of $B$.

$I_{16}\text{vec}4$ unpack_high($I_{16}\text{vec}4$ A, $I_{16}\text{vec}4$ B);

$I_{16}\text{vec}4$ unpack_high($I_{16}\text{vec}4$ A, $I_{16}\text{vec}4$ B);

$I_{16}\text{vec}4$ unpack_high($I_{16}\text{vec}4$ A, $I_{16}\text{vec}4$ B);

$I_{16}\text{vec}4$ unpack_high($I_{16}\text{vec}4$ A, $I_{16}\text{vec}4$ B);

$R_0 = A_2; R_1 = B_2; R_2 = A_3; R_3 = B_3$;

Corresponding intrinsic: ${\_}_m{\_}\text{unpackhi_pi16}$

Interleave the four 8-bit values from the high half of $A$ with the four 8-bit values from the high half of $B$.

$I_{8}\text{vec}8$ unpack_high($I_{8}\text{vec}8$ A, $I_{8}\text{vec}8$ B);

$I_{8}\text{vec}8$ unpack_high($I_{8}\text{vec}8$ A, $I_{8}\text{vec}8$ B);

$I_{8}\text{vec}8$ unpack_high($I_{8}\text{vec}8$ A, $I_{8}\text{vec}8$ B);

$I_{8}\text{vec}8$ unpack_high($I_{8}\text{vec}8$ A, $I_{8}\text{vec}8$ B);

$R_0 = A_4; R_1 = B_4; R_2 = A_5; R_3 = B_5; R_4 = A_6; R_5 = B_6; R_6 = A_7; R_7 = B_7$;

Corresponding intrinsic: ${\_}_m{\_}\text{unpackhi_pi8}$
Interleave the sixteen 8-bit values from the high half of A with the four 8-bit values from the high half of B.

\[ I\text{8vec16 unpack\_high}(I\text{8vec16 }A, I\text{8vec16 }B); \]
\[ I\text{s8vec16 unpack\_high}(I\text{s8vec16 }A, I\text{8vec16 }B); \]
\[ I\text{u8vec16 unpack\_high}(I\text{u8vec16 }A, I\text{8vec16 }B); \]

\[ R0 = A8; \]
\[ R1 = B8; \]
\[ R2 = A9; \]
\[ R3 = B9; \]
\[ R4 = A10; \]
\[ R5 = B10; \]
\[ R6 = A11; \]
\[ R7 = B11; \]
\[ R8 = A12; \]
\[ R9 = B12; \]
\[ R10 = A13; \]
\[ R11 = B13; \]
\[ R12 = A14; \]
\[ R13 = B14; \]
\[ R14 = A15; \]
\[ R15 = B15; \]

**Corresponding intrinsic:** \_mm\_unpackhi\_epi16

Interleave the 32-bit value from the low half of A with the 32-bit value from the low half of B

\[ R0 = A0; \]
\[ R1 = B0; \]

**Corresponding intrinsic:** \_mm\_unpacklo\_epi32

Interleave the 64-bit value from the low half of A with the 64-bit values from the low half of B

\[ I\text{64vec2 unpack\_low}(I\text{64vec2 }A, I\text{64vec2 }B); \]
\[ I\text{s64vec2 unpack\_low}(I\text{s64vec2 }A, I\text{s64vec2 }B); \]
\[ I\text{u64vec2 unpack\_low}(I\text{u64vec2 }A, I\text{u64vec2 }B); \]

\[ R0 = A0; \]
\[ R1 = B0; \]
\[ R2 = A1; \]
\[ R3 = B1; \]

**Corresponding intrinsic:** \_mm\_unpacklo\_epi32
Interleave the two 32-bit values from the low half of A with the two 32-bit values from the low half of B.

\[
\begin{align*}
I32vec4 \quad & \text{unpack\_low}(I32vec4 \ A, I32vec4 \ B); \\
Is32vec4 \quad & \text{unpack\_low}(Is32vec4 \ A, Is32vec4 \ B); \\
Iu32vec4 \quad & \text{unpack\_low}(Iu32vec4 \ A, Iu32vec4 \ B);
\end{align*}
\]

R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;

**Corresponding intrinsic:** \_mm\_unpacklo\_epi32

Interleave the 32-bit value from the low half of A with the 32-bit value from the low half of B.

\[
\begin{align*}
I32vec2 \quad & \text{unpack\_low}(I32vec2 \ A, I32vec2 \ B); \\
Is32vec2 \quad & \text{unpack\_low}(Is32vec2 \ A, Is32vec2 \ B); \\
Iu32vec2 \quad & \text{unpack\_low}(Iu32vec2 \ A, Iu32vec2 \ B);
\end{align*}
\]

R0 = A0;
R1 = B0;

**Corresponding intrinsic:** \_mm\_unpacklo\_pi32

Interleave the two 16-bit values from the low half of A with the two 16-bit values from the low half of B.

\[
\begin{align*}
I16vec8 \quad & \text{unpack\_low}(I16vec8 \ A, I16vec8 \ B); \\
Is16vec8 \quad & \text{unpack\_low}(Is16vec8 \ A, Is16vec8 \ B); \\
Iu16vec8 \quad & \text{unpack\_low}(Iu16vec8 \ A, Iu16vec8 \ B);
\end{align*}
\]

R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
R4 = A2;
R5 = B2;
R6 = A3;
R7 = B3;

**Corresponding intrinsic:** \_mm\_unpacklo\_epi16

Interleave the two 16-bit values from the low half of A with the two 16-bit values from the low half of B.
I16vec4 unpack_low(I16vec4 A, I16vec4 B);
Is16vec4 unpack_low(Is16vec4 A, Is16vec4 B);
Iu16vec4 unpack_low(Iu16vec4 A, Iu16vec4 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;

**Corresponding intrinsic:** _mm_unpacklo_pi16

Interleave the four 8-bit values from the high low of \(A\) with the four 8-bit values from the low half of \(B\).

I8vec16 unpack_low(I8vec16 A, I8vec16 B);
Is8vec16 unpack_low(Is8vec16 A, Is8vec16 B);
Iu8vec16 unpack_low(Iu8vec16 A, Iu8vec16 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
R4 = A2;
R5 = B2;
R6 = A3;
R7 = B3;
R8 = A4;
R9 = B4;
R10 = A5;
R11 = B5;
R12 = A6;
R13 = B6;
R14 = A7;
R15 = B7;

**Corresponding intrinsic:** _mm_unpacklo_epi8

Interleave the four 8-bit values from the high low of \(A\) with the four 8-bit values from the low half of \(B\).

I8vec8 unpack_low(I8vec8 A, I8vec8 B);
Is8vec8 unpack_low(Is8vec8 A, Is8vec8 B);
Iu8vec8 unpack_low(Iu8vec8 A, Iu8vec8 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
R4 = A2;
R5 = B2;
R6 = A3;
R7 = B3;

Corresponding intrinsic: _mm_unpacklo_pi8

Pack Operators
Pack the eight 32-bit values found in A and B into eight 16-bit values with signed saturation.

Is16vec8 pack_sat(Is32vec2 A, Is32vec2 B);
Corresponding intrinsic: _mm_packs_epi32

Pack the four 32-bit values found in A and B into eight 16-bit values with signed saturation.

Is16vec4 pack_sat(Is32vec2 A, Is32vec2 B);
Corresponding intrinsic: _mm_packs_pi32

Pack the sixteen 16-bit values found in A and B into sixteen 8-bit values with signed saturation.

Is8vec16 pack_sat(Is16vec4 A, Is16vec4 B);
Corresponding intrinsic: _mm_packs_epi16

Pack the eight 16-bit values found in A and B into eight 8-bit values with signed saturation.

Is8vec8 pack_sat(Is16vec4 A, Is16vec4 B);
Corresponding intrinsic: _mm_packs_pl16

Pack the sixteen 16-bit values found in A and B into sixteen 8-bit values with unsigned saturation.

Iu8vec16 packu_sat(Is16vec4 A, Is16vec4 B);
Corresponding intrinsic: _mm_packus_epi16

Pack the eight 16-bit values found in A and B into eight 8-bit values with unsigned saturation.

Iu8vec8 packu_sat(Is16vec4 A, Is16vec4 B);
Corresponding intrinsic: _mm_packus_pl16

Clear MMX™ State Operator
Empty the MMX™ registers and clear the MMX state. Read the guidelines for using the EMMS instruction intrinsic.

void empty(void);
Integer Functions for Streaming SIMD Extensions

NOTE. You must include fvec.h header file for the following functionality.

Compute the element-wise maximum of the respective signed integer words in A and B.
\[ \text{Is16vec4 } \text{simd_max}(\text{Is16vec4 } A, \text{Is16vec4 } B); \]
Corresponding intrinsic: \_mm\_max\_pi16

Compute the element-wise minimum of the respective signed integer words in A and B.
\[ \text{Is16vec4 } \text{simd_min}(\text{Is16vec4 } A, \text{Is16vec4 } B); \]
Corresponding intrinsic: \_mm\_min\_pi16

Compute the element-wise maximum of the respective unsigned bytes in A and B.
\[ \text{Iu8vec8 } \text{simd_max}(\text{Iu8vec8 } A, \text{Iu8vec8 } B); \]
Corresponding intrinsic: \_mm\_max\_pu8

Compute the element-wise minimum of the respective unsigned bytes in A and B.
\[ \text{Iu8vec8 } \text{simd_min}(\text{Iu8vec8 } A, \text{Iu8vec8 } B); \]
Corresponding intrinsic: \_mm\_min\_pu8

Create an 8-bit mask from the most significant bits of the bytes in A.
\[ \text{int } \text{move} \text{\_mask}(\text{I8vec8 } A); \]
Corresponding intrinsic: \_mm\_movemask\_pi8

Conditionally store byte elements of A to address p. The high bit of each byte in the selector B determines whether the corresponding byte in A will be stored.
\[ \text{void } \text{mask} \text{\_move}(\text{I8vec8 } A, \text{I8vec8 } B, \text{signed} \ \text{char} *p); \]
Corresponding intrinsic: \_mm\_maskmove\_si64

Store the data in A to the address p without polluting the caches. A can be any Ivec type.
\[ \text{void } \text{store} \text{\_nta}(\_m64 *p, \_M64 \ A); \]
Corresponding intrinsic: \_mm\_stream\_pi

Compute the element-wise average of the respective unsigned 8-bit integers in A and B.
\[ \text{Iu8vec8 } \text{simd} \text{\_avg}(\text{Iu8vec8 } A, \text{Iu8vec8 } B); \]
Corresponding intrinsic: \_mm\_avg\_pu8

Compute the element-wise average of the respective unsigned 16-bit integers in A and B.
Iu16vec4 simd_avg(Iu16vec4 A, Iu16vec4 B);

**Corresponding intrinsic:** \_mm\_avg\_pu16

### Conversions between Fvec and Ivec

Convert the lower double-precision floating-point value of \( A \) to a 32-bit integer with truncation.

```cpp
int F64vec2ToInt(F64vec4 A);
```

```cpp
r := (int)A0;
```

Convert the four floating-point values of \( A \) to two the two least significant double-precision floating-point values.

```cpp
F64vec2 F32vec4ToF64vec2(F32vec4 A);
```

```cpp
r0 := (double)A0;
r1 := (double)A1;
```

Convert the two double-precision floating-point values of \( A \) to two single-precision floating-point values.

```cpp
F32vec4 F64vec2ToF32vec4(F64vec2 A);
```

```cpp
r0 := (float)A0;
r1 := (float)A1;
```

Convert the signed int in \( B \) to a double-precision floating-point value and pass the upper double-precision value from \( A \) through to the result.

```cpp
F64vec2 InttoF64vec2(F64vec2 A, int B);
```

```cpp
r0 := (double)B;
r1 := A1;
```

Convert the lower floating-point value of \( A \) to a 32-bit integer with truncation.

```cpp
int F32vec4ToInt(F32vec4 A);
```

```cpp
r := (int)A0;
```

Convert the two lower floating-point values of \( A \) to two 32-bit integer with truncation, returning the integers in packed form.

```cpp
Is32vec2 F32vec4ToIs32vec2(F32vec4 A);
```

```cpp
r0 := (int)A0;
r1 := (int)A1;
```

Convert the 32-bit integer value \( B \) to a floating-point value; the upper three floating-point values are passed through from \( A \).

```cpp
F32vec4 IntToF32vec4(F32vec4 A, int B);
```

```cpp
r0 := (float)B;
r1 := A1;
```
r2 := A2;
r3 := A3;

Convert the two 32-bit integer values in packed form in B to two floating-point values; the upper two floating-point values are passed through from A.

F32vec4 Is32vec2ToF32vec4(F32vec4 A, Is32vec2 B);
r0 := (float)B0;
r1 := (float)B1;
r2 := A2;
r3 := A3;

Floating-point Vector Classes

Overview: Floating-point Vector Classes

The floating-point vector classes, F64vec2, F32vec4, and F32vec1, provide an interface to SIMD operations. The class specifications are as follows:

F64vec2 A(double x, double y);
F32vec4 A(float z, float y, float x, float w);
F32vec1 B(float w);

The packed floating-point input values are represented with the right-most value lowest as shown in the following table.

Single-Precision Floating-point Elements
This reference uses the following conventions for syntax and return values.

### Fvec Classes Syntax Notation

Fvec classes use the syntax conventions shown the following examples:

**Example 1:**
```
```

**Example 2:**
```
[Fvec_Class] R = [operator]([Fvec_Class] A, [Fvec_Class] B);
```

**Example 3:**
```
[Fvec_Class] R [operator] = [Fvec_Class] A;
```

where

- `[operator]` is an operator (for example, `&`, `|`, `^` or `^`)
- `[Fvec_Class]` is any Fvec class (F64vec2, F32vec4, or F32vec1)
R, A, B are declared Fvec variables of the type indicated.

**Return Value Notation**

Because the Fvec classes have packed elements, the return values typically follow the conventions presented in the Return Value Convention Notation Mappings table. F32vec4 returns four single-precision, floating-point values (R0, R1, R2, and R3); F64vec2 returns two double-precision, floating-point values, and F32vec1 returns the lowest single-precision floating-point value (R0).

**Return Value Convention Notation Mappings**

<table>
<thead>
<tr>
<th>Example 1:</th>
<th>Example 2:</th>
<th>Example 3:</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 := A0 &amp; B0;</td>
<td>R0 := A0 andnot B0;</td>
<td>R0 := A0;</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>R1 := A1 &amp; B1;</td>
<td>R1 := A1 andnot B1;</td>
<td>R1 := A1;</td>
<td>x</td>
<td>x</td>
<td>N/A</td>
</tr>
<tr>
<td>R2 := A2 &amp; B2;</td>
<td>R2 := A2 andnot B2;</td>
<td>R2 := A2;</td>
<td>x</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>R3 := A3 &amp; B3</td>
<td>R3 := A3 andhot B3;</td>
<td>R3 := A3;</td>
<td>x</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Data Alignment**

Memory operations using the Streaming SIMD Extensions should be performed on 16-byte-aligned data whenever possible. F32vec4 and F64vec2 object variables are properly aligned by default. Note that floating point arrays are not automatically aligned. To get 16-byte alignment, you can use the alignment __declspec:

```c
__declspec( align(16) ) float A[4];
```

**Conversions**

All Fvec object variables can be implicitly converted to __m128 data types. For example, the results of computations performed on F32vec4 or F32vec1 object variables can be assigned to __m128 data types.
Constructors and Initialization

The following table shows how to create and initialize F32vec objects with the Fvec classes.

### Constructors and Initialization for Fvec Classes

<table>
<thead>
<tr>
<th>Example</th>
<th>Intrinsic</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constructor Declaration</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F64vec2 A;</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>F32vec4 B;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F32vec1 C;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **__m128 Object Initialization** |                  |         |
| F64vec2 A(__m128d mm);         | N/A               | N/A     |
| F32vec4 B(__m128 mm);          |                  |         |
| F32vec1 C(__m128 mm);          |                  |         |

<p>| <strong>Double Initialization</strong>      |                  |         |
| / * Initializes two doubles. */ | __mm_set_pd      |         |
| F64vec2 A(double d0,           | A0 := d0;        |         |
| double d1);                   | A1 := d1;        |         |
| F64vec2 A =                   |                  |         |
| F64vec2(double d0,            |                  |         |
| double d1);                   |                  |         |
| F64vec2 A(double d0);         | __mm_set1_pd     |         |
| / * Initializes both return values with the same double precision value */. | A0 := d0;        |         |
|                                | A1 := d0;        |         |</p>
<table>
<thead>
<tr>
<th>Float Initialization</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F32vec4 A(float f3, float f2, float f1, float f0);</td>
<td>_mm_set_ps A0 := f0; A1 := f1; A2 := f2; A3 := f3;</td>
</tr>
<tr>
<td>F32vec4 A = F32vec4(float f3, float f2, float f1, float f0);</td>
<td></td>
</tr>
<tr>
<td>F32vec4 A(float f0); _mm_set1_ps A0 := f0; /* Initializes all return values with the same floating point value. */</td>
<td></td>
</tr>
<tr>
<td>F32vec4 A(double d0); _mm_set1_ps(d) A0 := d0; /* Initialize all return values with the same double-precision value. */</td>
<td></td>
</tr>
<tr>
<td>F32vec1 A(double d0); _mm_set_ss(d) A0 := d0; /* Initializes the lowest value of A with d0 and the other values with 0. */</td>
<td></td>
</tr>
<tr>
<td>F32vec1 B(float f0); _mm_set ss B0 := f0; /* Initializes the lowest value of B with f0 and the other values with 0. */</td>
<td></td>
</tr>
<tr>
<td>F32vec1 B(int I); _mm_cvtsi32_ss B0 := f0; /* Initializes the lowest value of B */</td>
<td></td>
</tr>
</tbody>
</table>
Float Initialization

with f0, other values
are undefined.*/

Arithmetic Operators

The following table lists the arithmetic operators of the `Fvec` classes and generic syntax. The operators have been divided into standard and advanced operations, which are described in more detail later in this section.

**Fvec Arithmetic Operators**

<table>
<thead>
<tr>
<th>Category</th>
<th>Operation</th>
<th>Operators</th>
<th>Generic Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>Addition</td>
<td>+</td>
<td>R = A + B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+=</td>
<td>R += A;</td>
</tr>
<tr>
<td></td>
<td>Subtraction</td>
<td>-</td>
<td>R = A - B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-=</td>
<td>R -= A;</td>
</tr>
<tr>
<td></td>
<td>Multiplication</td>
<td>*</td>
<td>R = A * B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*=</td>
<td>R *= A;</td>
</tr>
<tr>
<td></td>
<td>Division</td>
<td>/</td>
<td>R = A / B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/=</td>
<td>R /= A;</td>
</tr>
<tr>
<td>Advanced</td>
<td>Square Root</td>
<td>sqrt</td>
<td>R = sqrt(A);</td>
</tr>
<tr>
<td></td>
<td>Reciprocal</td>
<td>rcp</td>
<td>R = rcp(A);</td>
</tr>
<tr>
<td></td>
<td>(Newton-Raphson)</td>
<td>rcp_nr</td>
<td>R = rcp_nr(A);</td>
</tr>
<tr>
<td></td>
<td>Reciprocal Square</td>
<td>rsqrt</td>
<td>R = rsqrt(A);</td>
</tr>
<tr>
<td></td>
<td>Root</td>
<td>rsqrt_nr</td>
<td>R = rsqrt_nr(A);</td>
</tr>
</tbody>
</table>

**Standard Arithmetic Operator Usage**

The following two tables show the return values for each class of the standard arithmetic operators, which use the syntax styles described earlier in the Return Value Notation section.
### Standard Arithmetic Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>Operators</th>
<th>B</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 :=</td>
<td>A0</td>
<td>+</td>
<td>−</td>
<td>*</td>
<td>/</td>
<td>B0</td>
</tr>
<tr>
<td>R1 :=</td>
<td>A1</td>
<td>+</td>
<td>−</td>
<td>*</td>
<td>/</td>
<td>B1</td>
</tr>
<tr>
<td>R2 :=</td>
<td>A2</td>
<td>+</td>
<td>−</td>
<td>*</td>
<td>/</td>
<td>B2</td>
</tr>
<tr>
<td>R3 :=</td>
<td>A3</td>
<td>+</td>
<td>−</td>
<td>*</td>
<td>/</td>
<td>B3</td>
</tr>
</tbody>
</table>

### Arithmetic with Assignment Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>Operators</th>
<th>A</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 :=</td>
<td>+ - * / = /=</td>
<td>A0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>R1 :=</td>
<td>+ - * / = /=</td>
<td>A1</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>R2 :=</td>
<td>+ - * / = /=</td>
<td>A2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>R3 :=</td>
<td>+ - * / = /=</td>
<td>A3</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

This table lists standard arithmetic operator syntax and intrinsics.

### Standard Arithmetic Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
<th>Example Syntax Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>4 floats</td>
<td>F32vec4 R = _mm_add_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 A +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 R +=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 A;</td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = _mm_add_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 A +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 R +=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 A;</td>
</tr>
<tr>
<td>Operation</td>
<td>Returns</td>
<td>Example Syntax Usage</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_add_ss</td>
</tr>
<tr>
<td></td>
<td>F32vec1 A +</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec1 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec1 R +=</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec1 A;</td>
<td></td>
</tr>
<tr>
<td>Subtraction</td>
<td>4 floats</td>
<td>F32vec4 R = _mm_sub_ps</td>
</tr>
<tr>
<td></td>
<td>F32vec4 A -</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec4 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec4 R -=</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec4 A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = _mm_sub_pd</td>
</tr>
<tr>
<td></td>
<td>F64vec2 A +</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec2 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F64vec2 R -=</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F64vec2 A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 float</td>
<td>F32vec1 R = _mm_sub_ss</td>
</tr>
<tr>
<td></td>
<td>F32vec1 A -</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec1 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec1 R -=</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec1 A;</td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>4 floats</td>
<td>F32vec4 R = _mm_mul_ps</td>
</tr>
<tr>
<td></td>
<td>F32vec4 A *</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec4 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec4 R *=</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F32vec4 A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = _mm_mul_pd</td>
</tr>
<tr>
<td></td>
<td>F64vec2 A *</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F364vec2 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F64vec2 R *=</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F64vec2 A;</td>
<td></td>
</tr>
</tbody>
</table>
Intrinsic Example Syntax

Returns Operation

```
F32vec1 R = _mm_mul_ss
F32vec1 A *
F32vec1 B;
F32vec1 R *=
F32vec1 A;
```

```
F32vec4 R = _mm_div_ps
F32vec4 A /
F32vec4 B;
F32vec4 R /=
F32vec4 A;
```

```
F64vec2 R = _mm_div_pd
F64vec2 A /
F64vec2 B;
F64vec2 R /=
F64vec2 A;
```

```
F32vec1 R = _mm_div_ss
F32vec1 A /
F32vec1 B;
F32vec1 R /=
F32vec1 A;
```

Division 4 floats

```
F32vec4 R = _mm_div_ps
F32vec4 A /
F32vec4 B;
F32vec4 R /=
F32vec4 A;
```

```
F64vec2 R = _mm_div_pd
F64vec2 A /
F64vec2 B;
F64vec2 R /=
F64vec2 A;
```

Advanced Arithmetic Operator Usage

The following table shows the return values classes of the advanced arithmetic operators, which use the syntax styles described earlier in the Return Value Notation section.

### Advanced Arithmetic Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>Operators</th>
<th>A</th>
<th>F32vec</th>
<th>F64vec</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:</td>
<td>sqrt</td>
<td>rcp</td>
<td>rsqrt</td>
<td>rpr</td>
</tr>
<tr>
<td>R1:</td>
<td>sqrt</td>
<td>rcp</td>
<td>rsqrt</td>
<td>rpr</td>
</tr>
</tbody>
</table>
This table shows examples for advanced arithmetic operators.

### Advanced Arithmetic Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Square Root</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = sqrt(F32vec4 A);</td>
<td>_mm_sqrt_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = sqrt(F64vec2 A);</td>
<td>_mm_sqrt_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = sqrt(F32vec1 A);</td>
<td>_mm_sqrt_ss</td>
</tr>
<tr>
<td><strong>Reciprocal</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = rcp(F32vec4 A);</td>
<td>_mm_rcp_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = rcp(F64vec2 A);</td>
<td>_mm_rcp_pd</td>
</tr>
</tbody>
</table>
### Reciprocal

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 float</td>
<td><code>F32vec1 R = rcp(F32vec1 A);</code></td>
</tr>
</tbody>
</table>

### Reciprocal Square Root

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td><code>F32vec4 R = _mm_rsqrt_ps rsqrt(F32vec4 A);</code></td>
</tr>
<tr>
<td>2 doubles</td>
<td><code>F64vec2 R = _mm_rsqrt_pd rsqrt(F64vec2 A);</code></td>
</tr>
<tr>
<td>1 float</td>
<td><code>F32vec1 R = _mm_rsqrt_ss rsqrt(F32vec1 A);</code></td>
</tr>
</tbody>
</table>

### Reciprocal Newton Raphson

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td><code>F32vec4 R = _mm_sub_ps _mm_add_ps _mm_mul_ps _mm_rcp_ps rcp_nr(F32vec4 A);</code></td>
</tr>
<tr>
<td>2 doubles</td>
<td><code>F64vec2 R = _mm_sub_pd _mm_add_pd _mm_mul_pd _mm_rcp_pd rcp_nr(F64vec2 A);</code></td>
</tr>
<tr>
<td>1 float</td>
<td><code>F32vec1 R = _mm_sub_ss _mm_add_ss _mm_mul_ss _mm_rcp_ss rcp_nr(F32vec1 A);</code></td>
</tr>
</tbody>
</table>

### Reciprocal Square Root Newton Raphson

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 float</td>
<td><code>F32vec4 R = _mm_sub_pd _mm_mul_pd _mm_rsqrt_ps rsqrt_nr(F32vec4 A);</code></td>
</tr>
<tr>
<td>Reciprocal Square Root Newton Raphson</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td></td>
</tr>
<tr>
<td>F64vec2 R = rsqrt_nr(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>_mm_sub_pd _mm_mul_pd _mm_rsqrt_pd</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td></td>
</tr>
<tr>
<td>F32vec1 R = rsqrt_nr(F32vec1 A);</td>
<td></td>
</tr>
<tr>
<td>_mm_sub_ss _mm_mul_ss _mm_rsqrt_ss</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Horizontal Add</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 float</td>
</tr>
<tr>
<td>float f = add_horizontal(F32vec4 A);</td>
</tr>
<tr>
<td>_mm_add_ss _mm_shuffle_ss</td>
</tr>
<tr>
<td>1 double</td>
</tr>
<tr>
<td>double d = add_horizontal(F64vec2 A);</td>
</tr>
<tr>
<td>_mm_add_sd _mm_shuffle_sd</td>
</tr>
</tbody>
</table>

**Minimum and Maximum Operators**

Compute the minimums of the two double precision floating-point values of \(A\) and \(B\).

\[
\begin{align*}
F64vec2 R &= \text{simd\_min}(F64vec2 A, F64vec2 B) \\
R0 &= \min(A0,B0) \\
R1 &= \min(A1,B1)
\end{align*}
\]

**Corresponding intrinsic:** _mm_min_pd

Compute the minimums of the four single precision floating-point values of \(A\) and \(B\).

\[
\begin{align*}
F32vec4 R &= \text{simd\_min}(F32vec4 A, F32vec4 B) \\
R0 &= \min(A0,B0) \\
R1 &= \min(A1,B1) \\
R2 &= \min(A2,B2) \\
R3 &= \min(A3,B3)
\end{align*}
\]

**Corresponding intrinsic:** _mm_min_ps

Compute the minimum of the lowest single precision floating-point values of \(A\) and \(B\).

\[
\begin{align*}
F32vec1 R &= \text{simd\_min}(F32vec1 A, F32vec1 B) \\
R0 &= \min(A0,B0)
\end{align*}
\]

**Corresponding intrinsic:** _mm_min_ss
Compute the maximums of the two double precision floating-point values of \(A\) and \(B\).

\[
F64vec2 \text{ simd\_max}(F64vec2 A, F64vec2 B) \\
R0 := \text{max}(A0,B0); \\
R1 := \text{max}(A1,B1);
\]

**Corresponding intrinsic:** `_mm_max_pd`

Compute the maximums of the four single precision floating-point values of \(A\) and \(B\).

\[
F32vec4 \text{ R = simd\_man}(F32vec4 A, F32vec4 B) \\
R0 := \text{max}(A0,B0); \\
R1 := \text{max}(A1,B1); \\
R2 := \text{max}(A2,B2); \\
R3 := \text{max}(A3,B3);
\]

**Corresponding intrinsic:** `_mm_max_ps`

Compute the maximum of the lowest single precision floating-point values of \(A\) and \(B\).

\[
F32vec1 \text{ simd\_max}(F32vec1 A, F32vec1 B) \\
R0 := \text{max}(A0,B0);
\]

**Corresponding intrinsic:** `_mm_max_ss`

**Logical Operators**

The following table lists the logical operators of the Fvec classes and generic syntax. The logical operators for F32vec1 classes use only the lower 32 bits.

**Fvec Logical Operators Return Value Mapping**

<table>
<thead>
<tr>
<th>Bitwise Operation</th>
<th>Operators</th>
<th>Generic Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;</td>
<td>(R = A &amp; B;)</td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>(R &amp;= A;)</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>^</td>
<td>(R = A ^ B;)</td>
</tr>
<tr>
<td></td>
<td>^</td>
<td>(R ^= A;)</td>
</tr>
<tr>
<td>andnot</td>
<td>andnot</td>
<td>(R = \text{andnot}(A);)</td>
</tr>
</tbody>
</table>

The following table lists standard logical operators syntax and corresponding intrinsics. Note that there is no corresponding scalar intrinsic for the F32vec1 classes, which accesses the lower 32 bits of the packed vector intrinsics.
Logical Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>4 floats</td>
<td>F32vec4 &amp; = _mm_and_ps F32vec4 A &amp; F32vec4 B; F32vec4 &amp; &amp;= F32vec4 A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 doubles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 float</td>
</tr>
<tr>
<td>OR</td>
<td>4 floats</td>
<td>F32vec4 R = _mm_or_ps F32vec4 A</td>
<td>F32vec4 B; F32vec4 R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 doubles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 float</td>
</tr>
</tbody>
</table>
**Operation** | **Returns** | **Example Syntax Usage** | **Intrinsic Usage**
---|---|---|---
XOR | 4 floats | F32vec4 R = _mm_xor_ps F32vec4 A ^ F32vec4 B; F32vec4 R ^= F32vec4 A; | 2 doubles | F64vec2 R = _mm_xor_pd F64vec2 A ^ F364vec2 B; F64vec2 R ^= F64vec2 A; |
1 float | F32vec1 R = _mm_xor_ps F32vec1 A ^ F32vec1 B; F32vec1 R ^= F32vec1 A; | 2 doubles | F64vec2 R = _mm_andnot_pd andnot(F64vec2 A, F64vec2 B); |
ANDNOT | 2 doubles | | |

**Compare Operators**

The operators described in this section compare the single precision floating-point values of A and B. Comparison between objects of any Fvec class return the same class being compared.

The following table lists the compare operators for the Fvec classes.

**Compare Operators and Corresponding Intrinsic**

<table>
<thead>
<tr>
<th>Compare For:</th>
<th>Operators</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>cmpeq</td>
<td>R = cmpeq(A, B)</td>
</tr>
<tr>
<td>Inequality</td>
<td>cmpneq</td>
<td>R = cmpneq(A, B)</td>
</tr>
<tr>
<td>Greater Than</td>
<td>cmpgt</td>
<td>R = cmpgt(A, B)</td>
</tr>
</tbody>
</table>
Compare Operators

The mask is set to 0xffffffff for each floating-point value where the comparison is true and 0x00000000 where the comparison is false. The following table shows the return values for each class of the compare operators, which use the syntax described earlier in the Return Value Notation section.

Compare Operator Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>A0</th>
<th>For Any Operators</th>
<th>Bf (443)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:=</td>
<td>(A1)</td>
<td>cmp[eq</td>
<td>lt</td>
</tr>
<tr>
<td>!A1</td>
<td></td>
<td></td>
<td>1B</td>
</tr>
<tr>
<td>R1:=</td>
<td>(A1)</td>
<td>cmp[eq</td>
<td>lt</td>
</tr>
<tr>
<td>!A1</td>
<td></td>
<td></td>
<td>2B</td>
</tr>
<tr>
<td>R2:=</td>
<td>(A1)</td>
<td>cmp[eq</td>
<td>lt</td>
</tr>
<tr>
<td>!A1</td>
<td></td>
<td></td>
<td>3B</td>
</tr>
<tr>
<td>R3:=</td>
<td>A3</td>
<td>cmp[eq</td>
<td>lt</td>
</tr>
</tbody>
</table>
The following table shows examples for arithmetic operators and intrinsics.

## Compare Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare for Equality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = mm_cmpeq_ps ( \text{cmpeq}(\text{F32vec4} \ A) );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>mm_cmpeq_pd ( \text{cmpeq}(\text{F64vec2} \ A) );</td>
</tr>
<tr>
<td></td>
<td>1 float</td>
<td>mm_cmpeq_ss ( \text{cmpeq}(\text{F32vec1} \ A) );</td>
</tr>
<tr>
<td><strong>Compare for Inequality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = mm_cmpneq_ps ( \text{cmpneq}(\text{F32vec4} \ A) );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>mm_cmpneq_pd ( \text{cmpneq}(\text{F64vec2} \ A) );</td>
</tr>
<tr>
<td></td>
<td>1 float</td>
<td>mm_cmpneq_ss ( \text{cmpneq}(\text{F32vec1} \ A) );</td>
</tr>
<tr>
<td><strong>Compare for Less Than</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = mm_cmplt_ps ( \text{cmplt}(\text{F32vec4} \ A) );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>mm_cmplt_pd ( \text{cmplt}(\text{F64vec2} \ A) );</td>
</tr>
<tr>
<td></td>
<td>1 float</td>
<td>mm_cmplt_ss ( \text{cmplt}(\text{F32vec1} \ A) );</td>
</tr>
</tbody>
</table>
### Compare for Less Than or Equal

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>(_{\text{mm_cmple_ps}} \text{F32vec4 R = cmple(F32vec4 \ A)};)</td>
</tr>
<tr>
<td>2 doubles</td>
<td>(_{\text{mm_cmple_pd}} \text{F64vec2 R = cmple(F64vec2 \ A)};)</td>
</tr>
<tr>
<td>1 float</td>
<td>(_{\text{mm_cmple_pd}} \text{F32vec1 R = cmple(F32vec1 \ A)};)</td>
</tr>
</tbody>
</table>

### Compare for Greater Than

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>(_{\text{mm_cmpgt_ps}} \text{F32vec4 R = cmpgt(F32vec4 \ A)};)</td>
</tr>
<tr>
<td>2 doubles</td>
<td>(_{\text{mm_cmpgt_pd}} \text{F64vec2 R = cmpgt(F32vec42 \ A)};)</td>
</tr>
<tr>
<td>1 float</td>
<td>(_{\text{mm_cmpgt_ss}} \text{F32vec1 R = cmpgt(F32vec1 \ A)};)</td>
</tr>
</tbody>
</table>

### Compare for Greater Than or Equal To

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>(_{\text{mm_cmpge_ps}} \text{F32vec4 R = cmpge(F32vec4 \ A)};)</td>
</tr>
<tr>
<td>2 doubles</td>
<td>(_{\text{mm_cmpge_pd}} \text{F64vec2 R = cmpge(F32vec42 \ A)};)</td>
</tr>
<tr>
<td>1 float</td>
<td>(_{\text{mm_cmpge_ss}} \text{F32vec1 R = cmpge(F32vec1 \ A)};)</td>
</tr>
</tbody>
</table>

### Compare for Not Less Than

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>(_{\text{mm_cmplnt_ps}} \text{F32vec4 R = cmplnt(F32vec4 \ A)};)</td>
</tr>
<tr>
<td>2 doubles</td>
<td>(_{\text{mm_cmplnt_pd}} \text{F64vec2 R = cmplnt(F32vec42 \ A)};)</td>
</tr>
<tr>
<td>Compare for Not Less Than</td>
<td>1 float</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare for Not Less Than or Equal</td>
<td>4 floats</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 float</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare for Not Greater Than</td>
<td>4 floats</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 float</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare for Not Greater Than or Equal</td>
<td>4 floats</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 float</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Conditional Select Operators for Fvec Classes

Each conditional function compares single-precision floating-point values of A and B. The C and D parameters are used for return value. Comparison between objects of any Fvec class returns the same class.

<table>
<thead>
<tr>
<th>Conditional Select for:</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td><code>select_eq</code></td>
</tr>
<tr>
<td>Inequality</td>
<td><code>select_neq</code></td>
</tr>
<tr>
<td>Greater Than</td>
<td><code>select_gt</code></td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td><code>select_ge</code></td>
</tr>
<tr>
<td>Not Greater Than</td>
<td><code>select_gt</code></td>
</tr>
<tr>
<td>Not Greater Than or Equal To</td>
<td><code>select_ge</code></td>
</tr>
</tbody>
</table>

**Syntax**

- `R = select_eq(A, B)`
- `R = select_neq(A, B)`
- `R = select_gt(A, B)`
- `R = select_ge(A, B)`

1826
### Conditional Select Operator Usage

For conditional select operators, the return value is stored in C if the comparison is true or in D if false. The following table shows the return values for each class of the conditional select operators, using the **Return Value Notation** described earlier.

#### Compare Operator Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>A0</th>
<th>Operators</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F32x4</th>
<th>F64x2</th>
<th>F32x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:</td>
<td>(A1</td>
<td>select_eq</td>
<td>lt</td>
<td>le</td>
<td>gt</td>
<td>B0)</td>
<td>C0</td>
<td>D0</td>
</tr>
<tr>
<td></td>
<td>!(A1</td>
<td>select_ne</td>
<td>nlt</td>
<td>nle</td>
<td>ngt</td>
<td>nge]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

[1827]
The following table shows examples for conditional select operations and corresponding intrinsics.

### Conditional Select Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare for Equality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = _mm_cmpeq_ps select_eq(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = _mm_cmpeq_pd select_eq(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = _mm_cmpeq_ss select_eq(F32vec1 A);</td>
<td></td>
</tr>
<tr>
<td><strong>Compare for Inequality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = _mm_cmpneq_ps select_neq(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = _mm_cmpneq_pd select_neq(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>Compare for Inequality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 float F32vec1 R = _mm_cmpneq_ss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>select_neq(F32vec1 A);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Less Than</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats F32vec4 R = _mm_cmplt_ps</td>
</tr>
<tr>
<td>select_lt(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles F64vec2 R = _mm_cmplt_pd</td>
</tr>
<tr>
<td>select_lt(F64vec2 A);</td>
</tr>
<tr>
<td>1 float F32vec1 R = _mm_cmplt_ss</td>
</tr>
<tr>
<td>select_lt(F32vec1 A);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Less Than or Equal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats F32vec4 R = _mm_cmple_ps</td>
</tr>
<tr>
<td>select_le(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles F64vec2 R = _mm_cmple_pd</td>
</tr>
<tr>
<td>select_le(F64vec2 A);</td>
</tr>
<tr>
<td>1 float F32vec1 R = _mm_cmple_ss</td>
</tr>
<tr>
<td>select_le(F32vec1 A);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Greater Than</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats F32vec4 R = _mm_cmpgt_ps</td>
</tr>
<tr>
<td>select_gt(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles F64vec2 R = _mm_cmpgt_pd</td>
</tr>
<tr>
<td>select_gt(F64vec2 A);</td>
</tr>
<tr>
<td>1 float F32vec1 R = _mm_cmpgt_ss</td>
</tr>
<tr>
<td>select_gt(F32vec1 A);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Greater Than or Equal To</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats F32vec1 R = _mm_cmpge_ps</td>
</tr>
<tr>
<td>select_ge(F32vec4 A);</td>
</tr>
</tbody>
</table>
### Compare for Greater Than or Equal To

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmpge_pd select_ge(F64vec2 A);</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmpge_ss select_ge(F32vec1 A);</td>
</tr>
</tbody>
</table>

### Compare for Not Less Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1 R =</td>
<td>_mm_cmpnlt_ps select_nlt(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmpnlt_pd select_nlt(F64vec2 A);</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmpnlt_ss select_nlt(F32vec1 A);</td>
</tr>
</tbody>
</table>

### Compare for Not Less Than or Equal

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1 R =</td>
<td>_mm_cmpnle_ps select_nle(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmpnle_pd select_nle(F64vec2 A);</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmpnle_ss select_nle(F32vec1 A);</td>
</tr>
</tbody>
</table>

### Compare for Not Greater Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1 R =</td>
<td>_mm_cmpngt_ps select_ngt(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmpngt_pd select_ngt(F64vec2 A);</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmpngt_ss select_ngt(F32vec1 A);</td>
</tr>
</tbody>
</table>
### Compare for Not Greater Than or Equal

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td><code>F32vec1 R = select_nge(F32vec4 A);</code></td>
</tr>
<tr>
<td></td>
<td><code>_mm_cmpnge_ps</code></td>
</tr>
<tr>
<td>2 doubles</td>
<td><code>F64vec2 R = select_nge(F64vec2 A);</code></td>
</tr>
<tr>
<td></td>
<td><code>_mm_cmpnge_pd</code></td>
</tr>
<tr>
<td>1 float</td>
<td><code>F32vec1 R = select_nge(F32vec1 A);</code></td>
</tr>
<tr>
<td></td>
<td><code>_mm_cmpnge_ss</code></td>
</tr>
</tbody>
</table>

### Cacheability Support Operators

Stores (non-temporal) the two double-precision, floating-point values of `A`. Requires a 16-byte aligned address.

```c
void store_nta(double *p, F64vec2 A);
```

Corresponding intrinsic: `_mm_stream_pd`

Stores (non-temporal) the four single-precision, floating-point values of `A`. Requires a 16-byte aligned address.

```c
void store_nta(float *p, F32vec4 A);
```

Corresponding intrinsic: `_mm_stream_ps`

### Debug Operations

The debug operations do not map to any compiler intrinsics for MMX™ technology or Streaming SIMD Extensions. They are provided for debugging programs only. Use of these operations may result in loss of performance, so you should not use them outside of debugging.

### Output Operations

The two single, double-precision floating-point values of `A` are placed in the output buffer and printed in decimal format as follows:

```c
cout << F64vec2 A;
"[1]:A1 [0]:A0"
```

Corresponding intrinsics: none

The four, single-precision floating-point values of `A` are placed in the output buffer and printed in decimal format as follows:

```c
cout << F32vec4 A;
```
The lowest, single-precision floating-point value of \( A \) is placed in the output buffer and printed.

```cpp
std::cout << F32vec1 A;
```

Corresponding intrinsics: none

### Element Access Operations

```cpp
double d = F64vec2 A[int i];
```

Read one of the two, double-precision floating-point values of \( A \) without modifying the corresponding floating-point value. Permitted values of \( i \) are 0 and 1. For example:

If DEBUG is enabled and \( i \) is not one of the permitted values (0 or 1), a diagnostic message is printed and the program aborts.

```cpp
double d = F64vec2 A[1];
```

Corresponding intrinsics: none

Read one of the four, single-precision floating-point values of \( A \) without modifying the corresponding floating-point value. Permitted values of \( i \) are 0, 1, 2, and 3. For example:

```cpp
float f = F32vec4 A[int i];
```

If DEBUG is enabled and \( i \) is not one of the permitted values (0-3), a diagnostic message is printed and the program aborts.

```cpp
float f = F32vec4 A[2];
```

Corresponding intrinsics: none

### Element Assignment Operations

```cpp
F64vec4 A[int i] = double d;
```

Modify one of the two, double-precision floating-point values of \( A \). Permitted values of \( i \) are 0 and 1. For example:

```cpp
F32vec4 A[int i] = float f;
```

Modify one of the four, single-precision floating-point values of \( A \). Permitted values of \( i \) are 0, 1, 2, and 3. For example:
If DEBUG is enabled and int i is not one of the permitted values (0-3), a diagnostic message is printed and the program aborts.

Corresponding intrinsics: none.

**Load and Store Operators**

Loads two, double-precision floating-point values, copying them into the two, floating-point values of A. No assumption is made for alignment.

```c
void loadu(F64vec2 A, double *p)
Corresponding intrinsic: _mm_loadu_pd
```
Stores the two, double-precision floating-point values of A. No assumption is made for alignment.

```c
void storeu(float *p, F64vec2 A);
Corresponding intrinsic: _mm_storeu_pd
```

Loads four, single-precision floating-point values, copying them into the four floating-point values of A. No assumption is made for alignment.

```c
void loadu(F32vec4 A, double *p)
Corresponding intrinsic: _mm_loadu_ps
```
Stores the four, single-precision floating-point values of A. No assumption is made for alignment.

```c
void storeu(float *p, F32vec4 A);
Corresponding intrinsic: _mm_storeu_ps
```

**Unpack Operators**

Selects and interleaves the lower, double-precision floating-point values from A and B.

```c
F64vec2 R = unpack_low(F64vec2 A, F64vec2 B);
Corresponding intrinsic: _mm_unpacklo_pd(a, b)
```

Selects and interleaves the higher, double-precision floating-point values from A and B.

```c
F64vec2 R = unpack_high(F64vec2 A, F64vec2 B);
Corresponding intrinsic: _mm_unpackhi_pd(a, b)
```

Selects and interleaves the lower two, single-precision floating-point values from A and B.

```c
F32vec4 R = unpack_low(F32vec4 A, F32vec4 B);
Corresponding intrinsic: _mm_unpacklo_ps(a, b)
```

Selects and interleaves the higher two, single-precision floating-point values from A and B.

```c
F32vec4 R = unpack_high(F32vec4 A, F32vec4 B);
```
Corresponding intrinsic: _mm_unpackhi_ps(a, b)

Move Mask Operators

Creates a 2-bit mask from the most significant bits of the two, double-precision floating-point values of A, as follows:

```c
int i = move_mask(F64vec2 A)
i := sign(a1)<<1 | sign(a0)<<0
```

Corresponding intrinsic: _mm_movemask_pd

Creates a 4-bit mask from the most significant bits of the four, single-precision floating-point values of A, as follows:

```c
int i = move_mask(F32vec4 A)
i := sign(a3)<<3 | sign(a2)<<2 | sign(a1)<<1 | sign(a0)<<0
```

Corresponding intrinsic: _mm_movemask_ps

Classes Quick Reference

This appendix contains tables listing operators to perform various SIMD operations, corresponding intrinsics to perform those operations, and the classes that implement those operations. The classes listed here belong to the Intel C++ Class Libraries for SIMD Operations.

In the following tables,

- N/A indicates that the operator is not implemented in that particular class. Thus, in the Logical Operations table, the Andnot operator is not implemented in the F32vec4 and F32vec1 classes.

- All other entries under Classes indicate that those operators are implemented in those particular classes, and the entries under the Classes columns provide the suffix for the corresponding intrinsic. For example, consider the Arithmetic Operations Part1 table, where the corresponding intrinsic is _mm_add_[x] and the entry epi16 is under the I16vec8 column. It means that the I16vec8 class implements the addition operators and the corresponding intrinsic is _mm_add_epi16.

Logical Operations:
**Operators** | **Corresponding Intrinsic** | **Classes**
---|---|---
<, <= | _mm_and_[x] | si128 si64 pd ps ps
| | | I128vec1, I64vec1, F64vec2, F32vec4, F32vec1 I64vec2, I32vec2, I32vec1, I16vec4, I16vec8, I8vec8 I8vec16
| | _mm_or_[x] | si128 si64 pd ps ps
| | _mm_xor_[x] | si128 si64 pd ps ps
| | _mm_andnot_[x] | si128 si64 pd N/A N/A

### Arithmetic Operations: Part 1

**Operators** | **Corresponding Intrinsic** | **Classes**
---|---|---
+, += | _mm_add_[x] | epi64 epi32 epi16 epi8
| | | I64vec2 I32vec4 I16vec8 I8vec16
| | _mm_sub_[x] | epi64 epi32 epi16 epi8
| | _mm_mullo_[x] | N/A N/A epi16 N/A
| | _mm_div_[x] | N/A N/A N/A N/A
| | _mm_mulhi_[x] | N/A N/A epi16 N/A
| | _mm_madd_[x] | N/A N/A epi16 N/A
| | _mm_sqrt_[x] | N/A N/A N/A N/A
| | _mm_rcp_[x] | N/A N/A N/A N/A
| | _mm_rcp_nr_[x] | N/A N/A N/A N/A
### Arithmetic Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsqrt</td>
<td><em>mm_rsqrt</em>[x]</td>
<td>I64vec2 I32vec4 I16vec8 I8vec16</td>
</tr>
<tr>
<td>rsqrt_nr</td>
<td><em>mm_rsqrt</em>[x]</td>
<td>N/A N/A N/A N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_sub</em>[x]</td>
<td>N/A N/A N/A N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_mul</em>[x]</td>
<td>N/A N/A N/A N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, +</td>
<td><em>mm_add</em>[x]</td>
<td>pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pi8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ss</td>
</tr>
<tr>
<td>-, -</td>
<td><em>mm_sub</em>[x]</td>
<td>pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pi8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ss</td>
</tr>
<tr>
<td>*, *=</td>
<td><em>mm_mullo</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ss</td>
</tr>
<tr>
<td>/, /=</td>
<td><em>mm_div</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ss</td>
</tr>
<tr>
<td>mul_high</td>
<td><em>mm_mulhi</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>mul_add</td>
<td><em>mm_madd</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>sqrt</td>
<td><em>mm_sqrt</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ss</td>
</tr>
<tr>
<td>rcp</td>
<td><em>mm_rcp</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ss</td>
</tr>
<tr>
<td>rcp_nr</td>
<td><em>mm_rcp</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Classes:**
- I64vec2
- I32vec4
- I16vec8
- I8vec16

**Intrinsic Operators:**
- _mm_rsqrt_ [x]
- _mm_sub_ [x]
- _mm_mul_ [x]
- _mm_div_ [x]
- _mm_mullo_ [x]
- _mm_madd_ [x]
- _mm_sqrt_ [x]
- _mm_rcp_ [x]
- _mm_add_ [x]
### Shift Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&gt;&gt;, &gt;&gt;=</code></td>
<td><code>_mm_srl_[x]</code></td>
<td>N/A epi64 epi32 epi16 N/A</td>
</tr>
<tr>
<td></td>
<td><code>_mm_srli_[x]</code></td>
<td>N/A epi64 epi32 epi16 N/A</td>
</tr>
<tr>
<td></td>
<td><code>_mm_sra_[x]</code></td>
<td>N/A N/A epi32 epi16 N/A</td>
</tr>
<tr>
<td></td>
<td><code>_mm_srai_[x]</code></td>
<td>N/A N/A epi32 epi16 N/A</td>
</tr>
<tr>
<td><code>&lt;&lt;, &lt;&lt;=</code></td>
<td><code>_mm_sll_[x]</code></td>
<td>N/A epi64 epi32 epi16 N/A</td>
</tr>
<tr>
<td></td>
<td><code>_mm_slli_[x]</code></td>
<td>N/A epi64 epi32 epi16 N/A</td>
</tr>
</tbody>
</table>

### Shift Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&gt;&gt;, &gt;&gt;=</code></td>
<td><code>_mm_srl_[x]</code></td>
<td>si64 pi32 pi16 N/A</td>
</tr>
<tr>
<td></td>
<td><code>_mm_srli_[x]</code></td>
<td>si64 pi32 pi16 N/A</td>
</tr>
<tr>
<td></td>
<td><code>_mm_sra_[x]</code></td>
<td>N/A pi32 pi16 N/A</td>
</tr>
<tr>
<td></td>
<td><code>_mm_srai_[x]</code></td>
<td>N/A pi32 pi16 N/A</td>
</tr>
<tr>
<td><code>&lt;&lt;, &lt;&lt;=</code></td>
<td><code>_mm_sll_[x]</code></td>
<td>si64 pi32 pi16 N/A</td>
</tr>
<tr>
<td></td>
<td><code>_mm_slli_[x]</code></td>
<td>si64 pi32 pi16 N/A</td>
</tr>
</tbody>
</table>

### Comparison Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmpeq</code></td>
<td><code>_mm_cmpeq_[x]</code></td>
<td>epi32 epi16 epi8 pi16 pi32 pi16 pi8</td>
</tr>
<tr>
<td><code>cmpneq</code></td>
<td><code>_mm_cmpeq_[x]</code></td>
<td>epi32 epi16 epi8 pi16 pi32 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td><code>_mm_and_not_[y]</code></td>
<td>si128 si128 si128 si164 si64 si128 si64</td>
</tr>
<tr>
<td><code>cmpgt</code></td>
<td><code>_mm_cmpgt_[x]</code></td>
<td>epi32 epi16 epi8 pi16 pi32 pi16 pi8</td>
</tr>
</tbody>
</table>
## Intrinsic Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmpge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>i32vec4 I16vec8 I16vec2 I16vec4 I8vec8</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>epi32  epi16 epi8  pi32  pi16  pi0  si128 si128 si128 si64 si64 si64</td>
</tr>
<tr>
<td>cmpgt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>epi32  epi16 epi8  pi32  pi16  pi0  si128 si128 si128 si64 si64 si64</td>
</tr>
<tr>
<td>cmpge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>epi32  epi16 epi8  pi32  pi16  pi0  si128 si128 si128 si64 si64 si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>epi32  epi16 epi8  pi32  pi16  pi0  si128 si128 si128 si64 si64 si64</td>
</tr>
<tr>
<td>cmpnle</td>
<td><em>mm_cmpnle</em>[x]</td>
<td>N/A    N/A   N/A    N/A    N/A    N/A</td>
</tr>
<tr>
<td>cmpnlt</td>
<td><em>mm_cmpnlt</em>[x]</td>
<td>N/A    N/A   N/A    N/A    N/A    N/A</td>
</tr>
<tr>
<td>cmplt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>epi32  epi16 epi8  pi32  pi16  pi0  si128 si128 si128 si64 si64 si64</td>
</tr>
<tr>
<td>cmple</td>
<td><em>mm_cmple</em>[x]</td>
<td>epi32  epi16 epi8  pi32  pi16  pi0  si128 si128 si128 si64 si64 si64</td>
</tr>
<tr>
<td>cmpnlt</td>
<td><em>mm_cmpnlt</em>[x]</td>
<td>N/A    N/A   N/A    N/A    N/A    N/A</td>
</tr>
<tr>
<td>cmpnle</td>
<td><em>mm_cmpnle</em>[x]</td>
<td>N/A    N/A   N/A    N/A    N/A    N/A</td>
</tr>
</tbody>
</table>

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.

### Comparison Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmpeq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>F64vec2 F32vec4 F32vec1</td>
</tr>
<tr>
<td>cmpneq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>F64vec2 F32vec4 F32vec1</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>pd     ps    ss</td>
</tr>
<tr>
<td>cmpgt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>F64vec2 F32vec4 F32vec1</td>
</tr>
<tr>
<td>cmpte</td>
<td><em>mm_cmpte</em>[x]</td>
<td>F64vec2 F32vec4 F32vec1</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>pd     ps    ss</td>
</tr>
<tr>
<td>Operators</td>
<td>Corresponding Intrinsic</td>
<td>Classes</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 F32vec4 F32vec1</td>
</tr>
<tr>
<td>cmplt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>pd      ps      ss</td>
</tr>
<tr>
<td>cmple</td>
<td><em>mm_cmple</em>[x]</td>
<td>pd      ps      ss</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td>cmpngt</td>
<td><em>mm_cmpngt</em>[x]</td>
<td>pd      ps      ss</td>
</tr>
<tr>
<td>cmpnge</td>
<td><em>mm_cmpnge</em>[x]</td>
<td>pd      ps      ss</td>
</tr>
<tr>
<td>cmnplnt</td>
<td><em>mm_cmpnlt</em>[x]</td>
<td>pd      ps      ss</td>
</tr>
<tr>
<td>cmnple</td>
<td><em>mm cmpnle</em>[x]</td>
<td>pd      ps      ss</td>
</tr>
</tbody>
</table>

**Conditional Select Operations: Part 1**

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I32vec4 I16vec8 I8vec16 I32vec2 I16vec4 I8vec8</td>
</tr>
<tr>
<td>select_eq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>epi32   epi16   epi8   p132   p16   p8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_and-not</em>[y]*</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
<tr>
<td>select_neq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>epi32   epi16   epi8   p132   p16   p8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_and-not</em>[y]*</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
<tr>
<td>select_gt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>epi32   epi16   epi8   p132   p16   p8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_and-not</em>[y]*</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>si128   si128   si128  si64   si64  si64</td>
</tr>
</tbody>
</table>
### Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I32vec4 I16vec8 I16vec8 I8vec16 I32vec2 I16vec4 I8vec8</td>
</tr>
<tr>
<td>select_ge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>epi32 epi16 epi16 pi32 pi16 pi16 pi0</td>
</tr>
<tr>
<td></td>
<td>_mm_and_ge[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td></td>
<td>_mm_or_ge[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td></td>
<td>_mm_andnot_ge[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td>select_lt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>epi32 epi16 epi16 pi32 pi16 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td>_mm_and_lt[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td></td>
<td>_mm_or_lt[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td></td>
<td>_mm_andnot_lt[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td>select_le</td>
<td><em>mm_cmlple</em>[x]</td>
<td>epi32 epi16 epi16 pi32 pi16 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td>_mm_and_le[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td></td>
<td>_mm_or_le[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td></td>
<td>_mm_andnot_le[y]</td>
<td>si128 si128 si128 si64 si64 si64 si64</td>
</tr>
<tr>
<td>select_ngt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>N/A N/A N/A N/A N/A N/A N/A</td>
</tr>
<tr>
<td>select_nge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>N/A N/A N/A N/A N/A N/A N/A</td>
</tr>
<tr>
<td>select_nlt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>N/A N/A N/A N/A N/A N/A N/A</td>
</tr>
<tr>
<td>select_nle</td>
<td><em>mm_cmlplt</em>[x]</td>
<td>N/A N/A N/A N/A N/A N/A N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmple</em>[x]</td>
<td>N/A N/A N/A N/A N/A N/A N/A</td>
</tr>
</tbody>
</table>

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.

### Conditional Select Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F64vec2 F32vec4 F32vec1</td>
</tr>
<tr>
<td>select_eq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd ps ss</td>
</tr>
<tr>
<td></td>
<td>_mm_and_eq[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_mm_or_eq[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_mm_andnot_eq[y]</td>
<td></td>
</tr>
</tbody>
</table>

---

1840
<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F64vec2</td>
</tr>
<tr>
<td>select_neq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_gt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_ge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_lt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_le</td>
<td><em>mm_cmple</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_ngt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td>select_nge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td>select_nlt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td>select_nle</td>
<td><em>mm_cmple</em>[x]</td>
<td>pd</td>
</tr>
</tbody>
</table>

**Packing and Unpacking Operations: Part 1**
### Classes Corresponding Intrinsic Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I16vec2 I32vec4 I16vec8 I18vec16 I32vec2</td>
</tr>
<tr>
<td>unpack_high</td>
<td><em>mm_unpackhi</em>[x]</td>
<td>epi64  epi32  epi16  epi8  pi32</td>
</tr>
<tr>
<td>unpack_low</td>
<td><em>mm_unpacklo</em>[x]</td>
<td>epi64  epi32  epi16  epi8  pi32</td>
</tr>
<tr>
<td>pack_sat</td>
<td><em>mm_packs</em>[x]</td>
<td>N/A    epi32  epi16  N/A   pi32</td>
</tr>
<tr>
<td>packu_sat</td>
<td><em>mm_packus</em>[x]</td>
<td>N/A    N/A    epi16  N/A   N/A</td>
</tr>
<tr>
<td>sat_add</td>
<td><em>mm_adds</em>[x]</td>
<td>N/A    N/A    epi16  epi8  N/A</td>
</tr>
<tr>
<td>sat_sub</td>
<td><em>mm_subs</em>[x]</td>
<td>N/A    N/A    epi16  epi8  N/A</td>
</tr>
</tbody>
</table>

### Packing and Unpacking Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I16vec4 I18vec8 F64vec2 F32vec4 F32vec1</td>
</tr>
<tr>
<td>unpack_high</td>
<td><em>mm_unpackhi</em>[x]</td>
<td>pl64   pl8   pd    ps    N/A</td>
</tr>
<tr>
<td>unpack_low</td>
<td><em>mm_unpacklo</em>[x]</td>
<td>pl64   pl8   pd    ps    N/A</td>
</tr>
<tr>
<td>pack_sat</td>
<td><em>mm_packs</em>[x]</td>
<td>pl64   N/A   N/A   N/A   N/A</td>
</tr>
<tr>
<td>packu_sat</td>
<td><em>mm_packus</em>[x]</td>
<td>pl64   N/A   N/A   N/A   N/A</td>
</tr>
<tr>
<td>sat_add</td>
<td><em>mm_adds</em>[x]</td>
<td>pl64   pl8   pd    ps    ss</td>
</tr>
<tr>
<td>sat_sub</td>
<td><em>mm_subs</em>[x]</td>
<td>pl64   pl8   pl8   pl8   pd</td>
</tr>
</tbody>
</table>

### Conversions Operations:

Conversion operations can be performed using intrinsics only. There are no classes implemented to correspond to these intrinsics.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>F64vec2ToInt</td>
<td>_mm_cvtsd_si32</td>
</tr>
<tr>
<td>Operators</td>
<td>Corresponding Intrinsic</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>F32vec4ToF64vec2</td>
<td>_mm_cvtps_pd</td>
</tr>
<tr>
<td>F64vec2ToF32vec4</td>
<td>_mm_cvtpd_ps</td>
</tr>
<tr>
<td>IntToF64vec2</td>
<td>_mm_cvtsi32_sd</td>
</tr>
<tr>
<td>F32vec4ToInt</td>
<td>_mm_cvtt_ss2si</td>
</tr>
<tr>
<td>F32vec4ToI32vec2</td>
<td>_mm_cvttps_pi32</td>
</tr>
<tr>
<td>IntToF32vec4</td>
<td>_mm_cvtsi32_ss</td>
</tr>
<tr>
<td>I32vec2ToF32vec4</td>
<td>_mm_cvtpi32_ps</td>
</tr>
</tbody>
</table>
## Programming Example

This sample program uses the F32vec4 class to average the elements of a 20 element floating point array.

```c
#include <fvec.h>

#define SHUFFLE(a,b,i) (F32vec4)_mm_shuffle_ps(a,b,i)
#include <stdio.h>
#define SIZE 20

float result;
_MM_ALIGN16 float array[SIZE];

void Add20ArrayElements (F32vec4 *array, float *result)
{
    F32vec4 vec0, vec1;
    vec0 = _mm_load_ps ((float *) array); // Load array's first 4 floats
    // Add all elements of the array, 4 elements at a time
}```
vec0 += array[1]; // Add elements 5-8
vec0 += array[2]; // Add elements 9-12
vec0 += array[3]; // Add elements 13-16
vec0 += array[4]; // Add elements 17-20
//***************************************************************************
// There are now 4 partial sums.
// Add the 2 lowers to the 2 raises,
// then add those 2 results together
//***************************************************************************
vec1 = SHUFFLE(vec1, vec0, 0x40);
vec0 += vec1;
vec1 = SHUFFLE(vec1, vec0, 0x30);
vec0 += vec1;
vec0 = SHUFFLE(vec0, vec0, 2);
_mm_store_ss (result, vec0); // Store the final sum
}
void main(int argc, char *argv[])
{
    int i;

    //Initialize the array
    for (i=0; i < SIZE; i++)
    {
        array[i] = (float) i;
    }
    //Call function to add all array elements

    Add20ArrayElements (array, &result);
//Print average array element value

    printf("Average of all array values = \%f\n", result/20.);
    printf("The correct answer is \%f\n\n", 9.5);
}

C++ Library Extensions

Introduction

This section contains descriptions of Intel's C++ library extensions that assist users in parallel programming. The following C++ library specialization is included:

- **C++ valarray implementation**: enables users to leverage a custom valarray header file that uses the Intel® Integrated Performance Primitives (Intel® IPP) for performance benefit.

Intel's valarray implementation

Introduction to Intel's valarray Implementation

The Intel® Compiler provides a high performance implementation of specialized one-dimensional valarray operations for the C++ standard STL valarray container.

The standard C++ valarray template consists of array/vector operations for high performance computing. These operations are designed to exploit high performance hardware features such as parallelism and achieve performance benefits.

Intel's valarray implementation uses the Intel® Integrated Performance Primitives (IPP), which is part of the product. Make sure you select IPP when you install the product.

The valarray implementation consists of a replacement header, `<valarray>`, that provides a specialized high performance implementation for the following operators and types:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Valarrays of Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs, acos, asin, atan, atan2, cos, cosh, exp, log, log10, pow, sin, sinh, sqrt, tan, tanh</td>
<td>float, double</td>
</tr>
<tr>
<td>addition, subtraction, division, multiplication</td>
<td>float, double</td>
</tr>
</tbody>
</table>
Using Intel's valarray Implementation

Intel's valarray implementation allows you to declare huge arrays for parallel processing. Improved implementation of valarray is tied up with calling the IPP libraries that are part of Intel® Integrated Performance Primitives (Intel® IPP). Intel® IPP is part of the product.

Using valarray in Source Code

To use valarrays in your source code, include the valarray header file, `<valarray>`. The `<valarray>` header file is located in the path `<installdir>/perf_header`.

The example code below shows a valarray addition operation (+) specialized through use of Intel's implementation of valarray:

```c
#include <valarray>

void test() {
    std::valarray<float> vi(N), va(N);
    --
    vi = vi + va; // array addition
    --
}
```

**NOTE.** To use the static merged library containing all CPU-specific optimized versions of the library code, you need to call the `ippStaticInit` function first, before any IPP calls. This ensures automatic dispatch to the correct version of the library code at runtime. If you do not call `ippStaticInit` first, the emerged library will use the generic instance of the code. If you are using the dynamic version of the libraries, you do not need to call `ippStaticInit`. 
Compiling valarray Source Code

To compile your valarray source code, the compiler option, /Quse-intel-optimized-headers (for Windows* OS) or -use-intel-optimized-headers (for Linux* and MacOS* X OSes), is used to include the required valarray header file and all the necessary IPP library files.

The following examples illustrate two instances of how to compile and link a program to include the Intel valarray replacement header file and link with the Intel® Integrated Performance Primitives (Intel® IPP) libraries. Refer to the Intel® IPP documentation for details.

In the following examples, "merged" libraries means using a static library that contains all the CPU-specific variants of the library code.

Windows* OS examples:
The following command line performs a one-step compilation for a system based on IA-32 architecture, running Windows OS:

icl /Quse-intel-optimized-headers source.cpp

The following command lines perform separate compile and link steps for a system based on IA-32 architecture, running Windows OS:

**DLL (dynamic):**
icl /Quse-intel-optimized-headers /c source.cpp
icl source.obj /Quse-intel-optimized-headers

**Merged (static):**
icl /Quse-intel-optimized-headers /c source.cpp
icl source.obj /Quse-intel-optimized-headers

Linux* OS examples:
The following command line performs a one-step compilation for a system based on Intel® 64 architecture, running Linux OS:

icpc -use-intel-optimized-headers source.cpp

The following command lines perform separate compile and link steps for a system based on Intel® 64 architecture, running Linux OS:

**so (dynamic):**
icpc -use-intel-optimized-headers -c source.cpp
icpc source.o -use-intel-optimized-headers
Merged (static):

icpc -use-intel-optimized-headers -c source.cpp
icpc source.o -use-intel-optimized-headers
Introduction

This section contains descriptions of Intel's C/C++ Language Extensions that assist users in parallel programming. The following C/C++ language extensions are included:

- **C++ lambda expressions or functions**: enable easy usage of Intel® Threading Building Blocks (TBB), making parallelism more available to developers.

Intel's C++ lambda extensions

**Introduction**


C++ Lambda expressions are primary expressions that define function objects. Such expressions can be used wherever a function object is expected; for example, as arguments to Standard Template Library (STL) algorithms.

This section contains the following topics:

- Details on Using C++ Lambda Expressions
- Understanding Lambda-Capture
- Lambda Function Object

**Details on Using Lambda Expressions in C++**

This topic explains in some detail about using Intel's implementation of C++ Lambda expressions. In order to use lambda expressions, you need to request c++0x with the command-line option `-std` or `/Qstd`:

On Windows® systems: `/Qstd=c++0x`

On Linux® systems: `-std=c++0x`
Introducing Lambda Expressions in Code

Lambda expressions are introduced in the code by \([\text{lambda-capture}_{opt}]\). If the expression does not capture any local variables or references with automatic storage duration, \(\text{lambda-capture}_{opt}\) can be empty, leaving \([]\) as the introducer.

For example, the lambda expression
\[
[](\text{int } x) \{ \text{return } x \% 3 == 0; \}
\]
is equivalent to the expression \(\text{unique}()\), where \(\text{unique}\) is a secret identifier generated and defined by the compiler as
\[
\text{class unique} \{ \\
\text{   public:} \\
\text{      bool operator() (int x) const \{ return x \% 3 == 0; \}} \\
\};
\]

Parameter List in Lambda Expressions

The \(\text{lambda-parameter-declaration}\) follows the \(\text{lambda-introducer}\). If the parameter list is empty, the parentheses can be omitted. For example, the following lambda expressions are equivalent.

\[
[]\{ \text{return } \text{rand();} \}
\]

\[
[]() \{ \text{return } \text{rand();} \}
\]

Body and Return Type of Lambda Expressions

The body of a lambda expression is a compound statement. A return type \(T\) for a lambda expression can be specified by writing \(\rightarrow T\) after the parameter list.

For example, the following lambda expressions are equivalent:

\[
[](\text{int } x) \{ \text{return } x \% 3 == 0; \}
\]

\[
[](\text{int } x) \rightarrow \text{bool} \{ \text{return } x \% 3 == 0; \}
\]

\[
[](\text{int } x) \rightarrow \text{bool} \{ \text{if} ( x \% 3 == 0 ) \text{ return true; else return false; } \}
\]

If the return type is not explicitly specified, the return type is \(\text{void}\) unless the body has the form \(\{ \text{return expression; }\). In such a case the return type is the type of the expression. Consider the following example:

\[
[](\text{int } x) \{ \text{if} (x \% 3 == 0) \text{ return true; else return false; } \}
\]
This expression is an error because you can't return a boolean value from a function returning void.

**Exception Specification**

A lambda expression may include an exception specification after the parameter list and before the return type specification. The parameter list is necessary if there is an exception specification.

The following lambda expressions specify that they do not throw exceptions:

```
[](int x) throw() -> bool { return x%3==0; }
[](() throw() { return rand(); })
```

**See Also**

- Intel's C++ lambda extensions
- Understanding [lambda-capture]
- Formal Grammar for Lambda Expressions in "Lambda Expressions and Closures: Wording for Monomorphic Lambdas (Revision 4)"

**Understanding Lambda-Capture**

A lambda expression can refer to identifiers declared outside the lambda expression. If the identifier is a local variable or a reference with automatic storage duration, it is an up-level reference and must be "captured" by the lambda expression. Such a lambda expression must be introduced by `[lambda-capture_opt]`, where `lambda-capture_opt` specifies whether identifiers are captured by reference or by copy. The table below summarizes forms of `lambda-capture_opt`.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[]</code></td>
<td>Nothing to capture: an up-level reference is an error</td>
</tr>
<tr>
<td><code>&amp;x, y, ...</code></td>
<td>Capture as specified: identifiers prefixed by <code>&amp;</code> are captured by reference; other identifiers are captured by copy. An up-level reference to any variable not explicitly listed is an error</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>Capture by reference: an up-level reference implicitly captures the variable by reference</td>
</tr>
<tr>
<td><code>=</code></td>
<td>Capture by copy: an up-level reference implicitly captures the variable by copy</td>
</tr>
<tr>
<td><code>&amp;, x, y, ...</code></td>
<td>Capture by reference with exceptions: listed variables are captured by value/copy (no listed variable may be prefixed by <code>&amp;</code>)</td>
</tr>
</tbody>
</table>
Capture by copy with exceptions: listed variables are captured by reference only (every listed variable must be prefixed by &)

No identifier may appear twice in the list. In the following code that sets area to the sum of the areas of four circles, the notation [&area,pi] specifies that area is captured by reference and pi by copy.

float radius[] = {2,3,5,7};
float area=0;
float pi  = 3.14f;
for_each(radius, radius+4, [&area,pi](float r) {return area+=pi*r*r;})

**Specifying Default Capture**

When a default capture is specified, the list must specify only the other kind of capture. In other words, if you specify that the default capture is by reference, then you must list (and only list) the variables that should be captured by copy. For example, if your intent is to capture x by reference and y by copy, and both x and y appear in the function body, the following code illustrates what is correct and incorrect code:

[&, &x, y]

//ERROR - default is capture-by-reference; list only capture-by-copy variable, y

[&, y]

//CORRECT - default is capture-by-reference; listed variable, y, should not have & prefix

[=, &x, y]

//ERROR - default is capture-by-copy; listed variable, x, must be prefixed with &
\[=, &x\]
//CORRECT - default is capture by copy; listed variable must be prefixed with &

\[&x\]
//ERROR - no default capture; you must list y separately to be captured by copy

\[y\]
//ERROR - again no default capture is specified, so you must list &x separately to be captured by reference

\[&x, y\]
//CORRECT - since no default is specified, every variable is listed with its capture mode.

**Default Binding Modes**

The following lambda expressions demonstrate default binding modes. All three expressions are semantically equivalent. Each captures \(x\) and \(y\) by reference, and captures \(a\) and \(b\) by copy.

\[&x, &y, a, b\](float r) \{x=a; y=b;\}

\[&, a, b\](float r) \{x=a; y=b;\}

\[=, &x, &y\](float r) \{x=a; y=b;\}

**Referring to this**

If a lambda expression occurs inside a member function, it can refer to this. Because this is not a variable, it cannot be captured by reference. Even when it is captured implicitly in a lambda expression introduced by \([&]\), it is captured by copy.

**Lambda Function Object**

The compiler creates an anonymous function object upon evaluating a lambda expression.
This function object, created by a lambda expression, may live longer than the block in which it is created. You must ensure that it does not use variables that were destroyed before the function object is used.

The following example shows how a function object created by a lambda expression can outlive the function that creates it.

```cpp
struct Base {
    virtual bool test(int x) = 0;
};
template<typename F>
struct Derived: Base {
    F f;
    bool test(int x) {return f(x);}
    Derived(F f_) : f(f_) {}
};
template<typename F>
Base* MakeDerived( F f ) {
    return new Derived<F>(f);
}
Base* Foo( int k ) {
    return MakeDerived( [k](int x) {return x%k==3;} );
}
bool Bar() {
    Base* b = Foo(3);
    return b->test(6);
}
```

In the above example, Bar invokes Foo, which copies the function object generated by a lambda expression into an instance of template class Derived. The lambda expression refers to a local variable k. Although the code destroys k before using the copied function object, the code is safe because k was captured by copy.
Index

/Api64 compiler option 212
/arch compiler option 213
/As compiler option 215
/bigobj compiler option 223
/c compiler option 225
/C compiler option 226
/D compiler option 232
/debug compiler option 237
/E compiler option 268
/EHa compiler option 270
/EHc compiler option 270
/EHs compiler option 270
/EP compiler option 271
/Fa compiler option 276
/FA compiler option 277
/fast compiler option 286
/FC compiler option 290
/F compiler option 295
/FD compiler option 294
/Fe compiler option 295
/FI compiler option 300
/fixed compiler option 306
/Fm compiler option 309
/fo compiler option 323
/fp compiler option 325, 334, 1399
   how to use 1399
/Fp compiler option 333
/Fr compiler option 351
/FR compiler option 352
/G2 compiler option 376
/G5 compiler option 378
/G6 compiler option 378
/G7 compiler option 378
/GA compiler option 380
/Gd compiler option 386
/Ge compiler option 388
/Gf compiler option 389
/GF compiler option 390
/Gh compiler option 391
/GH compiler option 392
/Gm compiler option 393
/Gr compiler option 395
/GR compiler option 396
/Gs compiler option 397
/GS compiler option 357, 358, 398
/GT compiler option 399
/GX compiler option 400
/Gy compiler option 402
/Gz compiler option 403
/GZ compiler option 404
/H compiler option 405
/help compiler option 406
/homeparams compiler option 409
/hotpatch compiler option 410
/I compiler option 411
/J compiler option 450
/LD compiler option 455
/link compiler option 456
/MD compiler option 472
/ML compiler option 480
/MP compiler option 485, 497
/MT compiler option 492
/noBool compiler option 498
/nologo compiler option 502
/O compiler option 507
/Od compiler option 514

1857
/Og compiler option 515
/Oi compiler option 289, 516
/Op compiler option 517
/Os compiler option 558
/Ot compiler option 559
/Ow compiler option 297, 516
/Ox compiler option 561
/Oy compiler option 324, 562
/P compiler option 565
/QA compiler option 204, 609
/QA- compiler option 205, 610
/Qalias-args compiler option 206, 283, 611
/Qalias-const compiler option 207, 613
/Qansi-alias compiler option 211, 614
/Qauto-ilp32 compiler option 216, 615
/Qax compiler option 217, 616, 1110
/Qc99 compiler option 227, 620
/Qchkstk compiler option 621
/Qcomplex-limited-range compiler option 229, 622
/Qcontext-limit compiler option 907
/Qcov-dir compiler option 623
/Qcov-file compiler option 624
/Qcov-gen compiler option 625
/QCxx-features compiler option 626
/QD compiler option 233, 648
/Qdiag disable compiler option 240, 246, 627, 633
/Qdiag dump compiler option 245, 632
/Qdiag-enable
   sc compiler option 240, 246, 627, 633
   sc-include compiler option 251, 638
   sc-parallel compiler option 252, 640
   sv-include compiler option 251, 638
/Qdiag-enable compiler option 240, 246, 627, 633
/Qdiag-error compiler option 240, 246, 627, 633
/Qdiag-error-limit compiler option 254, 642
/Qdiag-file-append compiler option 257, 644
/Qdiag-file compiler option 255, 643
/Qdiag-id-numbers compiler option 258, 646
/Qdiag-once compiler option 260, 647
/Qdiag-remark compiler option 240, 246, 627, 633
/Qdiag-warning compiler option 240, 246, 627, 633
/Qdm compiler option 261, 649
/Qdn compiler option 262, 650
/Qeffc++ compiler option 651, 910
/Qfast-transcendentals compiler option 288, 653
/Qfma compiler option 310, 654
/Qfnalign compiler option 280, 655
/Qfnsplit compiler option 321, 656
Index

/QMT compiler option 491, 719
/Qmultibyte-chars compiler option 496, 720
/Qnbs-init compiler option 499, 721
/Qnpic compiler option 722
/Qopenmp compiler option 519, 722
/Qopenmp-lib compiler option 520, 724, 1125
/Qopenmp-link compiler option 522, 726
/Qopenmp-profile compiler option 524, 727
/Qopenmp-report compiler option 525, 728
/Qopenmp-stubs compiler option 526, 730
/Qopenmp-task compiler option 527, 731
/Qopenmp-analysis compiler option 532, 735
/Qoption compiler option 759
/Qopenmp-tables compiler option 533, 736
/Qopt-loadpair compiler option 535, 737
/Qopt-mem-bandwidth compiler option 538, 738
/Qopt-multi-version-aggressive compiler option 541, 741
/Qopt-prefetch compiler option 542, 742
/Qopt-prefetch-initial-values compiler option 543, 744
/Qopt-prefetch-issue-excl-hint compiler option 545, 745
/Qopt-prefetch-next-iteration compiler option 546, 746
/Qopt-ra-region-strategy compiler option 547, 748, 1371
/example 1371
/Qopt-report compiler option 549, 749, 1057
/Qopt-report-file compiler option 550, 751
/Qopt-report-help compiler option 551, 752
/Qopt-report-phase compiler option 552, 753
/Qopt-report-routine compiler option 554, 757
/Qpar-adjust-stack compiler option 760
/Qpar-parallel compiler option 566, 761
/Qpar-threads compiler option 567, 772
/Qpar-report compiler option 568, 763
/Qpar-runtime-control compiler option 570, 766
/Qpar-schedule compiler option 572, 767
/Qpar-threshold compiler option 575, 770
/Qpc compiler option 578, 773
/Qpc compiler option 774
/Qpreci compiler option 588, 777
/Qpreci-sqrt compiler option 486, 775
/Qpreci-sqrt compiler option 589, 778
/Qprof-data-order compiler option 591, 779
/Qprof-dir compiler option 592, 780
/Qprof-file compiler option 593, 781
/Qprof-func-order compiler option 596, 783
/Qprof-gen
/srcpos compiler option 1306, 1308, 1329
/Qprof-gen:srcpos compiler option
code coverage tool 1308
test prioritization tool 1329
/Qprof-gen compiler option 598, 785, 1306
/Qprof-hotness-threshold compiler option 600, 786
/Qprof-srcc-dir compiler option 601, 788
/Qprof-srcc-root compiler option 603, 789
/Qprof-srcc-root-cwd compiler option 605, 791
/Qprof-use compiler option 606, 793, 1308, 1338
code coverage tool 1308
profmerge utility 1338
/Qrcd compiler option 795, 839
/Qrct compiler option 796, 840
/Qrestrict compiler option 797, 842
/Qsafesep compiler option 798
/Qsave-temps compiler option 799, 846
/Qscalar-rep compiler option 801, 848
/Qserialize-volatile compiler option 490, 802
/Qsafalgin compiler option 803
/Qsox compiler option 807, 853
/Qstd compiler option 804, 859
/Qtbb compiler option 808, 863
/Qtcheck compiler option 809, 866
/Qcollect compiler option 810, 867
/Qcollect-filter compiler option 812, 868
/Qcollect-depth compiler option 360, 814
/Qprofile compiler option 872
/Qtrapuv compiler option 362, 815
/Qunroll-aggressive compiler option 816, 879
/Qunroll compiler option 817, 878
/Quse-asm compiler option 818, 880
/Quse-intel-optimized-headers compiler option 819, 881
/Quse-msasm-symbols compiler option 820
/Qvc compiler option 822
/QV compiler option 821, 884
/Qvec compiler option 823, 887
/Qvec-guard-write compiler option 825, 888
/Qvec-report compiler option 826, 889
/Qvec-threshold compiler option 827, 891
/Qwd compiler option 829, 907
/Qwe compiler option 830, 909
/Qwo compiler option 831, 924
/Qwq compiler option 832, 925
/Qwr compiler option 833, 930
/Qww compiler option 834, 940
/Qx compiler option 835, 942, 1106
/RTC compiler option 843
/S compiler option 845
/showIncludes compiler option 852
/Tc compiler option 864
/TC compiler option 865
/Tp compiler option 870
/TP compiler option 451, 871
/traceback compiler option 873
/u (W*) compiler option 876
/U compiler option 877
/V compiler option 878
/vd compiler option 886
/vmb compiler option 893
/vmg compiler option 894
/vmm compiler option 895
/vms compiler option 896
/vmv compiler option 897
/Wall compiler option 903
/Wcheck compiler option 905
/w compiler option 898
/W compiler option 899, 900
/Werror-all compiler option 913
/WL compiler option 919
/Wp64 compiler option 926
/Wport compiler option 928
/WX compiler option 912, 941
/X compiler option 949
/Yc compiler option 581, 952
/Y- compiler option 951
/Yd compiler option 954
/Yu compiler option 955
/YX compiler option 956
/Ze compiler option 960
/Zc compiler option 960
/Zd compiler option 961
/Ze compiler option 961
/Zg compiler option 962
/Zr compiler option 963
/ZI compiler option 374, 957, 963
/ZI compiler option 964
/ZI compiler option 965
/Zp compiler option 966
/Zs compiler option 967
/Zx compiler option 968
.dpi file 1308, 1329, 1338
.dyn file 1308, 1329, 1338
.dyn files 1308, 1329, 1338
.spi file 1308, 1329
.A compiler option 204, 609
-A compiler option 205, 610
-alias-args compiler option 206, 283, 611
-alias-const compiler option 207, 613
-align compiler option 209
-ansi-alias compiler option 211, 614
-ansi compiler option 210
-ax compiler option 217, 616, 1110
-B compiler option 220
-Bdynamic compiler option 222
-Bstatic compiler option 224
-c99 compiler option 227, 620
-c compiler option 113, 225
-C compiler option 226
-check-uninit compiler option 228
-complex-limited-range compiler option 229, 622
-xxlib compiler option 230
-D compiler option 232
-dD compiler option 233, 648
-debug compiler option 108, 234
-diag compiler option 240, 246, 627, 633
-diag-disable compiler option 240, 246, 627, 633
-diag-show compiler option 240, 246, 627, 633
-diag-enable compiler option 240, 246, 627, 633
-diag-enable port-win compiler option 147
-diag-enable sc compiler option 240, 246, 627, 633
-diag-enable sc-include compiler option 251, 638
-diag-enable sc-parallel compiler option 252, 640
-diag-show compiler option 251, 638
-diag-error compiler option 240, 246, 627, 633
-diag-error-limit compiler option 254, 642
-diag-error-limit compiled object option 257, 644
-diag-file-append compiler option 255, 643
-diag-include compiler option 258, 646
-diag-include port-win compiler option 260, 647
-diag-remark compiler option 240, 246, 627, 633
-diag-warning compiler option 240, 246, 627, 633
-dM compiler option 261, 649
-dN compiler option 262, 650
-dryrun compiler option 263
-dumpmachine compiler option 264
-dumpversion compiler option 265
-dynamiclib compiler option 113, 115, 267
-dynamic-linker compiler option 266
-early-template-check compiler option 269
-E compiler option 268
<table>
<thead>
<tr>
<th>Compiler Option</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>-EP</td>
<td>271</td>
</tr>
<tr>
<td>-export</td>
<td>272</td>
</tr>
<tr>
<td>-export-dir</td>
<td>273</td>
</tr>
<tr>
<td>-fabi-version</td>
<td>278</td>
</tr>
<tr>
<td>-falias</td>
<td>279</td>
</tr>
<tr>
<td>-falign-functions</td>
<td>280, 655</td>
</tr>
<tr>
<td>-falign-stack</td>
<td>281</td>
</tr>
<tr>
<td>-fargument-alias</td>
<td>206, 283, 611</td>
</tr>
<tr>
<td>-fargument-noalias-global</td>
<td>284</td>
</tr>
<tr>
<td>-fasm-blocks</td>
<td>285</td>
</tr>
<tr>
<td>-fast</td>
<td>286</td>
</tr>
<tr>
<td>-fast-transcendentals</td>
<td>288, 653</td>
</tr>
<tr>
<td>-fbuiltin</td>
<td>289, 516</td>
</tr>
<tr>
<td>-fcode-asm</td>
<td>291</td>
</tr>
<tr>
<td>-fcommon</td>
<td>292</td>
</tr>
<tr>
<td>-fdata-sections</td>
<td>299</td>
</tr>
<tr>
<td>-fexceptions</td>
<td>296</td>
</tr>
<tr>
<td>-ffnalias</td>
<td>297</td>
</tr>
<tr>
<td>-ffreestanding</td>
<td>298, 663</td>
</tr>
<tr>
<td>-ffunction-sections</td>
<td>299</td>
</tr>
<tr>
<td>-finline</td>
<td>301</td>
</tr>
<tr>
<td>-finline-functions</td>
<td>302</td>
</tr>
<tr>
<td>-finline-limit</td>
<td>303</td>
</tr>
<tr>
<td>-finstrument-functions</td>
<td>304, 687</td>
</tr>
<tr>
<td>-fjump-tables</td>
<td>307</td>
</tr>
<tr>
<td>-fkeep-static-consits</td>
<td>308, 704</td>
</tr>
<tr>
<td>-fma</td>
<td>310</td>
</tr>
<tr>
<td>-fmath-errno</td>
<td>312</td>
</tr>
<tr>
<td>-fminshered</td>
<td>313</td>
</tr>
<tr>
<td>-fmudflap</td>
<td>314</td>
</tr>
<tr>
<td>-fno-gnu-keywords</td>
<td>315</td>
</tr>
<tr>
<td>-fno-implicit-inline-templates</td>
<td>316</td>
</tr>
<tr>
<td>-fno-implicit-templates</td>
<td>317</td>
</tr>
<tr>
<td>-fnon-call-exceptions</td>
<td>319</td>
</tr>
<tr>
<td>-fnon-value-assign</td>
<td>320</td>
</tr>
<tr>
<td>-fno-operator-names</td>
<td>318</td>
</tr>
<tr>
<td>-fno-rtti</td>
<td>318</td>
</tr>
<tr>
<td>-fnssplit</td>
<td>321, 656</td>
</tr>
<tr>
<td>-fp</td>
<td>346</td>
</tr>
<tr>
<td>-fp-branch-compare</td>
<td>347</td>
</tr>
<tr>
<td>-fp-cpu</td>
<td>348</td>
</tr>
<tr>
<td>-fpic</td>
<td>113, 115, 348</td>
</tr>
<tr>
<td>-fpie</td>
<td>350</td>
</tr>
<tr>
<td>-fp-model</td>
<td>325, 334, 1399</td>
</tr>
<tr>
<td>-fp-port</td>
<td>341, 658</td>
</tr>
<tr>
<td>-fp-relaxed</td>
<td>342, 659</td>
</tr>
<tr>
<td>-fp-speculation</td>
<td>343, 660</td>
</tr>
<tr>
<td>-fp-stack-check</td>
<td>345, 662</td>
</tr>
<tr>
<td>-fr32</td>
<td>353</td>
</tr>
<tr>
<td>-freg-struct-return</td>
<td>354</td>
</tr>
<tr>
<td>-fshort-enums</td>
<td>355</td>
</tr>
<tr>
<td>-fsource-asm</td>
<td>356</td>
</tr>
<tr>
<td>-fstack-protector</td>
<td>357, 358, 398</td>
</tr>
<tr>
<td>-fstack-security-check</td>
<td>357, 358, 398</td>
</tr>
<tr>
<td>-fsyntax-only</td>
<td>359</td>
</tr>
<tr>
<td>-ftemplate-depth</td>
<td>360, 814</td>
</tr>
<tr>
<td>-ftls-model</td>
<td>361</td>
</tr>
<tr>
<td>-ftrapuv</td>
<td>362, 815</td>
</tr>
<tr>
<td>-fz compiler</td>
<td>363, 664, 1403</td>
</tr>
<tr>
<td>-func-groups</td>
<td>364</td>
</tr>
<tr>
<td>-funroll-all-loops</td>
<td>365</td>
</tr>
<tr>
<td>-funsigned-bitfields</td>
<td>367</td>
</tr>
<tr>
<td>-funsigned-char</td>
<td>368</td>
</tr>
<tr>
<td>-fverbose-asm</td>
<td>369</td>
</tr>
<tr>
<td>-fvisibility</td>
<td>370</td>
</tr>
<tr>
<td>-fvisibility-inlines-hidden</td>
<td>373</td>
</tr>
<tr>
<td>-g0</td>
<td>375</td>
</tr>
<tr>
<td>-gcc</td>
<td>381, 382</td>
</tr>
<tr>
<td>-gcc-name</td>
<td>383</td>
</tr>
<tr>
<td>-gcc-sys</td>
<td>381, 382</td>
</tr>
<tr>
<td>-gcc-version</td>
<td>385</td>
</tr>
<tr>
<td>-g</td>
<td>374, 957, 963</td>
</tr>
<tr>
<td>-gdwarf-2</td>
<td>387</td>
</tr>
<tr>
<td>-global-hoist</td>
<td>394, 666</td>
</tr>
<tr>
<td>-gxx-name</td>
<td>401</td>
</tr>
<tr>
<td>-H</td>
<td>404, 667</td>
</tr>
<tr>
<td>-help</td>
<td>406</td>
</tr>
<tr>
<td>-help-pragma</td>
<td>408, 668</td>
</tr>
<tr>
<td>-icc</td>
<td>412</td>
</tr>
<tr>
<td>-I</td>
<td>411</td>
</tr>
<tr>
<td>-idirafter</td>
<td>413</td>
</tr>
<tr>
<td>-imacros</td>
<td>414</td>
</tr>
<tr>
<td>-inline-calloc</td>
<td>415, 670</td>
</tr>
<tr>
<td>-inline-debug-info</td>
<td>416, 671</td>
</tr>
<tr>
<td>-inline-factor</td>
<td>417, 673</td>
</tr>
<tr>
<td>-inline-forceinline</td>
<td>419, 675</td>
</tr>
<tr>
<td>-inline-level</td>
<td>420, 512</td>
</tr>
<tr>
<td>-inline-max-per-compile</td>
<td>422, 676</td>
</tr>
<tr>
<td>-inline-max-per-routine</td>
<td>423, 678</td>
</tr>
<tr>
<td>-inline-max-size</td>
<td>425, 679</td>
</tr>
<tr>
<td>-inline-max-total-size</td>
<td>427, 681</td>
</tr>
<tr>
<td>-inline-min-size</td>
<td>428, 683</td>
</tr>
<tr>
<td>-ip</td>
<td>430, 689</td>
</tr>
<tr>
<td>-IPF-ftacc</td>
<td>435, 694</td>
</tr>
<tr>
<td>-IPF-ft-eval-method0</td>
<td>433, 692</td>
</tr>
</tbody>
</table>
-IPF-fma compiler option 310, 654
-IPF-fp-relaxed compiler option 342, 659
-ip-no-inlining compiler option 431, 690
-ipo compiler option 43, 697
-ipo compiler option 436, 695, 1275
-ipo-jobs compiler option 439, 698
-ipo-S compiler option 440, 699
-ipo-separate compiler option 441, 700
-ippc compiler option 442, 701
-iprefix compiler option 444
-iplquote compiler option 445
-iprefix compiler option 446
-ivdep-parallel compiler option 447, 703
-ivarshpxbefore compiler option 449
-ivarshpxcompiler option 448
-Kc++ compiler option 451, 871
-kernel compiler option 452
-i compiler option 453
-L compiler option 454
-m32 compiler option 460, 461
-m64 compiler option 460, 461
-malign-double compiler option 462
-malign-mac68k compiler option 463
-malign-natural compiler option 464
-malign-power compiler option 465
-map-opts compiler option 466, 709
-march compiler option 467
-m compiler option 457
-M compiler option 459, 708
-mcpu compiler option 493
-MD compiler option 471, 710
-mdynamic-no-pic compiler option 473
-MF compiler option 474, 711
-mfixed-range compiler option 475
-MG compiler option 476, 712
-msoe-fp compiler option 483
-minstruction compiler option 477, 685
-mlk compiler option 478, 713
-MM compiler option 481, 715
-MMD compiler option 482, 716
-mp1 compiler option 486, 775
-mp compiler option 483
-MP compiler option 484
-MQ compiler option 487
-mregparm compiler option 488
-mrelax compiler option 489
-mserialize-volatile compiler option 490, 802
-MT compiler option 491, 719
-mtune compiler option 493
-multibyte-chars compiler option 496, 720
-multiple-processes compiler option 485, 497
-nobss-init compiler option 499, 721
-nodefaultlibs compiler option 500
-nolib-inline compiler option 501
-nostartfiles compiler option 503
-nostdinc++ compiler option 504
-nostdlib compiler option 505
-Ob compiler option 420, 512
-o compiler option 506
-O compiler option 507
-openmp compiler option 519, 722
-openmp-lib compiler option 520, 724, 1125
-openmp-link compiler option 522, 726
-openmp-profile compiler option (Linux* only) 524, 727
-openmp-report compiler option 525, 728
-openmp-stubs compiler option 526, 730
-openmp-task compiler option 527, 731
-openmp-threadprivate compiler option 529, 732
-opt-block-factor compiler option 530, 734
-opt-calloc compiler option (Linux only) 531
-opt-class-analysis compiler option 532, 735
-opt-jump-tables compiler option 533, 736
-opt-loadpair compiler option 535, 737
-opt-malloc-options compiler option 536
-opt-mem-bandwidth compiler option 538, 738
-opt-mod-versioning compiler option 539, 740
-opt-multi-version-aggressive compiler option 541, 741
-opt-prefetch compiler option 542, 742
-opt-prefetch-initial-values compiler option 543, 744
-opt-prefetch-issue-excl-hint compiler option 545, 745
-opt-prefetch-next-iteration compiler option 546, 746
-opt-ra-region-strategy compiler option 547, 748
-opt-report compiler option 549, 749, 1057
-opt-report-file compiler option 550, 751
-opt-report-help compiler option 551, 752
-opt-report-phase compiler option 552, 753
-opt-report-routine compiler option 554, 755
-opt-report-file compiler option 555, 756
-opt-report-file compiler option 557, 757
-Os compiler option 558
-par-affinity compiler option (Linux* only) 566, 761
-par-parallel compiler option 577, 772
-par-num-threads compiler option 568, 763
-par-report compiler option 569, 764
-par-runtime-control compiler option 570, 766
-par-schedule compiler option 572, 767
-par-threshold compiler option 575, 770
-pc compiler option 578, 773
-pch compiler option 91, 584
-pch-create compiler option 91, 581, 952
-pch-dir compiler option 91, 583
-pch-use compiler option 91, 584
-p compiler option 564
-P compiler option 565
-pg compiler option 564
-pie compiler option 586
-pragmas compiler option 587
-prec-div compiler option 588, 777
-prec-sqrt compiler option 589, 778
-print-multi-lib compiler option 590
-prof-data-order compiler options 591, 779
-prof-dir compiler option 592, 780
-prof-file compiler option 593, 781
-prof-func-groups compiler option 595
-prof-func-order compiler options 596, 783
-prof-gen
srpos compiler option 1306, 1308, 1329
-prof-gen:srcpos compiler option
code coverage tool 1308
-test priorization tool 1329
-related options 1296
-prof-hotness-threshold compiler option 600, 786
-prof-src-dir compiler option 601, 788
-prof-src-root compiler option 603, 789
-prof-src-root-cwd compiler option 605, 791
-prof-use compiler option 606, 793, 1296, 1308, 1338
code coverage tool 1308
profmerge utility 1338
-related options 1296
-pthread compiler option 608
-Qinstall compiler option 684
-Qlocation compiler option 705
-Qoption compiler option 759
-qp compiler option 564
-rcd compiler option 795, 839
-rcx compiler option 796, 840
-reserve-kernel-regs compiler option 841
-restrict compiler option 797, 842
-save-temps compiler option 799, 846
-scalar-rep compiler option 801, 848
-S compiler option 845
-shared compiler option 113, 115, 849
-shared-intel compiler option 115, 850
-shared-libgcc compiler option 851
-sox compiler option 807, 853
-static compiler option 854
-static-intel compiler option 857
-staticlib compiler option 855
-static-libgcc compiler option 858
-standard compiler option 804, 859
-strict-ansi compiler option 861
-tbb compiler option 808, 863
-tcheck compiler option (Linux* only) 809, 866
-tcollect compiler option 810, 867
-tcollect-filter compiler option 812, 868
-T compiler option 862
-tprofile compiler option (Linux* only) 872
-traceback compiler option 873
-u (L*) compiler option 875
-U compiler option 877
-unroll-aggressive compiler option 816, 879
-unroll compiler option 817, 878
-use-asm compiler option 818, 880
-use-intel-optimized-headers compiler option 819, 881
-use-msasm compiler option 882
-V (L*) compiler option 821, 884
-v compiler option 883
-vec compiler option 823, 887
-vec-guard-write compiler option 825, 888
-vec-report compiler option 826, 889
-vec-threshold compiler option 827, 891
--version compiler option 892
-Wabi compiler option 902
-Wa compiler option 901
-Wall compiler option 903
-Wbriefer compiler option 904
-Wcheck compiler option 905
-Wcomment compiler option 906
-w compiler option 898, 899, 900
-Wcontext-limit compiler option 907
-wd compiler option 829, 907
-Wdeprecate compiler option 908
-we compiler option 830, 909
-Weffc++ compiler option 910
-Wextra-tokens compiler option 911
-Wformat compiler option 912, 941
-Wformat-secure compiler option 913
-Wformat-security compiler option 914
-Wformat-security compiler option 915
-Wformat-security compiler option 916
-Wincludemodule compiler option 917
-WL compiler option 918
-Wmain compiler option 920

1863
-Wmissing-declarations compiler option 921
- Wmissing-prototypes compiler option 922
- wn compiler option 831, 924
- Wnon-virtual-dtor compiler option 923
- wo compiler option 832, 925
- Wp64 compiler option 926
- Wp compiler option 925
- Wpointer-arithmetic compiler option 927
- Wpragma-once compiler option 929
- wr compiler option 833, 930
- Wreorder compiler option 931
- Wreturn-type compiler option 932
- Wshadow compiler option 933
- Wstrict-prototypes compiler option 934
- Wtrigraphs compiler option 935
- Wunknown-prototypes compiler option 937
- Wuninitialized compiler option 938
- Wunneccessary-variable compiler option 939
- wv compiler option 834, 940
- Wwrite-strings compiler option 941
- x (type) compiler option 947
- x compiler option 835, 942, 1106
- X compiler option 949
- Xlinker compiler option 950
- Zp compiler option 966

A

Advanced Encryption Standard (AES) Intrinsics 1671
advanced PGO options 1306
AES intrinsics
  __mm_aesdec_si128 1672
  __mm_aesdeclast_si128 1672
  __mm_aesenc_si128 1672
  __mm_aesenclast_si128 1672
  __mm_aesdec_si128 1672
  __mm_aesenclast_si128 1672
  __mm_aeskeygenassist_si128 1672
aliasing
  option specifying assumption in functions 297
  option specifying assumption in programs 279
aligning data 1379
alignment 1379
ALLOCATABLE (continued)
  visual presentation 1308
allocating registers 1371
alternate tools and locations 90
analyzing applications 1041, 1044, 1046
  Intel(R) Debugger 1044
  Intel(R) Threading Tools 1044
ANSI/ISO standard 133
application characteristics 1046
application performance 1046
applications
  option specifying code optimization for 507
application tests 1329
architectures
  coding guidelines for 1407
  option generating instructions for 213
array 1407
array operation 1407
ar tool 113
assembler
  option passing options to 901
  option producing objects through 818, 880
assembly files
  naming 90
automatic optimizations 1101, 1102
auto-parallelization 1036, 1086, 1237, 1243, 1244
diagnostic 1086
  enabling 1243
  environment variables 1243
guidelines 1244
overview 1237
programming with 1244
threshold 1086
auto-parallelized loops 1086
auto-parallelizer 569, 575, 577, 764, 770, 772, 1036, 1086, 1237
tools 1036, 1086
  enabling 1036
  option controlling level of diagnostics for 569, 764
  option enabling generation of multithreaded code 577, 772
  option setting threshold for loops 575, 770
autovectorization 1409
auto-vectorization 1036
autovectorization of innermost loops 1409
autovectorizer 1110, 1251
  optimization for systems based on IA-32
architecture 1110
avoid
inefficient data types 1409
mixed arithmetic expressions 1409

C

capturing IPO output 1275
Carry-less intrinsic
\_mm\_clmulepi64\_si128 1672
Carry-less Multiplication Instruction (PCLMULQDQ)
Intrinsic 1671
changing number of threads
summary table of 1177
checking
floating-point stacks 1412
stacks 1412
Checking the Floating-point Stack State 1412
Class Libraries
programming example 1844
Class Libraries (continued)
integer vector classes (continued)
debug operations
element access operator 1796
element assignment operators 1796
functions for SSE 1805
ivec classes 1775
logical operators 1782
multiplication operators 1787
pack operators 1804
rules for operators 1778
shift operators 1789
unpack operators 1799
overview 1767
Quick reference 1834
code coverage tool 1308
color scheme 1308
dynamic counters in 1308
exporting data 1308
syntax of 1308
codecov tool
option producing an instrumented file for 625
option specifying a directory for profiling output for 623
option specifying a file name for summary files for 624
code layout 1281
compilation units 313, 1290
option to prevent linking as shareable object 313
compiler
overview 45, 48
compiler information
saving in your executable 59
compiler installation
option specifying root directory for 684
compiler operation
compilation phases 54
default behavior 54
input files 88
invoking from the command line 85
output files 54, 89
compiler optimization 1102
compiler optimizations 1101
compiler option mapping tool 94
compiler options
command-line syntax 55
cross reference 969
deprecated and removed 192
for interoperability 125
compiler options (continued)
for optimization 144
for portability 1016
for templates 136
for visibility 118, 119
general rules for 201
linker-related 87
new 177
option categories 55
overview of descriptions of 201
preprocessor options 97, 98, 99
quick reference guide 969
unsupported Microsoft Visual Studio 1028
using 55
compiler reports 1055, 1057, 1060, 1069, 1088, 1093, 1285
High-Level Optimization (HLO) 1069
Interprocedural Optimizations (IPO) 1060
report generation 1057
requesting with xi* tools 1285
software pipelining 1088
vectorization 1093
compiler reports quick reference 1055
compilers
using multiple versions 62
compiling
gcc* code with Intel(R) C++ 144
compiling large programs 1279
compiling with IPO 1275
complex operations
option enabling algebraic expansion of 229, 622
conditional check
option performing in a vectorized loop 825, 888
conditional parallel region execution
auto-parallelizer diagnostics 1036, 1086
inline expansion 1288
configuration files 104
conventions
in the documentation 45
correct usage of countable loop 1258
COS
correct usage of 1258
counters for dynamic profile 1355
CPU
option generating code for specified 467
option performing optimizations for specified 493
CPU dispatch 1110, 1113
automatic 1110
CPU time 1286, 1329, 1380
DPI lists 1329
for inline function expansion 1286
create libraries using IPO 1283
D
data alignment 1379
dataflow analysis 1036, 1237
data format
dependence 1086
partitioning 1244
prefetching 1357
sharing 1036
type 1036, 1251
data ordering optimization 1342
data prefetches 1367
data types
efficiency 1412
DAZ flag 1403
debugging 107, 108, 234
compiler options for 107, 108
option affecting information generated 234
option specifying settings to enhance 234
symbolic 107
denormal exceptions 1408
denormalized numbers (IEEE*) 1415
NaN values 1415
deprecated compiler options 192
determining parallelization 1036
diagnostic messages
option affecting which are issued 240, 246, 627, 633
option controlling auto-parallelizer 240, 246, 627, 633
option controlling display of 240, 246, 627, 633
option controlling OpenMP 240, 246, 627, 633
option controlling source control 240, 246, 627, 633
option controlling vectorizer 240, 246, 627, 633
option enabling or disabling 240, 246, 627, 633
option enabling parallel lint 252, 640
option issuing only once 250, 647
option processing include files and source files for 251, 638
diagnostic reports 1086
Index

diagnostics 149, 152, 159, 161, 162, 163, 164, 1036, 1086, 1253
  auto-parallelizer 1036, 1086
difference operators 1164
differential coverage 1308
directory
  option adding to start of include path 446
  option specifying for executables 220
  option specifying for includes and libraries 220
  disabling
    function splitting 1296
    inlining 1288
  disabling optimization 1118
dllimport functions
  option controlling inlining of 672
DO constructs 1180, 1258
  numbers 1180
Documentation
  conventions for 45
driver tool commands
  option specifying to show and execute 883
  option specifying to show but not execute 263
dual-core 1036
dual core thread affinity 1200
dumping profile information 1352, 1353
dynamic information 1180, 1295, 1306, 1351, 1352, 1355
  dumping profile information 1352
  files 1306
  resetting profile counters 1355
  threads 1180
dynamic-information files 1296
dynamic libraries
  option invoking tool to generate 267
dynamic linker
  option specifying an alternate 266
dynamic-linking of libraries
  option enabling 222
dynamic-link libraries (DLLs)
  option searching for unresolved references in 472
dynamic shared object
  option producing a 849
dyn files 1296, 1306, 1351, 1352, 1355
E
  ebp register
    option determining use in optimizations 324, 562
  Eclipse* integration
    adding a source file 65
    building a project 66
    creating a new project 63
    exporting makefiles 78
    makefiles 77
    overview 61
    running a project 66
    setting properties 68
    starting 62
  Eclipse* projects 62
  efficiency 1409
  efficient
    auto-parallelizer 1036
    inlining 1288
    parallelizer 1036
    PGO options 1296
  efficient data types 1412
  EMMS Instruction
    about 1486
    using 1487
  endian data
    and OpenMP* extension routines 1185
    auto-parallelization 1243
    dumping profile information 1352
    for auto-parallelization 1243
    for profile-guided optimization 1351
    loop constructs 1258
    OpenMP* 1168
    parallel program development 1036
    PROF_DIR 1351
    PROF_DUMP_INTERVAL 1353
    routines overriding 1180
    using OpenMP* 1164
    using profile-guided optimization 1306
  enhancing optimization 1046
  enhancing performance 1046
  environment variables
    C_INCLUDE_PATH 101
    CPATH 101
    CPLPLUS_INCLUDE_PATH 101
    for Linux* 101
    IA32ROOT 101
    IA64ROOT 101
    ICCCFG 101
    ICPCCFG 101
    LD_LIBRARY_PATH 112
    LIBRARY_PATH 101
    modifying 101
environment variables (continued)
  PATH 101
  PROF_DIR 101
  PROF_DPI 101
  SUNPRO_DEPENDENCIES 101
  TEMP 101
  TMP 101
  TMPDIR 101
error parser 67
exception handling
  option generating table of 296
exceptions
  option allowing trapping instructions to throw 319
exclude code 1308
  code coverage tool 1308
exclusive hint
  option causing prefetches for stores with 545, 745
executable files 89
execution environment routines 1180
execution flow 1244
execution mode 1185
explicit-shape arrays
  .dpi 1296, 1308, 1329, 1338
  .dyn 1296, 1306, 1308, 1329, 1338, 1351, 1352, 1355
  .spi 1308, 1329
  OpenMP* header 1180
  pgopti.dpi 1308
  pgopti.spi 1308
  source 1306
exported templates 133
export keyword 133
expressions
  option evaluating floating-point 433, 692

F

floating-point accuracy
  option disabling optimizations affecting 435, 694
floating-point array operation 1407
floating-point calculations
  option controlling semantics of 325, 334
floating-point exceptions
  denormal exceptions 1408
floating-point numbers
  formats for 1415
  overview 1397
  overview of 1415
floating-point numbers (continued)
  special values 1415
floating-point operations
  option controlling semantics of 325, 334
  option enabling combining of 310, 654
  option rounding results of 341, 658
floating-point precision
  option controlling for significand 578, 773
  option improving for divides 588, 777
  option improving for square root 589, 778
  option improving general 486, 775
floating-point stack 345, 662, 1415
  option checking 345, 662
float-to-integer conversion
  option enabling fast 795, 839
flow dependency in loops 1361
format function security problems
  option issuing warning for 916
FPU rounding control
  option setting to truncate 796, 840
FTZ flag 1403
function entry and exit points
  option determining instrumentation of 304, 687
function expansion 1290
function grouping
  option enabling or disabling 595
function grouping optimization 1342
function ordering optimization 1342
function order list 1296, 1349
  enabling or disabling 1296
function order lists 1342
function preemption 1286
function profiling
  option compiling and linking for 564
functions
  option aligning on byte boundary 280, 655
function splitting
  option enabling or disabling 321, 656

G

g++* language extensions 121
gcc*
  porting from 147
gcc* built-in functions 129
gcc* compatibility 121
gcc* considerations 144
gcc* interoperability 125
gcc* language extensions 121
GCC* project
  updating to use the Intel(R) C++ compiler 73
gcc C++ run-time libraries
  include file path 413
  option adding a directory to second 413
  option removing standard directories from 949
  option specifying to link to 230
general compiler directives 1057, 1110, 1244, 1251,
  1253, 1286, 1295, 1296, 1350, 1367
  affecting data prefetches 1367
  for auto-parallelization 1244
  for inlining functions 1286
  for profile-guided optimization 1295
  for vectorization 1251, 1253
  instrumented code 1296
  processor-specific code 1110
  profile-optimized executable 1296
  profiling information 1350
  reports 1057
global symbols 116
GNU C++ compatibility 121

I
IA-32 architecture based applications
HLO 1357
  methods of parallelization 1036
  options 1106, 1110
  targeting 1106, 1110
IA-64 architecture based applications
auto-vectorization in 1036
HLO 1357
  methods of parallelization 1036
  options 1117
  report generation 1057
  targeting 1117
IA-64 intrinsics
  conversion intrinsics 1449
  Dual-Core Itanium® processor 9000 sequence 1464
  InterlockedCompare64Exchange128 1471
  InterlockedCompare64Exchange128_acq 1471
  InterlockedCompare64Exchange128_rel 1471
  load128 1471
  load128_acq 1471
  load128_rel 1471
  load and store intrinsics 1444
  lock and atomic intrinsics 1444
  multimedia addition intrinsics 1450
  MS compatible intrinsics 1471
  native intrinsics 1437
  OS-related addition intrinsics 1440
  register names 1460
  store128 1471
  store128_rel 1471
iccvars.sh environment script 85
IEEE*
  floating-point values 1415
ILO 1057
  include files 105
  inlined code
    option producing source position information for 416, 671

hot patching
  option preparing a routine for 410
hotspots 1041
Hyper-Threading Technology
  parallel loops 1245
  thread pools 1245

IEEE* floating-point values 1415
ILO 1057
  include files 105
inline function expansion
  option specifying level of 420, 512
inlining 303, 419, 422, 423, 425, 427, 428, 431, 432, 675, 676, 678, 679, 681, 683, 690, 691, 1286, 1288, 1290, 1295
  compiler directed 1288
developer directed 1290
  option disabling full and partial 431, 690
  option disabling partial 432, 691
  option forcing 419, 675
  option specifying lower limit for large routines 425, 679
  option specifying maximum size of function for 303
  option specifying maximum times for a routine 423, 678
  option specifying maximum times for compilation unit 422, 676
  option specifying total size routine can grow 427, 681
  option specifying upper limit for small routine 428, 683
  preemption 1286
inlining options
  option specifying percentage multiplier for 417, 673
input files 88
instruction-level parallelism 1036
instrumentation 812, 868, 1295, 1296, 1306, 1352
  compilation 1306
  execution 1306
  feedback compilation 1306
  generating 1296
  option enabling or disabling for specified functions 812, 868
  program 1295
INTEL_PROF_DUMP_CUMULATIVE environment variable 1351
INTEL_PROF_DUMP_INTERVAL environment variable 1351
Intel(R) 64 architecture based applications
  HLO 1357
    methods of parallelization 1036
    options 1106, 1110
    targeting 1106, 1110
Intel(R) C/C++ Error Parser 67
Intel(R) C++ project
  updating to use current compiler version 71
Intel(R) compatibility libraries for OpenMP* 1125
Intel(R) extension environment variables 1168
  Intel(R) extension routines 1185
  Intel(R) IPP libraries
    option letting you link to 442, 701
  Intel(R) linking tools 1271
  Intel(R) MKL
    option letting you link to 478, 713
  Intel(R) TBB libraries
    option letting you link to 808, 863
  Intel(R) Trace Collector API
    option inserting probes to call 810, 867
  Intel-provided libraries
    option linking dynamically 850
    option linking statically 857
  intermediate files
    option saving during compilation 799, 846
  intermediate language scalar optimizer 1057
  intermediate representation (IR) 1271, 1275
  interoperability
    with g++* 125
    with gcc* 125
  interoperability options 125
interprocedural optimizations 302, 430, 436, 689, 695, 1057, 1060, 1271, 1274, 1275, 1278, 1279, 1281, 1283, 1288, 1295, 1353
  capturing intermediate output 1275
  code layout 1281
  compilation 1271
  compiling 1275
  considerations 1279
  creating libraries 1283
  initiating 1353
  issues 1278
  large programs 1279
  linking 1271, 1275
  option enabling additional 430, 689
  option enabling between files 436, 695
  option enabling for single file compilation 302
  options 1274
  overview 1271
  performance 1278
  reports 1060
  using 1275
  whole program analysis 1271
  xiar 1283
  xild 1283
  xilibtool 1283
intrinsics 1367, 1419, 1420, 1423, 1424, 1475, 1477, 1655
  about 1419
intrinsics (continued)
data alignment 1475
data types 1420
Increased Performance across IA 1655
inline assembly 1475, 1477
memory allocation 1475, 1477
naming and syntax 1423
references 1424
registers 1420
Intrinsics for all IA 1427, 1428, 1431, 1432, 1434
arithmetic intrinsics 1427
floating point intrinsics 1428
miscellaneous intrinsics 1434
string and block copy intrinsics 1431
synchronization primitives 1432
introduction to Optimizing Applications 1033
introduction to the compiler 53
IPO
options 1013
option specifying jobs during the link phase of 439, 698
IR 1271, 1275
IVDEP
effect when tuning applications 1357

J
jump tables
option enabling generation of 533, 736

K
KMP_AFFINITY 1168, 1200
 modifier 1200
offset 1200
permute 1200
type 1200
KMP_ALL_THREADS 1168
KMP_BLOCKTIME 1168
KMP_LIBRARY 1168, 1189
KMP_MONITOR_STACKSIZE 1168
KMP_STACKSIZE 1168
KMP_VERSION 1168

L
Lambda expressions 1851
Lambda Expressions
details on using in C++ 1851
function object 1855
lambda-capture 1853
language extensions
g++* 121
gcc-* 121
LD_LIBRARY_PATH 112
libgcc library
 option linking dynamically 851
 option linking statically 858
libraries 111, 112, 113, 115, 222, 224, 225, 500, 590, 854,
  1180, 1185, 1380
-c compiler option 113
creating your own 113
default 111
-FPIC compiler option 113
Intel(R) Math Library 111
LD_LIBRARY_PATH 112
managing 112
OpenMP* run-time routines 1180, 1185
 option enabling dynamic linking of 222
 option enabling static linking of 224
 option preventing linking with shared 854
 option preventing use of standard 500
 option printing location of system 590
 shared 113, 115
 -shared compiler option 113
 static 113
library
option option searching in specified directory for 454
option to search for 453
Library extensions 1846
valarray implementation 1846
library functions 1180, 1185, 1286
Intel extension 1185
OpenMP* run-time routines 1180
library math functions
 option testing errno after calls to 312
linker
 option passing linker option relax to 489
 option passing linker option to 950
linking
 option preventing use of startup files and libraries when 505
 option preventing use of startup files when 503
 option suppressing 225
linking options 1013

1871
linking tools 1013, 1271, 1278, 1283
xild 1271, 1278, 1283
xilibttool 1283
xilink 1271, 1278
linking with IPO 1275
Linux* compiler options
-A 204, 609
-A- 205, 610
-alias-argument 206, 283, 611
-alias-const 207, 613
-align 209
-ansi 210
-ansi-alias 211, 614
-auto-Ilp32 216, 615
-ax 217, 616
-B 220
-Bdynamic 222
-Bstatic 224
-c 90, 225
-C 97, 98, 226
-c99 227, 620
-check-uninit 228
-complex-limited-range 229, 622
-cxilib 230
-D 97, 99, 232
-DD 233, 648
-debug 234
-diag 240, 246, 627, 633
-diag-dump 245, 632
-diag-enable sc-include 251, 638
-diag-enable sc-parallel 252, 640
-diag-error-limit 254, 642
-diag-file 255, 643
-diag-file-append 257, 644
-diag-id-numbers 258, 646
-diag-once 260, 647
-din 261, 649
-dN 262, 650
-dryrun 263
-dumpmachine 264
-dumpversion 265
-dynamic-linker 266
-E 97, 98, 268
-early-template-check 269
-EP 97, 98, 271
-export 272
-export-dir 273
-fabi-version 278
-falias 279

Linux* compiler options (continued)
-falign-functions 280, 655
-falign-stack 281
-fargument-alias 206, 283, 611
-fargument-noalias-global 284
-fast 286
-fast-transcendentals 288, 653
-fbuiltin 289, 516
-fcode-asn 291
-fcommon 292
-fdata-sections 299
-fexceptions 296
-ffnalias 297
-ffreestanding 298, 663
-ffunction-sections 299
-finline 301
-finline-functions 302
-finline-limit 303
-finstrument-functions 304, 687
-fjump-tables 307
-fkeep-static-consts 308, 704
-fma 310, 654
-fmath-errno 312
-fminshared 313
-fmodflap 314
-fnognu-keywords 315
-fnointerface-template-templates 316
-fnsplit 317
-fn Orexceptions 319
-fno-lex 320
-fno-operator-names 318
-fno-rtti 318
-fnsplit 321, 656
-fp 324, 562
-fpack-struct 346
-fpermissive 348
-fpic 348
-fpie 350
-fp-model 325, 334
-fp-port 341, 658
-fp-relaxed 342, 659
-fp-speculation 343, 660
-fp-stack-check 345, 662
-fr32 353
-freg-struct-return 354
-fshort-enums 355
-fsource-asm 356
-fstack-protector 357, 358, 398
-fstack-security-check 357, 358, 398
Linux* compiler options (continued)
- fsyntax-only 359
- ftemplate-depth 360, 814
- ftls-model 361
- ftz 363, 664
- funroll-all-loops 366
- funsigned-bitfields 367
- funsigned-char 368
- fverbose-asm 369
- fvisibility 370
- fvisibility-inlines-hidden 373
- g 107, 374, 957, 963
- g0 375
- gcc 381, 382
- gcc-name 383
- gcc-sys 381, 382
- gcc-version 385
- gdwarf-2 387
- global-hoist 394, 666
- gxx-name 401
- H 404, 667
- help 406
- help-pragmas 408, 668
- l 105, 411
- icc 412
- idirafter 413
- imacros 414
- inline-calloc 415, 670
- inline-debug-info 416, 671
- inline-factor 417, 673
- inline-forceinline 419, 675
- inline-level 420, 512
- inline-max-per-compile 422, 676
- inline-max-per-routine 423, 678
- inline-max-size 425, 679
- inline-max-total-size 427, 681
- inline-min-size 428, 683
- ip 430, 689
- IPF-fltacc 435, 694
- IPF-flt-quantum 433, 692
- IPF-fma 310, 654
- IPF-fp-relaxed 342, 659
- ip-no-inlining 431, 690
- ip-no-pinlining 432, 691
- ipo 436, 695
- ipo-c 438, 697
- ipo-jobs 439, 698
- ipo-S 440, 699

Linux* compiler options (continued)
- ipo-separate 441, 700
- ipp 442, 701
- iprefix 444
- irt 445
- isystem 446
- ivdep-parallel 447, 703
- ivdepprefixed 448
- ivdepprefixedbefore 449
- K++ 451, 871
- kernel 452
- l 453
- L 454
- m 457
- M 459, 708
- m32 460, 461
- m64 460, 461
- malign-double 462
- map-opts 466, 709
- march 467
- mcmode 469
- mcpu 493
- MD 471, 710
- MF 474, 711
- mfixed-range 475
- MG 476, 712
- miee-fp 483
- minstruction 477, 685
- mkf 478, 713
- MM 481, 715
- MMD 482, 716
- mp 483
- MP 484
- mregparm 488
- mrelex 489
- mserialize-volatile 490, 802
- MT 491, 719
- mtime 493
- multibyte-chars 496, 720
- multiple-processes 485, 497
- nobss-init 499, 721
- nodefaultlib 500
- nolib-inline 501
- nostartfiles 503
- nostdinc++ 504
- nostdlib 505
- o 89, 90, 506
### Linux® compiler options (continued)

- `-O` 507  
- `-Ob` 420, 512  
- `-openmp` 519, 722  
- `-openmp-lib` 520, 724  
- `-openmp-link` 522, 726  
- `-openmp-profile` 524, 727  
- `-openmp-report` 525, 728  
- `-openmp-stubs` 526, 730  
- `-openmp-task` 527, 731  
- `-openmp-threadprivate` 529, 732  
- `-opt-block-factor` 530, 734  
- `-opt-calloc` 531  
- `-opt-class-analysis` 532, 735  
- `-opt-jump-tables` 533, 736  
- `-opt-loadpair` 535, 737  
- `-opt-malloc-options` 538, 738  
- `-opt-mem-bandwidth` 539, 738  
- `-opt-mod-versioning` 540, 740  
- `-opt-multi-version-aggressive` 541, 741  
- `-opt-prefetch` 542, 742  
- `-opt-prefetch-initial-values` 543, 744  
- `-opt-prefetch-issue-excl-hint` 545, 745  
- `-opt-prefetch-next-iteration` 546, 746  
- `-opt-ra-region-strategy` 547, 748  
- `-opt-report` 549, 749  
- `-opt-report-file` 550, 751  
- `-opt-report-help` 551, 752  
- `-opt-report-phase` 552, 753  
- `-opt-report-routine` 554, 755  
- `-opt-streaming-stores` 555, 756  
- `-opt-subscript-in-range` 557, 757  
- `-Os` 558  
- `-p` 564  
- `-P` 97, 98, 565  
- `-par-affinity` 566, 761  
- `-parallel` 577, 772  
- `-par-num-threads` 568, 763  
- `-par-report` 569, 764  
- `-par-runtime-control` 570, 766  
- `-par-schedule` 572, 767  
- `-par-threshold` 575, 770  
- `-pc` 578, 773  
- `-pch` 579  
- `-pch-create` 581, 952  
- `-pch-dir` 583  
- `-pch-use` 584  
- `-pg` 564  
- `-pie` 586

---

### Linux® compiler options (continued)

- `-pragma-optimization-level` 587  
- `-prec-div` 588, 777  
- `-prec-sqrt` 589, 778  
- `-print-multi-lib` 590  
- `-prof-data-order` 591, 779  
- `-prof-dir` 592, 780  
- `-prof-file` 593, 781  
- `-prof-func-groups` 595  
- `-prof-func-order` 596, 783  
- `-prof-gen` 598, 785  
- `-prof-hotness-threshold` 600, 786  
- `-prof-src-dir` 601, 788  
- `-prof-src-root` 603, 789  
- `-prof-src-root-cwd` 605, 791  
- `-prof-use` 606, 793  
- `-pthread` 608  
- `-Qinstall` 684  
- `-Qlocation` 90, 705  
- `-Qoption` 90, 759  
- `-qp` 564  
- `-rcd` 795, 839  
- `-rcf` 796, 840  
- `-reserve-kernel-reg` 841  
- `-restrict` 797, 842  
- `-S` 90, 845  
- `-save-temps` 799, 846  
- `-scalar-rep` 801, 848  
- `-shared` 849  
- `-shared-intel` 850  
- `-shared-libgcc` 851  
- `-sox` 807, 853  
- `-static` 854  
- `-static-intel` 857  
- `-static-libgcc` 858  
- `-std` 804, 859  
- `-strict-ansi` 861  
- `-T` 862  
- `-tbb` 808, 863  
- `-tcheck` 809, 866  
- `-tcollect` 810, 867  
- `-tcollect-filter` 812, 868  
- `-tpresent` 872  
- `-traceback` 873  
- `-U` 97, 99, 877  
- `-u` 875  
- `-unroll` 817, 878  
- `-unroll-aggressive` 816, 879  
- `-use-as` 818, 880
Linux* compiler options (continued)
- use-intel-optimized-headers 819, 881
- use-msasm 882
- v 883
- V (L*) 821, 884
- vec 823, 887
- vec-guard-write 825, 888
- vec-report 826, 889
- vec-threshold 827, 891
- version 892
- w 898, 899, 900
- Wa 901
- Wabi 902
- Wall 903
- Wbrief 904
- Wcheck 905
- Wcomment 906
- Wcontext-limit 907
- wd 829, 907
- Wdeprecate 908
- we 830, 909
- Weff++ 651, 910
- Werror 912, 942
- X 105, 949
- X (type) 947
- Xlinker 950
- Zp 966
loadpair optimization
option enabling 535, 737
lock routines 1180
loop blocking factor
option specifying 530, 734
loop interchange 1375
loops 530, 570, 734, 766, 816, 817, 878, 879, 1036, 1244, 1254, 1258, 1357, 1359, 1361, 1375, 1381
anti dependency 1361
constructs 1258
dependencies 1244
distribution 1357
flow dependency 1361
independence 1361
interchange 1357, 1375, 1381
manual transformation 1381
option performing run-time checks for 570, 766
option specifying blocking factor for 530, 734
option specifying maximum times to unroll 817, 878
option using aggressive unrolling for 816, 879
output dependency 1361
parallelization 1036, 1244, 1254
reductions 1361
transformations 1357, 1375
unrolling 1359
vectorization 1254
loop unrolling 1057, 1253, 1357, 1359
limitations of 1359
using the HLO optimizer 1057, 1357

M

Mac OS* X compiler options
- A 204, 609
- A- 205, 610
- alias-args 206, 283, 611
- alias-const 207, 613
- align 209
- ansi 210
Mac OS* X compiler options (continued)
-ansi-alias 211, 614
-ax 217, 616
-B 220
-c 225
-C 226
-c99 227, 620
-check-uninit 228
-complex-limited-range 229, 622
-cxxlib 230
-D 232
-dD 233, 648
-debug 234
-diag 240, 246, 627, 633
-diag-dump 245, 632
-diag-enable sc-include 251, 638
-diag-enable sc-parallel 252, 640
-diag-error-limit 254, 642
-diag-file 255, 643
-diag-file-append 257, 644
-diag-id-numbers 258, 646
-diag-once 260, 647
-diM 261, 649
-dN 262, 650
-dryrun 263
-dumpmachine 264
-dynamiclib 267
-E 268
-early-template-check 269
-EP 271
-export 272
-export-dir 273
-F 274
-fabi-version 278
-falias 279
-falign-functions 280, 655
-falign-stack 281
-fargument-noalias-global 284
-fasm-blocks 285
-fast 286
-fast-transcendentals 288, 653
-fbuiltin 289, 516
-fcode-asm 291
-fcommon 292
-fexceptions 296
-fnalias 297
-ffreestanding 298, 663
-ffunction-sections 299
-finline 301

Mac OS* X compiler options (continued)
-finline-functions 302
-finline-limit 303
-finstrument-functions 304, 687
-fjump-tables 307
-fkeep-static-consts 308, 704
-fmath-errno 312
-fminshared 313
-fno-gnu-keywords 315
-fno-implicit-inline-templates 316
-fno-implicit-templates 317
-fnon-call-exceptions 319
-fnon-value-assign 320
-fno-omit-frame-pointer 324, 562
-fno-operator-names 318
-fno-rtti 318
-fp 324, 562
-fpack-struct 346
-fpascal-strings 347
-fpermissive 348
-fpic 348
-fp-model 325, 334
-fp-port 341, 658
-fp-speculation 343, 660
-fp-stack-check 345, 662
-freg-struct-return 354
-fshort-enums 355
-fsource-asm 356
-fstack-protector 357, 358, 398
-fstack-security-check 357, 358, 398
-fsyntax-only 359
-ftemplate-depth 360, 814
-ftls-model 361
-ftrapuv 362, 815
-ftz 363, 664
-funroll-all-loops 366
-funroll-loops 817, 878
-funsigned-bitfields 367
-funsigned-char 368
-fverbose-asm 369
-fvisibility 370
-fvisibility-inlines-hidden 373
-g 374, 957, 963
-g0 375
-gcc 381, 382
-gcc-name 383
-gcc-sys 381, 382
-gcc-version 385
-gdwarf-2 387

1876
Mac OS* X compiler options (continued)
- global-hoist 394, 666  
- gxx-name 401  
- H 404, 667  
- help 406  
- help-pragma 408, 668  
- I 411  
- icc 412  
- idirafter 413  
- imacros 414  
- inline-calloc 415, 670  
- inline-factor 417, 673  
- inline-forceinline 419, 675  
- inline-level 420, 512  
- inline-max-per-compile 422, 676  
- inline-max-per-routine 423, 678  
- inline-max-size 425, 679  
- inline-max-total-size 427, 681  
- inline-min-size 428, 683  
- ip 302  
- ip-no-inlining 431, 690  
- ip-no-pinlining 432, 691  
- ipo 436, 695  
- ipo-c 438, 697  
- ipo-jobs 439, 698  
- ipo-S 440, 699  
- ipo-separate 441, 700  
- ipp 442, 701  
- iprefix 444  
- iquote 445  
- isystem 446  
- withprefix 448  
- withprefixbefore 449  
- Kc++ 451, 871  
- kernel 452  
- L 454  
- m 457  
- M 459, 708  
- m32 460, 461  
- m64 460, 461  
- m align-double 462  
- m align-mac68k 463  
- m align-natural 464  
- m align-power 465  
- m march 467  
- mcm model 469  
- mcpu 493  
- MD 471, 710  
- mdynamic-no-pic 473  

Mac OS* X compiler options (continued)
- MF 474, 711  
- mfixed-range 475  
- MG 476, 712  
- minstruction 477, 685  
- mkl 478, 713  
- MM 481, 715  
- MMD 482, 716  
- MP 484  
- mp 486, 775  
- MQ 487  
- mregparm 488  
- mserialize-volatile 490, 802  
- MT 491, 719  
- mtune 493  
- multibyte-chars 496, 720  
- multiple-processes 485, 497  
- nobss-init 499, 721  
- nodefaultlibs 500  
- nolib-inline 501  
- nostartfiles 503  
- nostdinc 949  
- nostdinc++ 504  
- nostdlib 505  
- o 506  
- O 507  
- O1 507  
- O2 507  
- O3 507  
- Ob 420, 512  
- openspmd 519, 722  
- openmp-link 522, 726  
- openmp-report 525, 728  
- openmp-stubs 526, 730  
- openmp-task 527, 731  
- opt-block-factor 530, 734  
- opt-class-analysis 532, 735  
- opt-jump-tables 533, 736  
- opt-malloc-options 536  
- opt-multi-version-aggressive 541, 741  
- opt-prefetch 542, 742  
- opt-ra-region-strategy 547, 748  
- opt-report 549, 749  
- opt-report-file 550, 751  
- opt-report-help 551, 752  
- opt-report-phase 552, 753  
- opt-report-routine 554, 755  
- opt-streaming-stores 555, 756  

1877
Mac OS* X compiler options (continued)
- opt-subscript-in-range 557, 757
  - p 564
  - parallel 577, 772
  - par-num-threads 568, 763
  - par-report 569, 764
  - par-runtime-control 570, 766
  - par-schedule 572, 767
  - par-threshold 575, 770
  - pc 578, 773
  - pch 579
  - pch-create 581, 952
  - pch-dir 583
  - pch-use 584
  - pg 564
  - pragma-optimization-level 587
  - prec-div 588, 777
  - prec-sqrt 589, 778
  - print-multi-lib 590
  - prof-data-order 591, 779
  - prof-dir 592, 780
  - prof-file 593, 781
  - prof-func-groups 595
  - prof-func-order 596, 783
  - prof-gen 598, 785
  - prof-src-dir 601, 788
  - prof-src-root 603, 789
  - prof-src-root-cwd 605, 791
  - prof-use 606, 793
  - pthread 608
  - Qinstall 684
  - Qlocation 705
  - Qoption 759
  - qp 564
  - rcd 795, 839
  - rct 796, 840
  - reserve-kernel-regs 841
  - restrict 797, 842
  - S 845
  - save-temps 799, 846
  - scalar-rep 801, 848
  - shared-intel 850
  - static-intel 857
  - staticlib 855
  - std 804, 859
  - strict-ansi 861
  - traceback 873
  - u (L*) 875
  - unroll 817, 878

Mac OS* X compiler options (continued)
- unroll-aggressive 816, 879
  - use-asm 818, 880
  - use-intel-optimized-headers 819, 881
  - use-msasm 882
  - v 883
  - V 821, 884
  - vec 823, 887
  - vec-guard-write 825, 888
  - vec-report 826, 889
  - vec-threshold 827, 891
  --version 892
  - w 899, 900
  - Wa 901
  - Wabi 902
  - Wall 903
  - Wbrief 904
  - Wcheck 905
  - Wcontext-limit 907
  - wd 829, 907
  - Wdeprecated 908
  - we 830, 909
  - Wefc++ 651, 910
  - Werror 912, 941
  - Werror-all 913
  - Wextra-tokens 914
  - Wformat 915
  - Wformat-security 916
  - Winline 917
  - WI 918
  - Wmain 920
  - Wmissing-declarations 921
  - Wmissing-prototypes 922
  - wn 831, 924
  - Wnon-virtual-dtor 923
  - wo 832, 925
  - Wp 925
  - Wp64 926
  - Wpointer-arith 927
  - Wpragma-once 929
  - wr 833, 930
  - Wreorder 931
  - Wreturn-type 932
  - Wshadow 933
  - Wstrict-prototypes 934
  - Wtrigraphs 935
  - Wuninitialized 936
  - Wunknown-pragmas 937
  - Wunused-function 938

1878
Mac OS* X compiler options (continued)
-Wunused-variable 939
-ww 834, 940
-x (type) 947
-Xlinker 950
macro names
option associating with an optional value 232
macros 128, 140, 169
maintainability 1185, 1409
allocation 1185
main thread
option adjusting the stack size for 760
make, using 86
makefiles
modifying 138
manual transformations 1381
math functions
option enabling faster code sequences for 342, 659
Math library
Complex Functions
  cabs library function 1759
cacosh library function 1759
cacos library function 1759
carg library function 1759
casin library function 1759
catanh library function 1759
catan library function 1759
ccosh library function 1759
cosh library function 1759
cstin function 1759
csinh library function 1759
csin library function 1759
csqrt library function 1759
ctan library function 1759
ctanph library function 1759
Exponential Functions
  cbrt library function 1740
exp10 library function 1740
exp library function 1740
frexp library function 1740
hypot library function 1740
ldexp library function 1740
ldexp library function 1740
log10 library function 1740
log1p library function 1740
log2 library function 1740
log library function 1740
logb library function 1740
pow library function 1740
scalb library function 1740
scalb library function 1740
sqrt library function 1740
Hyperbolic Functions
  acosh library function 1738
asin library function 1738
atanh library function 1738
cosh library function 1738
sinh library function 1738
tanh library function 1738
Miscellaneous Functions
  copysign library function 1753
  fabs library function 1753
  fdim library function 1753
  finite library function 1753
  fma library function 1753
  fmax library function 1753
  fmin library function 1753
Miscellaneous Functions 1753
  nextafter library function 1753
Nearest Integer Functions
  ceil library function 1749
  floor library function 1749
  llrint library function 1749
  llround library function 1749
  lrint library function 1749
  lround library function 1749
  modf library function 1749
  nearbyint library function 1749
Nearest Integer Functions 1749
  rint library function 1749
  round library function 1749
  trunc library function 1749
Remainder Functions
  fmod library function 1752
Math library (continued)

Remainder Functions (continued)

remainder library function 1752
remquo library function 1752

Special Functions

annuity library function 1745
compound library function 1745
erfc library function 1745
erf library function 1745
gamma_r library function 1745
gamma library function 1745
j0 library function 1745
j1 library function 1745
jn library function 1745
lgamma_r library function 1745
lgamma library function 1745
tgamma library function 1745
y0 library function 1745
y1 library function 1745
yn library function 1745

Trigonometric Functions

acosd library function 1733
acos library function 1733
asind library function 1733
asin library function 1733
atan2 library function 1733
atan2d library function 1733
atan library function 1733
cosd library function 1733
cos library function 1733
cotd library function 1733
cot library function 1733
sincosd library function 1733
sincos library function 1733
sin library function 1733	
tand library function 1733
tan library function 1733

Math Library

C99 macros 1765
for Linux* OS 1723
function list 1727
using on Linux* OS 1723
memory aliasing 1375
memory bandwidth
option enabling tuning and heuristics for 538, 738

memory dependency
option specifying no loop-carried following IVDEP
447, 703
memory loads
option enabling optimizations to move 394, 666
memory model
option specifying large 469
option specifying small or medium 469
option to use specific 469
Microsoft* Visual C++
option specifying compatibility with 822
Microsoft* Visual Studio
option specifying compatibility with 822
mixing vectorizable types in a loop 1253
MMX(TM) 1251

MMX intrinsics

add_pi16 1491
add_pi32 1491
add_pi8 1491
adds_pi16 1491
adds_pi32 1491
adds_pi8 1491
adds_pu16 1491
adds_pu32 1491
adds_pu8 1491
and_si64 1496
andnot_si64 1496
arithmetic intrinsics 1491
cmpeq_pi16 1496
cmpeq_pi32 1496
cmpeq_pi8 1496
cmpgt_pi16 1496
cmpgt_pi32 1496
cmpgt_pi8 1496
cmpgt_si16 1496
cmpgt_si32 1496
cmpgt_si64 1496
cmpgt_si128 1496
cmpgt_sm16 1496
cmpgt_sm32 1496
cmpgt_sm64 1496
cmpgt_sm128 1496
cmpgt_xm16 1496
cmpgt_xm32 1496
cmpgt_xm64 1496
cmpgt_xm128 1496
cmpgt_xmm16 1496
cmpgt_xmm32 1496
cmpgt_xmm64 1496
cmpgt_xmm128 1496
compare intrinsics 1496
cvpkni64_si64 1488
cvpkni64_vi64 1488
cvpsq64_si64 1488
cvpsq64_si128 1488
cvpsq64_vq64 1488
cvpsq64_vq128 1488
cvpsq64_vq256 1488
cvpsb64_si64 1488
cvpsb64_vi64 1488
cvptp64_si64 1488
cvptp64_vi64 1488
cvptp64_vq64 1488
cvptp64_vq128 1488
cvptp64_vq256 1488
cvptp64_vq4 1488
cvpsd64_si64 1488
cvpsd64_vq64 1488
cvpsd64_vq128 1488
cvpsd64_vq256 1488
cvpsd64_vq4 1488
cvpsq64_vq64 1488
cvpsq64_vq128 1488
cvpsq64_vq256 1488
cvpsq64_vq4 1488
cvpsb64_vq64 1488
cvpsb64_vq128 1488
cvpsb64_vq256 1488
cvpsb64_vq4 1488
cvptp64_vq64 1488
cvptp64_vq128 1488
cvptp64_vq256 1488
cvptp64_vq4 1488
cvpsd64_vq64 1488
cvpsd64_vq128 1488
cvpsd64_vq256 1488
cvpsd64_vq4 1488
cvpsq64_vq64 1488
cvpsq64_vq128 1488
cvpsq64_vq256 1488
cvpsq64_vq4 1488
empty 1488
general support intrinsics 1488
logical intrinsics 1496
madd_pi16 1491
mulh_pi16 1491
mullo_pi16 1491
or_si64 1496
packs_pi16 1488
packs_pi32 1488
packs_pu16 1488
set_pi16 1498
set_pi32 1498
MMX intrinsics (continued)
  set_pi8 1498
  set1_pi16 1498
  set1_pi32 1498
  set1_pi8 1498
  set intrinsics 1498
  setr_pi16 1498
  setr_pi32 1498
  setr_pi8 1498
  setzero_si64 1498
shift intrinsics 1493
  sll_pi16 1493
  sll_pi32 1493
  sll_pi64 1493
  slli_pi16 1493
  slli_pi32 1493
  slli_pi64 1493
  sra_pi16 1493
  sra_pi32 1493
  sra_pi64 1493
  srai_pi16 1493
  srai_pi32 1493
  srai_pi64 1493
  srl_pi16 1493
  srl_pi32 1493
  srl_pi64 1493
  sral_pi16 1493
  sral_pi32 1493
  sral_pi64 1493
  sub_pi16 1491
  sub_pi32 1491
  sub_pi8 1491
  subs_pi16 1491
  subs_pi32 1491
  subs_pi8 1491
  subs_pu16 1491
  subs_pu32 1491
  unpackhi_pi16 1488
  unpackhi_pi32 1488
  unpackhi_pi8 1488
  unpacklo_pi16 1488
  unpacklo_pi32 1488
  unpacklo_pi8 1488
  xor_si64 1496

MMX intrinsics for IA-64 architecture 1500

MMX technology intrinsics
  data types 1485
  registers 1485
mock object files 1275
modifying the compilation environment 101
modulo operations
  option enabling versioning of 539, 740

MOVBE instructions
  option generating 477, 685
multiple processes
  option creating 485, 497
multithreaded programs 1036, 1237
multithreading 1189, 1244
multi-threading performance
  option aiding analysis of 872
multi-version compiler support 62
MXCSR register 1403

N
non-unit memory access 1375
NOPREFETCH
  using 1367
normalized floating-point number 1415
Not-a-Number (NaN) 1415

O
object file
  option increasing number of sections in 223
object files
  specifying 90
OMP_DYNAMIC 1168
OMP_NESTED 1168
OMP_NUM_THREADS 1168
OMP_SCHEDULE 1168
OMP directives 1036, 1099, 1125, 1130, 1135, 1161, 1164, 1168, 1180, 1189, 1200, 1230
  advanced issues 1161
  compatibility libraries 1125, 1189
  compatibility with other compilers 1125
  debugging 1161
directives 1161
evironment variables 1168, 1200
guidelines for using libraries 1125
KMP_AFFINITY 1168, 1200
KMP_ALL_THREADS 1168
KMP_BLOCKTIME 1168
KMP_LIBRARY 1168
KMP_MONITOR_STACKSIZE 1168
KMP_STACKSIZE 1168
KMP_VERSION 1168
KMP_Blocksize 1168
KMP_version 1168
library file names 1189
OMP directives (continued)
load balancing 1135
object-level interoperability 1125
OMP_DYNAMIC 1168
OMP_NESTED 1168
OMP_NUM_THREADS 1168
OMP_SCHEDULE 1168
omp.h 1161
parallel processing thread model 1130
performance 1161
reports 1099
run-time library routines 1180
source compatibility 1125
support libraries 1189
task 1230
OpenMP*
option controlling diagnostics 525, 728
option enabling 519, 722
option enabling analysis of applications 524, 727
option enabling programs in sequential mode 526, 730
option specifying threadprivate 529, 732
OpenMP* analysis 164
OpenMP* Fortran directives 1164, 1177, 1185
clauses for 1177
examples of 1164
Intel extensions for 1185
OpenMP* run-time library
option controlling which is linked to 522, 726
option specifying 520, 724
Open Source tools 96
optimal records to improve performance
analyzing applications 1046
application-specific 1046
hardware-related 1046
library-related 1046
methodology 1041
options
 restricting 1118
 setting 1118
 OS-related 1046
 reports 1057
 resources 1038
 restricting 1118
 setting 1118
 strategies 1046
 system-related 1046
optimization 144, 438, 440, 507, 515, 542, 697, 699, 742
 option enabling global 515
 option enabling prefetch insertion 542, 742
 option generating single assembly file from multiple files 440, 699
 option generating single object file from multiple files 438, 697
 option specifying code 507
optimization report
 option displaying phases for 551, 752
 option generating for routines with specified text 554, 755
 option generating to stderr 549, 749
 option specifying name for 550, 751
 option specifying phase to use for 552, 753
optimizations 514, 558, 559, 1033, 1036, 1101, 1102, 1106, 1117, 1118, 1295, 1296, 1357
 compilation process 1101
 for specific processors 1106
 high-level language 1357
 option disabling all 514
 option enabling all speed 559
 option enabling many speed 558
 options for IA-32 architecture 1106
 options for IA-64 architecture 1117
 options for Intel(R) 64 architecture 1106
 overview of 1101, 1295
 parallelization 1036
 PGO methodology 1296
 profile-guided 1295
 profile-guided optimization 1296
 optimization support 1033
 optimizer report generation 1057
 optimizing 1033, 1046, 1375
 applications 1046
 helping the compiler 1375
 overview 1033
 technical applications 1046
 optimizing performance 1041
 option mapping tool 94
 options for IA-32 architectures 1110
 options used for IPO 1013, 1274
 OptReport support 1057
 ORDERED
 overview of OpenMP* directives and clauses 1177
 output files 89, 506
 option specifying name for 506
overflow call to a runtime library routine 1180
overview 1033, 1036, 1102
of optimizing for specific processors 1102
of parallelism 1036

P

packed structures 1379
PARALLEL DO
  summary of OpenMP* directives and clauses 1177
parallel invocations with makefile 1296
parallelism 1036, 1180, 1237
parallelization 252, 640, 1036, 1086, 1237, 1244, 1254
diagnostic 1086
option enabling analysis in source code 252, 640
parallel lint 1150
PARALLEL OpenMP* directive 1177
parallel processing
  thread model 1130
parallel programming 1033, 1036
parallel region
  option specifying number of threads to use in 568, 763
parallel regions 1177
PARALLEL SECTIONS
  summary of OpenMP* directives 1177
performance 1409
Performance
  MMX Intrinsics 1659
  SSE intrinsics 1663
performance analyzer 1380
performance issues with IPO 1278
PGO 1295
PGO API
  _PGOPTI_Prof_Dump_And_Reset 1355
  _PGOPTI_Prof_Reset 1355
  _PGOPTI_Set_Interval_Prof_Dump 1353
  enable 1350
pgopti.dpi file 1296, 1351
pgopti.spi file 1329
PGO tools 1308, 1329, 1338
code coverage tool 1308
profmerge 1338
proforder 1338
test prioritization tool 1329
pgouser.h header file 1350

pipelining 1036
pointer aliasing
  option using aggressive multi-versioning check for 541, 741
porting
  from GNU gcc* to Microsoft Visual C++* 147
porting applications 137, 147
position-independent code
  option generating 348, 350
position-independent executable
  option producing 586
position-independent external references
  option generating code with 473
pragma alloc_section 1683
pragma distribute_point 1684
pragma intel_omp_task 1688
pragma intel_omp_taskq 1689
pragma ivdep 1691
pragma loop_count 1693
pragma memref_control 1695
pragma noparallel 1705
pragma noprefetch 1706
pragma noswp 1710
pragma nounroll 1711
pragma nounroll_and_jam 1713
pragma nvector 1700
pragma optimization_level 1703
pragma optimize 1701
pragma parallel 1705
pragma prefetch 1706
pragma
  option displaying 408, 668
Pragma
  Intel-specific 1682
  Intel-supported 1720
  overview 1681
pragma swp 1710
pragma unroll 1711
pragma unroll_and_jam 1713
pragma unused 1716
pragma vector
  aligned 1716
  always 1716
  nontemporal 1716
  temporal 1716
  unaligned 1716
precompiled header files 91
preempting functions 1286

preempting functions 1286
preempting functions 1286
preempting functions 1286
PREFETCH
  options used for 1365
  using 1367
prefetches before a loop
  option enabling 543, 744
prefetches for memory access in next iteration
  option enabling 546, 746
prefetches of data 1365, 1367
  optimizations for 1365
prefetch insertion
  option enabling 542, 742
preprocessor compiler options 97
prioritizing application tests 1329
PRIVATE
  summary of data scope attribute clauses 1177
processor
  option optimizing for specific 493
processors
  targeting IA-32 architecture processors using
  options 1106
  targeting IA-64 architecture processors using
  options 1117
  targeting Intel(R) 64 architecture processors using
  options 1106
processor-specific code
  option generating 217, 616
  option generating and optimizing 835, 942
PROF_DIR environment variable 1351
PROF_DUMP_INTERVAL environment variable (deprecated) 1351
PROF_NO_CLOBBER environment variable 1351
profile data records
  option affecting search for 601, 788
  option letting you use relative paths when searching
  for 603, 605, 789, 791
profile-guided optimization 1295, 1296, 1306, 1342,
  1350, 1351, 1353, 1355
  API support 1350
  data ordering optimization 1342
  dumping profile information 1355
  environment variables 1351
  example of 1306
  function grouping optimization 1342
  function ordering optimization 1342
  function order lists optimization 1342
  interval profile dumping 1353
  options 1296
  overview 1295
  phases 1306

profile-guided optimization (continued)
  resetting dynamic profile counters 1355
  resetting profile information 1355
  support 1350
  usage model 1295
  profile-optimized code 1296, 1350, 1352, 1353, 1355
  dumping 1352, 1353
  generating information 1350
  resetting dynamic counters for 1355
profiling
  option enabling use of information from 606, 793
  option instrumenting a program for 598, 785
  option specifying directory for output files 592, 780
  option specifying name for summary 593, 781
profiling information
  option enabling function ordering 596, 783
  option using to order static data items 591, 779
profmerge 1338
program loops 1130, 1237
  parallel processing model 1130
programs
  option maximizing speed in 286
  option specifying aliasing should be assumed in
  279

Q

quick reference 1055, 1102, 1274, 1296
  automatic optimizations 1102
  compiler reports 1055
  IPO options 1274
  PGO options 1296

R

REDUCTION
  summary of data scope attribute clauses 1177
  reductions in loops 1361
  register allocation 1371
  register allocator
    option selecting method for partitioning 547, 748
  remarks
    option changing to errors 913
  removed compiler options 192
report generation
  dynamic profile counters 1355
  Intel extension 1185
Index

report generation (continued)
  OpenMP® run-time 1180
  profile information 1355
  timing 1180
  using compiler commands 1057
  using xi* tools 1285
report software pipelining (SWP) 1088
response files 106
restricting optimization 1118
routine entry
  option specifying the stack alignment to use on
  run-time checking 1110
  run-time performance
    improving 1407

S

sample of timing 1380
scalar replacement
  option enabling during loop transformation 801, 848
  option using aggressive multi-versioning check for
  shareable objects 117
shared libraries 113
shared object
  option producing a dynamic 849
shared scalars 1164
signed infinity 1415
signed zero 1415
SMP systems 1237
software pipelining 1036, 1088
  reports 1088
source code analysis 152, 159, 161, 162, 163, 164
specialized code 1036, 1110
specifying file names
  for assembly files 90
  for executable files 89
  for object files 90
specifying symbol visibility 1392
SSE 1251
SSE2 1251
SSE2 intrinsics
  add_epi16 1577
  add_epi32 1577
  add_epi64 1577
  add_epi8 1577
SSE2 intrinsics (continued)

- cmpunord_pd 1555
- cmpunord_sd 1555
- comieq_sd 1555
- comige_sd 1555
- comigt_sd 1555
- comile_sd 1555
- comilt_sd 1555
- compare operation intrinsics 1592
- conversion operation intrinsics 1595
- cvtspi32_pd 1566
- cvtspi32_ps 1595
- cvtspd_epi32 1566
- cvtspd_pi32 1566
- cvtspd_ps 1566
- cvtspi32_pd 1566
- cvtfps_epi32 1595
- cvtfps_pi32 1566
- cvtfps_ps 1566
- cvtss_f64 1566
- cvtss_sd 1566
- cvtss_sd 1566
- cvtss_ss 1566
- cvtss_si128 1597
- cvtsi28_sd 1566
- cvtsi28_si64 1597
- cvtsi32_sd 1566
- cvtsi32_si128 1597
- cvtsi64_sd 1595
- cvtsi64_si128 1597
- cvtss_sd 1566
- cvtssp_epi32 1566
- cvttdp_epi32 1566
- cvttdp_pi32 1566
- cvttps_epi32 1595
- cvttsd_si32 1566
- cvttss_si64 1595
- div_pd 1550
- div_sd 1550
- extract_epi16 1608
- FP arithmetic intrinsics 1550
- FP compare intrinsics 1555
- FP conversion intrinsics 1566
- FP load intrinsics 1570
- FP logical operations intrinsics 1554
- FP set intrinsics 1573
- FP store intrinsics 1575
- insert_epi16 1608
- l fence 1605
- load_pd 1570
- load_sd 1570

SSE2 intrinsics (continued)

- load_si128 1598
- load1_pd 1570
- loadh_pd 1570
- loadl_epi64 1598
- loadl_pd 1570
- load operation intrinsics 1598
- loadcr_pd 1570
- loadu_pd 1570
- loadu_si128 1598
- logical operations intrinsics 1586
- madd_epi16 1577
- maskmoveu_si128 1603
- max_epi16 1577
- max_epi8 1577
- max_pd 1550
- max_sd 1550
- mfence 1605
- min_epi16 1577
- min_epi8 1577
- min_pd 1550
- min_sd 1550
- miscellaneous intrinsics 1608
- move_epi64 1608
- move_epi64 1608
- move_s 1573
- movemask_epi8 1608
- movemask_pd 1608
- move operation intrinsics 1597
- movepi64_epi64 1608
- movepi64_epi64 1608
- movemask_epi8 1608
- mul_epi32 1577
- mul_ep 1577
- mul_epi16 1577
- mul_epi16 1577
- mull_epi16 1577
- mullo_epi16 1577
- mullo_epi16 1577
- or_pd 1554
- or_si128 1586
- packs_epi16 1608
- packs_epi32 1608
- packus_epi16 1608
- pause intrinsic 1614
- sad_epi8 1577
- sad_epi8 1577
- set_epi16 1599
- set_epi32 1599
- set_epi64 1599
- set_epi8 1599
- set_pd 1573
- set_pd 1573

Intel® C++ Compiler User and Reference Guides
SSE2 intrinsics (continued)

set_sd 1573
set1_epi16 1599
set1_epi32 1599
set1_epi64 1599
set1_epi8 1599
set1_pd 1573
set operation intrinsic 1599
setr_epi16 1599
setr_epi32 1599
setr_epi64 1599
setr_epi8 1599
setr_pd 1573
setzero_pd 1573
setzero_si128 1603
shift operation intrinsics 1587
shuffle_epi32 1608
shuffle_pd 1608
shufflehi_epi16 1608
shufflelo_epi16 1608
shuffle macro 1616
sll_epi16 1587
sll_epi32 1587
sll_epi64 1587
sll_epi8 1587
slli_epi16 1587
slli_epi32 1587
slli_epi64 1587
slli_si128 1599
sqrt_pd 1550
sqrt_sd 1550
sra_epi16 1587
sra_epi32 1587
sra_epi64 1587
srai_epi16 1587
srai_epi32 1587
srai_epi64 1587
srli_epi16 1587
srli_epi32 1587
srli_epi64 1587
srli_si128 1599
store_pd 1575
store_sd 1575
store_si128 1603
store1_pd 1575
storeh_pd 1575
storel_epi64 1603
storel_pd 1575
store operation intrinsics 1603

SSE2 intrinsics (continued)

storer_pd 1575
storeu_pd 1575
storeu_si128 1603
stream_pd 1605
stream_si128 1605
stream_si32 1605
sub_epi16 1577
sub_epi32 1577
sub_epi64 1577
sub_epi8 1577
sub_pd 1550
sub_sd 1550
subs_epi16 1577
subs_epi8 1577
subs_epi16 1577
subs_epi8 1577
subs_epu16 1577
subs_epu8 1577
ucomieq_sd 1555
ucomige_sd 1555
ucomigt_sd 1555
ucomilte_sd 1555
ucomilt_sd 1555
unpackedhi_epi16 1608
unpackedhi_epi32 1608
unpackedhi_epi64 1608
unpackedhi_epi8 1608
unpackedhi_pd 1608
unpackedlo_epi16 1608
unpackedlo_epi32 1608
unpackedlo_epi64 1608
unpackedlo_epi8 1608
unpackedlo_pd 1608
xor_pd 1554
xor_si128 1586

SSE3 intrinsics

addsub_pd 1619
addsub_ps 1618
float32 vector intrinsics 1618
float64 vector intrinsics 1619
hadd_pd 1619
hadd_ps 1618
hsub_pd 1619
hsub_ps 1618
integer vector intrinsic 1617
loadupd_pd 1619
macro functions 1621
miscellaneous intrinsics 1621
<table>
<thead>
<tr>
<th>SSE3 intrinsics (continued)</th>
<th>SSE intrinsics (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>movedup_pd 1619</td>
<td>cmpord_ss 1513</td>
</tr>
<tr>
<td>movehdup_ps 1618</td>
<td>cmpunord_ps 1513</td>
</tr>
<tr>
<td>moveldup_ps 1618</td>
<td>cmpunord_ss 1513</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SSE4 intrinsic</td>
<td></td>
</tr>
<tr>
<td>cacheability support intrinsic 1648</td>
<td>comieq_ss 1513</td>
</tr>
<tr>
<td>DWORD multiply intrinsics 1645</td>
<td>comige_ss 1513</td>
</tr>
<tr>
<td>packed compare for equal intrinsic 1648</td>
<td>comileg_ss 1513</td>
</tr>
<tr>
<td>packed DWORD to unsigned WORD intrinsic 1648</td>
<td>comilt_ss 1513</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SSE4 intrinsics</td>
<td></td>
</tr>
<tr>
<td>application targeted accelerator intrinsics 1651</td>
<td>compare operations 1513</td>
</tr>
<tr>
<td>floating-point rounding intrinsics 1644</td>
<td>conversion operations 1523</td>
</tr>
<tr>
<td>FP dot product intrinsics 1641</td>
<td>cvtpi16_ps 1523</td>
</tr>
<tr>
<td>packed blending intrinsics 1640</td>
<td>cvtpi32_ps 1523</td>
</tr>
<tr>
<td>packed compare intrinsics 1649</td>
<td>cvtpi32x2_ps 1523</td>
</tr>
<tr>
<td>packed format conversion intrinsics 1641</td>
<td>cvtpi8_ps 1523</td>
</tr>
<tr>
<td>packed integer min/max intrinsics 1642</td>
<td>cvtpe16_ps 1523</td>
</tr>
<tr>
<td>register insertion/extraction intrinsics 1645</td>
<td>cvtpe32 1523</td>
</tr>
<tr>
<td>test intrinsics 1646</td>
<td>cvtpe8 1523</td>
</tr>
<tr>
<td></td>
<td>cvtpe16_ps 1523</td>
</tr>
<tr>
<td></td>
<td>cvtpe8_ps 1523</td>
</tr>
<tr>
<td></td>
<td>cvtss_i32 1523</td>
</tr>
<tr>
<td></td>
<td>cvtss_i64 1523</td>
</tr>
<tr>
<td></td>
<td>cvtss_f32 1523</td>
</tr>
<tr>
<td></td>
<td>cvtss_i32 1523</td>
</tr>
<tr>
<td></td>
<td>cvtss_i64 1523</td>
</tr>
<tr>
<td></td>
<td>cvtss_f32 1523</td>
</tr>
<tr>
<td></td>
<td>cvtss_i32 1523</td>
</tr>
<tr>
<td></td>
<td>cvtss_i64 1523</td>
</tr>
<tr>
<td></td>
<td>data types 1503</td>
</tr>
<tr>
<td></td>
<td>div_ps 1507</td>
</tr>
<tr>
<td></td>
<td>div_ss 1507</td>
</tr>
<tr>
<td></td>
<td>extract_pi16 1535</td>
</tr>
<tr>
<td></td>
<td>getcsr 1539</td>
</tr>
<tr>
<td></td>
<td>insert_pi16 1535</td>
</tr>
<tr>
<td></td>
<td>integer operations 1535</td>
</tr>
<tr>
<td></td>
<td>load_ps 1528</td>
</tr>
<tr>
<td></td>
<td>load_ps1 1528</td>
</tr>
<tr>
<td></td>
<td>load_ss 1528</td>
</tr>
<tr>
<td></td>
<td>loadh_pi 1528</td>
</tr>
<tr>
<td></td>
<td>loadl_pi 1528</td>
</tr>
<tr>
<td></td>
<td>load operations 1528</td>
</tr>
<tr>
<td></td>
<td>loadr_ps( 1528</td>
</tr>
<tr>
<td></td>
<td>loadu_ps 1528</td>
</tr>
<tr>
<td></td>
<td>logical operations 1512</td>
</tr>
<tr>
<td></td>
<td>maskmove_si64i 1535</td>
</tr>
<tr>
<td></td>
<td>matrix transposition macro 1547</td>
</tr>
<tr>
<td></td>
<td>max_pi16 1535</td>
</tr>
<tr>
<td></td>
<td>max_ps 1507</td>
</tr>
</tbody>
</table>

1888
SSE intrinsics (continued)
max_pu8 1535
max_ss 1507
min_pi16 1535
min_ps 1507
min_pu8 1507
min_ss 1507
miscellaneous operations 1539
move_ss 1539
movehl_ps 1539
movelh_ps 1539
movemask_pi8 1535
movemask_ps 1539
mul_ps 1507
mul_ss 1507
mulhi_pu16 1535
or_ps 1512
prefetch 1534
programming with SSE intrinsics 1506
rcp_ps 1507
rcp_ss 1507
read/write control register macros 1544
read/write register intrinsics 1539
registers 1503
rsqrt_ps 1507
rsqrt_ss 1507
sad_pu8 1535
set_ps 1530
set_ps1 1530
set_ss 1530
setcsr 1539
set operations 1530
setr_ps 1530
setzero_ps 1530
sfence 1534
shuffle_pi16 1535
shuffle_ps 1539
shuffle function macro 1543
sqr_ps 1507
sqr_ss 1507
store_ps 1531
store_ps1 1531
store_ss 1531
storeh_pi 1531
storel_pi 1531
store operations 1531
storer_ps 1531
storeu_ps 1531
stream_pi 1534
SSE intrinsics (continued)
stream_ps 1534
sub_ps 1507
sub_ss 1507
ucomieq_ss 1513
ucomige_ss 1513
ucomigt_ss 1513
ucomile_ss 1513
ucomilt_ss 1513
ucomineq_ss 1513
unpackhi_ps 1539
unpacklo_ps 1539
using SSE intrinsics on IA-64 architecture 1541
xor_ps 1512
SSSE3 intrinsics
absolute value intrinsics 1628
addition intrinsics 1623
concatenate intrinsics 1630
multiplication intrinsics 1627
negation intrinsics 1631
shuffle intrinsics 1629
subtraction intrinsics 1625
stack
option disabling checking for routines in 397
option enabling probing 621
option specifying reserve amount 275
stack alignment
option specifying for functions 803
stack probing
option enabling 621
stack variables
option initializing to NaN 362, 815
standard directories
option removing from include search path 949
standards conformance 133
static libraries 113, 855
option invoking tool to generate 855
strategies for optimization 1046
Streaming SIMD Extensions 1253, 1503
Streaming SIMD Extensions 2 1549
Streaming SIMD Extensions 3 1617
Streaming SIMD Extensions 4 1639
streaming stores
option generating for optimization 555, 756
subroutines in the OpenMP* run-time library 1180, 1189, 1237, 1367
for OpenMP* 1189
for prefetching 1367
parallel run-time 1237
Supplemental Streaming SIMD Extensions 3 1623
supported tools 96
SWP
SWP reports 1088
symbol names
option using dollar sign when producing 820
symbol preemption 117
symbol visibility 370, 1392
option specifying 370
specifying 1392
symbol visibility on Linux* 1392
symbol visibility on Mac OS* X 1392
synchronization 1036, 1130, 1185, 1237
parallel processing model for 1130
thread-level parallelism 1036
thread sleep time 1185

T
targeting 1106, 1110, 1117
IA-32 architecture processors 1106
Intel(R) 64 architecture processors 1106
Itanium(R) 2 processors 1117
run-time checking 1110
technical applications 1046
template instantation 136
templates
exported 133
test prioritization tool 1329
examples 1329
options 1329
requirements 1329
thread affinity 566, 761, 1200
option specifying 566, 761
threaded applications
option enabling analysis of 809, 866
thread-local storage 130
thread pooling 1245
threshold control for auto-parallelization 1086, 1180, 1253
OpenMP* routines for 1180
reordering 1253
tool options 1308, 1329, 1338
code coverage tool 1308
profmerge 1338
proforder 1338
test prioritization 1329

tools 705, 759, 1308
option passing options to 759
option specifying directory for supporting 705
topology maps 1200
traceback information
option providing 873
transcendental functions
option replacing calls to 288, 653
tselect tool
option producing an instrumented file for 625
option specifying a directory for profiling output for
623
option specifying a file name for summary files for
624

U
Unicode* 54
unvectorizable copy 1253
user functions 1036, 1164, 1180, 1237, 1288, 1290,
1306, 1351, 1380
auto-parallelization 1036, 1237
dynamic libraries 1180
OpenMP* 1164
PGO environment 1351
profile-guided optimization 1306
timing for an application 1380
using Intel(R) Performance Libraries 82
utilities 1308, 1338
profmerge 1338
proforder 1338

V
valarray implementation
compiling code 1847
using in code 1847
variables
option placing in DATA section 499, 721
option saving always 308, 704
vector copy
options 1251
options for 1036
overview 1251
programming guidelines 1251, 1253
reports 1093
vectorization
  option disabling 823, 887
  option setting threshold for loops 827, 891
vectorizer
  option controlling diagnostics reported by 826, 889
vectorizing 1253, 1258, 1295
loops 1258, 1295
version
  option saving in executable or object file 807, 853
visibility declaration attribute 118

W

warnings
  option changing to errors 912, 913, 941
warnings and errors 149
whole program analysis 1271
Windows* compiler options
  /Ap64 212
  /arch 213
  /As 215
  /bigobj 223
  /c 226
  /C 226
  /D 232
  /debug 237
  /E 268
  /EHa 270
  /Ehc 270
  /EHs 270
  /EP 271
  /F 275
  /Fa 276
  /FA 277
  /fast 286
  /FC 290
  /FD 294
  /Fe 295
  /FI 300
  /fixed 306
  /Fm 309
  /Fo 323
  /fp 325, 334
  /Fp 333
  /Fr 351
  /FR 352
  /G2 376
  /GS 378
  Windows* compiler options (continued)
    /G6 378
    /G7 378
    /GA 380
    /Gd 386
    /Ge 388
    /GF 389
    /GF 390
    /Gh 391
    /GH 392
    /Gm 393
    /Gr 395
    /GR 396
    /Gs 397
    /GS 357, 358, 398
    /GT 399
    /GX 400
    /Gy 402
    /Gz 403
    /GZ 404
    /H 405
    /help 406
    /homeparams 409
    /hotpatch 410
    /I 411
    /J 450
    /LD 455
    /link 456
    /MD 472
    /ML 480
    /MP 485, 497
    /MT 492
    /noBool 498
    /nologo 502
    /O 507
    /Oa 512
    /Ob 420, 512
    /Od 514
    /Oe 515
    /Oi 289, 516
    /Op 517
    /Os 558
    /Ot 559
    /Ow 560
    /Ox 561
    /Oy 324, 562
    /P 565
    /prof-func-order 596, 783
    /QA 204, 609

1891
### Windows* compiler options (continued)

- `/QA- 205, 610`
- `/Qalias-args 206, 283, 611`
- `/Qalias-const 207, 613`
- `/Qansi-alias 211, 614`
- `/Qauto-ilp32 216, 615`
- `/Qax 217, 616`
- `/Qc99 227, 620`
- `/Qchkstk 621`
- `/Qcomplex-limited-range 229, 622`
- `/Qcontext-limit 907`
- `/Qcov-dump 254, 622`
- `/Qcov-file 264`
- `/Qcov-gen 625`
- `/Qcxx-features 626`
- `/Qdiag 240, 246, 627, 633`
- `/Qdiag-dump 245, 632`
- `/Qdiag-enable:sc-include 251, 638`
- `/Qdiag-enable:sc-parallel 252, 640`
- `/Qdiag-error-limit 254, 642`
- `/Qdiag-file 255, 643`
- `/Qdiag-file-append 257, 644`
- `/Qdiag-id-numbers 258, 646`
- `/Qdiag-once 260, 647`
- `/QdM 261, 649`
- `/Qdn 262, 650`
- `/Qeffc++ 651, 910`
- `/Qfast-transcendentals 288, 653`
- `/Qfma 310, 654`
- `/Qfnalign 280, 655`
- `/Qfnsplit 321, 656`
- `/Qfp-port 341, 658`
- `/Qfp-relaxed 342, 659`
- `/Qfp-speculation 343, 660`
- `/Qfp-stack-check 345, 662`
- `/Qfreatstanding 298, 663`
- `/Qftz 363, 664`
- `/Qglobal-hoist 394, 666`
- `/QH 404, 667`
- `/Qhelp-pragmas 408, 668`
- `/QIA64-fr32 669`
- `/Qinline-calloc 415, 670`
- `/Qinline-debug-info 416, 671`
- `/Qinline-dllimport 672`
- `/Qinline-factor 417, 673`
- `/Qinline-forceinline 419, 675`
- `/Qinline-max-per-compile 422, 676`
- `/Qinline-max-per-routine 423, 678`

### Windows* compiler options (continued)

- `/Qinlining-max-size 425, 679`
- `/Qinlining-max-total-size 427, 681`
- `/Qinlining-min-size 428, 683`
- `/Qinstruct 477, 685`
- `/Qinstrument-functions 304, 687`
- `/Qip 430, 689`
- `/QIPF-fltacc 435, 694`
- `/QIPF-flt-eval-method 433, 692`
- `/QIPF-fma 310, 654`
- `/QIPF-fp-relaxed 342, 659`
- `/Qip-no-inlining 431, 690`
- `/Qip-no-pinning 432, 691`
- `/Qipo 436, 695`
- `/Qipo-c 438, 697`
- `/Qipo-jobs 439, 698`
- `/Qipo-S 440, 699`
- `/Qipo-separate 441, 700`
- `/Qipp 442, 701`
- `/Qivdep-parallel 447, 703`
- `/Qkeep-static-casts 308, 704`
- `/Qlocation 705`
- `/Qlong-double 707`
- `/QM 459, 708`
- `/Qmap-opts 466, 709`
- `/Qmcmodel 469`
- `/QMD 471, 710`
- `/QMF 474, 711`
- `/QMG 476, 712`
- `/Qmkl 478, 713`
- `/QMM 481, 715`
- `/QMMMD 482, 716`
- `/Qms 717`
- `/Qmspp 718`
- `/QMT 491, 719`
- `/Qnopenmp-threads 496, 720`
- `/Qnobsb-init 499, 721`
- `/Qnopic 722`
- `/Qopenmp 519, 722`
- `/Qopenmp-lib 520, 724`
- `/Qopenmp-link 522, 726`
- `/Qopenmp-profile 524, 727`
- `/Qopenmp-report 525, 728`
- `/Qopenmp-stubs 526, 730`
- `/Qopenmp-task 527, 731`
- `/Qopenmp-threadprivate 529, 732`
- `/Qopt-block-factor 530, 734`
- `/Qopt-class-analysis 532, 735`
- `/Qoption 759`
Windows* compiler options (continued)
/Qopt-jump-tables 533, 736
/Qopt-loadpair 535, 737
/Qopt-mem-bandwidth 538, 738
/Qopt-mod-versioning 539, 740
/Qopt-multi-version-aggressive 541, 741
/Qopt-prefetch 542, 742
/Qopt-prefetch-initial-values 543, 744
/Qopt-prefetch-issue-excl-hint 545, 745
/Qopt-prefetch-next-iteration 546, 746
/Qopt-ra-region-strategy 547, 748
/Qopt-report 549, 749
/Qopt-report-file 550, 751
/Qopt-report-help 551, 752
/Qopt-report-phase 552, 753
/Qopt-report-routine 554, 755
/Qopt-streaming-stores 555, 756
/Qopt-subscript-in-range 557, 757
/Qpar-adjust-stack 760
/Qpar-affinity 566, 761
/Qpar-parallel 577, 772
/Qpar-num-threads 568, 763
/Qpar-report 569, 764
/Qpar-runtime-control 570, 766
/Qpar-schedule 572, 767
/Qpar-threshold 575, 770
/Qc 578, 773
/Qpch 774
/Qprec 486, 775
/Qprec_div 588, 777
/Qprec-sqrt 589, 778
/Qprof-data-order 591, 779
/Qprof-dir 592, 780
/Qprof-file 593, 781
/Qprof-gen 598, 785
/Qprof-hotness-threshold 600, 786
/Qprof-src-dir 601, 788
/Qprof-src-root 603, 789
/Qprof-src-root-cwd 605, 791
/Qprof-vc 606, 793
/Qrcd 795, 839
/Qrct 796, 840
/Qrestrict 797, 842
/Qsafeseh 798
/Qsave-temps 799, 846
/Qserial-rew 801, 848
/Qserialize-volatile 490, 802
/Qsafalign 803
/Qsox 807, 853

Windows* compiler options (continued)
/Qstd 804, 859
/Qtbb 808, 863
/Qtcheck 809, 866
/Qtcollect 810, 867
/Qtcollect-filter 812, 868
/Qtemplate-depth 360, 814
/Qprofile 872
/Qtrapuv 362, 815
/Qunroll 817, 878
/Qunroll-aggressive 816, 879
/Quse-asm 818, 880
/Quse-intel-optimized-headers 819, 881
/Quse-msasm-symbols 820
/QV 821, 884
/Qvc 822
/Qvec 823, 887
/Qvec-guard-write 825, 888
/Qvc-report 826, 889
/Qvec-threshold 827, 891
/Qwd 829, 907
/Qwe 830, 909
/Qwn 831, 924
/Qwo 832, 925
/Qwr 833, 930
/Qww 834, 940
/Qx 835, 942
/RTC 843
/S 845
/showIncludes 852
/Tc 864
/TC 865
/Tp 870
/TP 451, 871
/traceback 873
/U 877
/u (W*) 876
/V (W*) 885
/vd 886
/vm 893
/vmm 894
/vms 895
/vms 896
/vmv 897
/w 898
/W 899, 900
/WW 903
/Wcheck 905
/WL 919
Windows* compiler options (continued)
/Wp64 926
/Wport 928
/X 949
/Y- 951
/Yc 581, 952
/Yd 954
/Yu 955
/YX 956
/Z7 374, 957, 963
/Za 959
/Zc 960
/Zd 961
/Ze 961
/Zg 962
/Zi 374, 957, 963
/ZI 964

Windows* compiler options (continued)
/ZI 965
/Zp 966
/Zs 967
/Zx 968
-Werror-all 913
-WX 912, 941
worker thread 1189
worksharing 1036, 1177, 1237

X
xiar 1278, 1283
xild 1271, 1278, 1283
xilib 1283
xilibtool 1283
xilink 1271, 1278, 1283