Quick Reference Guide to Optimization with Intel® C++ and Fortran Compilers v16

For IA-32 processors, Intel® 64 processors, compatible non-Intel processors, Intel® Xeon Phi™ coprocessors and Intel® Graphics Technology.

Contents

For product and purchase information, visit the Intel® Software Development Tools site at: http://intel.ly/sw-dev-tools.

§§ Compiler support for Intel® Graphics Technology depends on operating system support.

§ Intel® MIC architecture and Intel® Xeon Phi™ coprocessors are supported by compilers within Intel® Parallel Studio XE, but not within Intel® System Studio.

** Several of these options are available for both Intel® and non-Intel microprocessors but they may perform more optimizations for Intel microprocessors than they perform for non-Intel microprocessors.‡

Application Performance

A Step-by-Step Approach to Application Tuning with Intel Compilers

Before you begin performance tuning, check correctness of your application by building it without optimization using **/Od (-O0**).

1. Measure performance using the general optimization options (Windows* **/O1, /O2 or /O3**; Linux* and OS X* **-O1, -O2, or -O3**) to see which one works best for your application. Most users should start at **/O2 (-O2)**, the default, before trying more advanced optimizations.
   Next, try **/O3 (-O3)** for loop-intensive applications.**

2. Fine-tune performance using processor-specific options such as **/Qx (-x)** or **/arch (-m)**.
   Examples are **/QxCORE-AVX2 (-xcore-avx2)** for the 4th Generation Intel® Core™ processor family and **/arch:SSE3 (-msse3)** for compatible, non-Intel processors that support at least the Intel® SSE3 instruction set. Or, **/QxHOST (-xhost)** will use the most advanced instruction set for the processor on which you compiled.**

3. Add interprocedural optimization (IPO), **/Qipo (-ipo)** and/or profile-guided optimization (PGO), **/Qprof-gen** and **/Qprof-use (-prof-gen** and **-prof-use)**, then measure performance again to determine whether your application benefits from one or both of them.

4. Use Intel® Advisor and Intel® VTune™ Amplifier†† to help you identify serial and parallel performance "hotspots" within your application that could benefit from further performance tuning. Use the compiler optimization report **/Qopt-report (-qopt-report)** to help identify individual optimization opportunities.

5. Further optimize your application for SIMD through explicit vector programming using the Intel® Cilk™ Plus language extensions for C/C++ or the SIMD features of OpenMP* 4.0 with **/Qopenmp-simd (-qopenmp-simd)**.†

6. Optimize for parallel execution on multi-threaded, multi-core and multi-processor systems using: the auto-parallelization option **/Qparallel (-parallel)**; the Intel Cilk Plus language extensions for C/C++; OpenMP pragmas or directives along with the option **/Qopenmp (-qopenmp)** †; or by using the Intel® Performance Libraries included with the product.** Use Intel® Inspector to reduce the time to market for threaded applications by diagnosing memory and threading errors and speeding up the development process.

For more details, please consult the main product documentation at https://software.intel.com/intel-software-technical-documentation . The Intel® Compiler User and Reference Guide includes dedicated sections on compiling applications for Intel® MIC Architecture and for Intel® Graphics Technology.

** Several of these options are available for both Intel® and non-Intel microprocessors but they may perform more optimizations for Intel microprocessors than they perform for non-Intel microprocessors.‡

†OpenMP is supported by compilers within Intel® Parallel Studio XE, but not within Intel® System Studio.

††Some features of this product cannot be used on non-Intel microprocessors.

General Optimization Options**

Windows*	Linux* & OS X*	Comment
/Od	**-O0**	**No optimization**. Used during the early stages of application development and debugging.
/Os **/O1**	**-Os** **-O1**	**Optimize for size**. Omits optimizations that tend to increase object size. Creates the smallest optimized code in most cases. May be useful in large server/database applications where memory paging due to larger code size is an issue.
/O2	**-O2**	**Maximize speed**. Default setting. Enables many optimizations, including vectorization. Creates faster code than **/O1 (-O1)** in most cases.
/O3	**-O3**	Enables **/O2 (-O2)** optimizations plus more aggressive loop and memory-access optimizations, such as scalar replacement, loop unrolling, loop blocking to allow more efficient use of cache and additional data prefetching. The **/O3 (-O3)** option is particularly recommended for applications that have loops that do many floating-point calculations or process large data sets. These aggressive optimizations may occasionally slow down other types of applications compared to **/O2 (-O2)**.
/Qopt-report [:*n*]	**-qopt-report [*n*]**	Generates an optimization report, by default written to a file with extension *.optrpt*. ***n*** specifies the level of detail, from ***0*** (no report) to ***5*** (maximum detail). Default is ***2***.
/Qopt-report-file:*name*	**-qopt-report-file=*name***	Writes an optimization report to ***stderr***, ***stdout*** or to the file ***name***.
/Qopt-report-phase:*name1*, *name2*, ...	**-qopt-report-phase=*name1*, *name2*,...**	Optimization reports are generated for optimization phases ***name1*, *name2*,** *etc.* Possible phases include: **all** – Optimization reports for all phases (default) **loop** – Loop nest and memory optimizations **vec** – auto-vectorization and explicit vector programming **par** – auto-parallelization **openmp** – threading using OpenMP **cg** – code generation **ipo** – Interprocedural Optimization, including inlining **pgo** – Profile Guided Optimization **offload** – offload of data to and/or execution on Intel® MIC Architecture or Intel® Graphics Technology.
/Qopt-report-help	**-qopt-report-help**	Displays all possible values of ***name*** for **/Qopt-report-phase (-qopt-report-phase)** above. No compilation is performed.
/Qopt-report-routine: *substring*	**-qopt-report-routine= *substring***	Generates reports only for functions or subroutines whose names contain ***substring***. By default, reports are generated for all functions and subroutines.
/Qopt-report-filter:*"string"*	**-qopt-report-filter=*"string"***	Restricts reports to the file, function or subroutine and/or ranges of line numbers specified by ***"string"***, e.g. "myfile,myfun,line1-line2".

** Several of these options are available for both Intel® and non-Intel microprocessors but they may perform more optimizations for Intel microprocessors than they perform for non-Intel microprocessors.‡

Parallel Performance**

Windows*	Linux* & OS X*	Comment
/Qopenmp†	-qopenmp†	Multi-threaded code and/or SIMD code is generated when OpenMP* directives are present. For Fortran only, makes local arrays automatic and may require an increased stack size.
/Qopenmp-simd†	-qopenmp-simd†	SIMD code is generated when OpenMP SIMD directives are present.
/Qopenmp-stubs	-qopenmp-stubs	Ignores OpenMP directives and links references to OpenMP run-time library functions to stub (dummy) functions assuming single-threaded operation.
/Qparallel	-parallel	The auto-parallelizer detects simply structured loops that may be safely executed in parallel, including the DO CONCURRENT construct and loops implied by Intel® Cilk™ Plus array notation, and automatically generates multi-threaded code for these loops.
/Qpar-threshold[:*n*]	-par-threshold[*n*]	Sets a threshold for the auto-parallelization of loops based on the likelihood of a performance benefit. ***n=0*** (parallelize loops regardless of computational work volume) to ***n=100*** (default; parallelize loops only if a performance benefit is highly likely). Must be used in conjunction with **/Qparallel** (**-parallel**).
/Qpar-affinity:*name*	-par-affinity=*name*	Specifies thread-processor affinity for OpenMP or auto-parallelized applications. Typical values of ***name*** are ***none*** (default), ***scatter*** and ***compact***. Has effect only when compiling the main program. See the compiler user and reference guide for more settings and details.
/Qopt-matmul[-]	-q[no-]opt-matmul	This option enables [disables] a compiler-generated Matrix Multiply (matmul) library call by identifying matrix multiplication loop nests, if any, and replacing them with a matmul library call for improved performance. This option is enabled by default if options **/O3 (-O3)** and **/Qparallel** (**-parallel)** are specified. This option has no effect unless option **/O2 (-O2)** or higher is set.
/Qcilk-serialize	-cilk-serialize	This option includes a header file, cilk_stubs.h, that causes the compiler to ignore all Intel® Cilk™ Plus threading keywords, resulting in a serial executable. (C/C++ only). See the "Using Intel Cilk Plus" section of the user and reference guide for more detail.
/Qcoarray	-coarray	Enables the coarray feature of Fortran 2008 (Fortran only). Options are ***shared***, ***distributed***, ***coprocessor*** and ***single***. See the compiler reference guide for more detail.
/Qmkl:*name*	-mkl=*name*	Requests that the Intel® Math Kernel Library (Intel® MKL) be linked. Off by default. Possible values of ***name*** are: ***parallel*** Links the threaded part of Intel MKL (default) ***sequential*** Links the non-threaded part of Intel MKL ***cluster*** Links cluster and sequential parts of Intel MKL

** Several of these options are available for both Intel® and non-Intel microprocessors but they may perform more optimizations for Intel microprocessors than they perform for non-Intel microprocessors.‡

Parallel Performance Using Intel® Cilk™ Plus

Threading Keywords	Description (C/C++ only)
cilk_spawn	Allows (but does not require) a spawned function to be run in parallel with the caller, subject to dynamic scheduling by the Intel® Cilk™ Plus runtime.
cilk_sync	Introduces a barrier: function cannot continue until all spawned children are complete.
cilk_for	Introduces a for loop whose iterations are allowed (but not required) to run in parallel.

Reducers allow reduction operations, such as accumulation of a sum, to be executed safely in parallel.
E.g. **cilk::reducer< cilk::op_add<unsigned int> >** declares a reducer to sum unsigned ints.
Holders: The **cilk::holder** template class provides a convenient form of task local storage that is thread safe.

Array Notation: a readable, explicitly data-parallel C/C++ language extension that facilitates generation of SIMD parallel code by the vectorizer at optimization level **-O2** or higher and asserts absence of dependencies.
Syntax: **array[<lower bound>:<length>:<stride>]**.
Examples: **bb[:][:] = 0** zeros the entire two-dimensional array bb (size and shape must be known to compiler).
c[j:len] = sqrt(c[k:len:2]) takes the square root of alternate elements of c starting at c[k], and asserts that this is safe to vectorize (e.g., j<k).
Reduction functions are available, e.g. **__sec_reduce_add(a[:])** sums the elements of array a.

SIMD-enabled Functions: a language extension that permits functions to be called in either scalar or SIMD mode, allowing loops containing function calls to be vectorized efficiently. The compiler generates an alternate function version where one or more scalar arguments may be replaced by vectors.
C/C++ declaration syntax: **__declspec (vector(*clauses*)) func_name(arguments)** (or __attribute__)
Fortran equivalent: **!DIR$ ATTRIBUTES VECTOR: (clauses) :: func_name**
Optional clauses include *uniform, linear, mask, processor* and *vectorlength*.
The vector version of the function may be invoked directly using array notation, or indirectly via a loop, e.g.:

```
a[:] = func_name(b[:],c[:],d,..) ;
for (int i=0; i<n; i++)    a[i] = func_name(b[i],c[i],d,...) ;
DO J=1,N;    A(J) = FUNC_NAME(B(J),C(J),D,...) ;    ENDDO
```

Similar functionality is supported via the **DECLARE SIMD** feature of OpenMP* 4.0†. In certain cases, a SIMD pragma or directive may be needed to ensure vectorization of a loop containing a SIMD-enabled function.

Explicit vector programming using the SIMD pragma or directive
This tells the compiler to vectorize a loop using SIMD instructions. The programmer is responsible for correctness, e.g. by explicitly specifying private variables and reductions. Semantics are similar to those for OpenMP worksharing directives.
The compiler also supports similar functionality via the SIMD clause of OpenMP 4.0†.
C/C++ syntax: **#pragma simd [clauses]** Fortran syntax: **!DIR$ SIMD [clauses]**

Clause	Description
private(*var1,var2,...*)	Specifies which variables need to be private to each iteration of the loop, to avoid conflicts.
reduction(oper:var1,var2,...)	Instructs the compiler to accumulate a vector reduction into *var1, var2, ...* under operator oper.
linear(var1:step1,...)	var1 is incremented by step1 for each iteration of the scalar loop.

Other supported clauses: firstprivate, lastprivate, [no]assert, vectorlength, vectrolengthfor, vecremainder
The **_Simd** and **_Reduction** keywords provide an alternative to **#pragma simd reduction(...)**
For more information, see www.cilk.com and the Intel C++ Compiler User and Reference Guide

Recommended Processor-Specific Optimization Options**

Windows*	Linux* OS X*	Comment
/Qx*target*	**-x***target*	Generates specialized code for any Intel® processor that supports the instruction set specified by *target*. The executable will not run on non-Intel processors or on Intel processors that support only lower instruction sets. Possible values of *target*, from highest to lowest instruction set: **CORE-AVX512, MIC-AVX512, COMMON-AVX512, CORE-AVX2, AVX, SSE4.2, ATOM_SSE4.2, SSE4.1, ATOM_SSSE3, SSSE3, SSE3, SSE2** **Note:** This option enables additional optimizations that are not enabled by the /arch or -m options. On 64 bit OS X, options **SSE3** and **SSE2** are not supported.
/arch: *target*	**-m***target*	Generates specialized code for any Intel processor or compatible, non-Intel processor that supports the instruction set specified by *target*. Running the executable on an Intel processor or compatible, non-Intel processor that does not support the specified instruction set may result in a run-time error. Possible values of *target* : **AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2, IA32** **Note:** Option **IA32** generates non-specialized, generic x86/x87 code. It is supported on IA-32 architecture only. It is not supported on OS X.
/QxHOST	**-xhost**	Generates instruction sets up to the highest that is supported by the compilation host. On Intel processors, this corresponds to the most suitable **/Qx (-x)** option; on compatible, non-Intel processors, this corresponds to the most suitable of the **/arch (-m)** options **IA32, SSE2 or SSE3**. This option may result in additional optimizations for Intel® microprocessors that are not performed for non-Intel microprocessors.‡
/Qax*target*	**-ax***target*	May generate specialized code for any Intel processor that supports the instruction set specified by *target*, while also generating a default code path. Possible values of *target* : **CORE-AVX512, MIC-AVX512, COMMON-AVX512, CORE-AVX2, AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2** Multiple values, separated by commas, may be used to tune for additional Intel processors in the same executable, e.g. **/QaxAVX,SSE4.2**. The default code path will run on any Intel or compatible, non-Intel processor that supports at least **SSE2**, but may be modified by using in addition a **/Qx** (**-x**) or **/arch** (**-m**) switch. For example, to generate a specialized code path optimized for the 4th generation Intel® Core™ processor family and a default code path optimized for Intel processors or compatible, non-Intel processors that support at least **SSE3**, use **/QaxCORE-AVX2 /arch:SSE3 (-axcore-avx2 -msse3** on Linux*). At runtime, the application automatically detects whether it is running on an Intel processor, and if so, selects the most appropriate code path. If an Intel processor is not detected, the default code path is selected. **Note:** On 64 bit OS X*, options **sse3** and **sse2** are not supported. This option may result in additional optimizations for Intel microprocessors that are not performed for non-Intel microprocessors.‡

Please see the online article "Intel® compiler options for Intel® SSE and Intel® AVX generation and processor-specific optimizations" to view the latest recommendations for processor-specific optimization options.

** Several of these options are available for both Intel® and non-Intel microprocessors but they may perform more optimizations for Intel microprocessors than they perform for non-Intel microprocessors.‡

Compiling for Offload

Windows*	Linux*	Comment
/Qoffload[-] **/Qoffload[:*kywd*]**	**-q[no-]offload** **-qoffload=*kywd***	Controls whether the compiler honors language constructs for offloading to Intel® MIC architecture or Intel® Graphics Technology. ***kywd*** can take values: ***none***: offload constructs are ignored and all code is compiled for execution on the host only. Equivalent to **/Qoffload-** (**-qno-offload**) ***mandatory***: if target is not available, program fails unless status clause is present, in which case offload code is skipped. ***optional***: if target is not available, all code is executed on host cpu. Default is **/Qoffload:mandatory** (**-qoffload=mandatory**)
/Qoffload-option, ***target, tool, "option-list"***	**-qoffload-option,** ***target, tool, "option-list"***	Specifies options to be used for the target compilation but not for the host. ***target*** may be ***mic*** (for Intel® MIC architecture) or ***gfx*** (for Intel® Graphics Technology). ***tool*** may be ***compiler***, ***ld***, ***link*** or ***as***.
/Qoffload-attribute-target: ***target-name***	**-qoffload-attribute-target** **=*target-name***	Flags every file-scope function or data object with the offload attribute target(***target-name***), where ***target-name*** is ***mic*** (for Intel MIC architecture) or ***gfx*** (for Intel Graphics Technology).
/Qopt-report-phase:*offload*	**-qopt-report-phase *offload***	Generates a compile time report of variables that will be copied to or from the host and the coprocessor or processor graphics.
__INTEL_OFFLOAD	**__INTEL_OFFLOAD**	Predefined macro for use in offload programming on the host

Compiling for Intel® Graphics Technology §§

Windows* (32 and 64 bits)	Linux* (64 bits only)	Comment
/Qgpu-arch:*arch*	**-mgpu-arch=*arch***	The compiler generates native instructions for the graphics processor that is on the Intel® microarchitecture codenamed ***ivybridge***, ***haswell*** or ***broadwell***, as specified by ***arch***. Default is virtual instructions translated by the jit engine.
__GFX__	**__GFX__**	Predefined macro for use when programming for Intel Graphics Technology

Environment Variables for Intel® Graphics Technology §§

Variable	Comment
GFX_CPU_BACKUP=1	Offload code is executed on host when target is not available (default). If =***0***, application fails if target is not available.
GFX_MAX_THREAD_COUNT	Maximum number of target threads when parallelizing loop nests. Default=***-1*** (system default).
GFX_OFFLOAD_TIMEOUT=*n*	Offload tasks time out after ***n*** seconds (default=***60***). The system recovery timeout may need to be disabled or increased for this option to be effective.
GFX_SHOW_TIME=1	Prints offload timing information at end of execution. Default=***0*** (no printing)
GFX_LOG_OFFLOAD=*n*	Generates an offload log. ***n*** specifies the level of detail, from ***0*** (no log) to ***3*** (maximum detail). Default is ***0***.

§§ Compiler support for Intel® Graphics Technology depends on operating system support.

For more information, see the Getting Started guide at https://software.intel.com/articles/getting-started-with-compute-offload-to-intelr-graphics-technology and the Intel® Compiler User and Reference Guides.

Optimizing for the Intel® Xeon Phi™ Coprocessor x100 product family§

(formerly code name Knights Corner)

Windows*	Linux*	Comment
/Qmic	**-mmic**	Builds an application that runs natively on the Intel® Xeon Phi™ coprocessor x100 product family. (Off by default).
/Qopt-streaming-cache-evict:*n*	**-qopt-streaming-cache-evict=*n***	Controls whether compiler generates a cache line evict instruction after a streaming store. ***n=0*** no clevict; ***n=1*** L1 clevict only; ***n=2*** L2 clevict only (default); ***n=3*** L1 and L2 clevict generated.
/Qopt-assume-safe-padding	**-qopt-assume-safe-padding**	Asserts that compiler may safely access up to 64 bytes beyond the end of array or dynamically allocated objects as accessed by the user program. User is responsible for padding. Off by default.
/Qopt-threads-per-core:*n*	**-qopt-threads-per-core=*n***	Hint to the compiler to optimize for ***n*** threads per physical core, where ***n=1***, ***2***, ***3*** or ***4***.
/Qopt-prefetch:*n*	**-qopt-prefetch=*n***	Enables increasing levels of software prefetching for ***n=0*** to ***4.*** Default is ***n=3*** at optimization levels of **-O2** or higher.
/Qimf-domain-exclusion:*n*	**-fimf-domain-exclusion=*n***	Specifies special case arguments for which math functions need not conform to IEEE standard. The bits of n correspond to the domains: ***0*** – extreme values (e.g. very large; very small; close to singularities); ***1*** – NaNs; ***2*** – infinities; ***3*** – denormals; ***4*** – zeros.
/Qopt-gather-scatter-unroll	**-qopt-gather-scatter-unroll**	Specifies an alternative loop unroll sequence for gather and scatter loops
/align: array64byte	**-align array64byte**	Seek to align the start of arrays at a memory address that is divisible by 64, to enable aligned loads and help vectorization. (Fortran only)
__MIC__	**__MIC__**	Predefined macro for use in programming for Intel® MIC architecture

Environment Variables for the Intel® Xeon Phi™ Coprocessor§

Variable	Comment
OFFLOAD_REPORT=<*n*>	Provides a run-time report for offload applications ***n***=1 reports execution times on host and on coprocessor ***n***=2 also reports on data transfers between host and coprocessor ***n***=3 detail on device initialization and individual variable transfers
OFFLOAD_DEVICES= <*n1*,*n2*,...>	Restricts the process on the host to use only the physical coproccessors ***n1***, ***n2***, etc., numbered from ***0***.
MIC_STACKSIZE=<*n*>M	Sets the maximum stack size on the coprocessor for offload applications. In this example, ***n*** is in Megabytes.
MIC_ENV_PREFIX=<name>	Specify prefix to distinguish environment variables on the coprocessor from ones on the host, for offload applications. E.g. if ***name***=MIC, then MIC_OMP_ NUM_THREADS controls the number of OpenMP* threads on the coprocessor.
MIC_USE_2MB_ BUFFERS=<*n*>M	Offloaded pointer variables whose runtime data length exceeds ***n*** MB will be allocated in large, 2MB pages.

For more optimization detail, see https://software.intel.com/articles/advanced-optimizations-for-intel-mic-architecture ; for building for Intel® MIC architecture in general, see http://software.intel.com/mic-developer, https://software.intel.com/articles/programming-and-compiling-for-intel-many-integrated-core-architecture and the Intel® Compiler User and Reference Guides at https://software.intel.com/intel-cplusplus-compiler-16.0-user-and-reference-guide and https://software.intel.com/intel-fortran-compiler-16.0-user-and-reference-guide.

§ Intel® MIC architecture and Intel Xeon Phi coprocessors are supported by compilers within Intel® Parallel Studio XE, but not within Intel® System Studio.

Interprocedural Optimization (IPO) and Profile-Guided Optimization (PGO) Options

Windows*	Linux* OS X*	Comment
/Qip	**-ip**	Single file interprocedural optimizations, including selective inlining, within the current source file.
/Qipo[*n*]	**-ipo[*n*]**	Permits inlining and other interprocedural optimizations among multiple source files. The optional argument ***n*** controls the maximum number of link-time compilations (or number of object files) spawned. Default for ***n*** is ***0*** (the compiler chooses). Caution: This option can in some cases significantly increase compile time and code size.
/Qipo-jobs[*n*]	**-ipo-jobs[*n*]**	Specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO). The default is ***1*** job.
/Ob2	**-finline-functions -finline-level=2**	This option enables function inlining within the current source file at the compiler's discretion. This option is enabled by default at **/O2** and **/O3** (**-O2** and **-O3**). Caution: For large files, this option may sometimes significantly increase compile time and code size. It can be disabled by **/Ob0** (**-fno-inline-functions** on Linux* and OS X*)
/Qinline-factor:*n*	**-finline-factor=*n***	This option scales the total and maximum sizes of functions that can be inlined. The default value of ***n*** is 100, i.e., 100% or a scale factor of one.
/Qprof-gen [*:kywd*]	**-prof-gen [=kywd]**	Instruments a program for profile generation. ***kywd= threadsafe*** allows profile generation for threaded applications. ***kywd=srcpos*** and ***globdata*** collect additional data useful for function and data ordering.
/Qprof-use	**-prof-use**	Enables the use of profiling information during optimization.
/Qprof-dir *dir*	**-prof-dir *dir***	Specifies a directory for profiling output files, *.dyn and *.dpi.
PROF_DIR	**PROF_DIR**	Environment variable to specify a directory for profiling output files (alternative to **/Qprof-dir** or **-prof-dir**).
/Qprofile-functions	**-profile-functions**	Instruments functions so that a profile of execution time spent in each function may be generated.
/Qprofile-loops	**-profile-loops**	Instruments functions to generate a profile of each loop or loop nest in serial code. See "Profile Function or Loop Execution Time" in the main compiler documentation for additional detail and how to view profiles.

Floating-Point Arithmetic Options

Windows*	Linux* & OS X*	Comment
/fp:*name*	**-fp-model *name***	Controls tradeoffs between performance, accuracy and reproducibility of floating-point results at a high level. Possible values of ***name***: **Fast[=1|=2]** – Allows more aggressive optimizations at a slight cost in accuracy or reproducibility. (default **fast=1**) **precise** – Disallows optimizations that might produce slight variations in floating point results. **double/extended/source** – Requires intermediate results to be computed in the corresponding precision. Implies **precise** unless overridden. **double** and **extended** are not available for Fortran. **except** – Enforces floating point exception semantics. **strict** – enables both the **precise** and **except** options and does not assume the default floating-point environment. Suppresses generation of fused multiply-add (FMA) instructions by the compiler. **Recommendation: /fp:precise /fp:source (-fp-model precise -fp-model source)** is the recommended form for the majority of situations where enhanced floating point consistency and reproducibility are needed.
/Qopt-dynamic-align[-]	**-q[no-]opt-dynamic-align**	Allows [disables] certain optimizations that depend on data alignment at run-time, and that could cause small variations in floating-point results when the same, serial application is run repeatedly on the same input data. On by default unless **/fp:precise** (**-fp-model precise**) is set.
/Qftz[-]	**-ftz[-]**	When the main program or dll main is compiled with this option, denormals (resulting from Intel® SSE or Intel® AVX instructions) at run time are flushed to zero for the whole program (dll). Default is on except at **/Od (-O0)**.
/Qimf-precision: *name*	**-fimf-precision: *name***	Sets the accuracy for math library functions. Default is OFF (compiler uses default heuristics). Possible values of ***name*** are ***high***, ***medium*** and ***low***. Reduced precision may lead to increased performance and vice versa, particularly for vectorized code.
/Qimf-arch-consistency: *true*	**-fimf-arch-consistency= *true***	Ensures that math library functions produce consistent results across different Intel or compatible, non-Intel processors of the same architecture. May decrease run-time performance. The default is "***false***" (off).
/Qprec-div[-]	**-[no-]prec-div**	Improves [reduces] precision of floating point divides. This may slightly degrade [improve] performance.
/Qprec-sqrt[-]	**-[no-]prec-sqrt**	Improves [reduces] precision of square root computations. This may slightly degrade [improve] performance.
/Qprotect-parens[-] **/assume: [no]protect _parens**	**-fprotect-parens[-]** **-assume [no]protect _parens**	(C/C++ option) Expressions are evaluated in the order specified by parentheses (Fortran option) Default is off unless /fp:precise (-fp-model precise) is specified.
/Qfma[-]	**-[no]fma**	Suppresses generation of fused multiply-add (FMA) instructions by the compiler (may still be present in run-time libraries).

See also http://software.intel.com /articles/consistency-of-floating-point-results-using-the-intel-compiler

Fine-Tuning (All Processors)

Windows*	Linux* & OS X*	Comment
/Qunroll[*n*]	**-unroll[*n*]**	Sets the maximum number of times to unroll loops. ***n=0*** disables loop unrolling. The default is **/Qunroll (-unroll)**, which uses default heuristics.
/Qopt-prefetch:*n*	**-qopt-prefetch=*n***	Enables increasing levels of software prefetching from ***n=0*** (no prefetching) to ***n=4*** (aggressive prefetching). The default value is ***2*** when **/O3 (-O3)** is enabled. Warning: excessive prefetching may result in resource conflicts that degrade performance.
/Qopt-block-factor:*n*	**-qopt-block-factor=*n***	Specifies a preferred loop blocking factor ***n***, the number of loop iterations in a block, overriding default heuristics. Loop blocking, enabled at **/O3** (**-O3**), is designed to increase the reuse of data in cache.
/Qopt-streaming-stores:*mode*	**-qopt-streaming-stores *mode***	Controls streaming store generation. Values for ***mode***: **always** Encourages generation of streaming stores that bypass cache, assuming application is memory bound with little data reuse **never** Disables generation of streaming stores **auto** Uses default compiler heuristics
/Qrestrict[-]	**-[no]restrict**	Enables [disables] pointer disambiguation with the **restrict** keyword. Off by default. (C/C++ only)
/Oa	**-fno-alias**	Assumes no aliasing in the program. Off by default.
/Ow	**-fno-fnalias**	Assumes no aliasing within functions. Off by default.
/Qalias-args[-]	**-fargument-[no]alias**	Implies function arguments may be aliased [are not aliased]. On by default. (C/C++ only). **-fargument-noalias** often helps the compiler to vectorize loops involving function array arguments.
/Qansi-alias[-]	**-[no-]ansi-alias**	Enables [disables] ANSI and ISO C Standard aliasability rules. Defaults: disabled on Windows*; enabled on Linux* and OS X*.
/Qopt-class-analysis[-]	**-q[no-]opt-class-analysis**	C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time. If a C++ application contains non-standard C++ constructs, such as pointer downcasting, it may result in different behavior. Off by default, but enabled with **/Qipo** (Windows) or **-ipo** (Linux and OS X). (C++ only)
/Qvec-threshold:*n*	**-vec-threshold=*n***	Sets a threshold **n** for the auto-vectorization of loops based on the likelihood of a performance benefit. Default is ***n=100***. ***n=0*** (vectorize loops regardless of amount of computational work) to ***n=100*** (vectorize loops only if a performance benefit is highly likely).
/Qvec[-]	**-[no-]vec**	Enables [disables] auto-vectorization. On by default at **/O2** (**-O2**)
/align: array*nn*byte	**-align array*nn*byte**	Seek to align the start of arrays at a memory address that is divisible by ***nn***, to facilitate aligned loads and help vectorization. (Fortran only)
/assume [no] buffered_io	**-assume [no] buffered_io**	Accumulates data for successive sequential reads or writes into a buffer for more efficient I/O. Default is unbuffered. (Fortran only)
	-f[no-] exceptions	**-fexceptions** enables exception handling table generation (default for C++). **-fno-exceptions**, default for C or Fortran, may result in smaller code. For C++, it causes exception specifications to be parsed but ignored. Any use of exception handling constructs (such as try blocks and throw statements) will produce an error if any function in the call chain has been compiled with -fno-exceptions.

Debug Options

Windows*	Linux* OS X*	Comment
/Zi **/debug** **/debug:full** **/debug:all**	**-g** **-debug** **-debug full** **-debug all**	Produces debug information for use with any of the common platform debuggers, for full symbolic debugging of unoptimized code. Turns off optimization unless **/O2 (-O2)** (or another **O** option) is specified. Debug symbols will generally increase the size of object modules and may slightly degrade performance of optimized code.
/debug:none	**-debug none**	No debugging information is generated. (default)
/debug: **minimal**	**-debug** **minimal**	Generates line number information for debugging, but not local symbols.
/debug:inline **-debug-info**	**-debug** **inline-debug-** **info**	This option causes symbols for inlined functions to be associated with the source of the called function, instead of with the caller. Not enabled by **/debug:full** (**-debug full**) unless **-O2** is specified.
	-debug **extended**	produces additional information for improved symbolic debugging of optimized code. (Linux* only; not enabled by **-debug full**).
	-debug **parallel**	generates additional symbols and instrumentation for debugging threaded code. (Linux only; not enabled by **-debug full**).
/Qsox[-]	**-[no-]sox** (Linux only)	Embeds the compiler version and options used as strings in the object file (Windows* and Linux) and in the executable (Linux). Off by default.
/Qtraceback	**-traceback**	Compiler includes slight extra information in the object file to provide source file traceback information when a severe error occurs at run time. May be used with optimized code. (Fortran applications only)
/Qinit:snan, **arrays**	**-init=snan,** **arrays**	Initializes certain floating-point variables to a signaling NaN to facilitate run-time detection of uninitialized floating-point variables. (Fortran only). Sets /fpe:0 (-fpe0) to unmask floating-point exceptions.

‡ Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

For product and purchase information, visit the Intel® Software Development Tools site at: http://intel.ly/sw-dev-tools.