Intel® Software
Development Products

Software

Quick Reference Guide to Optimization with Intel® Compilers Version 13

For IA-32 processors, Intel® 64 processors,
and compatible, non-Intel processors

Application Performance

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

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

1. Use the general optimization options (Windows* **/O1, /O2, or /O3**; Linux* and OS X* **-O1, -O2, or -O3**) and determine which one works best for your application by measuring performance with each. Most users should start at **/O2 (-O2)** (default) before trying more advanced optimizations. Next, try **/O3 (-O3)** for loop-intensive applications.[1]

2. Fine-tune performance to target IA-32 and Intel® 64-based systems with processor-specific options. Examples are **/QxAVX (-xavx)** for the 2nd- and 3rd-Generation Intel® Core™ processor families, and /arch:SSE3 (-msse3) for compatible, non-Intel processors that support at least the SSE3 instruction set. Alternatively, you can use **/QxHOST (-xhost),** which will use the most advanced instruction set for the processor on which you compiled.[1] For a more extensive list of options that optimize for specific processors or instruction sets, see the table "Recommended Processor-Specific Optimization Options."

3. Add interprocedural optimization (IPO), **/Qipo (-ipo)** and/or profile-guided optimization (PGO), **/Qprof-gen** and **/Qprof-use (-prof-gen** and **-prof-use)**[1], then measure performance again to determine whether your application benefits from one or both of them.

4. Optimize your application for vector and parallel execution on multithreaded, multicore, and multiprocessor systems using: advice from the Guided Auto-Parallelism (GAP) feature, /Qguide (-guide); the Intel® Cilk™ Plus language extensions for C/C++; the parallel performance options **/Qparallel (-parallel)** or **/Qopenmp (-openmp)**; or by using the Intel® Performance Libraries included with the product.[1]

5. Use Intel® VTune™ Amplifier XE to help you identify serial and parallel performance hotspots, so that you know which specific parts of your application could benefit from further tuning. Use Intel® Inspector XE to reduce the time to market for threaded applications by diagnosing memory and threading errors and speeding up the development process. These products cannot be used on non-Intel microprocessors.

Please consult the main product documentation for more details. For tuning applications for Intel® Xeon Phi™ coprocessors, see the references in the section "Options for the Intel® Xeon Phi™ Coprocessor."

[1] 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.

Intel® Software Development Products

General Optimization Options

These options are available for both Intel and non-Intel microprocessors, but they may result in more optimizations for Intel microprocessors than for non-Intel microprocessors.

Windows*	Linux*/OS X*	Comment
/Od	**-O0**	No optimization. Used during the early stages of application development and debugging. Use a higher setting when the application is working correctly.
/O1	**-O1**	Optimize for size. Omits optimizations that tend to increase object size. Creates the smallest optimized code in most cases. This option is useful in many 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, code replication to eliminate branches, 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*]	**-opt-report [*n*]**	Generates an optimization report directed to stderr. *n* specifies the level of detail, from 0 (no report) to 3 (maximum detail). Default is 2.
/Qopt-report-file:*name*	**-opt-report-file=*name***	Generates an optimization report directed to the file *name*.

Windows*	Linux*/OS X*	Comment
/Qopt-report-phase:*name*	**-opt-report-phase=*name***	Optimization reports are generated for phase *name*. The option can be used multiple times in the same compilation to get output from multiple phases. Some commonly used *name* arguments are as follows: all—All possible optimization reports for all phases (default) ipo_inl—Inlining report from the Interprocedural Optimizer hlo—High-Level Optimizer (includes loop and memory optimizations) hpo—High-Performance Optimizer (includes vectorizer and parallelizer) pgo—Profile Guided Optimizer
/Qopt-report-help	**-opt-report-help**	Displays all possible values of *name* for /Qopt-report-phase (-opt-report-phase) above. No compilation is performed.
/Qopt-report-routine:*string*	**-opt-report-routine=*string***	Generates reports only for functions or subroutines whose names contain *string*. By default, reports are generated for all functions and subroutines.

Parallel Performance

Options that use OpenMP* or auto-parallelization are available for both Intel and non-Intel microprocessors, but these options may result in additional optimizations on Intel microprocessors that do not occur on non-Intel microprocessors.

Windows*	Linux*/OS X*	Comment
/Qopenmp	**-openmp**	Causes multithreaded code to be generated when OpenMP* directives are present. For Fortran only: makes local arrays automatic and may require an increased stack size. See http://www.openmp.org for the OpenMP API specification.
/Qparallel	**-parallel**	The auto-parallelizer detects simply structured loops that may be safely executed in parallel, including loops implied by Intel® Cilk™ Plus array notation, and automatically generates multithreaded code for these loops.

/Qpar-report[:*n*]	**-par-report[*n*]**	Controls the auto-parallelizer's diagnostic level. *n* specifies the level of detail, from 0 (no report) to 3 (maximum detail). Default is 0.
/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 to 100, default 100. 0: Parallelize loops regardless of computation work volume. 100: 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.

Windows*	Linux*/OS X*	Comment
/Qguide[:*n*]	**-guide[=*n*]**	Guided auto-parallelization. Causes the compiler to suggest ways to help loops to vectorize or auto-parallelize, without producing any objects or executables. Auto-parallelization advice is given only if the option /Qparallel (-parallel) is also specified. *n* is an optional value from 1 to 4 specifying increasing levels of guidance to be provided; level 4 being the most advanced and aggressive. If *n* is omitted, the default is 4.
/Qopt-matmul[-]	**-[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, which 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:shared	**-coarray=shared**	Enables coarrays from the Fortran 2008 standard on shared memory systems (Fortran only). See the Compiler Reference Guide for more coarray options and 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 the cluster and sequential parts of Intel MKL

Parallel Performance Using Intel® Cilk™ Plus

Threading Keywords (C/C++)	Description
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_opadd<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 threadsafe.

Array Notation: A readable, explicitly data-parallel C and C++ language extension that facilitates generation of SIMD parallel code by the compiler 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 of the array object must be known to the 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).

Elemental Functions: A language extension that allows functions to be called in either scalar or SIMD parallel 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)

Fortran equivalent: !DIR$ ATTRIBUTES VECTOR: (clauses) :: func_name

Optional clauses include *procname*, *vectorlength*, *vectorlengthfor*, *linear*, *uniform*, and *mask*.

The vector version of the function may be invoked directly using array notation, or indirectly via a loop, for example:

```
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
```

In certain cases, a SIMD pragma (C/C++) or directive (Fortran) may be needed to ensure vectorization of a loop containing an elemental function.

Explicit vector programming using the SIMD pragma (C/C++) and directive (Fortran)

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 the OpenMP* directives:

#pragma omp parallel for (!$OMP PARALLEL DO) .

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*.
[no]assert	Compiler emits an error if it is unable to vectorize the loop. (default noassert).
linear(var1:step1,...)	var1 is incremented by step1 for each iteration of the scalar loop.

Other supported clauses: vectorlength, vectorlengthfor, firstprivate, lastprivate, vecremainder.

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
/Qxtarget	**-xtarget**	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-AVX2, AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2 Note: On OS X*, options SSE3 and SSE2 are not supported. This option enables additional optimizations that are not enabled by the /arch or –m options.
/arch:target	**-mtarget**	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 runtime 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. On OS X, options SSE3, SSE2 and IA32 are not supported.
/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.‡

Windows*	Linux*/OS X*	Comment
/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-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 2nd 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 /QaxAVX /arch:SSE3 (-axavx –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 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 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. These options are described in greater detail in the Intel® Compiler User and Reference Guides.

Options for the Intel® Xeon Phi™ Coprocessor (Linux* only)

-mmic	Builds an application that runs natively on Intel® Xeon Phi™ coprocessors. Off by default.
-no-offload	The compiler ignores language constructs for offloading to Intel Xeon Phi coprocessors and builds for the host only. By default, offload constructs are honored and compiled for execution on the coprocessor.
-offload-option, mic, *tool, "option-list"*	Specifies options to be used for the target compilation but not for the host. *tool* may be *compiler*, *ld*, or *as*.

-opt-report-phase *offload*	Generates a report at compile time of variables that will be copied from the host to the coprocessor and vice versa.
-opt-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.
-opt-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.
-opt-threads-per-core=*n*	Hint to the compiler to optimize for *n* threads per physical core, where *n=1*, *2*, *3*, or *4*.
-opt-prefetch=*n*	Enables increasing levels of software prefetching for *n=0* to *4*. Default is *n=3* at optimization levels of –O2 or higher.
-fimf-domain-exclusion=*n*	Specifies special case arguments for which math functions need not conform to IEEE standard. The bits of *n* correspond to these domains: *0*: extreme values (e.g., very large; very small; close to singularities); *1*: NaNs; *2*: infinities; *3*: denormals; *4*: zeros.
-align array64byte *(Fortran only)*	Where possible, align the start of arrays at a memory address that is divisible by 64, to enable aligned loads and help vectorization.

Environment Variables for the Intel® Xeon Phi™ Coprocessor

Variable	Comment
OFFLOAD_REPORT=<*n*>	Provides a runtime 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
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>	Specifies prefix to distinguish environment variables on the coprocessor from ones on the host, for offload applications. For example, 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 http://software.intel.com/en-us/articles/advanced-optimizations-for-intel-mic-architecture.

For building for Intel® Many Integrated Core (MIC) architecture in general, see http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture, http://software.intel.com/mic-developer and the Intel® Compiler User and Reference Guides.

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	**-prof-gen**	Instruments a program for profile generation.

Windows*	Linux*/OS X*	Comment
/Qprof-use	**-prof-use**	Enables the use of profiling information during optimization.
/Qprof-dir *dir*	**-prof-dir *dir***	Specifies a directory for the profiling output files, *.dyn and *.dpi.
/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. See "Profile Function or Loop Execution Time" in the main compiler documentation for additional detail and information on how to view profiles.

Floating-Point Arithmetic Options

Windows*	Linux*/OS X*	Comment
/fp:*name*	**-fp-model *name***	May enhance the consistency of floating-point results by restricting certain optimizations. Possible values of *name*: fast=[1|2]—Allows more aggressive optimizations at a slight cost in accuracy or consistency. (fast=1 is the default) . This may include some additional optimizations that are performed on Intel® microprocessors but not on non-Intel microprocessors. precise—Allows only value-safe optimizations on floating point code. double/extended/source—Intermediate results are computed in double, extended, or source precision. Implies precise unless overridden. The double and extended options are not available for the Intel® Fortran compiler. except—Enforces floating-point exception semantics. strict—Enables both the precise and except options, and does not assume the default floating-point environment. 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.
/Qftz[-]	**-ftz[-]**	When the main program or dll main is compiled with this option, denormals (resulting from Intel® SSE or AVX instructions) at runtime are flushed to zero for the whole program (dll). The default is on except at /Od (-O0).

Windows*	Linux*/OS X*	Comment
/Qimf-precision:*name*	**-fimf-precision:*name***	This option defines the accuracy for math library functions. The 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. Many routines in the math library are more highly optimized for Intel microprocessors than for non-Intel microprocessors.
/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 runtime 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.

For more information, go to: http://software.intel.com/en-us/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. /Qunroll0 (-unroll0) disables loop unrolling. The default is /Qunroll (-unroll), which uses default heuristics.
/Qopt-prefetch:*n*	**-opt-prefetch=*n***	Controls the level of software prefetching. *n* is an optional value between 0 (no prefetching) and 4 (aggressive prefetching), with a default value of 2 when high-level optimization is enabled. Warning: excessive prefetching may result in resource conflicts that degrade performance.
/Qopt-block-factor:*n*	**-opt-block-factor=*n***	Specifies preferred loop blocking factor *n*, the number of loop iterations in a block, overriding default heuristics. Loop blocking is enabled at /O3 (–O3) and is designed to increase the reuse of data in cache.

/Qopt-streaming-stores:*mode*	**-opt-streaming-stores *mode***	Specifies whether streaming stores may be generated. Values for *mode*: always—Encourages compiler to generate streaming stores that bypass cache, assuming application is memory bound with little data reuse never—Disables generation of streaming stores auto—Default compiler heuristics for streaming store generation
/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.

Windows*	Linux*/OS X*	Comment
/Qopt-class-analysis[-]	**-[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. Default is off, but it is turned on by default with the /Qipo (Windows*) or –ipo (Linux* and OS X*) compiler option, enabling improved C++ optimization. (C++ only).
	-f[no-]exceptions	-f-exceptions, default for C++, enables exception handling table generation. -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.
/Qvec-threshold:*n*	**-vec-threshold *n***	Sets a threshold *n* for the vectorization of loops based on the probability of performance gain. $0 \le n \le 100$, default $n=100$. 0—Vectorize loops regardless of amount of computational work. 100—Vectorize loops only if a performance benefit is almost certain.

/Qvec-report:*n*	**-vec-report *n***	Controls the vectorizer's diagnostic levels. Values *n* = *0*, *1*, *2*, *3*, and *6* specify increasing levels of detail. Default is 0 (no report).

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 /O2 (-O2) and makes /Od (-O0) the default 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. Not enabled by /debug:full (-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 runtime. May be used with optimized code. (Fortran applications only)

ǂ 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 Products site at: www.intel.com/software/products/compilers.