Quick-Reference Guide to Optimization with Intel® Compilers version 12

For IA-32 processors and Intel® 64 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**). In this compiler version, all optimization levels assume support for the SSE2 instruction set by default. To run on older IA-32 processors such as the Intel® Pentium® III processor, the option **/arch:IA32** (Windows*) or **–mia32** (Linux*) must be added.

1. Use the general optimization options (Windows **/O1, /O2 or /O3**; Linux and Mac 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. 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.

2. Fine-tune performance to target IA-32 and Intel 64-based systems with processor-specific options. Examples are **/QxSSE4.2 (-xsse4.2)** for the Intel® Core™ processor family, e.g. the Intel Core i7 processor, 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. This option is available for both Intel® and non-Intel microprocessors but it may perform more optimizations for Intel microprocessors than it performs for non-Intel microprocessors. For a more extensive list of options that optimize for specific processors or instruction sets, see the table "Recommended Processor-Specific Optimization Options" ǂ.

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. Optimize your application for vector and parallel execution on multi-threaded, multi-core and multi-processor systems using: advice from the new 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. These optimization steps are applicable to both Intel and non-Intel microprocessors, but may result in a greater performance gain on Intel microprocessors than on non-Intel microprocessors.

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.

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* Mac 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-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* Mac OS* X	Comment
/Qopenmp	-openmp	Causes multi-threaded code to be generated when OpenMP directives are present. May require an increased stack size.
/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 multi-threaded 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).
/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 –parallel (Linux or Mac OS X) or /Qparallel (Windows) 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 causes serialization of code containing Intel® Cilk Plus language extensions. This means that the compiler will run such code as a serial C/C++ program. This option forces inclusion of a special header file (cilk_stubs.h) that includes preprocessor macros that make the Intel Cilk Plus keywords invisible to the compiler. This serialization and all Intel Cilk Plus keywords are fully described in the "Using Intel Cilk Plus" section of the user and reference guide.
/Qcoarray:shared	-coarray=shared	Enables coarrays from the Fortran 2008 standard on shared memory systems (Fortran only). See the compiler reference guide for more options and detail. This option is available for both Intel and non-Intel microprocessors but it may result in more optimizations for Intel microprocessors than for non-Intel microprocessors.

Recommended Processor-Specific Optimization Options‡

Windows*	Linux* Mac 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: **AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2** **Note:** On Mac 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*	**-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* : **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 Mac 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.‡
/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* : **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. **/QaxSSE4.2,SSE3**. 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 Intel Core™ processor family and a default code path optimized for Intel processors or compatible, non-Intel processors that support at least SSE3, use **/QaxSSE4.2 /arch:SSE3 (-axsse4.2 –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 Mac 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 SSE 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.

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

Windows*	Linux* Mac 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 Mac 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.
/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 how to view profiles.

Floating-Point Arithmetic Options

Windows*	Linux* Mac 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 SSE instructions at run time are flushed to zero for the whole program (dll). The default is on except at **/Od (-O0)**.
/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 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.

Fine-Tuning (All Processors)

Windows*	Linux* Mac 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 the 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.
/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 down-casting, it may result in different behavior. Default is off, but it is turned on by default with the **/Qipo** (Windows) or **-ipo** (Linux and Mac 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** ≤ ***n*** ≤ **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. ***n*** specifies the level of detail, from 0 (no report) to 3 (maximum detail). Default is 0.

Debug Options

Windows*	Linux* Mac OS* X	Comment
/Zi	-g	Generates debug information for use with any of the common platform debuggers. Turns off /O2 (-O2) and makes /Od (-O0) the default unless /O2 (-O2) (or another O option) is specified.
/debug[:*keyword]*	-debug [*keyword]*	*keyword* **none** No debugging information is generated (default) **full** (or **all**) produces debugging information for full symbolic debugging of unoptimized code. Same as **–g** (**/Zi**), or as **–debug** (**/debug**) with no keyword. **extended** produces additional information for improved symbolic debugging of optimized code (Linux and Mac OS X only) Debug symbols will generally increase the size of object modules and may slightly degrade performance of optimized code. Implies also –debug full. **parallel** generates additional symbols and instrumentation for debugging threaded code. Does not imply –debug full.

For product and purchase information, visit the Intel® Software Development Products site at:

www.intel.com/software/products/compilers