Quick-Reference Guide to Optimization with Intel® Compilers version 11

For IA-32 processors, Intel® 64 processors and IA-64[1] processors.

Intel® Software Development Products

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* **-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, especially on IA-64[1]-based systems.

2. Fine-tune performance to target IA-32 and Intel® 64-based systems with processor-specific options such as **/QxSSE4.2** (**–xsse4.2**) for the Intel® Core™ processor family, e.g. the Intel® Core™ i7 processor. Alternatively, you can use **/QxHOST** (**-xhost**) which will optimize for and use the most advanced instruction set for the processor on which you compiled. For a complete list of recommended options for specific processors, see the table "Recommended Processor-Specific Optimization Options for IA-32 and Intel® 64 Processors".

3. Use the Intel® VTune™ Performance Analyzer to help you identify performance "hotspots" so that you know which specific parts of your application could benefit from further tuning. The Intel Compilers' optimization reports also help by showing where you might be able to assist the compiler.

4. Add in 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.

5. Optimize your application for simultaneous multithreading, multi-core and multi-processor systems using the parallel performance options **/Qparallel** (**-parallel**) or **/Qopenmp** (**-openmp**), or by using the Intel® Performance Libraries included with the product.

6. Use Intel® Thread Profiler to help you understand the structure of your threaded applications and maximize their performance. Use Intel® Thread Checker to reduce the time to market for threaded applications by diagnosing threading errors and speeding up the development process. Both threading tools work with binary instrumentation. Using the Intel Compiler with source code instrumentation will give you more complete source code information.

Please consult the Compiler Documentation and the *Optimizing Applications with the Intel® C++ & Fortran Compilers* white paper for more details.

[1] IA-64 = Intel® Itanium® Processors

Included in this Guide:

General Optimization Options

Before you begin performance tuning, you may want to check correctness of your application by building it without optimization using **/Od** (**-O0**). Begin performance tuning with **/O1**, **/O2**, or **/O3** (**-O1**, **-O2**, or **-O3**). These are general optimization options that should be at the heart of any application tuning for all 32-bit and 64-bit Intel® processors. Measure your performance before proceeding with more advanced options.

Parallel Performance

For systems with simultaneous multithreading, multiple cores and/or multiple processors, Intel compilers support development of multi-threaded applications through two mechanisms, **/Qparallel** (**-parallel**) or **/Qopenmp** (**-openmp**).

Recommended Processor-Specific Optimization Options for IA-32 and Intel® 64 Architectures

The **/Qx** (**-x** on Linux* or Mac OS* X) switches optimize for, and are recommended for best performance on, the corresponding or later Intel processors. For example, use **/QxSSE4.1** (**-xsse4.1**) for the 45nm Hi-k next generation Intel® Core™ microarchitecture. The **/arch** (**-m** on Linux or Mac OS X) switches are recommended for excellent performance on all processors that support the corresponding instruction set, including processors from AMD, e.g. **/arch:sse3** (**-msse3**) for processors that support SSE3. The **/Qax** (**-ax**) options will produce a binary optimized for the corresponding Intel processor that contains a second, default code path optimized for any processor that supports SSE2 instructions, including those from AMD. This default code path can be modified by the **/Qx** (**-x**) or **/arch** (**-m**) switches, e.g. to produce a binary with a default code path that will execute on older IA-32 processors such as the Intel Pentium® III processor, add the option **/arch:IA32** (Windows) or **–mia32** (Linux).

The options described in the table allow you to make use of all the instructions supported and tune performance for specific Intel processors. As with each previous step, measure the performance benefit of each option to guide your decisions. Use the Intel compiler's optimization reports to assist in determining whether you can provide more help to the compiler to resolve possible dependencies or other issues preventing your loops from being executed in parallel via SSE instructions.

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

IPO includes function-inlining to reduce function call overhead and expose more optimization opportunities. PGO provides runtime feedback to guide optimization decisions about data and code layout to improve instruction-cache efficiency, paging and branch prediction. However, IPO can increase code size. Be sure to measure your execution performance, compile time, and code size tradeoffs with these options. IPO is best used in conjunction with PGO to guide which functions to inline.

Floating-Point Arithmetic Options

The Intel® Compilers provide options for enhancing the consistency or precision of floating-point results on all Intel® architectures, at some cost in performance. Refer to the Compiler Options section of the *Intel® C++ and Fortran Compiler Documentation* for detailed information on floating-point options.

Fine-Tuning (All Processors)

Once you have identified performance hot-spots, you may need to provide the compiler with more information to fine-tune specific functions. The optimization and vectorization reports may show places where loops could not be optimized fully due to pointer aliasing or memory-access overlaps, for example. The *Intel® C++ and Fortran Compiler Documentation* includes details on other #pragmas, directives, and intrinsics that can be used to control software-pipelining, loop unrolling, vectorization, and prefetching for further fine-tuning within your application code.

General Optimization Options

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**).
/Zi	-g	Generates debug information for use with any of the common platform debuggers. This option turns off **/O2** (**-O2**) and makes **/Od** (**-O0**) the default unless **/O2** (**-O2**) (or another **O** option) is specified.
/debug:full	-debug full	Produces full debugging information including symbol table information needed for full symbolic debugging of unoptimized code and global symbol information needed for linking. It produces the largest size object modules. If this option is specified for an application that makes calls to C library routines that will be debugged, the option **/dbglibs** must also be specified to link the appropriate C debug library. If this option is used with optimized code, full symbol information will be generated including the local symbol table information, regardless of the optimization level. This may result in minor performance degradation.

Parallel Performance

Windows*	Linux* Mac OS* X	Comment
/Qopenmp	-openmp	Instructs the parallelizer to generate multi-threaded code when OpenMP* directives are present. May require an increased stack size.
/Qopenmp-report {0|1|2}	-openmp-report {0|1|2}	Controls the OpenMP parallelizer's diagnostic levels. The default level 1 reports loops, regions and sections successfully parallelized.
/Qparallel	-parallel	Detects simply structured loops capable of being executed safely in parallel and automatically generates multi-threaded code for these loops.
/Qpar-report {0|1|2|3}	-par-report {0|1|2|3}	Controls the auto-parallelizer's diagnostic levels as follows: **0** – Displays no diagnostic information. **1** – Indicates loops successfully parallelized (default). **2** – Adds information on loops that were not parallelized. **3** – Adds information about any proven or assumed dependencies inhibiting auto-parallelization (reasons for not parallelizing).
/Qpar-threshold[n]	-par-threshold[n]	Sets a threshold for the auto-parallelization of loops based on the probability of profitable execution of the loop in parallel, **n=0** to **100**. Default: **n=100**. **0** – Parallelize loops regardless of computation work volume. **100** – Parallelize loops only if profitable parallel execution is almost certain. Must be used in conjunction with **/Qparallel** (**-parallel**).
/Qpar-schedule-*keyword*[:n]	-par-schedule-*keyword*[=n]	Specifies scheduling algorithm for parallel DO loops. **n** is the chunk size (in contiguous loop iterations). Possible keywords: **static** allocates predetermined chunks to each thread in turn. **dynamic** allocates fixed chunks to threads dynamically at runtime **guided** divides up loops dynamically into variable chunks of decreasing size, with a minimum chunk size of **n** **runtime** scheduling algorithm and chunk size may be specified by the environment variable OMP_SCHEDULE at runtime
/Qopt-mem-bandwidth<n> (IA-64 only)	-opt-mem-bandwidth<n> (IA-64 only)	Restricts certain optimizations that may increase memory bandwidth requirements for parallel applications. **/Qopt-mem-bandwidth0** (**-opt-mem-bandwidth0**) - no restriction (default for serial compilation) **/Qopt-mem-bandwidth1** (**-opt-mem-bandwidth1**) – restricts optimizations for loops in OpenMP parallel regions (default with **/Qparallel** (**-parallel**) or **/Qopenmp** (**-openmp**)) **/Qopt-mem-bandwidth2** (**-opt-mem-bandwidth2**) - restricts optimizations for all loops. May be useful for MPI or other parallel applications.

Recommended Processor-Specific Optimization Options for IA-32 and Intel® 64 Architectures

Windows*	Linux* Mac OS* X	Comment
/Qx {SSE4.2|SSE4.1|SSSE3|SSE3|SSE2|HOST}	-x {sse4.2|sse4.1|ssse3|sse3|sse2|host}	Processor-specific targeting. Generates specialized code for the indicated Intel processor. The executable should only be run on the targeted or later Intel processors.† **SSE4.2** – May generate SSE4, SSSE3, SSE3, SSE2, and SSE instructions for Intel processors, including SSE4 Efficient Accelerated String and Text Processing. Optimizes for the Intel® Core™ processor family, e.g. the Intel® Core™ i7 processor. **SSE4.1** – May generate SSE4 Vectorizing Compiler and Media Accelerators, SSSE3, SSE3, SSE2, and SSE instructions for Intel processors. Optimizes for Intel® 45nm Hi-k next generation Intel Core™ microarchitecture. **SSSE3** – May Generate SSSE3, SSE3, SSE2, and SSE instructions for Intel processors. Optimizes for the Intel® Core™2 processor family. **SSE3** – May optimize and generate SSE3, SSE2, and SSE instructions for Intel processors. Performs optimizations not enabled with **/arch:SSE3** (**-msse3**). **SSE2** – May optimize and generate SSE2 and SSE instructions for Intel processors. Performs optimizations not enabled with **/arch:SSE2** (**-msse2**). **HOST** – May optimize and generate any instructions that are supported by the compilation host. On Intel processors, this may correspond to the most suitable **/Qx** (**-x**) option; on non-Intel processors, this may correspond to the most suitable **/arch** (**-m**) option. **Note**: On Mac OS X, options SSE3 and SSE2 are not supported.
/arch: {SSE3|SSE2|IA32}	-m {sse3|sse2|ia32} (Linux only)	Generates optimized code that may make use of the specified instruction sets. The executable should only be run on processors supporting the specified instruction sets.† **SSE3** – May Generate SSE3, SSE2, and SSE instructions. Code path may execute on Intel® and non-Intel processors that support SSE3. **SSE2** – May Generate SSE2 and SSE instructions. Code path may execute on Intel® and non-Intel processors that support SSE2. (default) **IA32** – Generates code without any extended instruction sets that will run on any Pentium or later Intel processor or compatible non-Intel processor. [IA-32 architecture only].
/Qax {SSE4.2|SSE4.1|SSSE3|SSE3|SSE2}	-ax {sse4.2|sse4.1|ssse3|sse3|sse2}	Automatic Processor Dispatch. Generates specialized code for the corresponding Intel processors while also generating a default code path. Multiple values, separated by commas, may be used to tune for additional processors in the same executable, e.g. **/QaxSSE4.1,SSE3**. The default code path may be modified by using in addition a **/Qx** (**-x**) or **/arch** (**-m**) switch.† For example, for best performance on the Intel® 45nm Hi-k next generation Intel Core™ microarchitecture and also good performance on an AMD processor that supports only SSE3, use **/QaxSSE4.1 /arch:SSE3** (**-axsse4.1 -msse3** on Linux*). This will produce binaries with two code paths, using automatic processor dispatch technology. One code path will take full advantage of the Intel® 45nm Hi-k next generation Intel Core™ microarchitecture. The other code path will still run well on both Intel and non-Intel processors that support SSE3 but not SSE4. At runtime, the application automatically identifies the Intel processor on which it is running and selects the appropriate code path, either specialized or default. **Note**: On Mac OS X, options **sse3** and **sse2** are not supported.

† The **/arch** (**-m**) option values **SSE3**, **SSE2**, and **IA32** produce binaries that should run on non-Intel processors that implement the same capabilities as the corresponding Intel processors. The corresponding **/Qx** (**-x**) option values perform additional optimizations that are not enabled by **/arch** (**-m**), but will run only on Intel processors.

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.

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[value]	-ipo[value]	Permits inlining and other interprocedural optimizations among multiple source files. The optional **value** argument controls the maximum number of link-time compilations (or number of object files) spawned. Default for ***value*** 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.

Floating-Point Arithmetic Optimizations

Windows*	Linux* Mac OS* X	Comment
/fp:name	-fp-model name	This method of controlling the consistency of floating point results by restricting certain optimizations is recommended in preference to the **/Op** (**-mp**) and **/Qprec** (**-mp1**) switches which are deprecated. The possible values of **name** are: **fast=[1|2]** – Allows more aggressive optimizations at a slight cost in accuracy or consistency. (**fast=1** is the default) **precise** – Enables 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 Intel® Fortran. **except** – Enforces floating point exception semantics. **strict** – Strictest mode of operation, enables both the **precise** and **except** options and disables fma contractions. **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.
/Qfp-speculation *mode*	-fp-speculation *mode*	Enables floating-point speculations with one of the following ***modes***: **fast** – Speculate floating-point operations. (default) **off** – Disables speculation of floating-point operations. **safe** – Do not speculate if this could expose a floating-point exception. **strict** – This is the same as specifying off.
/Qftz[-]	-ftz[-]	When the main program or dll main is compiled with this option, denormal results generated at run time are flushed to zero for the whole program (dll). On IA-64-based systems, the default is off except at **/O3 (-O3)**. On IA-32- based systems and Intel® 64-based systems, the default is on except at **/Od** (**-O0**), but only denormals resulting from SSE instructions are flushed to zero.
/Qfast-transcendentals [-]	-[no-]fast-transcendentals	Enables the use of faster, slightly less accurate versions of math functions such as sin or exp. Default is **/Qfast-transcendentals** (**-fast-transcendentals**) unless **/fp:precise** or **/fp:strict** (**-fp-model precise** or **-fp-model strict**) is specified, when the default becomes **/Qfast-transcendentals-** (**-no-fast-transcendentals**).
/Qfp-relaxed[-] (IA-64 only)	-[no-]fp-relaxed (IA-64 only)	Enables [disables] faster but slightly less accurate code sequences for math functions such as divide and square root. Disabled by default.
/Qprec-div[-]	-[no-]prec-div	Improves precision of floating point divides with a slight impact on speed.
/Qprec-sqrt[-]	-[no-]prec-sqrt	Improves precision of square root computations with a slight impact on speed.

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[-]	-[no-]opt-prefetch	Enables or disables prefetch insertion.
/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++ 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++ only)
/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**.
/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) **ecg_swp** – Gives only the report on the software pipelining component of the Code Generator (Windows* and Linux* on IA-64 only) **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:*rtn*	-opt-report-routine *rtn*	Generates reports only for functions or subroutines whose names contain the string ***rtn***. By default, reports are generated for all functions and subroutines.
/Qvec-report [n]	-vec-report [n]	Controls the vectorizer's diagnostic levels as follows: n = **0**: no information n = **1**: indicates vectorized loops (default) n = **2**: indicates vectorized and non-vectorized loops n = **3**: indicates vectorized loops and explains why other loops were not vectorized

For product and purchase information, visit the Intel® Software Development Products site at:
www.intel.com/software/products/compilers

0607/DAM/OMD/PP/3000 254349-006

Software
Products