Intel® C++ Compiler

Intel® Compilers – Product Summary

Fully support IA-32, Intel® 64, Itanium®, and PCA processors

C/C++ and Fortran 95

Common compiler technology for Windows*, Linux*, IA-32, Intel 64, and Itanium

Microsoft* C++ source, object compatible on Windows

GCC/G++ source, object compatible on Linux* and Mac OS*

IDE integration into Eclipse* for C++ Linux*; MSVC .NET*

Code Coverage and Test Prioritization tools for testing

Advanced optimizations:

- Profile-Guided Optimization
- Inter-Procedural and Whole-Program Optimization
- High Level optimizations and loop transformations
- Auto-vectorization for MMX, SSE, SSE2, SSE3
- OpenMP for Fortran and C++
- Auto-parallelization
- Optimization reports
- Floating Point Model

Intel® Debugger (IDB) and interoperability with GDB* and Etnus* Totalview*

Software

Win with Technology Leadership. Leap Ahead

Intel C++ Compilers 9.1 – Available now

IA-32, Intel® 64 and Itanium® 2 Compilers

- First Intel Compilers to support Mac OS 10.4.3 or higher
- Better Performance at default optimization
- Improved stability, quality
- Dual Core Support
  - Montecito Support – 1st IPF with DC
  - Yonah support – 1st Intel Pentium® M Processor w/ DC

Improved Performance

- axT (-xT) – optimize for Intel Core™ Microarchitecture
- Improved inlining control options (inline-forceinline, etc)
- Compiler Optimization Reports readable by Intel® VTune Analyzer

Latest IDE Integration:

- Support for MSVC* 2005 (Whidbey)
- Eclipse 3.1/CDT 3.0 – supports Intel® Itanium® 2 processor

Cluster OpenMP* – Extends OpenMP to across clusters

Intel Core™ Microarchitecture Optimizations Merom/Conroe/Woodcrest processors

New Option for Core™ Microarchitecture –QxT (-xT) for Windows* (Linux* and Mac OS*)

Auto-generation of new SSSE3 instructions via vectorization

SSSE3 only available on Intel Core™ Microarchitecture

Special compiler tuning for Merom/Conroe/Woodcrest

- Improved instruction scheduling
- Prefetch tuning
- Loop unrolling heuristics tuned

General Optimizations

Linux*	Windows*	
-O0	/Od	Disables optimizations
-g	/Zi	Creates symbols
-O1	/O1	Optimizes for speed without increasing code size (disables library function inlining)
-O2	/O2	Optimizes for speed (default)
-O3	/O3	High-level optimizations

Win with Technology Leadership. Leap Ahead

Advanced Optimizations OpenMP*

Easy multithreading using directives

Use Intel® tools to optimize for IA in tandem with OpenMP

- Fortran and C++ Compilers support OpenMP 2.0
- VTune Analyzer

Linux*	Windows*
-openmp	/Qopenmp
-openmp_report[n]	/Qopenmp_report[n]

OpenMP switches:

```
#pragma omp parallel for
    for (I=0;I<MAX;I++)
        A[I]=c*A[i] + B[i];
```

Loops must meet certain criteria…

Advanced Optimizations
Auto-parallelization

Auto-parallelization: Automatic threading of loops without having to manually insert OpenMP* directives.

Linux*	Windows*
-parallel	/Qparallel
-par_report[n]	/Qpar_report[n]

It is better to use OpenMP directives.

- Compiler can identify "easy" candidates for parallelization, but large applications are difficult to analyze.

Multi-pass Optimization
Profile Guided Optimizations (PGO)

- Use execution-time feedback to guide many other compiler Optimizations
- Helps I-cache, paging, branch-prediction
  - Basic block ordering
  - Better register allocation
  - Better decision of functions to inline
  - Function ordering
  - Switch-statement optimization
  - Better vectorization decision

Multi-pass Optimization
Interprocedural Optimizations (IPO)

ip:	Enables additional interprocedural optimizations for single file compilation
ipo:	Enables interprocedural optimizations across files

Linux	Windows*
-ip	/Qip
-ipo	/Qipo

Enhances optimization when used in combination with other compiler features

Multi-pass Optimization
Interprocedural Optimizations (IPO)

More benefits than just inlining

- Partial inlining
- Interprocedural constant propagation
- Passing arguments in registers
- Loop-invariant code motion
- Dead code elimination
- Helps vectorization, memory disambiguation

Miscellaneous Switches
Floating Point Precision

Switches (Linux*)	Switches (Windows*)	Description
-mp1	/Qprec	Precision closer to - but not quite – ANSI ; faster than ANSI
-prec_div	/Qprec_div	Turn off - division into reciprocal multiply.
-pc*n*	/Qpc*n*	Enable floating-point significand precision control. n={32,64,80}
-fp_port	/Qfp_port	Round fp results at assignments and casts (some speed impact).
-rcd	/Qrcd	Remove code that truncates during float to integer conversions.

Miscellaneous Switches
Other Switches

See *Compiler User's Guide and Reference*

(Linux) icc –help, man icc

(Win) icl /help

http://www.intel.com/software/products

Math Libraries

(Linux) Intel® libimf

(Win) Intel® LIBM

Short vector math library (SVML)

- Used when vectorizing loops that have math functions in them

Automatically used when needed

- LD_LIBRARY_PATH environment variable

Common math functions

- sin/cos/tan/exp/sqrt/log …

Processor dispatch for every IA processor

Libraries on Linux*

-i_dynamic (default)	Link to shared libraries
-static	Link to static libraries
-shared	*Create* a shared object
-Vaxlib (Fortran)	Link to portability library

Compiler Diagnostic – Global Variable Accesses

Problem: Thread legacy code that contains large number of global variables. Need to protect access to globals throughout application.

Intel C++ Compiler has compile time diagnostics to identify when global variable are accessed, available since 9.0 release (2005)

Linux* / Mac OS*: Enabled via -ww1710,1711,1712 –fsyntax-only

Windows*: /Qww1710,1711,1712 /Zs

Can enable each diagnostic separately:

1710 warns about reference to statically allocated variables

1711 warns about assignment to statically allocated variables

1712 warns about address taken of statically allocated variables

10.0: Better C++ diagnostics – Effective C++

The warnings generated with these compiler option are based on the following books from Scott Meyers:

- *Effective C++ Second Edition* - 50 Specific Ways to Improve Your Programs and Designs
- *More Effective C++* - 35 New Ways to Improve Your Programs and Designs

- Enabled via -Weffc++ (/Qeffc++)
- Examples include
  - Use const and inline rather than #define
  - Use <iostream> rather than <stdio.h>.
  - Use new and delete rather than malloc and free
  - Use delete on pointer members in destructors (diagnoses any pointer that does not have a delete)
  - have a user copy constructor and assignment operator in classes containing pointers.
  - Use initialization rather than assignment to members in constructors

  etc