Intel® C++ Compiler 8.1 for Linux Release Notes*

For Intel® IA-32 and Itanium® Processors

Contents

Overview

This product provides tools for Linux* software developers to create applications to run at top speeds on all Intel® IA-32 processors and the Intel® Itanium® processors. Optimizations include support for Intel® Streaming SIMD Extensions 2 (SSE2) in the Intel® Pentium® 4 and Pentium M processors, Intel® Streaming SIMD Extensions 3 (SSE3) in the Intel Pentium 4 processors with SSE3 support, and software pipelining in the Intel Itanium 2 processor. Inter-procedural optimization (IPO) and profile-guided optimization (PGO) can provide greater application performance. Intel® Compilers support multi-threaded code development through autoparallelism and OpenMP* support.

The paper, *Optimizing Applications with the Intel® C++ and Fortran Compilers for Windows* and Linux**, explains how to use the Intel compilers to optimize for the Pentium 4 and Itanium processors and is available at http://www.intel.com/software/products/compilers/ . Additional information on the Intel Software Development Products is available at http://www.intel.com/software/products/ .

Product Contents

Intel® C++ Compiler for IA-32 Based Applications

The Intel® C++ Compiler for IA-32 based applications contains the following components:

- Intel C++ Compiler for Linux for IA-32 applications, version 8.1
- Intel® Debugger for IA-32 applications, version 8.1
- Intel Compiler code-coverage tool
- Intel Compiler test-prioritization tool
- A version of the Eclipse* Integrated Development Environment with C/C++ Development Tools for the Intel C++ Compiler, and associated components

- The product documentation, version 8.1
    - The documentation index is provided for easy access of all the documents. It's located at `<install-dir>/doc/ccompindex.htm`
    - A training tutorial *Enhancing Performance with Intel Compilers* is also included

Intel® C++ Compiler for Itanium®-Based Applications

The Intel C++ Compiler for Itanium-based applications contains the following components:

- Intel C++ Compiler for Linux for Itanium-based applications, version 8.1
- Intel Debugger for Itanium-based applications, version 8.1
- Intel® Itanium Assembler to produce Itanium-based applications, version 7.0
- Substitute headers for use with the Intel C++ Compiler, version 8.1
- Intel Compiler code-coverage tool
- Intel Compiler test-prioritization tool
- The product documentation, version 8.1
    - The documentation index is provided for easy access of all the documents. It's located at `<install-dir>/doc/ccompindex.htm`
    - A training tutorial *Enhancing Performance with Intel Compilers* is also included

What's New in Version 8.1

The following section discusses new features and changes in version 8.1.

`-cxxlib-gcc` Is Now the Default for C++

The STL and gcc* C++ libraries are now used by default when linking C++ applications, rather than those from Dinkumware* used in previous releases. If you wish to use the Dinkumware libraries, specify the new switch `-cxxlib-icc`.

Change in Compiler Drivers

The C compiler command driver, `icc`, determines the language to use based on the filename extension. However, when compiling preprocessed files (`*.i`), the `icc` driver assumes the C language, whereas the C++ driver, `icpc`, assumes the C++ language. It is recommended to use the C++ compiler driver for C++ applications. The C compiler driver (`icc`) no longer links against the C++ libraries, resulting in smaller executables. If your application requires the use of the C++ libraries, please use the C++ compiler driver (`icpc`) instead.

New Predefined Macros

The following new predefined macros are now available:

- `__INTEL_STRICT_ANSI__` specifies that the strict ansi dialect has been selected. The strict ansi dialect is selected when the -strict-ansi option is specified.

This option provides more strict ANSI conformance than the -ansi option, which is intended to be compatible with the gcc -ansi option. This predefined macro should only be used for behavior unique to the -strict-ansi dialect.

- `__INTEL_RTTI__` specifies that RTTI has been enabled for the compilation. The macro is not defined if the -no-rtti option is specified.
- `__INTEL_COMPILER_BUILD_DATE` specifies the build date of the compiler in YYYYMMDD format. It matches the build date shown on the version banner. You can use this predefined macro if you have a need to conditionalize code based on a specific Inte®l compiler update. The YYYYMMDD string is guaranteed to be an increasing integral value with each new release.
- `__INTEL_CXXLIB_ICC` specifies that -cxxlib-icc was present on the compile command line. This may be used in header files dependent on the choice of C++ libraries being used.

`-O3` enables high-level loop and memory optimizations

In this release, specifying -O3 enables additional loop transformations (interchange, distribution, collapsing) and memory access optimizations which can improve performance.

Change in Meaning of `-fast`

As of the 8.1 release, specifying `-fast` implies the following options: `-O3 -ipo -static -xP`

New `-[no-]global-hoist` Optimization Option

`-[no-]global-hoist` is an option that controls certain optimizations, load hoisting and speculative loads, that can move memory loads to a point earlier in the program execution than where they appear in the source. In most cases, these optimizations are safe and can improve performance. The default is `-global-hoist`, enabling the optimizations.

However, some applications, such as those that use shared or dynamically mapped memory, may fail if a load is moved too early in the execution stream (for example, before the memory is mapped.) If you wish to disable these optimizations, specify `-no-global-hoist` when compiling the source files which reference the mapped or shared memory.

`-ipo` Intermediate Language Now Contained Within Object Files

In previous releases, when `-ipo` was used, separate `.il` files were generated to contain intermediate language. The use of these separate files could cause difficulty building with makefiles. In this version, the intermediate language is embedded in the `.o` files and no separate `.il` file is created.

New `-ipo`***`N`*** Option to Create Multiple Objects

In previous versions, when `-ipo` was specified to perform multifile interprocedural optimization, one object file was generated as input to the linker; this is still the default for version 8.1. New in version 8.1 is the ability to request that the compiler create multiple object files for input to the linker; this can, in some cases, reduce link time for large applications. To specify the maximum number of object files to be produced, use the `-ipo`*`N`* form of the option where *`N`* is the maximum number of object files to be created. For example, `-ipo4` specifies a maximum of 4 object files. The compiler may choose to create fewer files than the maximum depending on the application size. If `-ipo0` is specified, the compiler will choose an appropriate number of object files based on the total application size.

`__thread` Keyword for Thread-Local Storage Now Supported

The compiler now supports the gcc-compatible `__thread` keyword for thread-local storage. For details on the usage of this feature, see the gcc documentation.

`-fno-exceptions` is Now Supported for IA-32

The `-fno-exceptions` option turns off exception handling table generation, resulting in smaller code. Any use of exception handling constructs such as try blocks, exception handling specifications, or throw statements will be ignored or produce an error.

A preprocessor symbol `__EXCEPTIONS` is defined when this option is not used. It is undefined when this option is present.

`-fno-exceptions` is not currently supported on Itanium-based systems.

`KMP_SCHEDULE` Environment Variable for OpenMP* Scheduling Control

A new environment variable, `KMP_SCHEDULE`, can be used to fine tune the load balancing of parallel loops that are statically scheduled under OpenMP with no chunk size specification. The default value is `KMP_SCHEDULE="static,greedy"`. This results in (#iterations/#threads) iterations, rounded to the next higher integer, being allocated to most threads, but the final thread(s) may be allocated much fewer, or even zero, iterations. This corresponds to previous compiler behavior. The alternative, `KMP_SCHEDULE="static,balanced"`, results in (#iterations/#threads) iterations, rounded to the next lower integer, being allocated to most threads, with at most one additional iteration being allocated to some threads. Although the largest number of iterations assigned to any thread remains the same, this results in a more even sharing of iterations between threads, which may sometimes lead to a performance improvement.

For example, consider a loop of 9 iterations running on 4 threads:

`KMP_SCHEDULE`	Number of iterations			
	Thread 0	Thread 1	Thread 2	Thread 3
`"static,greedy"`	3	3	3	0
`"static,balanced"`	3	2	2	2

Additional New Options

For information on these options, please see the *New Options* section of the on-disk *Compiler Options Quick Reference Guide*.

- -cxxlib-gcc
- -debug inline_debug_info
- -debug variable_locations
- -debug extended
- -export
- -export_dir
- -fabi-version
- -finline-functions
- -fno-implicit-inline-templates
- -fno-implicit-templates
- -ftls-model
- -g0
- -ipo_separate
- -kernel
- -MP
- -MQ
- -MT
- -Os
- -Qlocation,gas
- -Qlocation,gld
- -reserve-kernel-regs
- -std-gnu89
- -std-gnu++98
- -traceback

Eclipse*

Version 8.1 of the Intel C++ compiler for IA-32 installs the Eclipse Integrated Development Environment (IDE) with C/C++ Development Tools (CDT), and the associated components required to use Eclipse.

What is Eclipse?

Eclipse is an open source software development project dedicated to providing a robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools. It is an extensible, open source IDE. It is platform and language-neutral. Intel Corporation is a member of the Eclipse Foundation, whose web site can be found at www.eclipse.org

What is the CDT?

The CDT (C/C++ Development Tools) Project is working towards providing a fully functional C and C++ Integrated Development Environment (IDE) for the Eclipse platform. The CDT is fully open-source and implemented purely in Java* as a set of plugins to the Eclipse platform.

What versions of Eclipse, CDT and JRE are shipped with Intel® C++ Compilers for 32-bit applications?

Intel C++ Compilers for 32-bit applications are packaged with Eclipse Version 2.1.3, CDT version 1.2.1 and BEA* WebLogic* JRockit* JRE version 1.4.2_04.

I have installed the Eclipse, CDT and JRE along with Intel C++ Compilers for 32-bit applications. How do I launch Eclipse now?

You will need to execute `<INSTALLDIR>/bin/iccec` where `<INSTALLDIR>` is the location where Intel C++ Compilers are installed. By default, `<INSTALLDIR>` is `/opt/intel_cc_80`.

I do not want to install or use the version of Eclipse which is included with Intel C++ Compiler for 32-bit applications. Can I use the Intel compiler with a differentversion of Eclipse?

We do not prevent users from using Eclipse and CDT versions of their choice. However, it should be kept in mind that Intel has tested Intel C++ Compilers for 32-bit applications with Eclipse version 2.1.3 and CDT version 1.2.1 only.

Can I use a different JRE than the one supplied with the Intel C++ Compiler?

Yes. You can use a compatible JRE of your choice. Eclipse requires an appropriate JRE to run/start. Each Eclipse build page (www.eclipse.org) contains a link to a page of suggested Java developer kit and Java runtime downloads that are said to work with the specified Eclipse build. You will need to decide for yourself which JRE works best for you, and which licensing terms work best for you. Please note that Intel has tested the included JRE for compatibility with the Intel C++ package and this combination is supported.

I have installed my own copy of a JRE. How do I use this JRE with the Eclipse that I installed along with the Intel C++ Compiler?

You will need to set the environment variable `OTHER_JVM_BINDIR`. Set the value of the variable `OTHER_JVM_BINDIR` to the full path of the folder of the `java` file from the JRE installed on your system. If you are using the *bash* shell, make sure that you `export` this environment variable.

I already have a functional version of Eclipse, CDT and JRE and I do not want to install another instance of these from the Intel C++ package. Can I still use the Intel C++ Compiler with my already-installed Eclipse, CDT and JRE?.

Yes, you can. You will need to create appropriate files and folders so that the Intel C++ Compiler gets plugged into your pre-installed Eclipse environment. If you want to use the `<INSTALLDIR>/bin/iccec` file (`<INSTALLDIR>` is the location where the Intel C++ Compiler is installed), you will need to make the following changes in the `iccec` file:

1. Set the value of the variable `OTHER_JVM_BINDIR` to the full path of the folder of the `java` file from the JRE installed on your system. If you are using the *bash* shell, make sure that you `export` this environment variable.
2. Set the value of the variable `OTHER_ECLIPSE_BIN` to the full path of the `eclipse` binary in the Eclipse installation folder. If you are using the *bash* shell, make sure that you `export` this environment variable. For example, if you have installed Eclipse in `/opt/intel/eclipse`, then `OTHER_ECLIPSE_BIN` should be set to `/opt/intel/eclipse/eclipse`. (Make sure that this file exists.)
3. Create a folder called `links` under your Eclipse installation folder at the same level as `plugins` and `features`. Create a file called `intel.compiler.cdt.link` in the `links` folder you just created. Put the following line in the `intel.compiler.cdt.link` file:
   ```
   path="<INSTALLDIR>"
   ```
   where `<INSTALLDIR>` is the location where Intel C++ Compilers are installed (By default, `/opt/intel_cc_80`).
4. You can then execute the `iccec` script and your chosen versions of Eclipse, CDT and JRE will be used. If you get an error regarding the loading of libraries, make sure that you set `LD_LIBRARY_PATH` to include the appropriate folder in which Eclipse is installed.

During the process of installation of the Intel C++ Compiler, I chose not to install intel-icc_ide8-8.1-xxx.i386.rpm for "Intel(R) C++ Compiler features and plugins for integration into Eclipse* CDT development environment, Version 8.1". I changed my mind and would now like to install this component. How do I do this?

After installing Intel C++ Compilers on IA-32, if you did not install intel-icc_ide8-8.1-xxx.i386.rpm RPM for "Intel(R) C++ Compiler features and plugins for integration into Eclipse* CDT development environment, Version 8.1", you can install the component in one of two ways:

1. Run the compiler installation program and select option 3 (Eclipse Package) in the main menu. When you choose this option, the installation program will detect that the Intel C++ features and plugins component is not installed and will provide you with an option to install it. Choose to install this component. After the component installation is complete, you can either proceed with the installation of the rest of the Eclipse Package or you can exit out of the installation program.
2. Uninstall the Intel C++ compiler and then reinstall the compiler, selecting the "features and plugins" component. Note that you will then have to reinstall any product updates provided as patches.

Where can I get more information about Eclipse?

The Eclipse Foundation web site is at http://www.eclipse.org/. The Eclipse FAQ contains introductory material and links to online documentation - it can be found at http://www.eclipse.org/eclipse/faq/eclipse-faq.html.

What's New in Version 8.0

New features in the 8.0 release of the Intel® C++ Compiler include optimization support for new Intel processors, improved source and binary compatibility with gcc for C and C++ programs, improved debug support, new code-coverage and test-prioritization tools and several other features driven by user demand.

Compiler Driver Names Changed

If you use the Intel C++ Compiler for Itanium®-based systems, note the compiler driver names for Itanium-based systems have changed from `ecc` to `icc`, and `ecpc` to `icpc` to be consistent with the IA-32 compiler driver names. The old driver names are currently supported, but are deprecated.

New Package Directory Structure

The package directory structure has changed to be compliant with the Filesystem Hierarchy Standard, part of the Linux Standard Base. See http://www.pathname.com/fhs/2.2/fhs-3.12.html for details on the FHS.

Changes to Default Linking Behavior

The compiler configuration files, <install-dir>/bin/icc.cfg and <install-dir>/bin/icpc.cfg, have been modified to remove the RPATH command to the linker. This was used in previous releases to set the location of the Intel shared libraries in the executable. The disadvantage of putting the RPATH directive in the configuration file is that just typing the compiler, icc, would result in a cryptic error:

```
/usr/lib/crt1.o: In function `_start':
/usr/lib/crt1.o(.text+0x18): undefined reference to `main'
```

Which would confuse a large number of users, thinking that there is a problem with their compiler or installation. The current release removes the RPATH directive, and typing icc gives a more informative error:

```
icc: Command line error: no files specified; for help type
"icc -help"
```

However, as executables no longer contain the location of the Intel shared libraries, you need to specify the location of the shared libraries. Any of the following techniques can be used to do this:

1. Modify the LD_LIBRARY_PATH environment variable to contain the location of the Intel shared libraries. For sh type shells, enter: `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<install-dir>/lib`.
2. Use the `ldconfig` system utility. Note: this requires root user privileges to run.
3. Modify the compiler configuration files, `<install-dir>/bin/icc.cfg`

If you see an error message such as this:

```
error while loading shared libraries: libcprts.so.5: cannot
open shared object file: No such file or directory
```

, you need to apply one of the 3 methods listed above to set the location of the Intel shared libraries.

Optimization support for Intel Pentium 4 Processors with SSE3 Instruction Set

A new generation of Intel Pentium 4 processors supports the Streaming SIMD Extensions 3 (SSE3) instruction set, which can improve performance of vectorized loops containing complex data types, float-to-integer conversions, and horizontal adds.

Version 8.0 adds the ability to optimize for Intel processors that support SSE3. To do so, specify the `-xP` or `-axP` options. For further details, please consult the sections on optimizations in the *Intel C++ Compiler User's Guide*.

New IA-32 Optimization Options

This release includes two new code generation options. `-xB` and `-axB` direct the compiler to generate code for best performance on the Intel Pentium M processor. The new `-xN` and `-axN` options enable additional optimizations for all Intel Pentium 4 processors. Intel recommends the use of `-xN` and `-axN` for best performance with Pentium 4 processors, and suggests trying `-xB` or `-axB` to see if it helps your application on the Pentium M processor. For more information, please refer to the sections on optimization in the *Intel C++ Compiler User's Guide*

Obsolete and Obsolescent Optimization Options

The optimization options `-[a]xi` (optimize for Pentium Pro and Pentium II) and `-[a]xM` (optimize for MMX instruction set) are no longer supported by the Intel C++ compiler. If these options are present on the compile command line, an informational message is displayed and the options are ignored. If you use `-[a]xi` or `-[a]xM`, you should discontinue their use. The default is to generate generic code that will run on Pentium processor as well as newer IA-32 processors.

The`-[a]xW` (lower optimization level for Pentium 4) may be removed in a future compiler version. If you use `-[a]xW`, use `-[a]xN` as a replacement when generating code for Intel Pentium 4 processors.

For more information, please refer to the sections on optimization in the *Intel C++ Compiler User's Guide*.

Binary Compatibility with gcc 3.2

The Intel C++ Compiler 8.0 for Linux has a high level of binary compatibility with gcc 3.2.

The new `-cxxlib-gcc` option allows you to build your applications using the C++ run-time provided by gcc. The gcc C++ run-time includes the libstdc++ standard C++ header files, library and language support. When this option is specified, these components are used instead of the libcprts standard C++ headers, library, and libcxa and libunwind C++ language support provided with the Intel Compiler.

When your applications are compiled and linked with the `-cxxlib-gcc` option, the resulting C++ object files, libraries, and executables can interoperate with C++ object files, libraries, and executables generated by gcc 3.2. This means that third party C++ libraries built with gcc 3.2 will work with C++ code generated by the Intel Compiler.

Source Compatibility with gcc 3.2

The predefined macros enabled by gcc are now enabled by the Intel Compiler. For example, the Intel Compiler now defines the `__GNUC__, __GNUC_MINOR__, and GNUC_PATCHLEVEL__` macros. If you do not want these macros to be defined, you can specify the `-no-gcc` option.

Additionally, this version of the Intel C++ Compiler for Linux uses the C headers shipped with the version of Linux you are running on, with the exception of two small substitute headers used only by the Itanium compiler for Itanium applications. Also, the `-cxxlib-gcc` switch now compiles the GNU* C++ library headers if you use the `-ansi` language mode.

By default, the same path (conditional code) in the headers will be used as is used by gcc 3.2, with two exceptions; The Intel Compiler 8.1 pre-defines `-D__NO_INLINE__` and `-D__NO_STRING_INLINES`. Note, these pre-defines have no impact on interoperability with gcc 3.2.

The GNU C++ min/max operators are now implemented. See http://gcc.gnu.org/onlinedocs/gcc/Min-and-Max.html for additional information.

The following new gcc attributes have been implemented:

- noinline
- always_inline
- used

Please see http://gcc.gnu.org/onlinedocs/gcc-3.2/gcc/Attribute-Syntax.html for additional information.

A large number of gcc built-in functions have been implemented in version 8.0 of the Intel C++ Compiler. The gcc built-in functions are documented at http://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html. Refer to the *Intel C++ Compiler User's Guide* for the list of supported gcc built-in functions.

The `-ansi` switch has been updated

The `-ansi` switch has been updated to be compatible with the gcc command line option of the same name. The Intel compiler can support stricter conformance of semantics to ISO C and C++ and this is implemented under the `-strict_ansi` command line option.

GNU environment variables

GNU environment variables that influence the preprocessor are now supported. These include

- `CPATH , C_INCLUDE_PATH , CPLUS_INCLUDE_PATH`
  The value of these variables specifies a list of directories that determines what include directories are searched. The driver would translate these into `-I` options.

- `DEPENDENCIES_OUTPUT , SUNPRO_DEPENDENCIES`
  The value of these variables can be set to a filename or a filename and target name. The driver would translate the filename into -MF filename, and the target name would be used by the compiler when generating the dependency output.

Please see http://gcc.gnu.org/onlinedocs/gcc/Environment-Variables.html for additional information

Precompiled headers

The Intel C++ Compiler for Linux now supports both automatic and manual precompiled header processing. Under the right circumstances, the use of this new feature can dramatically improve compilation time. Please see the User Guide for additional details.

To use automatic precompiled header processing add `-pch` to your compile options. The first compilation may take slightly longer as it creates a `.pchi` file. Subsequent compiles will use the `.pchi` file and can be potentially much faster.

```
> icc -c -pch file1.cpp
"file1.cpp": creating precompiled header file
"file1.pchi"

> icc -c -pch file1.cpp
"file1.cpp": using precompiled header file
"file1.pchi"
```

If disk space is an issue you could reduce the number of precompiled headers to one. To do this you could create a new header file common.h and include all of your important headers as illustrated in the example below.

common.h:

```
#include "myheader1.h"
#include "myheader2.h"
#include <iostream>
```

In each source file include `common.h` followed by `#pragma hdrstop`

file1.cpp:

```
#include "common.h"
#pragma hdrstop
#include "otherheader1.h"
```

file2.cpp:

```
#include "common.h"
#pragma hdrstop
#include "otherheader2.h"
```

Compiling:

```
> icc -c -pch file1.cpp
"file1.cpp": creating precompiled header file
"file1.pchi"

> icc -c -pch file2.cpp
"file2.cpp": using precompiled header file
"file1.pchi"
```

Note that only one precompiled header is created. This produces most of the compile time improvement with a smaller amount of disk usage.

Better debug support for `-ip` and `-ipo` options

Better debug support is now provided for `-ip` and the `-ipo` options. Some information about variables will now be available (although values may not be completely accurate due to optimizations).

New diagnostic switches

Following new diagnostic switches have been made available

- `-Wbrief` print brief one-line diagnostics
- `-Wcheck` enables diagnostics about non-portable constructs
- `-Wp64` enables 64-bit porting specific warnings

IA-32 Floating Point Stack Checker Option`-fpstkchk`

This option would cause extra code to be generated after every function/subroutine call that would assure that the FP stack was in the state the compiler expected. When a customer calls a function that returns an FP value, the FP value is supposed to be returned on the top of the FP stack. If the return value is unused the compiler must pop the value off the FP stack to keep the FP stack in the correct state. However, if the application has called such a function, but either has left out the function's prototype, or incorrectly prototyped the function such that the compiler doesn't know the function is returning an FP value, then the FP stack will not get popped as needed. This tends to cause the FP stack to fill up over time, and eventually overflow. When the stack overflows this generally results in a NAN value being put into FP calculations, and the program's results differ, or other errors may manifest themselves. Unfortunately the point where the errors manifest can be arbitrarily far away from the point of the actual bug. This option will force an access violation exception immediately after such an incorrect call occurred, thus making it very easy for the user to find these issues.

`libguide` can only be dynamically linked

The statically linked library, `libguide`, can potentially cause performance issues that are hard to debug. The compiler will now link `libguide` dynamically regardless of the command line options.

Debug support for variable in registers

More reliable debug object location information are provided with location lists. This allows for better debug support for variables in registers.

Intel® Compilers code-coverage tool

The Intel Compilers code-coverage tool leverages the Intel Compilers profile-guided optimization technology to present developers a complete picture of the coverage of their application code on a particular workload. To find the application's code coverage the tool combines static profile information generated by the compiler with dynamic profile information generated by running the user's instrumented binaries on the workload. The coverage tool uses this information together with the application sources to create HTML pages with color annotations that highlight the coverage of the code. Navigation is through frames that make it particularly easy to sort the application's files and functions and see the least-covered modules and functions. Developers can then use their favorite browser to display the coverage of their code.

The Intel Compilers code coverage tool can be used in a number of ways to improve development efficiency, reduce defects, and improve application performance. When applied to the profile of the application on its test space, the tool can be used to measure the overall quality of testing based on the coverage information. Similarly, when applied to the profile of a performance workload, the code-coverage information indicates how well the workload exercises the application's critical code. High coverage of performance-critical modules is essential to taking full advantage of the profile-guided optimizations that Intel Compilers offer. The tool also provides an option, useful for both coverage and performance tuning, through which the users can display the dynamic execution count of each basic block of the application. Lastly, the coverage tool provides the ability to compare the profile of two different runs of the application. This feature can be used to find the portion of the application's code that is not covered by the application's tests but is exercised when the application is used outside the test space, such as by a customer.

The Intel Compilers code-coverage tool is supported on Intel Architecture 32-bit and the Itanium Processor Family on both Windows and Linux and seamlessly supports C, C++, and Fortran.

Intel® Compilers test-prioritization tool

The Intel Compilers test-prioritization tool leverages the Intel Compilers profile-guided optimizations technology to select and prioritize application's tests based on prior execution profiles of the application. Using this tool, users can select and prioritize the tests that are more relevant for any subset of the application's code. When certain modules of an application are changed, the Intel Compilers test-prioritization tool suggests the tests that are most probably affected by the given change set. The tool mines the profile data from previous runs of the application, discovers the dependency between the application's components and its tests, and uses this information to guide the process of testing. The tool can be used for devising an effective hierarchical testing based on the application's code coverage. For instance, the tool may be used to find the smallest subset of the application tests that achieve exactly the same code coverage as the entire set of tests. The tool can also be used to dramatically reduce the turn-around time of testing. Instead of spending a large amount of time and finding a possibly-large number of failures, the tool may enable the users to quickly find a small number of tests that expose the defects associated with the regressions caused by a change set. The tool offers the potential of significant time saving in testing and development of large-scale applications where testing is major bottleneck. The tool can be used to minimize the number of tests that are required to achieve a given overall coverage for any subset of the application. Moreover, when the execution times of the tests are available, the tool may also be used to select and prioritize the tests to achieve certain level of code coverage in a minimum amount of time.

The Intel Compilers test-prioritization tool is supported on Intel Architecture 32-bit and the Itanium Processor Family on both Windows and Linux and seamlessly supports C, C++, and Fortran.

Please refer to the following link for additional details:
http://www.intel.com/software/products/compilers/techtopics/pgt.htm .

Versioned Intermediate files (`.il`) during interprocedural optimization (IPO)

Each `.il` file generated by IPO will have a version number. The compiler will only accept `.il` files with matching versions. The version numbers will be automatically generated and updated as part of the build process.

Fast Memory Copy Routines

The following is only an issue if you are not linking against the standard Intel libraries, either as a result of specifying `-nostdlib` on the command line or as a result of calling the linker directly rather than from the Intel C++ Compiler driver.

The Intel C++ Compiler uses two routines `_intel_fast_memcpy` and `_intel_fast_memset` to perform memcpy and memset operations that are not macro expanded to `__builtin_memcpy` and `__builtin_memset` in the source code. These are found in `libirc`. If you use the `gcc` compiler to link your application or if you directly call the linker, `ld`, you might find these unresolved symbols. For this reason, Intel recommends using the Intel C++ Compiler for linking, using the same compiler options used during the compilation phase. However, if you see these as undefined externals, either add `-lirc` to your link line, or change your includes so that `memcpy` and `memset` will be macro expanded to the builtin forms and recompile. The Intel C++ Compiler for IA-32 based applications calls a routine `intel_proc_init` from the main routine of any program to ensure that the processor is correctly set up. This routine is also found in `libirc`. These routines used further entry points from `glibc`, so `-lirc` needs to be placed before `-lc` on your command line.

Change in Read-Only Data Behavior

The 7.1 release of the IA-32 compiler placed all constants and string literals in a writeable data section. Starting with 8.0 release, the default behavior will change and will match the `gcc` behavior. Const data will be placed in a read only data section and string literals will be placed in the read-only section as well section. Applications that depend on the old behavior will need to use the new compiler option, `-fwritable-strings`, added in the 8.0 release.

The 7.1 release of the Itanium compiler placed all dimensioned constants and string literals in a writeable data section. Starting with the 8.0 release, the default behavior will change and will match the gcc behavior. In 8.0, dimensioned const data will be placed in a read only data section and string literals will be placed in the read-only section by default. Applications that depend on the old behavior will need to use the new compiler option, `-fwritable-strings`, added in the 8.0 release.

The option `-fwritable-strings` is a gcc compatible option that will cause string literals to be places in a writeable data section. It is provided for backward compatibility for applications that rely on writing to strings.

System Requirements

IA-32 Processor System Requirements

- A computer based on an Intel® Pentium processor or subsequent IA-32 based processor. (Intel Pentium 4 processor recommended).
- 128 MB (256MB recommended).
- 100 MB of disk space, plus an additional 200 MB during installation for the download and temporary files.
- Linux* system with glibc 2.2.4, 2.2.5, 2.2.93 or 2.3.2 and the 2.4.X or 2.6.X Linux kernel as represented by the following distributions. **Note:** Not all distributions listed are validated and not all distributions are listed.
  - Red Hat* Linux 7.3, 8, 9
  - Red Hat Enterprise Linux* 2.1, 3
  - SUSE* Linux 8.2, 9.1
  - SUSE Linux Enterprise Server* 8 or 9
- Linux Developer tools component installed, including gcc, g++ and related tools.

Itanium® Processor System Requirements

- A computer with an Intel® Itanium or Itanium 2 processor.
- 512 MB (1GB recommended).
- 150 MB of disk space, plus an additional 200 MB. during installation for the download and temporary files.
- Linux system with glibc 2.2.4, 2.2.5 or 2.3.2 and the 2.4.X or 2.6.X Linux kernel as represented by the following distributions. **Note:** Not all distributions listed are validated and not all distributions are listed.
  - Red Hat Linux 7.2
  - Red Hat Enterprise Linux* AS 2.1, AS 3, WS 3
  - SUSE Linux Professional 9.1

    - SUSE Linux Enterprise Server 8, 9
    - United Linux* 1.0
- Linux Developer tools component installed, including gcc, g++ and related tools.

We recommend using binutils 2.14 or later, especially if using shared libraries as there are a known issues with binutils 2.11.

Note: Compiling very large source files (several thousands of lines) using advanced optimizations such as `-O3, -ipo` and `-openmp`, may require substantially larger amounts of RAM.

Installation

If you have Intel® C++ Compiler 8.0 or 8.1 installed, you must uninstall it prior to installing this release of Intel C++ Compiler. Please refer to the Uninstalling the Compiler and Debugger below for details on uninstallation.

The installation script of the Intel C++ Compiler uses the system utility RPM to install files. Note, both RPM 4.0.2 and RPM 4.1 have a limitation, please see Known Limitations below for details.

Installing the license

The Intel C++ Compiler uses Macrovision Corporation's FLEXlm* electronic licensing technology. License management is transparent. The installation program of the Intel C++ Compiler 8.1 checks for a valid license before installing any component of the product. Also, the license must remain in place on the system in order to use the Intel C++ Compiler 8.1 to compile and build programs.

The FLEXlm license daemon for Intel software, used for floating and node-locked licenses only, is available for many popular platforms. The daemon may be installed on any supported platform accessible on your local network. The compiler CD contains license daemons for several Linux* distributions. If you do not have the CD, or need a license daemon for an additional platform, you can find all available license daemons in the *Downloads* section of your Intel® Premier Support account.

Note: Your existing license for the Intel C++ Compiler for Linux will work with the 8.1 compiler provided your support services have not expired.

Here is how to setup the license file before installation.

- If you have an electronically downloaded version of the Intel C++ Compiler 8.1, the license will be sent to you via email. Please follow the instructions in the email to install the license file.
- If you have a CD version of the Intel C++ Compiler 8.1, a valid license is included on the CD and the installation program can locate it automatically. But, in order to obtain access to technical support and to be able to download and execute product updates, as **a CD-ROM user** you must do the following:
    1. **Register your product:** First, locate the serial number found on the inside flap of the product box. Then, visit the web site

http://www.intel.com/software/products/registrationcenter/ and follow the instructions. After the registration you will receive an email within 24 hours containing a new license.

2. **Install the new license:** The new license in the email entitles you to one year of support services that allow you to download and execute product updates and obtain full technical support. The email also contains the instructions on how to install the license. Please follow the instructions to finish the new license installation.

Note:
The license file must have an extension "`.lic`".
The default license directory is `/opt/intel_cc_80/licenses/`

.
For details about the support service license, please see http://www.intel.com/software/products/compilers/clin/pricelist.htm .

Installing the Intel® C++ Compiler

Perform the following steps to install the compiler.

1. Download the compiler package.
2. Unpack the compiler package in a directory to which you have write access.
   ```
   > tar -xvf l_cc_p[c]_8.1.xxx.tar
   ```
   or
   ```
   > tar -zxvf l_cc_p[c]_8.1.xxx.tar.gz
   ```
3. Run the installation script
   Become the **root user**, needed to run the rpm command, and execute the install script in the directory where the tar file was extracted.
   ```
   > source ./install.sh
   ```
   If you do not have access to the root account, it is possible to install the compiler without root access by manually unpacking the RPM files with `rpm2cpio` and editing the `iccvars.sh` (`.csh`) files to include the directory where the compiler is installed. The install script automates this procedure.
4. Enter the directory for the license file
   It is the directory where you saved the license file (`*.lic`) above. The installation program will validate the license before installing any Intel C++ Compiler for Linux component.
5. After the license checking, the installation program will display the Intel software products that are already installed, and the following menu items for you to install:
   - `Intel Compiler for 32-bit applications` if you're installing on an IA-32 system or
     `Intel Compiler for Itanium architecture` if you're installing on an Itanium-based system
   - `Linux Application Debugger for 32-bit applications` or
     `Linux Application Debugger for Itanium®-based applications`
   - `Eclipse Package`

6. Select a package to install. All necessary packages needed to use the product will also be installed. If an RPM package has already been installed, the install script will report this and ask you to verify that you want to overwrite the existing installation. The default RPM options `-U --replacefiles --force` are recommended to force the update of existing files. The default installation directory is `/opt/intel_cc_80/` for the Intel C++ Compiler, and `/opt/intel_idb_80/` for the Intel Debugger.
7. After installation, the Intel packages installed will be redisplayed, followed by a redisplay of the install menu. Enter `'x'` to exit the install script.

Note: Installing the compiler also installs the Eclipse integration components (but not Eclipse itself, which is installed when you select Eclipse Package). These components are installed into `<install_dir>/eclipse/features` and `<install_dir>/eclipse/plugins`. Because of this, you will also be asked to agree to the Eclipse Disclaimer during compiler installation.

Setting Up The Compiler Environment

The programs in the Intel C++ Compiler 8.1 for Linux product rely on the environment variables PATH and LD_LIBRARY_PATH. The installation script `(install.sh)` creates compiler environment script files `(iccvars.sh/ idbvars.sh)` that set these variables. It is strongly recommended that you add those script files into your login script `(.login` file). Once the variables are set in the `".login"` file there is no need to run the script files for each session.

Source the script to setup the compiler environment:

- `> source <install-dir>/bin/iccvars.sh(.csh)`
  to use icc on an IA32 system or Itanium-based system
- `> source <install-dir>/bin/idbvars.sh(.csh)`
  to use idb on an IA32 system or Itanium-based system

The installation program also creates compiler configuration files named `<install-dir>/bin/icc.cfg` on an IA32 system or an Itanium-based system that contain common settings for all compilations. You can edit these files to add additional default options. **Note**, if you install a compiler update package, you need to save the configuration file if you have modified it to another filename so that the installation doesn't overwrite your modified file.

Please register for support after you install this product. See Technical Support for registration instructions.

Uninstalling the Compiler and Debugger

Please follow the steps below to uninstall the Intel Compiler and Debugger.

1. become the root user
2. to uninstall the compiler:
   `<compiler-install-dir>/bin/uninstall.sh`

or if you've installed the compiler to the default directory, use
`/opt/intel_cc_80/bin/uninstall.sh`

3. to uninstall the debugger:
   `<debugger-install-dir>/bin/uninstall.sh`
   or if you've installed the debugger to the default directory, use
   `/opt/intel_idb_80/bin/uninstall.sh`

Known Limitations

Installation Warning for RPM 4.0.2 and RPM 4.1

RPM 4.0.2 cannot install to a non-default directory. This has been resolved in RPM 4.0.3. RPM 4.1 cannot install to a non-default directory. This has been resolved in RPM 4.11 to 4.2.

Note about installing the Intel® Debugger

When installing the Intel® Debugger version 8.1 for IA-32 or Itanium-based applications from the Intel® C++ Compiler 8.1 package, if the Intel Debugger version 7.1, 7.2 or 7.3 is already installed on the system, it will be upgraded to the Intel Debugger version 8.1.

OpenMP* Limitations

POSIX threaded programs that require a large stack size may not run correctly on some versions of Linux* because of hard-coded stack size limits in some versions of the Linux POSIX threads libraries. These limits also apply to OpenMP programs (-openmp) and automatically generated parallel programs (`-parallel`) with the Intel compilers, because the Intel compilers use the POSIX threads library to implement OpenMP based and automatically generated parallelism. Threaded programs that exceed the stack space limit usually experience segmentation violations or addressing errors.

To avoid these limitations, use a version of glibc built with the `FLOATING_STACKS` parameter defined. For some distributions, this implies using the shared rather than the static version of the pthreads library. Then use the `ulimit -s` or `limit stacksize` command to set the maximum shell stack size to an explicit large value, in units of KBytes, (not `unlimited`), and also set the `KMP_STACKSIZE` environment variable to the needed thread stacksize in bytes. Note, in the bash shell, `ulimit -s` can be used to set a large maximum stack size only once. In the C shell (csh), `limit stacksize` , with no dash before the argument, can be used to reset the maximum stacksize repeatedly.

This solution has been tested on glibc version 2.2.4-13 for IA-32 and glibc 2.2.4-19 for the Itanium Processor Family as found in the Red Hat 7.2 Linux distribution. For glibc 2.2.4-13 on IA-32, the shared version of the POSIX threads library must be used, (there should not be a `-static` flag in the compiler .cfg file or on the command line).

Compile time slow down when using both `-g` and inlining

There will be an increase in compile time when `-g` is used together with inlining. Inlining can happen if the user specifies `-ipo, -ip` or compiles a `C++/C99` program at option levels

`-O1` or above. This is due to the generation of debug information. For many applications, this combination of compiler options will not increase compile time or compile-time memory use.

gnu asm aliases

Under the `-use_msasm` compilation flag, GNU asm aliases will work only if you use the `__asm__` keyword; they will not work correctly if you use the alternate `__asm` or `asm` keywords.

Issues relating to Multiple Object File Interprocedural Optimization

The following issues are expected to be resolved in a future update:

`-qipo_separate` is not recognised by `xild`

The `-qipo_separate` option is not recognized by `xild`. This causes IPO compilations using this option to fail.

Explicit naming of .o and .s files ignored with `-ipo` multiple objects

When using `-ipo_c` or `-ipo_S` (explicit .o or ..s files, respectively), options to explcitly name these files are ignored by the compiler for when generating multiple objects.

.s files generated by `-ipo_S` fail to assemble with multiple object IPO

There are two classes of errors:

- The assembler complains that routines being called haven't been defined.
- The assembler complains that constant data (e.g. strings in `printf` statements) hasn't been defined. This bug affects Itanium-based Linux systems only.

Limited Debug Information with Automatic CPU Dispatching (`-ax*`)

Compilation using `-ax{W|N|B|P}` results in two copies of generated code for each function. One for IA-32 generic code and one for CPU specific code. The symbol for each function then refers to an Auto CPU Dispatch routine that decides at run-time which one of the generated code sections to execute. Debugger breakpoints that are set on these functions by name cause the application to stop in the dispatch routine. This may cause unexpected behavior when debugging. This issue may be addressed in a future version of the Intel Debugger and Compilers.

Cannot Debug or View Traceback for IA-32 Programs Built Without `-fp`

Compilation using `-fp` specifies that the IA-32 EBP register be used as a frame pointer rather than a general purpose register. Debuggers and traceback handlers may not be able to properly unwind through a stack that contains a call to a function that is compiled without `-fp` in effect. If you compile with `-g` or `-O0`, `-fp` is implicitly enabled, but not if you specify a higher optimization level explicitly (such as `-O2`). If you intend to use the debugger or traceback on an application, and are using some level of optimization higher than `-O0`, you should also specify `-fp` to ensure that the debugger and traceback handler can use frame pointers.

GNU Assembler May Not Recognize `-xP` Generated Code

Older versions of the GNU Assembler may not be able to process assembly code generated by compiling with the `-[a]xP` option. Use binutils version 2.14.90.0.4.1 or later, or FSFbinutils 2.15 or later if this is an issue for you.

Using Older `gdb` Versions with Intel® Compilers

Intel compilers for Linux generate Dwarf2-format debugging information, including several advanced features in Dwarf2 such as declarations nested within classes. Older `gdb` debuggers, such as version 5.3.90-*, are sometimes unable to correctly handle these Dwarf features. For best success on source code which uses the full expressiveness of the C++ language, please consider using `gdb` version 6.1 or newer.

Other Issues

Please click on the appropriate link below to see additional notes and known limitations in the latest version of the compiler.

- Intel C++ Compiler to produce IA-32 applications. Note, this file is available only if the compiler for IA-32 applications is installed.
- Intel C++ Compiler to produce Itanium-based applications. Note, this file is available only if the compiler for Itanium-based applications is installed.

Technical Support

Your feedback is very important to us. To receive technical support for the tools provided in this product and technical information including FAQ's and product updates, you need to be registered for an Intel® Premier Support account on our secure web site, https://premier.intel.com. Please register at http://support.intel.com/support/performancetools/support.htm and click on "`Registration Center`".

Note:

- Registering for support varies for release products or pre-release products (alpha, beta, etc) - only released products have support web pages on http://support.intel.com.
- If you are having trouble registering or are unable to access your Premier Support account, contact developer.support@intel.com. Please do not email your technical issue to developer.support@intel.com as it is not a secure medium.

- If you have forgotten your password, please email a request to: quad.support@intel.com. Please do not email your technical issue to this email address as it is not a secure medium.

For information about the Intel C++ Compiler's Users Forums, FAQ's, tips and tricks, and other support information, please visit: http://support.intel.com/support/performancetools/c/linux/. For general support information please visit http://www.intel.com/software/products/support/.

Submitting Issues

Steps to submit an issue:

1. Go to https://premier.intel.com/.
2. Type in your Login and Password. Both are case-sensitive.
3. Click the "`Submit`" button.
4. Read the Confidentiality Statement and click the "`I Accept`" button.
5. Click on the "`Go`" button next to the "`Product`" drop-down list.
6. Click on the "`Submit Issue`" link in the left navigation bar.
7. Choose "`Development Environment (tools,SDV,EAP)`" from the "`Product Type`" drop-down list.
8. If this is a software or license-related issue, choose "`Intel C++ Compiler, Linux*`" from the "`Product Name`" drop-down list.
9. Enter your question and complete the fields in the windows that follow to successfully submit the issue.

Guidelines for problem report or product suggestion:

1. Describe your difficulty or suggestion.
   For problem reports please be as specific as possible, so that we may reproduce the problem. For compiler problem reports, please include the compiler options and a small test case if possible.
2. Describe your system configuration information.
   Get the version of glibc and kernel with following commands:
   ```
   > uname -a
   > rpm -qa | grep glibc
   ```
   If you don't have `rpm` installed, use the command below:
   ```
   > ls /lib/libc*
   ```
   And copy the information into the corresponding Premier Support fields.

   Get the Intel C++ Compiler's Package ID with the following commands:
   ```
   > icc -V
   ```
   And copy the "Package ID" (e.g. **`l_cc_p[c]_8.1.xxx`**) from the output into the corresponding Premier Support field. Please include any other specific information that may be relevant to helping us to reproduce and address your concern.

3. If you were not able to install the compiler or cannot get the Package ID, enter the filename you downloaded as the package ID.

Resolved Issues

Please review `<package ID>_README` (e.g. `l_cc_p[c]_8.1.xxx_README`), available for download from Intel® Premier Support, https://premier.intel.com/, to see which issues have been resolved in the latest version of the compiler.

Documentation

You can view the Intel® compiler and related HTML-based documentation with your Web browser, which provide full navigation, index look-up, and hyperlink capabilities. The documents also have PDF versions for easier printing.

The documentation is installed in the `<install-dir>/doc` (default `/opt/intel_cc_80/doc`) directory. An HTML index document can be found at `<install-dir>/doc/ccompindex.htm` (default `/opt/intel_cc_80/doc/ccompindex.htm`). An interactive HTML-based training tutorial *Enhancing Performance with Intel Compilers* is also available from links in the documentation index. This provides a tutorial on using compiler options that help you optimize your application for IA-32 and Itanium-based systems and describes the Itanium Assembler. *The Intel® Debugger Manual* can be found from the Intel® Debugger directory (default `/opt/intel_idb_xx/doc` (xx: is the idb version number, run "`idb -V`" for the version)).

The document *Intel® C++ Compiler User's Guide* is now organized into separate parts:

- An Options Quick Reference Guide
- User's Guide Volume I for Building Applications
- User's Guide Volume II for Optimizing Applications
- Reference Information

For information on the GNU glibc C language library, documentation can be obtained from the Linux OS vendor or from the GNU web site, www.gnu.org.

Viewing Manpages

The `icc`(1) manpage provides a list of command-line options and related information for the `icc` and `icpc` compiler commands. To display the `icc`(1) manpage, type the following command after you set up your environment by using a source command to execute the `<install-dir>/bin/iccvars.*sh` file:

```
$ man icc
```

The `man` command provides single keys or key combinations that let you scroll through the displayed content, search for a string, jump to a location, and perform other functions. For example, type the **`z`** to view the next screen or **`w`** to view the previous screen. To obtain help about the man command, type the **`h`** key; when you are done viewing help, type the **`q`** key to return to the displayed manpage. To search, type **`/`** character followed by the search string

(**`/string`**) and press Enter. After viewing the man command text, type **`q`** to return to the shell command prompt.

Viewing HTML Documentation

To view the compiler *User's Guide* HTML-based documentation, you no longer need to use a Java*-enabled Web browser. The documentation format has been tested to work with Web browsers shipped on standard Red Hat* distributions. To allow the HTML-based *User's Guide* help to be browser-neutral and not require java support, the Search capability has been removed. If you need to search the *User's Guide*, please use the supplied *User's Guide* PDF files (same content as HTML-based *User's Guide*) with the xpdf viewer. To effectively use the Index tab, you may need to enlarge the left pane so that index entries do not wrap.

Viewing PDF Documentation Files

You can read the PDF files using the `xpdf` utility (provides search capability), or use the `gv` or `ghostview` command. On some Linux distributions, using `mozilla` will display PDF files using a PDF helper.

Additional Information

Related Products and Services

Information on Intel software development products is available at http://www.intel.com/software/products.

Some of the related products include:

- The Intel® Software College provides training for developers on leading-edge software development technologies. Training consists of online and instructor-led courses covering all Intel architectures, platforms, tools, and technologies.
- The VTune™ Performance Analyzer enables you to evaluate how your application is utilizing the CPU and helps you determine if there are modifications you can make to improve your application's performance.
- The Intel® C++ and Fortran Compilers are an important part of making software run at top speeds with full support for the latest Intel IA-32 and Itanium® processors.
- The Intel® Performance Library Suite provides a set of routines optimized for various Intel processors. The Intel® Math Kernel Library, which provides developers of scientific and engineering software with a set of linear algebra, fast Fourier transforms and vector math functions optimized for the latest Intel Pentium® and Intel Itanium processors. The Intel® Integrated Performance Primitives consists of cross-platform tools to build high performance software for several Intel architectures and several operating systems.

Copyright and Legal Information