White Paper

Om P Sachan
Developer Products Division

Intel® Compilers for Linux*: Compatibility with GNU Compilers

Abstract

This paper describes the compatibility between Intel® C++ Compiler 11.0 Professional Edition for Linux* and GNU compilers in terms of source, binary, and command-line compatibility. The Intel® C++ and Fortran Compilers help make your software run at top speed on Intel platforms, including the IA-32, Intel® 64, and IA-64 architectures. The compilers also provide compatibility with commonly used Linux software development tools.

Table of Contents

Abstract 1
Table of Contents 3
Introduction 4
Overview 5
C and C++ Source Compatibility 6
Linux* Kernel Build 6
OpenMP* Source Compatibility and Interoperability 8
Binary Compatibility 9
Linking and Libraries 15
  Linking C Language with Intel® Compiler and gcc Compiler 15
  Intel® Compiler Linking Conventions 18
  Optimized Math Function Library 19
  Build Environment Enhancements 19
Command Line Options 20
Intel® Fortran Compiler 22
Customer Feedback 22
Conclusion 23
Appendix A: Linux Distribution Support Information 23
References 24

Introduction

Intel® C++ Compiler 11.0 Professional Edition for Linux provides excellent source, binary, and command-line compatibility with the GNU GCC and g++ compilers. Intel C++ Compiler for Linux now only generates C++ code that is binary compatible with g++ and requires a Linux distribution with a released version of g++ 3.2 or higher. In addition, compared to previous versions, Intel C++ Compiler for Linux has improved compiler option compatibility with GCC and includes general improvements related to the GCC build environment.

The motivation for improved compatibility comes from customer requests in several areas:

- Mixing and matching binary files created by g++, including third-party libraries
- Generating C++ code compatible with g++ 3.2 or higher
- Building the Linux kernel with the Intel Compiler with fewer workarounds
- Improved support for command-line options offered in the GNU compilers

Overview

Intel C++ Compiler for Linux supports the ANSI and ISO C and C++ standards, most GNU C and C++ language extensions, and the OpenMP* 3.0 standard. Intel® C++ Compiler 11.0 supports some features that will be in the future C++0x standard, consult the compiler documentation for details. Intel C++ Compiler for Linux is binary or object-file compatible with C and C++ binary files created with GNU GCC and g++ versions 3.2, 3.3, 3.4, 4.0, 4.1, 4.2 and 4.3.

Building the Linux kernel with Intel C++ Compiler is an ongoing project at Intel. The goal is to improve the GCC source compatibility of the Intel C++ Compiler, and to find opportunities to improve kernel performance.

Intel Corporation and Red Flag Software Co., Ltd, announced that Red Flag is the first company to use the Intel C++ Compiler for Linux to compile a commercial version of its Linux operating system. Details of this announcement are available at http://www.intel.com/pressroom/archive/releases/20040803net.htm.

Intel® Fortran Compiler 11.0 for Linux supports the Fortran 95 and OpenMP* 3.0 standards. Intel® Fortran Compiler 11.0 supports some features from the Fortran 2003 standard, consult the compiler documentation for details.

Intel Fortran Compiler for Linux is not binary compatible with GNU g77 or GNU gfortran compiler, nor is binary compatibility a future goal for the Intel Fortran Compiler. Intel Fortran Compiler for Linux is binary compatible with C-language object files created with either Intel C++ Compiler for Linux or the GNU GCC compiler.

Intel C++ and Fortran Compilers support a large number of the commonly used GNU compiler command-line options. See the Command-Line Options section of this document for details.

We encourage customers to submit feature requests for additional compatibility through their Intel® Premier Support accounts. See *Customer Feedback* below.

C and C++ Source Compatibility

Intel C++ Compiler supports the ANSI and ISO C and C++ standards. The *Intel® C++ Compiler* documentation provides information on supported C99 features. Intel C++ Compiler for Linux also supports most GNU C and C++ language extensions. Excellent source compatibility allows the definition of the following GCC predefined macros:

- **`__GNUC__`**
- **`__GNUC_MINOR__`**
- **`__GNUC_PATCHLEVEL__`**
- **`__GNUG__`**

The definition of these macros greatly reduces the amount of compiler-specific code, including the system header files. These macro definitions can be disabled using the -no-gcc option, although using this option is discouraged as system header files may not compile correctly. A new option has been added, -gcc-sys, which is similar to -no-gcc, except that the GNU macros are only defined when preprocessing system include headers files, so these will compile correctly. Alternatively, the following macro is defined by Intel C++ Compiler for Linux and can be used to conditionally compile code:

- **`__INTEL_COMPILER`**

GNU inline-assembly format has been supported on IA-32 processors since Intel C++ Compiler 6.0 for Linux. The Intel Compiler also supports GNU inline-assembly for Intel® 64 processors (previously referred to as Intel EM64T). The Intel C++ Itanium Compiler provides intrinsic functions that provide equivalent inline assembly functionality without inhibiting compiler optimizations important on IA-64. Intel C++ Compiler implements a large number of GCC built-in functions, which are documented at http://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html. See the compiler documentation for information on intrinsic functions and supported GCC built-in functions.

Table 1 summarizes GNU C-language extensions supported by Intel C++ Compiler 8.1, 9.1, 10.0 and 11.0, highlighting continuous source compatibility improvements. Table 2 shows supported GNU C++ language extensions. Note that some C extensions are not supported when compiling C++ source files. The GCC C and C++ language extensions are documented in the GCC manual, available at http://gcc.gnu.org.

Linux* Kernel Build

Intel C++ Compiler for Linux has excellent support for GNU C-language extensions. As a demonstration of compatibility with GCC, Intel C++ Compiler for Linux has successfully built and run Linux kernels on IA-32, Intel® 64, and IA-64 using a limited number of temporary source patches. The temporary nature of these patches will be addressed in the future, either by adding extensions to the Intel C++ Compiler or by working with the Linux community to replace or reduce usage of less frequently used, non-standard language features in the kernel sources.

GCC C Language Extension	Intel C++ Compiler 8.1	Intel C++ Compiler 9.1	Intel C++ Compiler 10.0	Intel C++ Compiler 11.0
Statement Expressions	Yes	Yes	Yes	Yes
Locally Declared Labels	Yes	Yes	Yes	Yes
Labels as Values	Yes	Yes	Yes	Yes
Nested Functions	No	No	No	No
Constructing Function Calls	No	No	No	No
Naming an Expression's Type	Yes	Yes	Yes	Yes
Referring to a Type with Typeof	Yes	Yes	Yes	Yes
Generalized Lvalues	Yes	Yes	Yes	Yes
Conditionals with Omitted Operands	Yes	Yes	Yes	Yes
Double Word Integers	Yes	Yes	Yes	Yes
Complex Numbers	Yes	Yes	Yes	Yes
Hex Floats	Yes	Yes	Yes	Yes
Arrays of Zero Length	Yes	Yes	Yes	Yes
Arrays of Variable Length	Yes	Yes	Yes	Yes
Macros with Variable Number of Arguments	Yes	Yes	Yes	Yes
Looser Rules for Escaped Newlines	No	No	No	Yes
Strings Literals with Embedded Newlines	Yes	Yes	Yes	Yes
Non-Lvalue Arrays Subscripts	Yes	Yes	Yes	Yes
Arithmetic on Void Pointers	Yes	Yes	Yes	Yes
Arithmetic on Function Pointers	No	Yes	Yes	Yes
Non-Constant Initializers	Yes	Yes	Yes	Yes
Compound Literals	Yes	Yes	Yes	Yes
Designated Initializers	Yes	Yes	Yes	Yes
Cast to Union Type	Yes	Yes	Yes	Yes
Case Ranges	Yes	Yes	Yes	Yes
Mixed Declarations and Code	Yes	Yes	Yes	Yes
Function Attributes	Most	Yes	Yes	Yes

Table 1. GCC C-Language Extensions Supported in the Intel® C++ Compiler for Linux*

GCC C Language Extension	Intel C++ Compiler 8.1	Intel C++ Compiler 9.1	Intel C++ Compiler 10.0	Intel C++ Compiler 11.0
Prototype and Old-Style Function Definitions	No	No	No	No
C++ Style Comments	Yes	Yes	Yes	Yes
Dollar Sign in Identifier Names	Yes	Yes	Yes	Yes
The ESC Character in Constants	Yes	Yes	Yes	Yes
__alignof__ (types, variables)	Yes	Yes	Yes	Yes
Attributes of Variables	Most	Yes	Yes	Yes
Inline Function as Fast as a Macro	Yes	Yes	Yes	Yes
Inline ASM (IA-32)	Yes	Yes	Yes	Yes
Controlling Names Used in ASM Code	Yes	Yes	Yes	Yes
Variables in Specified Registers	Yes	Yes	Yes	Yes
Alternate Keywords	Yes	Yes	Yes	Yes
Incomplete enum Types	Yes	Yes	Yes	Yes
Function Name as Strings	Yes	Yes	Yes	Yes
Getting the Return or Frame Address of a Function (IA-32)	Yes	Yes	Yes	Yes
Other Built-in Functions	Most	Yes	Yes	Yes
Using Vector Instructions Through Built-in Functions	No	No	No	No
Built-in Functions Specific to Particular Target Machines	No	No	No	No
Pragmas Accepted by GCC	No	No	Yes	Yes

Table 1 [Con't]. GCC C-Language Extensions Supported in the Intel® C++ Compiler for Linux*

GCC C++ Language Extensions	Intel C++ Compiler 11.0
Minimum and Maximum Operators in C++	Yes
When Is a Volatile Object Accessed?	No
Restricting Pointer Aliasing	Yes
Vague Linkage	Yes
Declarations and Definitions in One Header	No
Where's the Template?	External Template Supported
Extracting the Function Pointer from a Bound Pointer to Member Function	Yes
C++-Specific Variable, Function, and Type Attributes	Yes
Java* Exceptions	No
Deprecated Features	No
Backward Compatibility	No

Table 2. GCC C++ Language Extensions Supported in the Intel® C++ Compiler for Linux*

OpenMP* Source Compatibility and Interoperability

Intel compilers include two sets of OpenMP libraries:

- The Compatibility OpenMP libraries, which provide compatibility with OpenMP support provided by certain versions of the GNU* compilers, as well as the Intel compiler version 10.x (and later).
- The Legacy OpenMP libraries, which provide compatibility with OpenMP support provided by Intel compilers, including those prior to version 10.0.

To select the Compatibility (default) or Legacy OpenMP libraries, use the Intel compiler to link your application and specify the Intel compiler option **`-openmp-lib`**.

The term ***object-level interoperability*** refers to the ability to link object files and libraries generated by one compiler with object files and libraries generated by the second compiler, such that the resulting executable runs successfully. In contrast, ***source compatibility*** means that the entire application is compiled and linked by one compiler, and you do not need to modify the sources to get the resulting executable to run successfully.

Different compilers support different versions of the OpenMP specification. Based on the OpenMP features your application uses, determine what version of the OpenMP specification your application requires. If your application uses an OpenMP specification level equal to, or less than the OpenMP specification level supported by all the compilers, your application should have source compatibility with all compilers, but you need to link all object files and libraries with the same compiler's OpenMP libraries. You may consult OpenMP compatibility manual http://www.astro.uio.no/ita_lokal/data/manualer/Intel_fortran/pdf/345900_openmp_compa_rtl.pdf for more information.

Binary Compatibility

Intel® C++ Compiler 11.0 Professional Edition provides C and C++ binary compatibility with released versions of GCC including 3.2, 3.3, 3.4, 4.0, 4.1,4.2 and 4.3. Since Release 8.1, the default C++ library implementation is the GCC provided C++ libraries. New in release 11.0, C++ code is always binary compatible with g++–due to this, a Linux system with a released version of GCC 3.2 or higher is required. Previous versions of Intel C++ Compiler for Linux had an alternative C++ library provided by Intel with the **`–cxxlib-icc`** option. This did not provide C++ binary compatibility with GCC due to differences in the C++ library implementation. This option, **`–cxxlib-icc`**, had been deprecated and is no longer available as of version 10.0. The benefit is that C++ code is now always binary compatible with supported versions of g++.

Intel C++ Compiler for Linux supports the C++ ABI, a convention for binary object code interfaces between C++ code and the implementation-provided system and runtime libraries. The GCC 3.x and 4.x compilers also conform to the C++ ABI, allowing Intel C++ Compiler to be binary compatible with released versions of g++ 3.2 or higher, with slight differences to account for minor ABI changes in g++. Consult http://gcc.gnu.org/releases.html for information on the latest GCC releases. The goal is the implementation of a stable ABI for C++ applications and libraries, which is a benefit to the Linux community.

The Intel C++ Compiler 11.0 uses the C++ runtime provided by GCC. The GCC C++ runtime includes the **`libstdc++`** standard C++ header files, library, and language support. By default, C++ code generation is compatible with the GCC version in your **`PATH`** environment variable. The compiler option **`–gcc-name`** allows specifying the full-path location of GCC. Use this option when using a GCC with a non-standard installation.

The **-gcc-version=<version>** option allows great flexibility to generate code for the supported GCC versions:

- **-gcc-version=320** for gcc 3.2.x compatibility
- **-gcc-version=330** for gcc 3.3.x compatibility
- **-gcc-version=340** for gcc 3.4.x compatibility
- **-gcc-version=400** for gcc 4.0.x compatibility
- **-gcc-version=410** for gcc 4.1.x compatibility
- **-gcc-version=420** for gcc 4.2.x compatibility
- **-gcc-version=430** for gcc 4.3.x compatibility

Future product versions may include compatibility options for more recent versions of g++, including g++ 4.3. Contact Intel through your Intel® Premier Support account for additional information.

The following examples demonstrate binary compatibility with g++ and the usage of the g++ runtime libraries. Figure 1 shows building the "Hello World" example program.

```
prompt> cat hello.cxx
   #include <iostream>
   int main(){ std::cout<<"Hi"<<std::endl; }
prompt> icpc hello.cxx
```

Figure 1. "Hello World" example showing the default runtime libraries

Figure 2 shows the default runtime libraries linked against applications on a Intel® 64 system, including the g++ provided C++ runtime libraries. The g++ libraries are the standard GNU C++ library, **libstdc++**, and C++ language support library, **libgcc_s**.

The system utility **ldd** is used to show the dynamic libraries linked to the application, and the **awk** utility is used to make the output easier to read in this paper.

```
prompt>  ldd a.out | awk '{print $1}'
libm.so.6
libstdc++.so.6
libgcc_s.so.1
libc.so.6
libdl.so.2
/lib64/ld-linux-x86-64.so.2
```

Figure 2. Default runtime libraries linked against "Hello World" example program

The example in Figure 3 shows C++ binary compatibility with g++ by mixing binary files created by g++ and Intel C++ Compiler.

```
prompt> cat main.cxx
void pHello();
int main() { pHello(); }
prompt> cat pHello.cxx
#include <iostream>
void pHello()
{ std::cout << "Hello" << std::endl; }
prompt> icpc -c main.cxx
prompt> g++ -c pHello.cxx
prompt> icpc main.o pHello.o
prompt> ./a.out
Hello
```

Figure 3. Mixing binary files created by g++ and Intel® C++ Compiler

Figure 4 shows how to mix binary files created by g++ and Intel C++ Compiler, using g++ to link. Intel recommends linking with the Intel Compiler to correctly pass the Intel libraries to the system linker, **`ld`** On IA-32 and Intel® 64 processors, Intel C++ Compiler calls the function **`__intel_new_proc_init`** from the main routine to determine the ability to run processor-specific code. This routine is found in the Intel library **`libintlc`**. To link this example correctly, **`libintlc`** needs to be added to the link command. The compiler for IA-64 requires linking in the additional library **`libipr`** via the option **`-lipr`** for g++ interoperability support. Different compiler optimizations, such as OpenMP* and vectorization, may require additional libraries provided by Intel C++ Compiler.

Linking with the Intel Compiler removes the necessity of knowing the details of which Intel libraries are required, provided the same compiler options are used when compiling and linking. Without passing the correct libraries and library location, the initial link fails. For the example in Figure 4, linking with the Intel **`libintlc`** library is required.

```
prompt> icpc -c main.cxx
prompt> g++ -c pHello.cxx

# Fails without Intel provided libraries
prompt> g++ main.o pHello.o
main.o: In function `main':
main.o(.text+0xd): undefined reference to
 `__intel_new_proc_init'
collect2: ld returned 1 exit status

# Links with g++ and explicitly list
# Intel provided libraries in the default
# library location for release 10.0.014
prompt> g++ main.o pHello.o -L /opt/intel/
Compiler/11.0/027/lib/in
tel64/ -lirc -lcxaguard
prompt> ./a.out
Hello
```

Figure 4. Example of linking using g++

Linking and Libraries

Linking C Language with Intel® Compiler and GCC Compiler

C-language object files can be linked with either Intel Compilers or GCC compilers. Linking with the Intel Compiler is recommended, as the Intel libraries will be correctly passed to the linker. The examples shown in Figure 5 through 10 uses GCC to link the application with the file **`main.c`**, compiled with GCC, and the file **`calcSin.c`**, compiled with the Intel C++ Compiler for IA-32 applications and optimized for the Intel® Core™ 2 Duo processor. The paper, *Optimizing Applications with the Intel® C++ and Fortran Compilers*, available at http://www.intel.com/software/products/compilers/clin, describes advanced compiler optimizations available with the Intel Compiler.

Figure 5 uses the Intel Compiler to automatically vectorize the loop in **`calcSin.c`** when the C99 restrict keyword is used and enabled via command-line option.

```
prompt> cat calcSin.c
#include <math.h>
void calcSin(double *a,double * restrict b,
int N){
  int i;
  for (i=0; i<N; i++)
    b[i] = sin(a[i]); }
prompt> icc -c  -xSSSE3  calcSin.c -restrict
calcSin.c(4): (col. 3) remark: LOOP WAS
VECTORIZED.
```

Figure 5. Compiling C language function with the Intel® C++ Compiler, using the restrict command line option to enable usage of the C99 keyword restrict

The main function is compiled with GCC in Figure 6.

```
prompt> cat main.c
void calcSin(double *a,double *b,int N);
 int main() {
  const int N=100000;
  double a[N], b[N], c[N], x[N];
  int i;
  for (i=0;i<N;i++)
    a[i] = i;
  for (i=0;i<100;i++)
    calcSin( a, b, N); }
prompt> gcc -c main.c
```

Figure 6. Main function compiled with GCC

The application is linked with GCC, and the necessary libraries from the Intel Compiler are passed to GCC. In the example in Figure 7, the short vector math library, **libsvml**, and the optimized math function library provided by Intel, **libimf**, are required to link.

```
prompt> gcc main.o calcSin.o -L /opt/intel/
Compiler/11.0/027/lib/intel64/
-lsvml -limf  -o calcSin
```

Figure 7. Linking with GCC using additional Intel provided libraries: libsvml and libimf

The **nm** utility can determine the location of unresolved symbols in the Intel libraries. If the Intel libraries are not passed to GCC, the link fails with undefined references, as shown in Figure 8.

```
prompt> gcc main.o calcSin.o
calcSin.o: In function `calcSin':
calcSin.c:(.text+0xb6): undefined reference
to `__svml_sin2'
calcSin.c:(.text+0x10c): undefined reference
to `__svml_sin2'
calcSin.c:(.text+0x135): undefined reference
to `__svml_sin2'
calcSin.c:(.text+0x16b): undefined reference
to `__svml_sin2'
calcSin.c:(.text+0x194): undefined reference
to `sin'
collect2: ld returned 1 exit status
```

Figure 8. Failed link due to missing libraries when linking with GCC

Running the **nm** utility on the Intel libraries helps determine which libraries are required. See the Intel C++ and Fortran User's Guides, library section for documentation on the different Intel libraries. Next, the **grep** utility (Figure 9) verifies that the symbols are defined in the Intel libraries. Examine the symbol file, **icc-symbols.txt**, to determine which library contains the missing symbol and add this to the link options.

```
prompt> nm /opt/intel/Compiler/11.0/027/lib/
intel64/* > icc-symbols.txt

prompt> grep "__svml_sin2" icc-symbols.txt

0000000000000030 T __svml_sin2
```

Figure 9. Utilities to determine libraries required to link with GCC correctly

Link with the Intel Compiler, passing the same options (**-xT** in Figure 10) used during compilation, to avoid the necessity of finding out which Intel supplied libraries are required.

```
prompt> gcc -c main.c

prompt> icc -c -xSSSE3  calcSin.c

prompt> icc -xSSSE3 calcSin.o main.o
```

Figure 10. Avoiding missing libraries when linking by using Intel® C++ Compiler with the same options as used during compilation

Intel Compiler Linking Conventions

This section describes conventions used by Intel Compilers for Linux, and compiler options for changing the default behavior. The **icc** compiler driver for the C language does not link the C++ runtime libraries when compiling C applications. The **icc** compiler driver links the C++ runtime libraries if the input source files have C++ file extensions. The **icpc** compiler driver is meant to be used for C++ files, and it automatically links in the C++ runtime libraries, similar to the behavior of g++.

By default, Intel Compilers for Linux use Dynamic Shared Object (DSO) versions, also known as shared libraries, of the Linux system libraries. For the Intel provided libraries, by default the DSO version of **libcxaguard**, the g++ compatibility support library is used and static versions of all other Intel libraries are used. For user-provided libraries, a DSO version is searched for first. If no DSO version is found, a static version is searched for.

The following command line options modify the default behavior:

- **-static**: Link in all libraries statically, and create a statically linked executable.
- **-Bstatic**: Use the static version of all libraries specified after this point or until the option -**Bdynamic** is used. This option can be used to statically link all libraries.
- **-Bdynamic**: Use dynamic (**DSO**) version of all libraries specified after this point or until the -**Bstatic** option is used. Note that -**Bstatic** and -**Bdynamic** are toggles.
- **-static-intel**: Statically link all compiler libraries provided by Intel. This option can avoid the need to redistribute the libraries with your application.
- **-shared-intel**: Dynamically link all compiler libraries provided by Intel. In other words, use **DSO** versions of Intel libraries.
- **-shared**: Instructs the linker to create a **DSO** instead of an application binary.

  Note: **-shared** is not the opposite of the **-static** option.

The **ldd** utility lists the DSOs that an application is linked with, and is useful in understanding the command-line options described previously.

Optimized Math Function Library

The math function library, **`libimf`**, is an optimized math library provided with the Intel Compilers for Linux. The default Linux math library, **`libm`**, contains functions in addition to those provided in the math function library. The Intel Compilers first link to the **`libimf`** library, then to the **`libm`** library. If an optimized function is available, it will be found in the math-function library, **`libimf`**. Otherwise, it is linked from the Linux math library, **`libm`**.

Note: Since Intel C++ Compiler for Linux version 9.0, the compiler always links to functions in the optimized **`libimf`** library if available, regardless of whether the user specifies **`–lm`** on the link command line or if the user adds both **`libimf`** and **`libm`**, regardless of the order.

Build Environment Enhancements

Intel C++ Compiler continues to improve compatibility with the GCC build environments. We have enhanced the ability to use the GNU C++ library on systems with non-standard GCC configurations. The Intel Compiler installation supports traditional install with root account access using rpm, as well as non-root account install that does not use the rpm package manager.

The following GNU environment variables are supported:

- **`CPATH`**: Path to include directory for C/C++ compilations.
- **`C_INCLUDE_PATH`**: Path include directory for C compilations.
- **`CPLUS_INCLUDE_PATH`**: Path include directory for C++ compilations.
- **`DEPENDENCIES_OUTPUT`**: If this variable is set, its value specifies how to output dependencies for make based on the non-system header files processed by the compiler. System header files are ignored in the dependency output.
- **`GCC_EXEC_PREFIX`**: This variable specifies alternative names for the linker (**`ld`**) and assembler (**`as`**).
- **`LIBRARY_PATH`**: The value of **`LIBRARY_PATH`** is a colon-separated list of directories, much like **`PATH`**.
- **`SUNPRO_DEPENDENCIES`**: This variable is the same as **`DEPENDENCIES_OUTPUT`**, except that system header files are not ignored.
- **`GXX_INCLUDE`**: Specifies the location of the GCC headers. Set this variable only when the compiler cannot locate the GCC headers when using the **`-gcc-name`** option.
- **`GXX_ROOT`**: Specifies the location of the GCC binaries. Set this variable only when the compiler cannot locate the GCC binaries when using the **`-gcc-name`** option..

Command Line Options

Intel C++ Compiler for Linux supports a large number of common GNU compiler command-line options. Table 3 lists the GCC compiler options for which support has been recently added in the Intel C++ Compiler for Linux. Due to the large number of supported options, the full list of supported options is not included in this document. Information on Intel C++ command options can be found in the compiler documentation, compiler main pages, and summary information via **icc –help**.

In addition to the recently added GCC options listed in Table 3, the following options have been added and deal with GNU compatibility. The option **-gcc-sys** is similar to **-no-gcc** (do not predefine the **__GNUC__**, **__GNUC_MINOR__**, and **__GNUC_PATCHLEVEL__** macros), except that the GNU macros are only defined when preprocessing system include headers files, so these will compile correctly. The option **-pragma-optimization-level=[Intel|GCC]** enables / disables processing **#pragma optimize** using Intel (default) or GCC syntax.

The meaning of the **-ansi** switch was changed starting with Intel C++ Compiler 8.1, 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++. This support is implemented by using the **-strict-ansi** command-line option.

Note: the Intel C++ and Fortran Compilers have a large number of features to optimize applications for the latest Intel processors. The paper "*Optimizing Applications with the Intel C++ and Fortran Compilers*," available at http://www.intel.com/software/products/compilers/clin > Evaluation Center > White Papers, explains how to use Intel Compilers to optimize for the latest Intel processors.

Intel encourages customers to submit feature requests for command-option compatibility through their Intel Premier Support account. See *Customer Feedback* below.

Compiler Command Linux* Option	Description
-f[no-]keep-static-consts	Enable/disable (DEFAULT) emission of static const variables even when not referenced
-trigraphs	Support ISO C trigraphs; also enabled in ANSI and C99 modes
-fno-rtti	Disable RTTI support
-fsyntax-only	Perform syntax and semantic checking only (no object file produced)
-fvisibility-inlines-hidden	Do not mark inline member functions as hidden
-dD	Same as **-dM**, but output **#define directives** in preprocessed source
-dN	Same as **-dD**, but **#define directives** contain only macro names
-fargument-alias	Same as **-alias-args**
-fargument-noalias	same as **-alias-args-**
-fargument-noalias-global	arguments do not alias each other and do not alias global storage
-fdata-sections	same as -ffunction-sections
-f[no-]exceptions	enable(DEFAULT)/disable exception handling
-ffunction-sections	separate functions for the linker (COMDAT)
-finline	inline functions declared with **__inline**, and perform C++ inlining
-fno-inline	Do not inline functions declared with **__inline**, and do not perform C++ inlining
-finline-functions	inline any function, at the compiler's discretion (same as **-ip**)
-finline-limit=<n>	set maximum number of statements a function can have and still be considered for inlining

Table 3. Recently added GCC compiler command-line options for Intel® C++ Compiler for Linux*

Compiler Command Linux* Option	Description
-f[no-]math-errno	set **ERRNO** after calling standard math library functions
-fno-builtin	disable inline expansion of intrinsic functions
-fno-builtin-<func>	disable the **<func>** intrinsic
-fno-gnu-keywords	do not recognize '**typeof**' as a keyword
-fno-operator-names	disable support for operator name keywords
-f[no-]omit-frame-pointer	negative version same as **-fp**
-fpack-struct	pack structure members together
-fpermissive	allow for non-conformant code
-freg-struct-return	return struct and union values in registers when possible
-ftemplate-depth-<n>	control the depth in which recursive templates are expanded
-funroll-loops	unroll loops based on default heuristics
-I-	any directories you specify with '**-I**' options before the '**-I-**' options are searched only for the case of '**#include** "**FILE**"; they are not searched for '**#include** <**FILE**>'
-imacros <file>	treat **<file>** as an **#include** file, but throw away all preprocessing while macros defined remain defined
-iprefix <prefix>	use **<prefix>** with **-iwithprefix** as a prefix
-iquote <dir>	add directory **<dir>** to the front of the include file search path for files included with quotes, but not brackets
-iwithprefix <dir>	append **<dir>** to the prefix passed in by **-iprefix** and put it on the include search path at the end of the include directories
-iwithprefixbefore <dir>	similar to **-iwithprefix** except the include directory is placed in the same place as **-I** command line include directories
-malign-double	same as **-align**
-msse	generate code for Intel® Pentium® III and compatible Intel processors (IA-32 only)
-msse2	generate code for Intel processors that support SSE2 extensions (IA-32 only, enabled by default on Intel®64)
-msse3	generate code for Intel processors that support SSE3 extensions
-mtune=<cpu>	optimize for a specific CPU
-W[no-]abi	warn if generated code is not C++ ABI compliant (DEFAULT)
-W[no-]deprecated	print warnings related to deprecated features
-Winline	enable inline diagnostics
-W[no-]comment[s]	warn when /* appears in the middle of a /* */ comment
-W[no-]main	warn if return type of main is not expected
-W[no-]missing-prototypes	warn for missing prototypes
-W[no-]pointer-arith	warn for questionable pointer arithmetic
-W[no-]return-type	warn when a function uses the default int return type and warn when a return statement is used in a void function
-W[no-]uninitialized	warn if a variable is used before being initialized
-W[no-]unknown-pragmas	warn if an unknown **#pragma** directive is used (DEFAULT)
-W[no-]unused-function	warn if declared function is not used
--version	display GCC style version information

Table 3 [Con't]. Recently added GCC compiler command-line options for Intel® C++ Compiler for Linux*

Intel Fortran Compiler

Intel Fortran Compiler supports the Fortran 95 and OpenMP* 3.0 standards. Intel Fortran Compiler for Linux is not binary compatible with the GNU g77 or gfortran compiler, nor is this a future goal. In general, Fortran compilers are not binary compatible, due to using different runtime libraries. Intel Fortran Compiler is binary compatible with C-language object files created with either Intel C++ Compiler for Linux or the GNU GCC compiler. The *Intel Fortran Compiler* documentation has further details on calling C language functions from Fortran.

Intel Fortran Compiler for Linux uses a different name-mangling scheme than the GNU Fortran compiler. Intel does not recommend mixing object files created by the Intel Fortran Compiler and the GNU Fortran compiler.

Customer Feedback

Intel is committed to providing compilers that deliver the highest Linux-based application performance for applications running on platforms that use the latest Intel processors. Intel Premier Support is included with every compiler purchased; see http://www.intel.com/software/products/compilers for more information.

Intel strongly values customer feedback and is interested in suggestions for improved compatibility with the GNU compilers. If your applications require additional compatibility features, please submit a feature request by creating a customer-support issue through your Intel Premier Support account that explains your request and its impact.

Developers should register for an Intel Premier Support account to obtain technical support and product updates. The product-release notes describe how to register.

Conclusion

Intel C++ Compilers for Linux provide outstanding source-language, binary, and command-line compatibility with the GNU C and C++ Compilers. Intel encourages users to submit feature requests for compatibility enhancements. Many enhancements have been made based on customer feedback. This compatibility gives significant advantages to developers by increasing flexibility while enabling software to run at top speeds on IA-32, Intel® 64, and IA-64 systems.

Appendix A: Linux Distribution Support Information

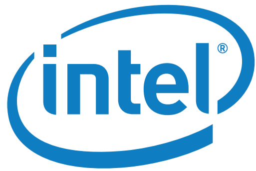
The following table lists Linux OS support information for recent releases of the Intel C++ Compiler for Linux. Supported are IA-32, Intel® 64, and Itanium® processor. Consult the product release notes for the most recent product information.

Note: Intel C++ Compiler for Linux version 11.0 requires a Linux distribution with a released gcc 3.2 or higher.

Intel® C++ Compiler for Linux*	Processor Architecture	Supported glibc versions	Supported kernel versions	Supported g++ versions
11.0	IA-32	2.3.2, 2.3.4, 2.3.5, 2.8	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.2.x, 4.3.x
11.0	Intel® 64	2.3.2, 2.3.4, 2.3.5, 2.8	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.2.x, 4.3.x
11.0	Itanium® Architecture	2.3.2, 2.3.3, 2.3.4, 2.5	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.2.x, 4.3.x
10.0	IA-32	2.3.2, 2.3.3, 2.3.4, 2.3.5	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x, 4.2.x
10.0	Intel® 64	2.3.2, 2.3.3, 2.3.4, 2.3.5	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x, 4.2.x
10.0	Itanium® Architecture	2.3.2, 2.3.3, 2.3.4	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x, 4.2.x
9.1	IA-32	2.2.5, 2.3.2, 2.3.3, 2.3.4	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x
9.1	Intel® 64	2.3.2, 2.3.3, 2.3.4	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x
9.1	Itanium® Architecture	2.2.4, 2.3.2, 2.3.4	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x
8.1	IA-32	2.2.4, 2.2.5, 2.3.2	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x
8.1	Intel 64	2.2.4, 2.2.5, 2.3.2	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x
8.1	Itanium Architecture	2.2.4, 2.2.5, 2.3.2	2.4.x	3.2.x, 3.3.x, 3.4.x

References

- General information on Intel® software development tools, including the Intel C++ and Fortran Compilers: http://www.intel.com/software/products
- Intel® C++ Compiler for Linux* documentation is available at http://www.intel.com/software/products/compilers/clin
- Documentation, application notes, and source code examples: http://developer.intel.com/software/products/opensource/
- The OpenMP standard: http://www.openmp.org
- ANSI C and C++ standards: http://www.ansi.org
- The GNU project including GNU GCC and gfortran compilers and glibc, the GNU C library: http://www.gnu.org
- Conventions for object code interfaces between C++ code and implementation-provided system and libraries: http://www.codesourcery.com/cxx-abi/
- Press release from Intel Corporation and Red Flag Software Co., Ltd, announcing that Red Flag is the first company to use the Intel C++ Compiler for Linux to compile a commercial version of its Linux operating system: http://www.intel.com/pressroom/archive/releases/20040803net.htm



For product and purchase information visit:

www.intel.com/software/products

Intel, the Intel logo, Intel. Leap ahead. and Intel. Leap ahead. logo, Pentium, Intel Core, and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.

Copyright © 2008, Intel Corporation. All Rights Reserved.

0505/DXM/OMD/PDF 300348-002