



Building Linux* Kernel with Intel® C++ Compiler for Linux

White Paper
Feilong Huang

Developer Products Division



Introduction

Intel® C++ Compilers have been in the market for over 10 years. More and more software developers are interested in using it to extract better performance on Windows* and Linux*.

As the most essential part of a Linux operating system, the Linux kernel is highly-optimized by kernel developers. Additionally there are many GNU C Language extensions, programming tricks and inline assembly code. This makes it challenging for compilers to further optimize the kernel. Building the Linux kernel with the Intel C++ Compiler (icc) is an ongoing project at Intel. The goal is to improve gcc source compatibility with the Intel C++ Compiler and to find opportunities to improve kernel performance.

Red Flag* Software Co., Ltd uses the Intel C++ Compiler for Linux to compile the Linux kernel in its commercial version of Linux operating system. Details of this are available at <http://www.intel.com/pressroom/archive/releases/20040803net.htm>

While this is not a how-to guide, this document provides extensive hints for the readers to build the Linux kernel using the Intel C++ Compiler.

Compiler Options Compatibility

icc supports most gcc options, but not all of them, therefore, unsupported options must either be replaced with icc equivalent options, or removed. Most of them are not critical and can be removed without breaking the Linux kernel. For example, the following unsupported options can be removed without any risks.

```
-fno-unit-at-a-time  
-msoft-float  
-gstabs  
-pipe
```

Regarding the following unsupported options in icc 10.x, we must invoke appropriate compilers, either icc for IA-32 or icc for Intel 64, to compile the source file.

```
-m32  
-m64
```

Ignoring the following options may break the Linux kernel. Replacing them with icc equivalent options is a must.

```
-mno-mmx  
-mno-sse2  
-mno-sse3
```

Table 1 summarizes some gcc options and support in Intel C++ Compiler 10.1 and 11.0.

Table 1. Some gcc Options and Support in Intel® C++ Compiler for Linux*

gcc Options	Intel C++ Compiler 10.1	Intel C++ Compiler 11.0
-f[no-]unit-at-a-time	No	No
-frename-registers	No	No
-msoft-float	No	No
-gstabs	No	No
-mfixed-range	Yes ¹	Yes ¹
-mregparam	Yes	Yes
-m32	No	Yes
-m64	No	Yes

¹ Partially implemented.

The Intel C++ Compiler is stricter with syntax checking and reports more warnings than the GNU compiler. As a result, the `-Werror` option may cause the compilation to stop due syntax errors, and therefore should be removed.

A simple wrapper script to remove or replace unsupported compiler options and invoke the Intel C++ Compiler can simplify the compiler command line needed to compile the Linux kernel with Intel C++ Compiler. In the example provided here, environment variables `HOSTCC` and `CC` will need to be set to the name of the wrapper script.

```
make menuconfig
make HOSTCC=<name of wrapper> CC=<name of wrapper>
make modules_install
```

For an example wrapper script, see Appendix A.

Source Compatibility

The Intel C++ Compiler does not support a few gcc's builtin functions and IA-64 inline assembly code. Workarounds are needed.

- `__builtin_va_arg()`

The Intel C++ Compiler for IA-64 defines `va_arg` macro in `<icc installation dir>/include/xarg.h`, however, compilation with `-nostdinc` inhibits inclusion of this header and instead causes `icc` to use the following GNU definition of `va_arg`:

```
#define va_arg(v,l) __builtin_va_arg(v,l)
```

Since `icc` does not support `__builtin_va_arg`, the `-nostdinc` option must be removed.

- `__builtin_offsetof()`

The Intel C++ Compiler does not allow a variable subscript in the builtin function `__builtin_offsetof()` as shown in the following example.

```
struct C
{
    int arr[1];
};
void foo(int i)
{
    __builtin_offsetof(struct C, arr[i]);
}
```

A possible workaround is to add the following line at the end of `include/linux/compiler-intel.h`.

```
#undef __compiler_offsetof
```

We expect to support variable subscripts for `__builtin_offsetof()` in an upcoming product update.

- Inline assembly code

The Intel C++ Compiler supports inline assembly code on IA-32 and Intel 64. IA-64 compilers do not support inline assembly. Instead intrinsics that are C-like functions are recommended. Assembly code on IA-64 needs to be rewritten using corresponding intrinsics. The Intel C++ Compiler documentation includes a mapping of assembly instructions to intrinsics. Most of these changes have been checked into the Linux kernel source tree.

Linux Kernel Source Defects

Some Linux kernel source defects were observed during compilation of the Linux kernel with the Intel C++ Compiler. These defects may have been fixed in the newer Linux kernel already.

- volatile attribute

Look at the following code snippet from `include/asm-ia64/spinlock.h`

```
# define _raw_spin_lock(x) \
do { \
    __u32 *ia64_spinlock_ptr = (__u32 *) (x); \
    __u64 ia64_spinlock_val; \
    ... \
    if (unlikely(ia64_spinlock_val)) { \
        do { \
            while (*ia64_spinlock_ptr) \
                ia64_barrier(); \
            ... \
        } while (ia64_spinlock_val); \
    } \
}
```

```
    } while (0)
```

In the above code snippet, `ia64_spinlock_ptr` points to a 32-bit volatile data in memory. Without a “volatile” keyword here, the compiler may generate the following asm code (shown in pseudo code) for the while loop, which is legal:

```
        load ia64_spinlock_ptr, register
label: test register
        jump-if-not-zero label
```

Unfortunately, the above code results in a dead lock of the Linux kernel because the 32-bit data pointed to by `ia64_spinlock_ptr` is not reloaded. The GNU compiler occasionally generates the “right” code, which is what kernel developers want:

```
label: load ia64_spinlock_ptr, register
        test register
        jump-if-not-zero label
```

In this case, a “volatile” attribute is needed for the variable `ia64_spinlock_ptr`, to make sure other compilers do not fail.

- `inline` keyword

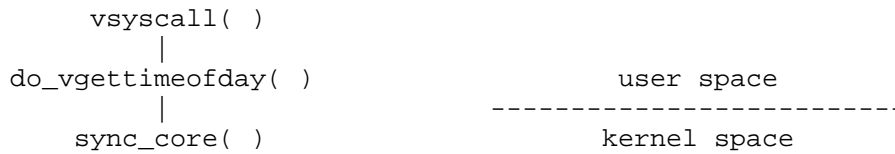
The `inline` keyword is just a hint to the compiler. Compilers may or may not inline an function declared as `inline`. For example, in some applications `gettimeofday()` is done very often like when the kernel is timestamping all transactions. It would help performance if it could be implemented with very low overhead.

One way of obtaining a fast `gettimeofday()` is by writing the current time in a fixed place on a page mapped into the memory of all applications and updating this location on each clock interrupt. These applications could then read this fixed location with a single instruction - no system call required.

There might be other data that the kernel could make available in a read-only way to the process, like perhaps the current process ID. A `vsyscall` is a “system” call that avoids crossing the userspace-kernel boundary.

`vsyscall()` and `do_vgettimeofday()` are in a special page, which can be accessed in user mode.

The Intel C++ Compiler doesn’t inline the function `sync_core()`, which is marked as an inline function in `include/asm-x86_64/processor.h`. Thus, the function is compiled as a separate function in the kernel image. `vsyscall()` calls `do_vgettimeofday()` and `do_vgettimeofday()` calls `sync_core()`. The first two functions are called by user applications while `sync_core()` is a kernel function. This will cause a page fault. The following illustrates the call-graph of these 3 functions.



gcc inlines `sync_core`. As a result, gcc avoids the page fault. To fix this source defect, adding a `always_inline` attribute to `sync_core()`.

Conclusion

Intel® C++ Compiler is highly compatible with the GNU Compiler. We've successfully compiled Linux kernels 2.4.21, 2.6.9 and 2.6.18 with the Intel C++ Compilers on IA-32, Intel® 64 and IA-64 with a small wrapper script and a limited number of temporary source patches. Since newer Linux kernels may have fewer source compatibility issues, we recommend that you build newer Linux kernels with the latest Intel C++ Compiler.

Additional Information and Links

Intel® Compilers for Linux*: Compatibility with GNU Compilers

Appendix A: An Example Wrapper Script

The following is an example of wrapper script.

```
#-----  
# This is a wrapper script for icc 10.1 on IA-32  
#-----  
ARGS=$@  
ICCARGS="-fno-builtin"  
  
# For loop to change options of icc  
for ARG in $@  
do  
case $ARG in  
-fno-stack-protector  )  
    ICCARGS="$ICCARGS -fno-stack-security-check"  
    ;;  
  
-Wno-pointer-sign | -msoft-float | -ffreestanding )  
    ;;  
  
* )  
    ICCARGS="$ICCARGS $ARG"  
    ;;  
esac  
done  
  
icc $ICCARGS  
  
exit $?
```



For product and purchase information visit:
www.intel.com/software/products

Intel, the Intel logo, Intel. Leap ahead. and Intel. Leap ahead. logo, Pentium, 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.