A look at the Intel C/C++ Compiler 9.0

COMPILER RALLY

Intel presented Version 9.0 of the C++ compiler for Intel processors in June, raising the bar for highly optimized code.

BY RENÉ REBE

The interim version 8.1 of the Intel C++ Compiler (ICC) [1] introduced the AMD64/x86-64 architecture (EM64T for Intel). Version 9 is a full-fledged major release with new extensions and optimizations [2]. As in previous versions, the compiler can handle the IA-32, x86-64, and Intel Itanium architectures. Intel's own debugger, a code coverage tool, and the Eclipse developer environment round out the package. An assembler is additionally available for the Itanium CPU, although we will not be discussing the assembler in this article. Itanium developers have not benefited from Eclipse integration thus far.

The licensing model is similar to the previous version. A non-commercial license without support is available free of charge for open source projects. Binaries created with this version cannot be sold. A license is required for commercial development. Depending on the size of the installation, you can either specify the serial number or a license file. The compiler can also use a network-based Flex license manager. The Intel C++ compiler costs about 300 Euros from distributors or about 400 US dollars from Intel.

SSP

Software-based Speculative Pre-Computation (SSP), as triggered by the *-ssp*

Box 1: OpenMP

OpenMP [4] is an extensive standard that adds expressions to the C, C++, and Fortran languages to explicitly tell the compiler how to distribute programs across parallel threads.

The central elements are the *#pragma* directives, which provide instructions on how the compiler should split up the code into concurrent fragments. For example, *#pragma omp parallel* tags a block for parallel execution; *shared()* specifies common variables, and *private()* specifies the variables restricted to exclusive use by a process.

The *#pragma omp for schedule* directive specifies distribution across threads. The *#pragma* directives thus tag a processing loop for parallel execution.

The threads share the variables *a*, *b*, *c*, and *chunk*; the iteration variable *i* is private in each thread. The expression tells the compiler to perform parallel execution of the *for* loop and to split the iteration space into blocks of size *chunk*.

OpenMP defines more instructions to specify parallelization, however, these instructions are currently only implemented by special compilers, typically in clustering applications. Listing 1 shows a short example of how to use OpenMP.

option, is a feature that prompted heated discussions prior to the release. This features involves the compiler adding a number of auxiliary threads to the program, which pre-compute a few instructions contained in the program to fill the CPU instruction cache with data. This method is claimed to be particularly effective with hyper-threading.

SSP is just one of the so-called profile guided optimizations. To enable profile guided optimization, the developer first has to compile the program with a special instrument (*-prof-gen-sampling*), and then run it against a representative set of input data using *profrun*. Finally, the developer recompiles the whole program once more, referencing the data collected by the previous processes [3].

In contrast to previous versions, *-O2* or above now enables various inter-procedural optimizations, and the compiler optimizes more loops. ICC gives the developer even more feedback: the reports about optimized loops and other warnings are now more detailed. When compiling multiple files, the compiler now offers more optimization options. Additionally, the *-ipo* option in the new version now supports individual object files when compiling multiple files.

RPM-based Installation

Version 8 of the Intel Compiler weighed in at about 65 MBytes, but the current version easily beats that at no less than 192 MBytes. The tarball comprises various RPM packages for I-386, EM64T, and IA64. It also includes the complete Eclipse platform with the C Development Toolkit (CDT) in versions for GTK and Motif.

A shell script takes care of installing the packages, but problems have been known to occur with distributions other than the RPM-based Red Hat and Suse distributions officially supported by Intel. On some systems, the script simply stated that it did not recognize the machine type, glibc, and the kernel. In contrast, the install ran without any trouble on the Debian-based Ubuntu distribution. Whatever distribution you have, you will need the *rpm* command to unpack the RPM archive.

All in all, the compiler is not as easily managed as GCC, which integrates so neatly with the system. ICC will not look in the library and header directories of its own installation by default, and this means setting *-I/opt/intel/cc/9.0/include*, and possibly *-I/opt/intel/cc/9.0/include/c++*, in many cases. As the programs are typically linked against the ICC runtime libraries, you will probably want to set the *LD_LIBRARY_PATH* environmental variable to */opt/intel/cc/9.0/lib* when running ICC.

This is normally done by *source/opt/intel/cc/9.0/bin/iccvars.sh*, either added to the *.profile* or entered manually at the console. To make things easier, these defaults can be stored in the *.../bin/icc.cfg* and *.../bin/icpc.cfg* configuration files. Setting the *LD_LIBRARY_PATH* will save you an entry in */etc/ld.so.conf*. By default, ICC uses the C++ system library, although it does provide its own implementation. The *-cxxlib-icc* and *-cxxlib-gcc* options support explicit switching.

ICC Version 8.1 or newer understands the *-march* and *-mcpu* GNU compiler options. However, Intel uses a somewhat cryptic notation, *-x{K|W|N|B|P}*, based on Intel's internal codename: K stands for the Pentium III (Katmai), B for the Pentium M (Banias). The *-ax* handles more than one CPU type allowing ICC to translate code passages multiple times – once for each CPU type – wherever it is worth the effort. When a program is then launched, optimized blocks are enabled to match the processor type.

Cooperation with GCC

In principle, it is possible to compile mixed object files for a single program using ICC and GCC, for example, if ICC refuses to compile a file or if GCC produces far superior code. If you use *icc* for the linking process, the linker will bind the required Intel runtime libraries without being told to do so:

```
icc -o prog main.o math.o
```

However, if you want to link these object files using *gcc*, you will need to explicitly specify the auxiliary libraries. GCC needs additional linker parameters:

```
-L/opt/intel/cc/9.0/lib/ -lirc
```

Behind the scenes, the Intel Compiler for IA-32 uses the GNU Binutils anyway.

Manual and Automatic Parallelization

The OpenMP working group [4] devises specifications to provide parallelization aids to compilers for C, C++, or Fortran programs. The *-openmp* option tells the

Table 1: Vectorized Loops

ICC -O2	-O2 -ip	GCC	
Botan	(171) 36	0	2
Bzip2	0	58	6
GnuPG	132	192	6
Gzip	48	62	2
Lame	118	112	22
Libmad	24	24	4
OpenSSL	148	170	30
Tramp-3D	(30) 8	0	2

Listing 1: OpenMP Example

```
01 01
02 #include <omp.h>
03 02
04 main ()
05 03
06 {
07 04
08   const int N = 10000;
09 05
10   int i;
11 06
12   float a[N], b[N], c[N];
13 07
14   const int chunk = 100;
15 08
16
17 09
18   #pragma omp parallel
     shared(a,b,c,chunk) private(i)
19 10
20   {
21 11
22     #pragma omp for
     schedule(dynamic,chunk) nowait
23 12
24     for (i=0; i < N; i++)
25 13
26       c[i] = a[i] + b[i];
27 14
28
29 15
30   } /* End of parallel block */
31 16
32 }
```

Intel compiler to produce programs that parallelize CPU intensive fragments (such as loops) on Unix systems. When the program is launched, the binary uses the Pthread library to create multiple threads, which then process these fragments in parallel. This makes better use of multi-processor systems (see the "OpenMP" box).

ICC 9 detects segments of code in which individual threads can be parallelized without help from OpenMP. The *-parallel* option enables automatic parallelization. This means users who want to support multiple CPUs or CPU cores, without modifying the source code, can now expect the compiler to accelerate their programs. Table 1 shows the number of loops the compiler vectorizes per benchmark. The values in brackets are the expressions caused by code from the C++ STL library.

Debugging Tips

The detailed warnings issued by the Intel compiler are another useful aid for developers. In some cases they say a lot more than warnings issued by the GNU compiler, particularly in the case of constant expressions, use of temporary objects, comparisons between floating point numbers or loss of precision in computations and allocations. Just like in previous versions, the Intel compiler quotes the source code and tags the character at the point where the warning or error occurred. This is a useful aid to debugging (Listing 2, lines 3 and 4).

The debugger included with the ICC package has a text-based interface, just like its GDB counterpart, but it also has a minimal graphical front-end. The IDB debugger normally runs in DBX mode, where it expects input in a syntax defined by Intel. When launched with the *-gdb* flag, it provides GDB (Version 6.1 or higher) compatibility.

Listing 2: Detailed ICC Warning

```
01 01
02 lib/__dtostr.c(47): remark
   #1572: floating-point \
03 02
04 equality and inequality
   comparisons are unreliable
05 03
06     if (d==0.0) {
07 04
08           ^
```

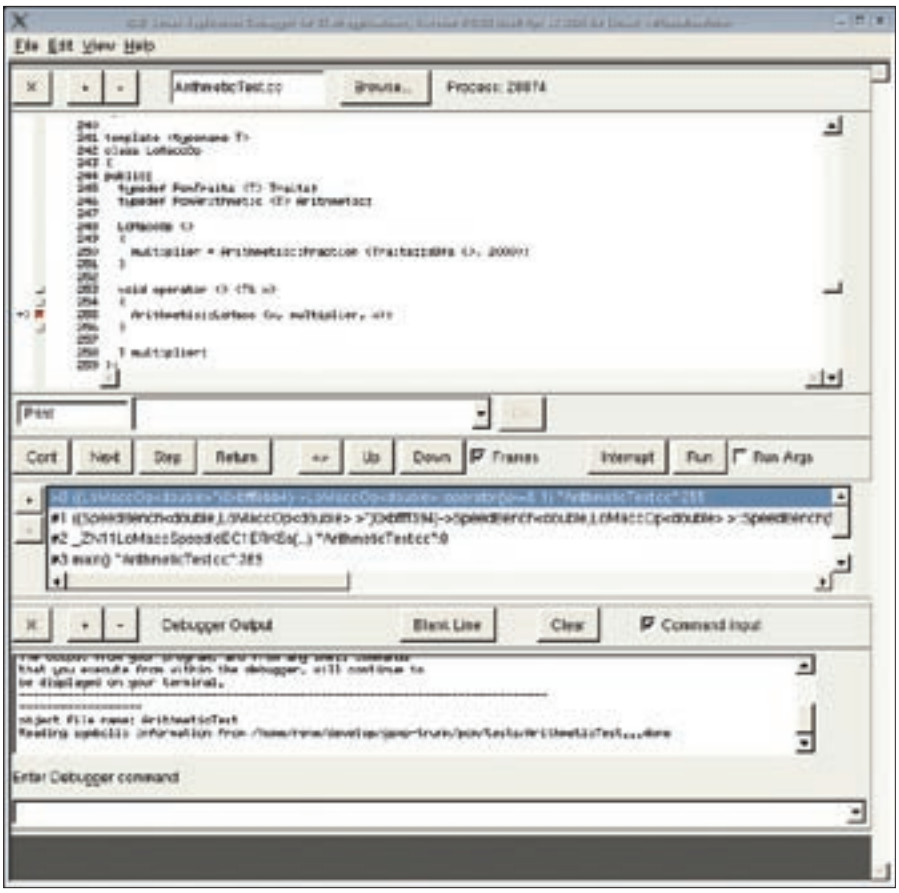

Figure 1: The Intel Compiler front-end is functional but not exactly modern.

Spartan GUI

When you launch IDB with the *-gui* option, it comes up with a fairly antique looking graphical user interface (see Figure 1). Unfortunately, the GUI provides very little automation support and does not simplify work much compared with the console. The GUI debugger does not give you anything in the way of visualizations.

In contrast to GDB, IDB does not support auto-completion using the tab key. But you can set breakpoints in instantiations of C++ templates, provided they have not been tagged inline by the compiler. GDB might accept these breakpoint entries but will not interrupt program execution when it reaches them.

Conclusions

We compared the ICC 9.0 to GCC 3.4 and 4.0 by compiling some representative Open Source tools (including OpenSSL, Libmad, Bzip2, and Gzip), and we found that recent versions of GCC have reduced or eliminated the performance advantage once held by ICC. Our tests didn't specifically include the high-end hardware or hardware-optimized software sometimes associated with ICC. ICC 9.0 still has an advantage in case of automatic vectorization and clearly demonstrates how well compiled C and C++ code can be distributed across SIMD (Single Instruction, Multiple Data) units. Support for OpenMP or automatic parallelization are beneficial for high performance computing. And the Intel Compiler's generous warnings allow developers to detect potential problems in the development phase rather than at the debugging stage. ■

INFO

[1] Intel C++ Compiler *http://www.intel.com/software/products/compilers/clin*

[2] Overview of ICC Optimizations: *http://www.intel.com/software/products/compilers/docs/qr_guide.htm*

[3] Ingo A. Kubbilun, "Kernel Tuning: Compiling the Linux kernel with the Intel compiler": Linux Magazine 08/04, pg. 44.

[4] OpenMP: *http://www.openmp.org*