GNU Compiler Collection 4.2

PARALLEL SPEED

The latest GNU compiler provides better support for parallel programming, and GCC also rolls out some new optimization features. We took GCC 4.2 for a test drive. **BY RENÉ REBE AND SUSANNE KLAUS**

After much debate and the usual delays, the latest version of the GNU C/C++ compiler (GCC) has finally materialized. Version 4.2 of GCC [1] follows in the trail of many major and minor changes. For a complete list of changes, refer to the GCC homepage [2].

The most significant change with version 4.2 is support for OpenMP [3], an open standard for program parallelization – especially for systems with shared memory. OpenMP lets programmers specify how the compiler and run-time systems will distribute code segments over multiple threads for parallel execution on multi-core systems.

The implementation of OpenMP in the new GCC version simplifies parallel programming on supported systems. Developers no longer need to go through the complicated and error-prone process of manually customizing their source code to match architecture-specific APIs.

OpenMP Features

The current GCC version supports the full set of OpenMP features. During the build, keywords determine how parallel execution of the code takes effect.

Pragmas allow compilers to build code with OpenMP extensions without being OpenMP-aware, which avoids platform-specific code and an inpenetrable *Ifdef* jungle. The *#pragma omp ...* keyword tells the compiler that parallelized optimization can take place. Behind the scenes, GCC uses the Pthread library to create the matching threads for Unix-style systems such as Linux.

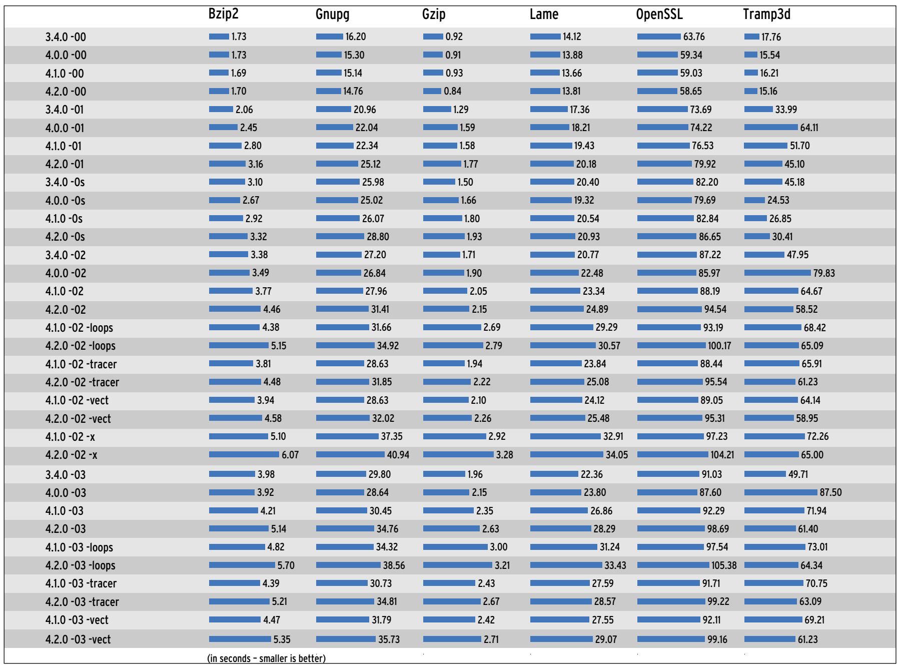
In the simplest case, loops with a significant iteration space – that is, wherever it is worthwhile creating new threads – are tagged as follows:

```
#pragma omp for
for(i = 0; i < N; i++)
  a[i] += b[i];
```

OpenMP has flexible controls; for example, you can tell a thread in a complex algorithm to use a local variable, which the program adds at the end (reduction):

```
#pragma omp parallel for⮐
private(w) reduction(+:sum)⮐
 schedule(static,1)
for(i = 0; i < N; i++)
{
  w = i*i;
  sum = sum + w*a[i];
}
```

The ability to ascertain the thread ID is more useful for testing than it is for algorithms:



	Bzip2	Gnupg	Gzip	Lame	OpenSSL	Tramp3d
3.4.0 -O0	1.73	16.20	0.92	14.12	63.76	17.76
4.0.0 -O0	1.73	15.30	0.91	13.88	59.34	15.54
4.1.0 -O0	1.69	15.14	0.93	13.66	59.03	16.21
4.2.0 -O0	1.70	14.76	0.84	13.81	58.65	15.16
3.4.0 -O1	2.06	20.96	1.29	17.36	73.69	33.99
4.0.0 -O1	2.45	22.04	1.59	18.21	74.22	64.11
4.1.0 -O1	2.80	22.34	1.58	19.43	76.53	51.70
4.2.0 -O1	3.16	25.12	1.77	20.18	79.92	45.10
3.4.0 -Os	3.10	25.98	1.50	20.40	82.20	45.18
4.0.0 -Os	2.67	25.02	1.66	19.32	79.69	24.53
4.1.0 -Os	2.92	26.07	1.80	20.54	82.84	26.85
4.2.0 -Os	3.32	28.80	1.93	20.93	86.65	30.41
3.4.0 -O2	3.38	27.20	1.71	20.77	87.22	47.95
4.0.0 -O2	3.49	26.84	1.90	22.48	85.97	79.83
4.1.0 -O2	3.77	27.96	2.05	23.34	88.19	64.67
4.2.0 -O2	4.46	31.41	2.15	24.89	94.54	58.52
4.1.0 -O2 -loops	4.38	31.66	2.69	29.29	93.19	68.42
4.2.0 -O2 -loops	5.15	34.92	2.79	30.57	100.17	65.09
4.1.0 -O2 -tracer	3.81	28.63	1.94	23.84	88.44	65.91
4.2.0 -O2 -tracer	4.48	31.85	2.22	25.08	95.54	61.23
4.1.0 -O2 -vect	3.94	28.63	2.10	24.12	89.05	64.14
4.2.0 -O2 -vect	4.58	32.02	2.26	25.48	95.31	58.95
4.1.0 -O2 -x	5.10	37.35	2.92	32.91	97.23	72.26
4.2.0 -O2 -x	6.07	40.94	3.28	34.05	104.21	65.00
3.4.0 -O3	3.98	29.80	1.96	22.36	91.03	49.71
4.0.0 -O3	3.92	28.64	2.15	23.80	87.60	87.50
4.1.0 -O3	4.21	30.45	2.35	26.86	92.29	71.94
4.2.0 -O3	5.14	34.76	2.63	28.29	98.69	61.40
4.1.0 -O3 -loops	4.82	34.32	3.00	31.24	97.54	73.01
4.2.0 -O3 -loops	5.70	38.56	3.21	33.43	105.38	64.34
4.1.0 -O3 -tracer	4.39	30.73	2.43	27.59	91.71	70.75
4.2.0 -O3 -tracer	5.21	34.81	2.67	28.57	99.22	63.09
4.1.0 -O3 -vect	4.47	31.79	2.42	27.55	92.11	69.21
4.2.0 -O3 -vect	5.35	35.73	2.71	29.07	99.16	61.23

(in seconds – smaller is better)

Figure 1: Benchmark results for build times of programs in the test.

```
#pragma omp parallel private(id)
int id = omp_get_thread_num();
printf §§
("This is thread %d\n", id);
```

With the x86 processor family continuing to grow, the x86 back-end now has two new architecture options. The *native* architecture directive tells GCC to apply best-possible optimization for the existing process at build time, based on *cpuid* instructions, while *generic* creates programs that will run equally well on AMD, Intel, or Via CPUs.

Bits and Bobs

A new warning, which can be enabled using the *-Waddress* option, and which is contained in *-Wall*, points out typical programming errors occurring in comparisons between function pointers and string literal addresses. The *-Wextra* option issues a warning in the case of an *if* expression followed by a semicolon to avoid typos like this one:

```
if (a);
return 1;
return 0;
```

The new compiler promises to reduce program launch time and, more specifically, the time the dynamic linker needs to resolve symbols, which is an issue that developers have been unhappy with for some time, especially in the case of

Advertisement

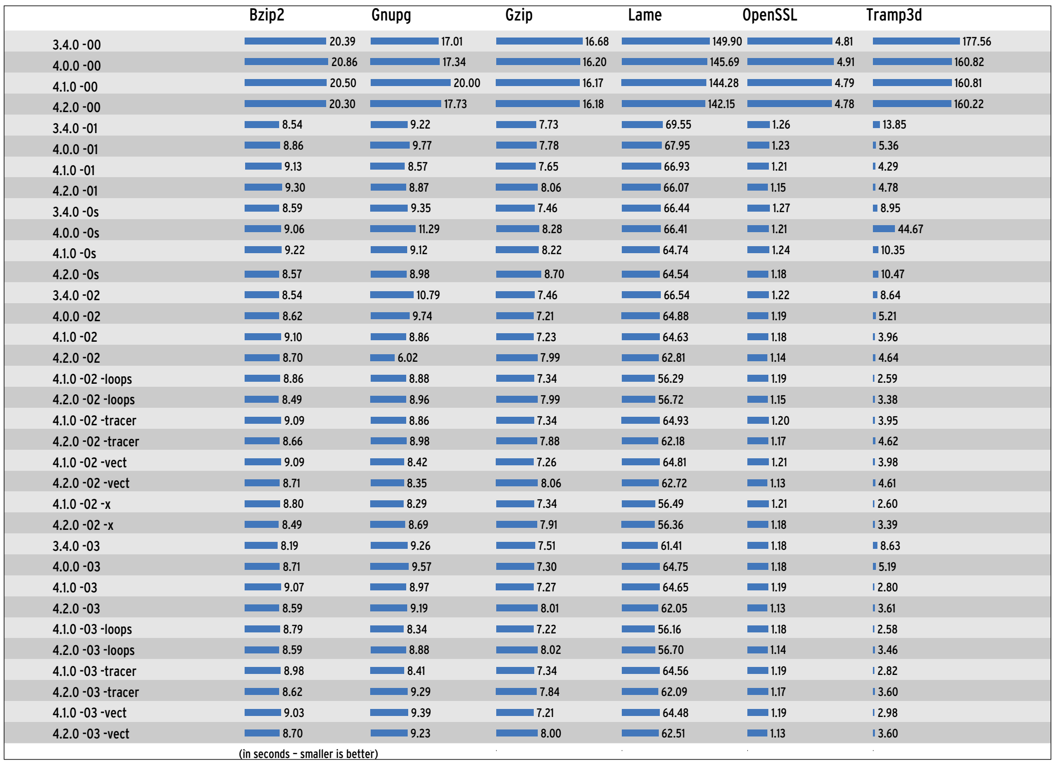


Figure 2: This diagram shows the run times for various compiler versions.

C++. Local symbols are no longer visible by default, and the compiler automatically applies class *visibility* attributes to members.

The *-fno-toplevel-reorder* option now makes it possible to output functions and variables in source code file order for code such as inline assembler that relies on a specific code order.

Note that new overflow optimization takes place as of optimization level *-O2*. The new complier can assume that an overflow will not occur for a loop such as *for (int i = 1; i > 0; i* = 2)* and thus optimize to form an infinite loop.

The GCC developers have added functionality to the new "200x" C++ standard, which is still in the standardization phase. For example, the TR1 namespace now includes *<random>*, *<complex>*. The lock-free container templates developed during Google's Summer of Code have also been integrated.

Regression

The good news is that GCC version 4.2 does not introduce many new bugs. A short test, in which we used the new compiler to build a complete system with T2 [4], yielded just two errors.

For one thing, far more memory was needed to build a couple of files that belong to the Xorg server [5] package, forcing the kernel to terminate the compiler on systems with less than 1GB RAM. For another, OpenSSL uses function pointer typecasts [6] in a way that the C standard does not define; this causes the program to quit at run time [7].

Benchmarks

The lab machine was an Intel Core 2 Duo with a clock speed of 2GHz and 1GB RAM.

We used the current version of Open Bench to test GCC versions 3.4, 4.0, 4.1, and 4.2 and compiled with the CPU in 64-bit mode for 64-bit programs. We measured the build time (Figure 1) and the run time in seconds (or the run time per iteration in milliseconds in the case of OpenSSL) (Figure 2).

Shorter Run Time

On initial inspection, we noticed that version 4.2 spends more time optimizing than its predecessors. The reward for this effort is a shorter run time, even for some legacy C programs.

Faster Build Time

Although the compiler is far slower when the typical *-O2* and *-O3* optimization levels are enabled, the build time during software development using *-O0* is faster.

A quick inspection of the logfiles for the benchmark build reveals that the new compiler vectorizes more loops – 14 for Gzip compared with 12 for GCC 4.1.

Conclusions

OpenMP integration in GCC 4.2 facilitates the task of programming on multi-core systems, which helps the free compiler project keep pace with commercial compilers.

Thanks to the widespread introduction of multi-core CPUs, parallelization has become a big topic for many programmers. The fact that each version of the compiler has taken more time to optimize code is slightly worrying.

New Projects

The new projects scheduled for completion before the new GCC version 4.3 is released include the Eclipse project's Java compiler, which has full support for Java 1.5. Integration of the MPFR library will help standardize calls to standard mathematical functions.

Support for the future 200x C++ standard will be extended in the next version of GCC. Optimization functions for more recent CPU types, such as Core 2 Duo and AMD Geode, have already made their way into the current GCC developer version. ■

INFO

[1] GCC homepage: *http://gcc.gnu.org*

[2] GCC 4.2 changelog: *http://gcc.gnu.org/gcc-4.2/changes.html*

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

[4] T2 SDE: *http://www.t2-project.org*

[5] Bugzilla report on Xorg server: *http://gcc.gnu.org/bugzilla/show_bug.cgi?id=31172*

[6] Patch for OpenSSL with GCC 4.2: *http://www.nabble.com/-PATCH--OpenSSL-vs-GCC-4.2.0-t3795606.htm*

[7] Open Bench: *http://www.exactcode.de/site/open_source/openbench*