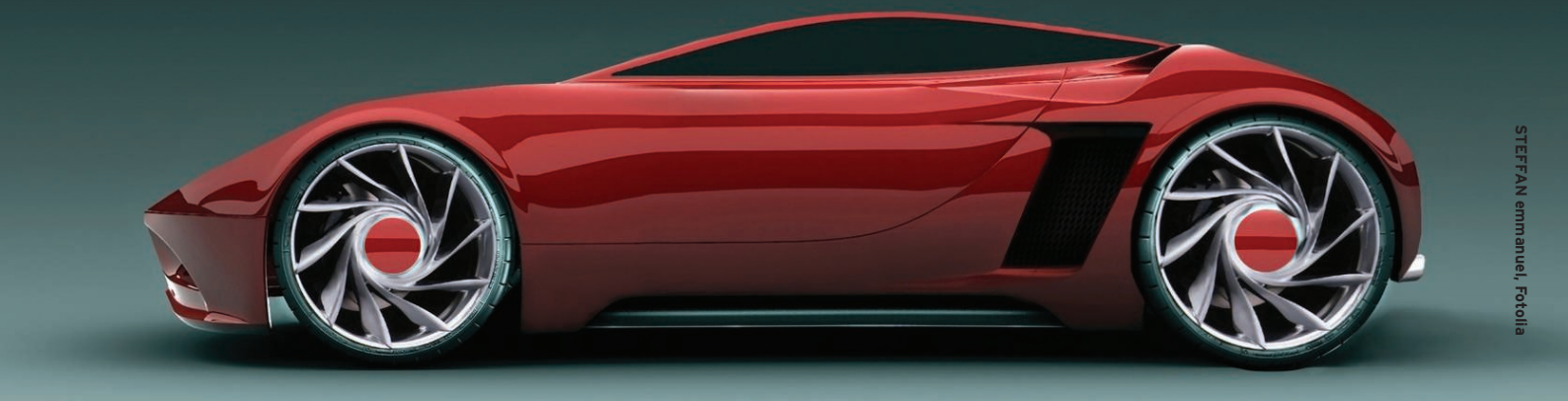
GNU Compiler Collection 4.3

OPTIMIZED!



STEFFAN emmanuel, Fotolia

The latest GCC 4.3 is set to take the programming world by storm with new optimizations, experimental support for the next-generation C++ 200x standard, an optional parallelized C++ STL, and a new Java compiler courtesy of the Eclipse project. **BY RENÉ REBE**

Following hot on the heels of GCC 4.2, the GNU Compiler Compilation [1] version 4.3 is now available [2]. As could be expected, many functions tagged as obsolete have now been dropped, such as the *-m386*, *-m486*, *-mpentium*, and *-mpentiumpro* optimization options. If you really do need these legacy CPUs, you can reanimate them with the *-march=* and *-mtune=* options. Users with newer CPUs will appreciate dedicated optimization options for the AMD Geode and Intel Core 2, as well as the SSE3 (*-msse3*), SSE 4.1 (*-msse4.1*), and SSE 4.2 (*-msse4.2*) features.

From an architectural point of view, GCC 4.3 adds ARM version 7 and the Thumb-2 extension for size optimization. Direct support for the IBM Synergistic Processor Unit (SPU) Cell Broadband Engine Architecture, found in both PlayStation 3 and IBM servers, is another GCC first. The release notes [2] detail various changes for MIPS, Motorola 68000, Coldfire, Cris, and PowerPC.

Some of the new optimizations rely on the MPFR (multiple-precision floating-point computations with correct rounding [3]) library, which helps GCC evaluate complex expressions and calls to embedded mathematical functions and truncate to equivalent functions or constants at build time. The MPFR library returns correct results, independent of floating-point precision and the target CPU. On the downside, a new dependency on the MPFR library, and thus on the GMP libraries, makes cross-compiling GCC itself more complex because these two libraries also need to be cross-compiled with the C++ code.

x86 Optimization Reveals Kernel Bug

GCC 4.3 code for x86 no longer creates an explicit *cld* instruction before each autorepeat string operation (*REP MOV...*), thus saving between 4 and 52 cycles on an Intel Pentium. This development revealed that some Linux and BSD kernels do not reset the direction flag themselves during signal handling. This in turn can mean that the kernel performs string operations in signal handlers in the opposite direction, which leads to incorrect addressing – a fairly obvious vulnerability [4].

New Developments Compiling C

Version 4.3 of GCC is the first to detect out-of-bounds access to arrays at build time, with the use of constants or offsets to do so. The *-Warray-bounds* option for this is also enabled by setting *-Wall*.

The new decimal floating-point arithmetic [5] adds more precision for financial and scientific applications. The use of base 10 instead of 2 also means that operations perform more accurate rounding; for example, the result of 0.9:10 is now 0.09 and not 0.089999996.

A GCC extension lets developers specify binary integer constants with the *0b* prefix or bit maps with *0B*. Support for fixed-point data types from the Embedded C specification is available but has only been implemented for MIPS thus far.

ISO-Compliant C++

G++ now implements more components of the next-generation ISO 200x Standard, or C++0x for short. The *-std=c++0x* or *-std=gnu++0x* options enable the standard [6]. G++ now supports templates with a variable number of parameters and static assertions for this reason. In case of nested templates, programmers no longer need to insert blanks between double angle brackets:

```
std::vector<
<std::vector<int> >
```

INFO

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

[2] Changes compared with the previous version: *http://gcc.gnu.org/gcc-4.3/changes.html*

[3] MPFR library: *http://www.mpfr.org*

[4] Direction flag handling by the Linux kernel: *http://nvd.nist.gov/nvd.cfm?cvename=CVE-2008-1367*

[5] Decimal floating-point arithmetic: *http://www2.hursley.ibm.com/decimal/*

[6] C++ 200x (0x) status: *http://gcc.gnu.org/gcc-4.3/cxx0x_status.html*

[7] Overview of source code modifications for GCC 4.3: *http://gcc.gnu.org/gcc-4.3/porting_to.html*

The syntax prevents the compiler from incorrectly identifying a shift operator. In the future, things like this will also work:

```
std::vector<
<std::vector<int>>
```

In production use, faster build times are quite noticeable as less critical includes have been removed from the C++ STL header. This might mean developers explicitly need to include headers such as *limit.h*, *string.h*, or *stdlib.h* with their code [7].

Users with multi-core CPUs will appreciate that some STL classes and algorithms can be parallelized by defining the *_GLIBCXX_PARALLEL* macro. On the other hand, if you use early GNU STL extensions, such as *hash_set* or *hash_map*, in your programs, you will have to face the fact that G++ will be removing them soon; the C++0x standard envisages *tr1/unordered_set*, *tr1/hash_set*, and the like instead.

Inline Functions

In case of inline functions, the new GCC takes stack growth into account. When trial runs are performed in the course of feedback optimization, the C compiler now uses the block sizes of string operations such as *memcpy()*, *memset()*, and *bzero()* to create code for particularly small blocks. The *memcpy()* and *memset()* operations have been reworked so that GCC now uses the best choice algorithm, depending on the block size and target CPU.

C++ and object-oriented emulations in C benefit from an early inline optimization run, particularly for inline fragments such as access by *set()* and *get()* methods to properties in which the function code is smaller than its call overhead. Automatic vectorization is now enabled by default for *-O3* and is said to be capable of handling complex loops. Some new optimizations replace legacy, low-performance algorithms, again reducing build times.

A Different Kind of Coffee

One of the biggest changes relates to the GCJ Java compiler, which the GCC developers have completely replaced with the Eclipse Java Compiler.

This radical change means that Java 1.5 is fully supported, making it possible to create a complete Java stack from free software with GCC and the Iced Tea Open JDK fork.

Some Java tools, such as *fastjar*, have been lost because of the move, but *gjar* is provided as a replacement. Others, such as *gcjh*, have been completely reworked and do not support the full set of arguments supported by earlier versions.

Mainly Faster

For the *Linux Magazine* speed tests, I used an Apple Mac Pro with an Intel Xeon, 3GHz clock speed, and 8GB RAM with a Linux system running in x86 64-bit mode.

The good news is that build times are again below the values achieved by the previous version. Figure 1 shows a comparison between GCC 4.2 and 4.3. The new compiler is just slightly slower if you disable optimization altogether (with the *-O0* flag).

The resulting programs typically run slightly faster – the benchmarks in Figure 2 show improvements after the decimal point. The step backward the previous version took in program size optimization (*-Os*) seems to be a thing of the past, probably because of the inline heuristics referred to earlier. ■

THE AUTHOR

René Rebe is the CEO of Exactcode, in Berlin, Germany, and is involved with various open source projects. Find out more about the author and his projects on his blog at: *http://rene.rebe.name/*

	Botan	Bzip2	Gzip	Lame	Tramp3d
GCC 4.2.0 »-O0«	49.7	0.98	0.54	7.3	10.0
GCC 4.3.0 »-O0«	52.6	1.01	0.54	7.8	14.5
GCC 4.2.0 »-O1«	93.0	2.10	1.09	11.8	32.6
GCC 4.3.0 »-O1«	71.9	1.98	1.00	11.8	25.8
GCC 4.2.0 »-Os«	79.3	2.08	1.24	12.7	21.0
GCC 4.3.0 »-Os«	67.9	2.04	1.12	12.4	18.7
GCC 4.2.0 »-O2«	118.3	2.93	1.35	15.4	42.9
GCC 4.3.0 »-O2«	87.5	3.94	1.35	16.1	34.9
GCC 4.2.0 »-O3«	124.5	3.44	1.60	17.9	44.4
GCC 4.3.0 »-O3«	94.0	5.24	1.78	20.3	38.3

(in seconds – smaller is better)

Figure 1: Comparison of build times between GCC 4.2 and the latest 4.3 version.

	Botan	Bzip2	Gzip	Lame	Tramp3d
GCC 4.2.0 »-O0«	70.6	13.3	10.2	89.1	108.4
GCC 4.3.0 »-O0«	69.7	13.3	10.2	89.8	109.4
GCC 4.2.0 »-O1«	13.4	6.10	5.25	38.8	3.09
GCC 4.3.0 »-O1«	13.3	6.10	5.30	38.0	3.30
GCC 4.2.0 »-Os«	26.7	5.74	5.50	38.9	7.03
GCC 4.3.0 »-Os«	14.6	5.93	4.92	38.4	3.59
GCC 4.2.0 »-O2«	13.1	5.67	5.11	37.6	2.91
GCC 4.3.0 »-O2«	13.1	5.62	5.11	37.1	3.27
GCC 4.2.0 »-O3«	13.0	5.61	5.12	37.2	2.44
GCC 4.3.0 »-O3«	13.2	5.58	5.05	36.5	2.72

(in seconds – smaller is better)

Figure 2: Five benchmarks compiled with the use of both GCC versions.