ORACLE®

The latest GCC release series and the special modes of its runtime C++ library

Paolo Carlini

Overall summary

- Highlights of the GCC 4.6 series (and beyond)
  - C++ front-end
  - Optimizers
  - x86/x86_64 backend
- The special modes of the C++ runtime library
  - Namespace association
  - Debug-mode
  - Parallel-mode
  - Profile-mode
  - Your-mode ;)

The GCC 4.6 release series (and beyond)

The GCC 4.6 release series (and beyond)

- GCC 4.6.0 released March 25th, after an unusually long “Stage 4” phase
  - 4.6.1 forthcoming, many serious bugs fixed
- Definitely too many interesting improvement for half a talk, I'll concentrate on some topics
  - No Fortran 2003 / 2008 (see Release Notes about those)
  - No backend != x86/x86_64
  - Some discussion of the general outlook *post* 4.6.x
  - General advice: do not trust *too much* the Release Notes in terms of coverage (and accuracy too)
    - in my experience developers don't like writing docs, eg, a lot is missing from the C++ runtime section (also my fault)

The GCC 4.6 release series (and beyond)

- Interesting new warnings:
  - -Wunused-but-set-variable / -Wunused-but-set-parameter
  - -Wsuggest-attribute=[const|pure|noreturn]
- Much better diagnostics for common mistakes
  - Misplaced/missing colons, semicolons, etc
  - Solicited by CLANG
- #pragma gcc diagnostic

- A new GO (http://golang.org/) front-end

  >>> -fsplit-stack in C/C++

  - Useful for threaded programs, in that it is no longer necessary to specify the maximum stack size when creating a thread

C++ front-end & C++0x

- The C++0x effort continues both in the front-end and in the runtime library:
  - constexpr (more later)
    - The first publicly available implementation
    - Runtime library completely updated to exploit it
    - Still a bit buggy, improved a lot in GCC 4.6.1
  - nullptr
    - nullptr_t in the library (+ all the additional overloads)
  - noexcept
    - Sort-of compile-time `throw()` (std::terminate called)
    - -fnothrow-opt
      - “Microsoft-style” `throw()`

C++ front-end & C++0x

- Forward declaration of enums
  - Completing the enums package begun in GCC 4.5
- Range-based `for` loops
  - Including the library bits
    - By the way these bits will not be necessary anymore in the updated specs, implemented for GCC 4.7
- Unrestricted unions
- More...

- See http://gcc.gnu.org/gcc-4.6/cxx0x_status.html for links to ISO papers including rationale for each one
  - http://gcc.gnu.org/projects/cxx0x.html can be also useful to see the progress from one series to the next

C++0x constexpr (crash intro)

- Consider the following snippet, in C++03:

```
template<int> struct F { };

F<std::numeric_limits<int>::max()> f;  // Error!
```

- Thus the C-style way of using limits, via macros like INT_MAX, was still unavoidable with templates. Also, code like:

```
const int z = numeric_limits<int>::max();
```

is legal in C++03 but z is dynamically (ie, at run-time), *not* statically initialized

C++0x constexpr (2)

- In GCC 4.6, C++ runtime library functions like max above are decorated with the **`constexpr`** keyword, ie (modulo irrelevant details):

```
static constexpr
int max()
{ return __INT_MAX__; }
```

- Only sufficiently simple functions (eg, the body must consist of a single return statement, no iteration, no changes to the arguments, etc.) can be syntactically declared as such but then (*assuming the arguments are in turn constant expressions*) the function is completely evaluated at compile-time and the return value “inlined” at each call site.

C++0x constexpr (examples)

```
constexpr int square(int x) { return x * x; }

constexpr int abs(int x)
{ return x < 0 ? -x : x; }

constexpr int
fact(int x)
{ return x > 2 ? x * fact(x - 1) : 1; }

float array[square(9)];   // Ok (not C99 VLA!)

std::bitset<abs(-87)> s;  // Ok

enum { Max = fact(5) };   // Ok
```

C++0x constexpr (4)

- Important clarification. Code like:

```
extern const int medium;
const int high = square(medium); // Ok, dynamic init
```

is also legal in C++0x, but the call boils down to a *normal* function call, thus high is initialized at run-time, because at compile-time the value of medium is unknown. Indeed:

```
constexpr int high = square(medium); // error!
```

vs

```
constexpr int s = square(5);         // Ok
```

s is called constexpr data (compile-time, “rodata”).

C++0x constexpr (5)

- constexpr *constructors* also exist (see std::complex):

```
struct complex {
  constexpr complex(double r, double i) : re(r), im(i) {}
  constexpr double real() { return re; }
  constexpr double imag() { return im; }

private:
  double re; double im;
};

constexpr complex I(0, 1);        // Ok
constexpr double i = I.imag();    // Ok
```

Optimizers: inlining

- Partial inlining: -fpartial-inlining (enabled by -O2)
- Inlining of callbacks is now more aggressive
  - Example: testcase fmtflags_manipulators.cc in the performance testsuite of the library

```
ostringstream os;

os.setf(ios_base::uppercase);
```

vs

```
os << uppercase;
```

Optimizers: WHOPR, LTO, FDO...

- Scalable Whole Program Optimizer (WHOPR)
  - Link time optimization can now split itself into multiple parallel compilations. Can be controlled in various ways, eg:
    - -flto=n
    - -flto-partition=[none|balanced|none]
- Rather recent blog entry by Mike Hommey on Firefox vs FDO with GCC 4.5
  - http://glandium.org/blog/?p=1975
- In GCC 4.6 LTO too works for Firefox (and other large applications, like GCC itself)
- Vastly improved in terms of memory use, performance, bugs fixed.

GCC 4.6: backend tidbits & varia

- libquadmath
  - Primary motivation: Fortran
- -march(-mtune)=[core2|corei7|corei7-avx|btver1]

- Ongoing work on OpenMP 3.1, will be in GCC 4.7
  - 3.0 delivered in GCC 4.4
  - A couple of serious bugs affecting OpenMP vs C++ being fixed in GCC 4.6

GCC 4.6 / 4.7: looking forward

- Quite a few active development branches
  - See both:
    - http://gcc.gnu.org/svn.html
    - http://gcc.gnu.org/wiki
  - Many Google people involved, besides the traditional Red Hat, Novell, etc.
- Pre-parsed Headers
- Profile Feedback Based Lightweight IPO
- Graphite targeting OpenCL (vs OpenMP)
- C++0x Memory Model
  - In particular bitfields-related issues, currently being worked on
- Transactional memory

The special modes of the C++ runtime library

A Chronology

- 2004 (GCC 3.4): debug-mode
  - Contributed by Doug Gregor
  - Exploits the “strong using” GNU extension
- 2008 (GCC 4.3): parallel-mode
  - Contributed by Johannes Singler and Leonor Frias
- 2009 (GCC 4.4): “inline namespace” mechanism
- 2010 (GCC 4.5): profile-mode
  - Contributed by Silvius Rus, Lixia Liu, and Changhee Jung
- 2011 (GCC 4.6): debug-mode performance work
  - More profile-mode forthcoming

Namespace association everywhere

- The idea is segregating the code for each special mode in a separate namespace and then importing it on demand in namespace std.
- However, the normal using-declaration mechanism is way too *weak* for that
  - A template can only be specialized in its actual namespace.
  - Argument-dependent lookup (aka “Koenig lookup”) breaks down if library components are split across multiple namespaces.
- The “inline namespace” mechanism, standardized in C++0x, solves all those issues!
  - See N2535 on the WG21 web site for details...
  - Available in GCC in C++03 mode too as an extension (like, eg, variadic templates)

Namespace association (N2535 example)

```
namespace Lib
{
  inline namespace Lib_1    // Lib_1 is an inline namespace of Lib
  {
    template <typename T> class A;
  }
  template <typename T> void g(T);
}
struct MyClass { … };

namespace Lib
{
  template <> class A<MyClass> { … }; // Ok, can specialize
}

int main()
{
  Lib::A<MyClass> a;
  g(a);  // Ok, Lib is an associated namespace of A, is searched
}
```

Debug-mode

- Today, most implementations of the C++ standard library provide a debug-mode, at least performing runtime checks via
  - Some kind of safe iterators, keeping track of the container whose elements they reference (eg, trying to increment past-the-end iterators, dereferencing iterators pointing to destructed container, are all easily detected)
  - Pre-conditions in the algorithms (eg, valid ranges, sorted ranges)
- Well established in GCC, -D_GLIBCXX_DEBUG
  - Pedantic mode also available
- Refer to the documentation about the specific design choices of the implementation

Debug-mode issues

- Many still today!
- *Issues with std::string, exported, weaker checking*
  - The **`extern template`** mechanism (standard in C++0x, by the way) is disabled in order to always check pre-conditions
  - No safe iterators
- *std::bitset vs C++0x*
  - Would not be a literal type anymore
- *Performance can be poor in some cases*
  - Improvements in GCC 4.6 thanks to Francois Dumont' help (see libstdc++/46659 for some rather impressive numbers)
  - More can be probably done, Francois is on it..
  - … do you care?

Debug-mode issues (2)

- *Behavior vs threads*
  - Ideally, the debug-mode library, should be *indistinguishable* from the normal library, but the safe iterators are a pain!
  - Not anticipated in the original design
  - First fixes: rather brutal locking strategies
  - Good improvements in GCC 4.6: essentially a pool of locks, randomly selected via hashing. We can certainly do better!
- *What about exceptions instead of assert?*
  - Long standing libstdc++/23888, differing opinions
  - C++0x knows about throwing checking libraries (see N3248)

Parallel-mode

- Enabled by -D_GLIBCXX_PARALLEL -fopenmp
- Stems from an University of Karlsruhe project aimed at parallelizing the C++ library via OpenMP.
- At the current stage of development, many algorithms are already available, both in <algorithm> proper and in <numeric>.
- Tuning and customization are easy (see docs), in any case the defaults are often sensible (at least on x86 / x86_64-linux).

Parallel-mode, some (rough) numbers

- A very simple experiment
  - On an i7-980x Linux machine, using /dict/words: 3878904 chars, 380646 words
  - Everything default, -O2 vs -O2 + parallel-mode
  - Relative real times in the Table
  - (# of iterations, etc, full details available)

	serial	parallel
sort & random_shuffle	15	3
find (“thing”)	7	1
stable_sort & random_shuffle	25	4

Parallel-mode issues

- *Dynamic memory allocation*
  - As happens for other scientific computing software, the code assumes that memory is just available and no memory allocation throws.
  - This is of course a big issue if the parallel replacements are supposed to behave exactly like the serial counterparts (besides performance).
- *Correctness vs C++0x about “move-only types”*
  - Quite a few parallel algorithms (eg, std::sort) assume that the types are just CopyConstructible and CopyAssignable, C++03 way. But in C++0x only MoveConstructible and MoveAssignable are required.
    - See “xfailed” testcases in the testsuite

Parallel-mode issues (2)

- *Integration with debug-mode*
  - Currently the special modes are mutually exclusive
  - As noticed by Francois Dumont, doesn't have to be like that, at least for debug-mode and parallel-mode. Will be hopefully fixed in GCC 4.7
- *Vectorization?*
  - For bits of <numeric> seems an obvious choice
  - How does that mix with OpenMP / OpenCL?
- *Other forms of parallelization?*

Profile-mode

- Silvius Rus @ google is the main contributor of the original code and maintainer today
- Enabled by -D_GLIBCXX_PROFILE
- Focused on the selection of the optimal std:: container (or of its parameters) for each problem
- During representative runs the instrumented library records the call patterns, collects statistics
- Basing on a performance model, which also includes details of the architecture (eg, Opteron vs Core2), diagnostics is produced about whether a different container would be more efficient in each “context”
  - normally the granularity is an individual function call

Profile-mode (2)

- *Examples of diagnostics (various subsets)*
  - Vector-to-list
  - Ordered-to-unordered
  - …
  - Hashtable-too-small
  - Hashtable-too-large
  - …
  - Vector-too-small
  - Vector-too-large
  - …
  - (see on-line docs for a detailed list & status table)
- Adding more is work in progress

Profile-mode, simple example (from Silvius)

```
#include <vector>

int main()
{
  std::vector<int> v;
  for (int k = 0; k < 1024; ++k)
    v.insert(v.begin(), k);
}
```

- It works! Profile-mode suggests to switch from std::vector to std::list and indeed the code runs about *two* times faster.
- Also...

Profile-mode (4)

- … the current - ie, as delivered in GCC 4.5 and 4.6 - profile-mode is already able to detect cases where std::vector is instead preferable to std::list - thanks to the compact memory layout - even if many insertions in the middle happen, something badly known in the community until quite recently.
  - A typical simple case (by Bjarne) would be inserting while maintaining the sequential container ordered
- http://gcc.gnu.org/ml/libstdc++/2010-12/msg00080.html
  - "A call for libstdc++ profile mode diagnostic ideas"
  - A lot of improvements forthcoming in 2011
  - Please get in touch with Silvius!

Profile-mode issues

- Of course still at an initial stage, needs testing
- Make sure it works well also on non-x86/x86_64 and/or non-Linux machines

- The memory footprint of the instrumented code could be optimized (too much is inline). Known issue.
- Double check and likely fix some parts of the models vs C++0x. Tricky.
  - For example, internal bookkeeping operations of containers like std::vector can be *much* faster for “moveable” types: the performance model cannot be the same.

Profile-mode issues (2)

- Probably do something about controlling granularity in a case by case way
- *Science-fiction:* automatic decisions, without asking the user to change himself the code, thus adjust the container, etc.

Conclusions

- Let's stop here today.
- Please also send your ideas, observations, etc, to:

  libstdc++@gcc.gnu.org
- ... or simply to me ;)

  paolo.carlini@oracle.com

Bibliography about the special modes

- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2535.htm
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3248.pdf
- http://gcc.gnu.org/onlinedocs/libstdc++/manual/debug_mode.html
- http://gcc.gnu.org/onlinedocs/libstdc++/manual/parallel_mode.html
- http://gcc.gnu.org/onlinedocs/libstdc++/manual/profile_mode.html
- Parallelization of Bulk Operations for STL Dictionaries. Johannes Singler. Leonor Frias. Copyright © 2007 Workshop on Highly Parallel Processing on a Chip (HPPC) 2007. (LNCS).
- The Multi-Core Standard Template Library. Johannes Singler. Peter Sanders. Felix Putze. Copyright © 2007 Euro-Par 2007: Parallel Processing. (LNCS 4641).
- Perflint: A Context Sensitive Performance Advisor for C++ Programs. Lixia Liu. Silvius Rus. Copyright © 2009. Proceedings of the 2009 International Symposium on Code Generation and Optimization.

Thanks!