ORACLE®

The GNU C++ Library and its special modes

Paolo Carlini
PhD

Outline

- Give a feeling of the current status of the special modes of the GNU C++ runtime library.
  - The maintainers spend quite a bit of work even simply keeping everything in sync and functional when bits of the normal mode is changed!
- Emphasize, not hide, the open issues, dark corners, beyond what's available in the form of Bugzilla PRs and discussions on the mailing lists.
- … encourage help and contributions from the community!

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

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, part of the forthcoming C++1x Standard, solves all those issues!
  - See N2535 on the WG21 web site for details...
  - Available in GCC in C++03 mode too (like, eg, variadic templ)

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, which keep track of the container whose elements they reference (eg, trying to increment past-the-end iterators, dereferencing iterators pointing to destructed container, 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!

- *Issues with std::string, exported, weaker checking*
  - The `extern template` mechanism (standard in C++1x, by the way) is disabled in order to always check pre-conditions
  - No safe iterators
- *std::bitset vs C++1x*
  - 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 impressive numbers)
  - More can be probably done, Francois is on it..

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!
  - Rather brutal locking strategies
  - Not part of the original design
  - 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++1x 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.
- In the current form many algorithms are already available, both in <algorithm> proper and in <numeric>.
- Tuning and customization is easy (see docs), in any case the defaults are often sensible (at least on x86 / x86_64-linux).
- Among the original contributors, Johannes Singler is certainly still quite responsive for normal bugs.
  - Not quite sure about enhancements and extensions

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 a lot of scientific computing software, the code assumes that memory is just available and no memory allocation throws.
  - This is of course a very bad problems if the parallel replacements are supposed to behave exactly like the serial counterparts (besides performance).
- *Correctness vs C++1x 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++1x only MoveConstructible and MoveAssignable are required.
    - See "xfailed" testcases in the testsuite (but some can be actually enabled, do not really fail anymore, I'll adjust that)

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 4.7
- *Vectorization?*
  - For bits of <numeric> seems an obvious choice
  - How does that mix with OpenMP?
- *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 a work in progress

Profile-mode, trivial 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 would be inserting while maintaing 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 non-Linux too ;) machines

- The memory footprint of the instrumented code could be optimized (too many inlines). Known issue.
- Double check and likely fix some parts of the models vs C++1x
  - 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

- 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!