Towards better C++ binding for OpenMP™

Christian Terboven
Center for Computing and Communication
RWTH Aachen University, Germany
terboven@rz.rwth-aachen.de

Michael Wong
IBM
Canada
michaelw@ca.ibm.com

Introduction and Previous Work

OpenMP C++ working group

- Studies how to improve C++ bindings for OpenMP
- Achievements in OpenMP 3.0:
  - Enhanced capabilities for parallelizing C++ loops
  - Refined specification for private non-POD variables enabling C++ idioms in a thread-safe manner

C++0x

- Current standard from 1998/2003 has no notion of threads
- Next standard will add support for concurrency:
  - C++ memory model
  - Threading API and additional language support
  - Atomic variables
  - Explicit support for initialization and destruction of static duration variables
  - Abstractions for Shared-Memory parallelization

Recent compiler releases support some of the new C++0x language features already. `BOOST.threads` as a reference implementation of C++0x threading is available already.

C++0x: Static Duration Variables

Function-local static variables

- C++98/03: Shall be initialized before first use (lazy init)
  - Current implementations usually use a guard variable to ensure that initialization occurs during first function call
- C++0x: Initialization of function-local variables in multiple threads will be sequenced and thus thread-safe.

Global static variables

- Access to different translation units was indeterministic in C++98/03, will become undefined in C++0x.
- Interaction only with globals in the same translation unit → concurrent initialization of different translation units.

Dynamic initialization / Expressions as initializers

- Dynamic initialization is not forbidden, but one needs to apply `std::call_once`.
- New `constexpr` keyword guarantees that qualified expressions are constant.

Added value:

- OpenMP does not specify how static duration objects are initialized safely and relies on the base language, but C++03 requires locking to be implemented by the user.
- Current C++ singleton in OpenMP requires critical section or locks done by the user, probably in a non-portable way.
- C++0x static duration variables will help single-locking be done properly.

Improved OpenMP support for C++0x

Improvements to the current Loop Construct

- Limitation to `RandomAccessIterator` and prohibition of `!=` operator makes many iterator loops unparallelizable.
- BUT: Increment of one is most common usage case and is always well-defined with `RandomAccessIterator`.

Parallelization of Range-based Loops

- A `Boost.Range` represents a controlled list, will be in C++0x.
- `for (for-range-decl: expression) statements`
- Language construct built around the range concept:
  - i: Declaration of a variable to iterate over the range
  - ii: Representation of the range concept being iterated on
- Any expression for which a concept map exists can be converted to a concept.
- *Parallelization with OpenMP*: Pose certain requirements on a range to be parallelizable (i.e. RAI) and allow range-based loops for the Loop Construct.

Increased Expressiveness for Reductions

- Must-Have: Min/Max. Nice-to-Have: Array Reductions.
- Added value: Reductions on non-POD variables.

C++0x: Thread-Local Storage

New keyword `thread_local`

- Standardizes existing practice, but C++0x will
  - enable dynamic initialization

```
thread_local std::vector<int> v = f();
```

  - allow for non-POD types

```
thread_local std::string s = “hello”;
```

Added value

- Implement OpenMP’s `threadprivate` with TLS, instead of Posix-Threads, for better speed.
- Allow `thread_local` to work with OpenMP threads.

Conclusion and Future Work

- **Make OpenMP and C++0x work well together by specifying their interactions.**
- **Split OpenMP specification into separate C/C++ and FORTRAN parts?**
- **We have only illustrated a few C++0x concurrency features, there are more to investigate:**
  - **Thread exception propagation**
  - **Futures and tasks**
  - **C++0x atomics**

RWTH AACHEN UNIVERSITY Center for Computing and Communication

IBM®