Transactional Memory for C++: Standardization efforts and commercial implementations

Torvald Riegel
Red Hat
15/01/19

Transactional memory for C++

- Lots of large C++ code bases
- For commercial software to evaluate/adopt TM...
  - Standardization
  - Commercial implementations

- This talk:
  - ISO C++ standardization efforts around TM
  - TM support in the GNU Compiler Collection

ISO C++

- ISO C++ committee (JTC1/SC22/WG21)
  - 100+ people attending 2-3 6-day meetings per year
  - Study groups on various subtopics
  - Produce drafts that are voted on by ISO National Bodies

- C++14 has been published recently
  - Only three years since previous release (C++11)! ;-)
- Technical specifications
  - Outlooks on features that may become part of the standard in the future

ISO C++ Study group 5 (SG5)

- Focus: TM
  - Language features
  - Integration with C++ Standard Library
- SG5 members: Mix of industry and academia
- Current goal: Produce a Technical Specification (TS)
- Current state:
  - Draft TS out for initial ballot and comments (N4302)
  - Considered experimental – aim is to get feedback
  - When no consensus on one way to do something, provide different ways
- Research on TM is biggest source of input for SG5

C++ TM TS: Transactions as a language construct

- Four ways to demarcate a transaction(-like code region):
  ```
  atomic_commit { /*...*/ }
  atomic_noexcept {  /*...*/ }
  atomic_cancel {  /*...*/ }
  synchronized {  /*...*/ }
  ```

- If no nested non-transactional synchronization and no exceptions, all have the same semantics:
  - As if a single recursive global lock is acquired before and released after the compound statement
  - Default C++ data-race-freedom requirement on programs makes such transactions strongly atomic

C++ TM TS: Checking atomicity at compile time

- `synchronized {}` allows non-transactional synchronization in the compound statement
- `atomic_* {}` disallow this, require *transaction-safe* code
  - In all code that may be executed from `atomic_* {}`
- Transaction safety is part of type system
  - Functions can be declared `transaction_safe` (e.g., many standard library functions, memory allocation, ...)
  - Most kinds of non-transactional synchronization are currently not considered transaction-safe
  - Compiler checks that transaction-safe code is indeed that
- Results: `atomic_* {}` is an atomic transaction:
  No deadlock, …
- Both `synchronized {}` and `atomic_* {}` are useful

C++ TM TS: Failure atomicity

- `atomic_* {}` differ in behavior when exceptions thrown across transaction boundaries (i.e., *escape*)
  - `atomic_commit {}` behaves like sequential or lock code
  - `atomic_noexcept {}` terminates program
  - `atomic_cancel {}` rolls back transaction. But:
    - Only a few exception types allowed
    - Safely copying data out of a to-be-rolled-back transaction is difficult
      - Depends on program semantics
      - Needs at least additional code annotations
    - Programmer must make exceptions logically self-contained
    - Constrains implementations

C++ TM TS: Outlook

- Open questions:
  - Make empty transactions no-ops (e.g., allow compiler to remove them)?
  - Allow locks in atomic transactions?
  - Allow `atomic<T>` operations in atomic transactions?
  - Low-level escape mechanism for non-transactional code in transactions?

- Next steps for SG5:
  - Address ISO feedback and publish TS
  - Get feedback through implementations of TS
  - Adapt TS
  - Repeat

GNU Compiler Collection (GCC)

- Most widely used open-source C/C++ compiler
  - System compiler for all of the major Linux distributions
  - Support for and used by many embedded systems
- Versions:
  - GCC 4.9 is the last stable release
  - GCC 5 is what will become the next stable release
    - In stabilization mode currently (no new big features allowed)
    - What this talk refers to

TM support in GCC's C/C++ compiler

- Initial support built as part of the Velox project
- Implements basically an older version of the spec
  - `__transaction_atomic {}`, `__transaction_relaxed {}`
    - Very similar to `atomic_commit {}`, `synchronized {}`
  - Don't support the newer exception constructs and semantics (but have something for noexcept)
  - Don't support the newer `transaction_safe` type annotation, but...
  - Attributes for functions: `transaction_safe`, `transaction_unsafe`, `transaction_pure`, `transaction_wrap`

TM support in GCC: Recent changes

- After Velox: new features and general maintenance
- Compiler generates instrumented and uninstrumented code paths for each transaction
    - Instrumented for STM – uninstrumented for HTM
- Rewrote most of GCC’s TM Runtime library (libitm)
    - Portable C++11 code base (x86, x86_64, powerpc, arm, aarch64, ...)
    - Several TM algorithms:
        - Serial, single-lock write-through, ...
        - Multiple-lock write-through with global timebase (ie, LSA)
        - HTM with serial as fallback
    - HTM support on x86_64, powerpc, s390

TM support in GCC: Outlook

- Once C++ TM Technical Specification is about to be published, implementing it will make sense
  - Lots of overlap with what’s already implemented
  - Exception handling, transaction safety rules, and deciding how to ship transactional clones of functions are probably the biggest tasks
- Working with one of Paolo’s students on integrating their HTM auto-tuning
- Working with MIT on getting their HyTM as a contribution to GCC