Document Number: P1070R0
Date: 2018-05-07
Authors: Michael Wong
Project: Programming Language C++, SG5 Transactional Memory
Reply to: Michael Wong <michael@codeplay.com>

SG5: Transactional Memory (TM) Meeting Minutes 2018/04/09

Contents

Minutes for 2018/04/09 SG5 Conference Call 2

Minutes for 2018/04/09 SG5 Conference Call

Minutes by Michael Spear

1.1 Roll call of participants

Maged, Mike Spear, Victor, Michael W, Michael Scott, and Herb Sutter

1.2 Adopt agenda

Adopted.

1.3 Approve minutes from previous meeting, and approve publishing previously approved minutes to ISOCPP.org

Approved.

1.4 Review action items from previous meeting (5 min)

Only action item was to invite Herb. It has been done.

1.5 Call schedules (please add your away days)

Apr 9: this call
Apr 23
May 7 mailing deadline

May 21

June 4 RAP C++ Meeting

2. Main issues (50 min)

2.1 Future of TM Discussion with Herb

Herb, if you like to send any pre-call material or discussion, please go ahead.

Michael W.

We published a TS in 2015, but held back because we wanted usage experience. Apart from GCC (4.7 and beyond), there hasn't been any implementation. So we haven't had much usage experience. The GCC 6 implementation is very close to our TS. But we don't have much usage experience.

We have also been looking at different interfaces that may reduce some of the implementation difficulty.

There have also been changes in terms of the employment of personnel, which makes leadership (especially by Michael W.) difficult. We are also wondering what we need to do with the group: continue normal calls? suspend calls? transfer into SG1?

We are especially looking for Herb's guidance.

Mike: discussion of how we don't see a lot of use, and how the implementation is complex. Only GCC supports, so using TM is only an option for GCC-only environments.

Herb: does anyone else use it.

Michael Scott: mostly academic research.

Mike Spear: brief summary of TS. Discussion of lambda proposal.

Herb agrees on avoiding viral annotations. It's a huge blocker. When we talk about "as-if" lock semantics, it is worrysome because C++ is a zero-overhead language. You shouldn't pay for what you don't use. If the value proposition includes "but we can compile it away", then it isn't C++, it's Java. Another piece of C++'s charter is to be close to hardware.

Questions: is the lambda/executor proposal worth it, or is TM not worth it at all.

Herb:

Is it worth trying? It sounds like it's still researchy, but it could bear out in practice.

Herb sees rollback as a problem. You can't call any function, so you need viral annotations, which is a death knell for adoptability. A transactional flag into the type is viral, and is difficult

even with language support. It's possible, but not easy. And I/O can't be undone. But there are some really hopeful levels still for TM
- It won't be the holy grail for avoiding mutexes.
- But a useful level would (which Herb has suggested in the past) is MCAS. We want lock-free data structures that need MCAS. Could we have an "atomic block" with a simple programming model that has no options, tags, hints, or anything, and all that can go into it is raw memory reads and writes (maybe just 16, or 64), no opaque function calls. And then maybe add a few specific types, like unique_ptr and shared_ptr. This would enable high-performance lock-free data structures. It avoids annotation, I/O, etc.

Michael Scott: the proposal is similar, in that you pass a lambda, and how it behaves depends on the implementation, but it would be easy for the compiler to say "if the lambda only does XYZ, we can guarantee a speculative implementation and good performance on certain classes of machines". This wouldn't require the compiler to enforce the rules on the lambda. And the documentation could say "16 load/store or less will do well". So this proposal could be a special case of the lambda proposal.

Victor: is the requirement to enforce an important requirement? That seems like more work than our proposal.

Herb would prefer that we do not rely on hardware support.

Michael Scott: custom MCAS in software can be better than TM.

Mike Spear: Herb's ideas to simplify could really save a lot of overhead (e.g., logging, setjmp, etc).

Herb: We need zero overhead when we don't use it. And there should be nothing that I could write more efficiently by hand. Otherwise the abstraction doesn't belong in C++.

Herb: Totally uninterested in code that accesses the same variable inside and outside of a transaction. But two transactions should be able to use the same variables at the same time.

Michael Scott: what about accessing an atomic variable inside and outside of transaction?

Herb: Expect the answer to either be (a) forbidden, or (b) it's allowed, and it is just as performant as other atomic accesses. Note: we don't usually touch atomics inside of critical sections.

Herb: Also assuming no nesting is fine.

Herb: It is fine to say "the more you do inside the block, the more you expect to pay (esp. wrt contention)". But we don't say "usability even if it costs you performance". If performance drops, at least the programmer knows she couldn't have done better. But we don't specify performance. We do require things to be implementable with the zero overhead principle.

Michael W.: Where do we go from here? Do we want to have Herb come back?

MLS: Our executor idea is compatible with Herb's proposal. Do we have an AA to think about a narrower API that gives an optimal implementation for MCAS?

Herb would love it if we could keep thinking about this.

Brief discussion of when the performance is contingent on contention.

<< Out of Time >>

2.2: Interaction with Executors and Synchonized proposal

https://groups.google.com/a/isocpp.org/forum/#!topic/tm/jG9XPJetNkc

The last discussion has us considering an alternative lambda form.

See Paper emailed out on Lambda proposal

https://docs.google.com/document/d/1ICmcrCdigq3ataoM2Jl7m19h_Sa3aE3KfU6AVkPyT-4/edit#

2.3 future issues list:

1. llvm synrhonized blocks
2. more smart ptrs?how fast can atomics and smart ptrs be outside tx if they have to interact with tx (for world that does not care about tx), the atomic nature of smart ptrs as a way towards atomics inside atomic blocks
3. more papers?
4. Issue 1-4 paper updates to current TM spec
5. std library

2.4 Discuss defects if any work done since last call
Issue 1: https://groups.google.com/a/isocpp.org/forum/#!topic/tm/SMVEiVLbdig
Issue 2: https://groups.google.com/a/isocpp.org/forum/#!topic/tm/Th7IFxFuIYo
Issue 3:https://groups.google.com/a/isocpp.org/forum/#!topic/tm/CXBycK3kgo0
Issue 4: https://groups.google.com/a/isocpp.org/forum/#!topic/tm/Ood8sP1jbCQ

3. Any other business

Skipped due to time.

4. Review

4.1 Review and approve resolutions and issues [e.g., changes to SG's working draft]
N4513 is the official working draft (these links may not be active yet until ISO posts these documents)
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4513.pdf

N4514 is the published PDTS:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4514.pdf

N4515 is the Editor's report:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4514.html

Github is where the latest repository is (I have updated for latest PDTS published draft from post-Leneaxa):
https://github.com/cplusplus/transactional-memory-ts

Bugzilla for filing bugs against TS:
https://issues.isocpp.org/describecomponents.cgi

4.2 Future backlog discussions:

4.2.1 Write up guidance for TM compatibility for when TM is included in C++ standard (SG5)

4.2.2 Continue Retry discussion
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/qB1Ib__PFfc
https://groups.google.com/a/isocpp.org/forum/#!topic/tm/7JsuXIH4Z_A

4.2.3 Issue 3 follow-up

Jens to follow up to see if anything needs to be done for Issue 3.

4.2.5 Future C++ Std meetings:

2018 06-04 RAP C++ Std meeting

4.3 Review action items (5 min)

All: We should work to see how our ideas match with the zero overhead principle, and how Herb's advice can help us steer towards an acceptable TM proposal for C++.

5. Closing process

5.1 Establish next agenda

Next call April 23rd. Michael W. will not be available.

5.2 Future meeting
Next call: TBD

Apr 9: this call
Apr 23
May 7 mailing deadline

May 21

June 4 RAP C++ Meeting