Document Number: P1472R0
Date: 2019-01-18
Authors: SG5 minutes takers
Project: Programming Language C++, SG5 Transactional Memory
Reply to: Hans Boehm<hboehm@google.com>
Audience:: SG5

SG5: Transactional Memory (TM) Meeting Minutes (June 2018 - January 2019)

Contents

Minutes for 2018/06/25 SG5 Conference Call 1
Minutes for 2018/07/09 SG5 Conference call 4
Minutes for 2018/08/06 SG5 Conference call 7
Notes for 2018/10/15 SG5 Conference call 9
Notes for 2018/10/29 SG5 Conference call 11
Minutes for 2018/11/26 SG5 Conference Call 12
Informal notes for 2019/01/07 Conference Call 14

2018-06-25 Minutes

Agenda:

1. Opening and introductions

1.1 Roll call of participants
Michael Scott, Michael Spear, Victor, Michael W, Maged,

1.2 Adopt agenda
Approve.

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

1.4 Review action items from previous meeting (5 min)
None.
MS. Herb and Tim seem consistent
2 questions
a. strongly restrict in the lambda that can be passed into library, assignment stmt and primitive types
b. fallback to global lock to be hidden inside library

Spear: Tim and Herb are far apart:
Tim: small local tx that enables whole class of lock free data structures, 98 % overlap with our recent proposal
Herb is asking for research project, sharedptr adds unbounded amount of work

Victor: Herb wants multiword cas that is dynamic
Sp: Herb saids it enables a whole class of data structures but Tim wants lock elision
MS: Herb may be using non-blocking data strcture casually,
Data structure that dont suffer from inopportune preemption
VL: a class Michael Greenwald's ... 2 headed emulation are all non-blocking
may retry in background several times, so what,

MS: only on hw with hardware tm
VL: only guarantee to make progress on HW TM, wont perform that well, TX is small then it wont be bad

MS: or get preempted for 10 ms,
VL: I can do this with HTM,
MM: user is expectign this to run on a signal handler then will not get what they are asking for as it will deadlock if impl is locked based
VL: for the most u should be OK, if contention is not too high
MM: then we will need stm that is non-blocking, will not interact well with TX that can not be restarted
irrevocablity
Sp: have C++ features that is not supposed to use from some context, no eh from signal handler
what did Herb mean by non-blocking? locked based stm are slow, but get non-blockign behaviour in informal sense
if actually non-blocking, then other parts of C++ e.g. allocators, eh are not specified to be non-blocking, both are lock based
problem if these interact

VL: current non-blocking algos that are hard to implement will not be , is maybe Herb's goal
MW: Herb wants to come back to talk to us about tm-lite

MS: informal use of non-blocking will not lead to performance problems due to preemption while holding a lock
VL: Agree, does htm with retry qualify as non-blocking
MS: lots of people think that is NB in practice
VL if I have SGL impl of TX, and guarantee no preemption while holding a lock, OK?
Yes OK
VL: I think thats what Herb means
Sp: we still need some blocking boundary condition, else cant deallocate data touched by tx
MW: allocation and deallocation? or just dealloc
Sp: if stm, then impl must ensure a doomed tx , that is deleted by another op does not seg fault what are the semantics of this?
VL: no restrctions?
Sp: 3 location Tx that touches 3 std:atomic vars?
VL: hmm need to think more on this
MM: need to check back with Herb, std has definiton for lockfree, similar to what is in literature, it is really about progress
All agree to get Herb
VL: is he saying the looser sense, MS and I think so
MM: or is it closer to strict
N4700 WD and it has lock fere defined.
4.7.2 para 2.2
July 9th or July 23
Sp: MS and I have been working on research paper on this document

MS: a bit too reactive here, think what we think is reasonable to provide first before talking to Herb or Tim
how much control flow inside, aritrary subroutines, std atomics inside, expose failure
VL: dissecting Herbs words, this would be a minimal approach
Tim is talking about persistent non volatile memory

AI: MW get Herb on for July 9
AI: MW +reflector forward our lambda proposal to Tim for feedback, then MS adds the current paper
AI: all to reply with their thoughts of the perfrect minimalist proposal

SG5 Transactional Memory
2018-07-09 19:00 UTC

Michael Wong, Jens Maurer, Michael Scott,
Herb Sutter, Victor Luchangco, Michael Spear

1.2 Adopt agenda
Herb Sutter's thoughts on TM light
No objections.

1.3 Approve the minutes from previous meetings
No objections.

1.4 Review action items
- Contact Herb Sutter to have him join today's call. Done.
- Contact Tim Sweeney for feedback. Tim sent e-mail.

2.1 Herb Sutter's thoughts on TM light

Alternate proposal for reduced TM interface (Oct 2017) by Michael Spear.
(atomic blocks, synchronized blocks, annotating transaction-safe functions)

Herb suggests "TM even lighter": In lock-free applications, we would like to do a multi-word compare/swap without data being contiguous. Just N (e.g. 10) memory operations in a transaction block; no exceptions, no function calls, no I/O, just plain memory operations.
Is that palatable to the group?

Michael Spear: Lambda-Executors proposal is defined in terms of locks, so transactions commit no matter what. The biggest fear under hardware TM that we have no strong progress guarantees. One proposal from 2008 attempts to be provably non-blocking, but no progress since then.

Herb Sutter: Lock-free for me means mostly obstruction-free; single global lock semantics imposed by a brace-enclosed block specially marked. We should go practical and useful.

Michael Scott: On Solaris, you have a kernel call that says "please don't preempt me for a little bit"; that might be all you need.

Herb Sutter: On older Windows, you could force a context switch and then have a good chance of not being interrupted again after you get the CPU again.

Michael Scott: Oracle uses the Solaris system call quite a lot.

Jens Maurer: Yes, a limited facility such as Herb's is useful.

Michael Scott: When we started talking about TM-light, we wanted to reduce the annotation and syntax burden. A lambda seemed useful. Details need to be nailed down (atomics, shared_ptr).

Herb Sutter: My suggestion: ordinary memory reads and writes only.

Jens Maurer: Fine with me. Lambda proposal with restrictions on the code appearing in the lambda is not good; instead expose the restricted code environment at the core language level.

Victor Luchangco: Syntax issues are important, but are orthogonal to the question what we could implement. Exploring lock-free requirement?

Herb Sutter: There is no requirement on a lock-free implementation for the "10 memory operations, but fast" proposal.
Tim Sweeney's point of view is that hardware vendor will make the chosen model fast, because they will complete on it.

Herb Sutter: A proposal that supports exceptions is more costly than one that doesn't. Both in terms of implementation and in terms of programming model complexity.

Michael Scott: Maybe Michael Spear is considering a lambda-passing proposal where you don't support STM. What's the extra runtime cost when passing a lambda?

Jens Maurer, Herb Sutter: A core language extension is much more optimizable.

Herb Sutter: If we have a very limited atomic{} block today, we can expand later to allow e.g. function calls etc. Don't close the door to future relaxation. We want static diagnostics.

Victor Luchangco: Where do we have a hard limit of 10 in the current

specification? Allow for larger sizes, with presumably worse performance.

Herb Sutter: Having a number was intending to make the implementation easier. If it doesn't, I'm fine with leaving the size unlimited.
See Annex on implementation limits in the C++ standard.

Herb Sutter: Yes, while loops would be forbidden as the first step.

Herb Sutter: All algorithms using MCAS work immediately with my proposal.

Michael Spear: I strongly disagree; all such algorithms are written in Java and assume garbage collection. We must be able to traverse a read-black tree within a single transaction. Relying on hazard pointers or similar is a non-starter.

Michael Spear: This will only perform well on HTM.

Next meeting scheduled for July 23.

20:00 UTC

"Minutes" for SG5 meeting on 6 Aug 2018 (Victor Luchangco)

Gist: We agreed in principle with what we want to specify as a first step for a stripped-down version of TM.

Participants: Mike Spear, Michael Scott, Herb, Hans, Victor

Secretary rota: Maged, Jens, Hans, Michael Scott, Michael Spear, Michael W, Victor

Mike Spear raised questions about how restrictive we want to be within atomic blocks:

- disallowing function calls discourages good modular programming
- what constitute "ordinary reads and writes"? what about atomics, volatiles?
- what is the purpose of these restrictions?

Herb:
- Not opposed to allowing more functionality in principle, but want to start with minimal useful proposal.
- Ordinary memory accesses are defined: they do not include atomics or volatiles. (Volatiles may be changed asynchronously, so it probably will never make sense to allow transactional access.)
- Perhaps we can use the wording for auto return functions to determine whether a function can be called within an atomic block.
- The goal is to have something easy to teach.
- Tim Sweeney is interested in having this functionality to use in Fortnite.

Michael Scott?:
- What about exceptions? (No, exceptions involve synchronization.)
- What about control flow (e.g., ifs, loops)? (These should be fine since there is no limit to the number of access in an atomic block.)

Summary of what we want in the proposal:

Introduce atomic block (contextual keyword).
Only ordinary memory accesses: no volatiles, atomics, etc.
  - may want to add atomics in the future, but keep it simple for now.
Local control flow (e.g., for loops, if statements) are allowed.
Primitive operations (e.g., addition, multiplication) are allowed.
No synchronization, throw statements, system calls, etc.
Only call functions that can be checked to obey restrictions.
 - perhaps require functions to be auto return value functions, so that the restrictions can be

checked

Action Item: Victor and Mike Spear will work on preparing a more complete draft description (i.e., in vernacular, not necessarily in the language of the specification).

Next meeting: Aug 20.

SG5, Oct 15 notes from Victor Luchangco

Michael Scott, Hans Boehm, and I called into the meeting (there seemed to be someone else who called in on a phone, but never actually identified themselves), and though we weren't really a quorum, and we didn't have an agenda, we reminded ourselves of what we think we agreed on and what we need to do. I think our last meeting was all the way back on August 20, so I took this secretary rota from there (and moved myself to the end):

Maged, Jens, Michael Scott, Michael Spear, Michael W, Hans, Victor

First, Mike Spear did send Herb Sutter a draft TM support proposal based on the LLVM-based plug-in that he and his students developed at Lehigh, so that Herb could share this with Tim Sweeney when they met last month. It would be great to hear any feedback that Herb or Tim has about the proposal.

Second, we agreed that the next step was to write a stripped-down proposal, or actually two: one that would be to explain the feature to programmers, and the other would be language to amend the actual specification. I think for now, we should focus on the first, but it may be that the second will consist primarily of stripping stuff out of the old TS.

We couldn't remember whether we had decided to go with a library interface (i.e., start a transaction by passing a lambda to a library function) or introduce an atomic block. I thought the latter, but I'm not sure, and our prototypes are likely to have a library interface in any case (certainly the Lehigh plug-in is so). That said, it will require some compiler support, because it needs to check that the only memory accesses within a transaction are ordinary reads and writes. The goal is to have a proposal that is easy to implement and easy to explain. We want something that is useful for noncontrived applications, even though it may not be useful for all the various ways that various of us would like to use them. But we also want to make design decisions that do not foreclose these other uses.

Some other aspects:
- No explicit limit on the number of accesses within a transaction, but definite guidance that fewer is better, analogous to guidance that smaller critical sections are better.
- Only ordinary reads and writes: no access to volatiles, locks, atomics, etc.
- Local computation and control flow (e.g., if statements, loops, switch statements) are okay.
- No exceptions can be thrown within a transaction (even if it is caught within the transaction).
- Function calls are okay provided that their definition is available (and the function body satisfies the constraints above). We can leverage the language for const-expr functions.

We may also want to provide a list of some extensions to consider, such as allowing locks or atomic accesses within a transaction, or allowing functions declared to be transaction-safe (as

in the TS), or dealing with exceptions. But even if we can implement these without run-time performance penalty, we aren't including them for now. We can decide what to actually add based on feedback from user experience.

Michael Scott suggested that we could have an NCAS library function, which might be useful if a programmer wanted to do their own plumbing for some reason. It is trivial to add this to a proposal that has transactions (since NCAS is trivial to implement as a transaction). The only "difficulty" is deciding on the interface. Providing this function might also be useful as a way to get people to start to exploit TM without having to think at all about transactions (and in some cases, it might be easier to provide just the NCAS library function).

Hans said we should try to provide an implementation that provides some benefit to the programmer even if there is no hardware TM support; that is, we shouldn't always just fall back to a single global lock. Certainly some STM seems appropriate in such cases. And if we can establish static separation of the locations accessed by a set of transactions, then they could be protected by their own lock. In any case, we don't provide any performance guarantees, so all this amounts only to some kind of guidance to give programmers and implementors.

October 29 notes from Mike Spear

Michael Scott, Hans Boehm, and Mike Spear called into the meeting. As with the previous call, we did not have enough people for a quorum, and we didn't have an agenda, but we reviewed some details and outlined some steps going forward.

The main task was discussion of TMLite. We have a document (attached), which isn't really in the proper format, but it could be the start of a TMLite document (technically, it is a hasty user manual for the TM system from Lehigh).

We discussed the following two issues, where the attached document may not be what we want for TMLite:

1 - Static checking - In our discussions of TMLite, we seemed to favor statically checking certain properties (no calls to functions outside of the translation unit; no syscalls, atomics, or volatiles, etc.). The alternative would be to serialize in those cases. We discussed how pragmatism favors the latter: a programmer might want to insert some printf debugging, or an assert, in a transaction. Static checking forbids these approaches. Note that the two are equivalent in terms of implementation effort. In any case where one compiler would insert a serialization instruction, the other would output an error. Note: No decision was made, since we did not have a quorum.

2 - Memory model - In the TMTS, synchronized blocks have the same ordering as locks. Atomic blocks do not. The attached document claims lock-based semantics, but it is sloppy, and does not precisely say how its TM proposal interacts with the memory model. Note: No decision was made, since we did not have a quorum.

I think that summarizes the discussion to a satisfactory degree. If any attendee wishes to correct or expand upon anything in this email, please do.

We anticipate that the next call will be on Monday, 12 November.

Minutes, SG5
26 November 2018

Participants: Hans Boehm, Mike Spear, Michael Scott, Victor Luchangco, Michael Wong

Michael Scott taking minutes.

Secretary rota: Maged Michael, Jens Maurer, Mike Spear, Michael Wong, Hans Boehm, Victor Luchangco

Hans is now chairing the group, with thanks to Michael Wong for his past service.

How often to meet?
Every 4 weeks, starting Jan. 7.
(Four weeks from now would be Christmas Eve.)
For what it's worth, that schedule will avoid PPoPP, NVMW, and ASPLOS.

Where are we?
No one has yet written anything up for "TM Lite".
Mike Spear "under water" until Dec. 7.

Remembering/clarifying where we are:
Tentative agreement to use atomic{...} syntax rather than executors (pass-a-lambda).
What restrictions to put on the code in atomic blocks?
  Victor: Herb would prefer something where "whether it's ok" is self-evident -- prefer error messages to performance anomalies.
Michael Scott: can always extend later.
Mike Spear: forbid bad things, or make behavior undefined?
  Victor: forbidding is better, if we can.
  Hans: undefined is more typical C++.
  Mike Spear: want life to be simpler for the implementor.
  General agreement: undefined (compiler can of course do more).

Michael Scott: what _is_ allowed in an atomic block?
Victor:
  ordinary reads & writes
  ordinary control flow
  no volatiles or atomics

no exceptions
calls to known-to-be-safe functions -- definitely including those of the current compilation unit (as in constexpr)

Library functions?
Victor: not yet?
General agreement: whatever constexpr allows
Michael Scott: that is, if constexpr says you can use foo() if you pass constant arguments, we would say you can use foo() (w/out necessarily passing constants)
Hans: that probably rules out vectors.

Hans: note that constexpr is a moving target. People are looking at vectors and even exceptions.

Semantics should be (a slightly less flexible version of) what we have for atomic_noexcept in the current tech. spec.
Hans: except nonconforming blocks will have undefined behavior, so the compiler is not required to check conformance.
Want to admit stupid global lock implementation.

Looks like we're in enough agreement that somebody should be writing. Michael Wong created a Google Doc for this. We should edit it as time is available.

Mike Spear's older lambda-based proposal (for reference):
https://docs.google.com/document/d/1ICmcrCdigq3ataoM2Jl7m19h_Sa3aE3KfU6AVkPyT-4/edit

Skeleton for a new atomic block proposal (edit this one):
https://docs.google.com/document/d/1GN_97YpPmxs9byKpzzlxusNtSapCFl6Bm9NdCExrl0U/edit#heading=h.tj9hitg7dbtr

Next meeting: 7 Jan., noon PST.

Informal notes for Jan. 7, 2019 SG5 Conference call

We did not have a quorum, and cut the call short.

Attendees: Hans Boehm, Victor Luchangco, Michael Wong

Brief discussion as to how to proceed when we finally have time.

HB: Looks like we may be able to proceed largely by deletion from current TS.

VL: Have to get the form right.

Agreement that we need to look more carefully at constexpr to understand the extent to which constexpr functions currently or eventually will cover everything essential for “light” TM.

VL: Maybe also want to add other visible functions?

Adjourned early.

Next meeting Feb. 4.