Document Number: P0569R0
Date: 2017-02-06
Authors: Michael Wong
Project: Programming Language C++, SG5 Transactional Memory
Reply to: Michael Wong <michael@codeplay.com>

SG5: Transactional Memory (TM) Meeting Minutes 2016/07/18-2016/10/10

Contents

Minutes for 2016/10/24 SG5 Conference Call

Meeting minutes by Jens

19:07 UTC

1.1 Attendees: Jens Maurer (minutes), Michael Scott, Victor Luchangco,
Michael Spear

1.2 Agenda was adopted unanimously.

1.3 Minutes are approved.

1.4 Action items
 - Jens: Defect 3 is still open with me.

2.1 Continue the calls?
Michael Scott: I don't get a lot out of the calls right now, but I fear
   the whole thing falls apart if we stop the calls.
Michael Spear: Agreed. Nobody is using the TS we have out there.
Victor: We ought to be productive. We should have a plan of action.
Jens: TM adoption will take time.
Victor: We have a publicity problem.
Jens: Someone should give a talk at CppCon, explaining e.g. a synced
   doubly-linked list.
Michael Scott: What about Splash? (OOPSLA is the umbrella conference);
Victor: The paper deadline for Splash is in April; conference is in October.
Michael Spear: What about Paul McKenney?
Victor: Yes, Paul McKenney gave a talk at the last CppCon.
Consensus: Keep the calls for the time being.
Action item (all): Find speakers for the conferences.

Michael Spear: If TM is not interesting without HTM, why not just use
   HTM-based lock elision?
Victor: HTM has limits, and we need the software fallback anyway.
Jens: It's easier to show a performance advantage with HTM.
Michael Scott: HTM-based lock elision is like synchronized blocks,
   but there are other advantages for atomic blocks.
Jens: Suggest to use the example of an LRU cache (scalability with/without TM)
   for a CppCon presentation.
(Michael Spear leaves.)

2.3 synchronized vs. atomics

Victor: We decided that we shouldn't preemptively remove features that might
be useful. But a synchronized block that goes irrevocable often is likely
to detrimentally affect performance for everyone, including atomic blocks.
So maybe separate atomic from synchronized blocks.
Jens: You need to redesign your program for concurrency anyway, so
you should go for atomics from the start. That makes synchronized
blocks dispensable. But I know there are strong opinions the other way.

20:00 UTC closed

Minutes for 2016/12/05 SG5 Conference Call

Meeting minutes by Michael
Michael WOng, Michael Scott, Mike Spear, Victor, Hans

Agenda:

1. Opening and introductions

1.1 Roll call of participants

1.2 Adopt agenda

Yes

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

Yes

1.4 Review action items from previous meeting (5 min)

1.4.1:

2. Main issues (50 min)

2.1 Review of Issaquah status and Discuss future work plan

Review of iSsaquah, about 400 NB comments, may be 80 % done,

Parallelism TS2: task block, simd vectors, vector execution policy, may benefit from annotations, allows for some orderign dependency between iterations

Concurrency Ts2: ostream sync
atomic views
fp atomics
synchronized value allows you to wrap things in locks
queues counters

Executor TS

C++ shared ptr gaining in prominence, some interaction with TM. these have atomic reference count, or DCAS, or just plain locks, so these can affect how we affect TM

shared_ptr +parallelism TS into TM TS 2?

shared_ptr become unusually racy, racing assignment, lead to decrement reference count in the wrong pointer, lead to dangling pointer, ownership, less ref count decrements slices of memory

they are for memory management, not for atomicity

1. shared ptr, implementation uses an atomic , it is not allowed inside any atomic blocks
if all accesses inside atomic transactions, then it should be easy
Spear: only matters in STM, which is same as the malloc case, in HTM it does not matter, TM needs to use it and book keeps it

Should just talk about enabling atomics inside atomic blocks
Hans: use_count is an issue

atomics , can we get by with the same implementation
WITHOUT CHANGING THE TX-UNSAFE ACCESS, if it is same as tx-safe access, then interaction among threads wont be a problem, this seems wrong, let me retry:
have tx-safe method, and make it friendly with tx-unsafe code, then its going to OK

Scott: hope to only need to make small change top shared_ptr library
Hans: use_count is a mistake in design, as it retrieves the reference count
Spear: can say use_count is not tx-safe
Hans: but it is used in non-tx code to observe state
Spear: tx increase count is OK, if it lowers count, then not ok
Hans: still have relaxed accesses to reference count
if multiple affect use_count, result is approx, because of weak ordering
people wanted to preserve use_count in single threaded case
Spear+victor: use this as a trial run for general problem of atomics

2.2 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

2.3 Continue General discussion on which part of TS we should keep (Synchronized vs atomics)

Notes from Oct 10:

https://docs.google.com/document/d/1RU4XaBH_sKW0MsLgoAby_oNAgiZKGExWQ5DiUSHQdkU/edit

Decided to keep both parts
Spear: how much of TS we want to keep if we just have synchronized blocks?

Scott: those are roads we go down only if we get pushed back

3. Any other business

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 Smart Ptr
https://groups.google.com/a/isocpp.org/forum/#!topic/tm/TJ2oUYO6bkU

We concluded to not go in this direction.

4.2.4 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:

N4573 2017-02 Kona WG21 Meeting Information

N4607 Toronto Meeting Information

4.3 Review action items (5 min)

None

5. Closing process

5.1 Establish next agenda

Resolve the atomic nature of smart ptrs as a way towards atomics inside atomic blocks

5.2 Future meeting
Next call:Dec 19

Past and future Meeting dates (offfset with WG14 CPLEX calls on Monday 1-3 ET)
Dec 19: smart ptr interface review, characteristics of smart ptr and atomic smart ptr, Maged and Michael

Jan 16:

Jan 30:Feb 6 is mailing deadline

Feb 13:

Feb 27: Kona Meeting

Minutes for 2016/12/19 SG5 Conference Call

Minutes by Michael Scott 19 December 2016

Start Time: Monday, 19 Dec 2016, 12:00 PM US Pacific Time (07:00 PM in GMT)
End Time: 1:00 PM US Pacific Time (duration: one hour)

Notes by Michael Scott.
The current secretary rota list is (the person who took notes at the last meeting is moved to the end)
  Torvald, Tatiana, Mike Spear, Maged, Victor, Hans, Jens Maurer,
  Michael Wong, Michael Scott

Agenda:

1. Opening and introductions

1.1 Roll call of participants

  Hans Boehm, Victor Luchangco, Jens Maurer, Michael Scott,
  Mike Spear, Michael Wong.

1.2 Adopt agenda

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

1.4 Review action items from previous meeting (5 min)

Michael Wong sent discussion materials on smart pointers; Hans sent follow-up.

1.4.1:

2. Main issues (50 min)

2.1 Resolve the atomic nature of smart ptrs as a way towards atomics inside atomic blocks

Jens: Is this hard?

Hans: Yes: atomic_shared_ptr might work ok, but plain (not atomic) shared_ptr may be quite hard. Strong isolation would require that we protect reference count operations, since there are queries to request the count. That protection would slow down nontransactional accesses a lot.

Jens: maybe just "roughen" the semantics inside transactions?

Hans: destructor timing anomalies may also be a problem.

Jens: is the destructor for shared_ptr transaction-safe?
Hans: needs to be

Michael Scott: how can we possibly require the destructor of the pointed-at object to be transaction-safe?

Jens: maybe require _all_ pointed-at shared objects to have transaction-safe destructors.

Michael Scott: defer execution of destructor to end of transaction -- actually _outside_ transaction? Hans: not sure that's always safe; might require access to state with limited extent.

Michael Scott: so what about requiring the implementation to resort to a global lock when you eliminate the last pointer to a shared object?

Hans: OK, but still worried about isolation wrt nontransactional accesses...
Michael/Jens: ... like seeing destruction and/or querried use counts happen one at a time from the perspective of nontransactional code.

Michael Wong: So how about Herb's atomic_deferred_ptr?
Initially meant to handle cyclic structures.
(Not garbage collection: happens synchronously with destructors.)

Michael Scott: Seems like a bind: can't defer destructors that need local state; can't execute them right away if they do non-transaction-safe things.

Victor/Jens: current semantics specify that destructors are called one at a time -- but in unspecified order if objects go out of scope at the same time.

Can we assume that any destructor that needs local state is transaction-safe? Probably not...

Michael Scott: lean toward requiring destructors to be transaction-safe if shared_ptrs to their objects are used w/in transactions.
Jens: That seems to imply isolation wrt to all other code.

But may require a new pointer name to avoid suprises when people try to compose things.

Jens: probably possible (but not desirable) to use a mini-transaction implementation of shared_ptr outside transactions when dealing with objects that have transaction-safe destructors.
Michael Scott: an alternative might be to give up on strong isolation: allow nontransactional code to see that destructors inside atomic blocks happen one at a time.
Whether that's acceptable is a judgment call.

Jens: maybe both "heavy" and "light" varieties of shared pointers? One defers destruction, but requires the destructor to be "deferrable" -- no use of state that won't be there at the end of an outermost transactions.

2.2 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

No action.

3. Any other business

None.

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

Michael Wong will try to summarize issues based on today's discussion.

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:

No action on any of these:

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.

5. Closing process

5.1 Establish next agenda

5.2 Future meeting

Next SG5 call: Jan 16

Past and future Meeting dates (offfset with WG14 CPLEX calls on Monday 1-3 ET)

Dec 19:
Jan 16:
Jan 30: Feb 6 is mailing deadline
Feb 13:
Feb 27: Kona Meeting

Minutes for 2017/01/16 SG5 Conference Call

Minutes by Mike Spear

Michael Scott, Michael Wong, Victor, Maged, Mike Spear, Jens Maurer
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)
(No action items)
2. Main issues (50 min)
2.1 Continue to Resolve the atomic nature of smart ptrs as a way towards atomics inside atomic blocks
https://groups.google.com/a/isocpp.org/forum/#!topic/tm/R91g34JNjT8
Michael Scott: We have some fairly sticky problems, where neither alternative is attractive for ordinary shared pointers inside of transactions. When last copy of object goes away, if we need to invoke the destructor, and it requires local state (which it might), it can't be deferred. But if the destructor does anything nontransactional, that's a problem too.

The second problem is that we want operations on shared pointers *outside* of transactions to keep their current costs, and remain strongly atomic with respect to transactions.

Victor: do we really want that?

Michael Scott: That's the natural desire (strongly atomic).

Mike Spear: are we concerned about the arrow going in the other direction (e.g., transaction sees intermediate state of a shared pointer)

Michael Scott: That's a concern too.

Maged: Is there an example of the local state problem?

Michael Scott: It would be bad code, but if there's a shared pointer to something on the stack, it can't be deferred.

Maged: If it's a shared pointer, where we don't control when object is destroyed, due to interleavings, then it's hard to imagine such a case.

Michael Scott: can only imagine it in a bad program. But it is technically valid to have a pointer to stack space, as long as the programmer keeps it safe.

Maged: That still seems unlikely

Mike Spear: It seems that this case can only happen if the shared pointer is on the stack, but outside of the stack frames of the transaction. Otherwise, it seems like the transaction would have leaked a shared pointer it created (to its transaction stack), and that seems to break atomicity/isolation.

Victor: We need a contrived example to show the complicated behavior. The "nontransactional destructor" part is easy to understand, but the other issue (pointer to stack) is difficult to grasp. It seems like the semantics could be "undefined".

Michael Scott: To paraphrase Spear: Hans is worried about deferring destructor to end of transaction because it requires local state that won't exist at end of transaction. But Mike says for that state to be an issue, it must not be visible at end of transaction, and if that is the case, then the shared pointer must have been created inside of a transaction, and it can't be visible until the end of the transaction.

Victor: Described situation with unnecessary-but-legal use of shared pointer. If we don't want two kinds of shared pointers, we have to be able to handle this case. This seems like Chandler's concern about atomic vs synchronized, where we need to have the good performance when we know the bad behavior won't happen.

Mike: Ok, an example is lock free data structures inside of transaction. Can transaction weaken the shared pointer operations in that case?

Michael Scott: It would be nice, but needs to be expressed formally.

Victor: we make multiple versions of a function already. That should influence the solution.

Maged: If shared pointer is not superfluous, it is questionable that this program is correct.

Michael Scott: Two more things to add to the discussion. First, in last meeting we floated idea of transactional shared pointer, that programmers could use instead. It would imply a bit more expense for nontransactional uses, but would guarantee transaction safety. Second, perhaps we're focusing on the "hard" question first, and we might have an easier time with atomic variables first, and then move to shared pointers.

<<Jens Joins>>

Victor: Transactional shared pointers probably should be "last resort". (all agree)

Victor: at a high level, atomic ints (and locks) should be quite easy.

Maged: If a transaction writes to atomic int x, then atomic int y, how to keep others from seeing intermediate updates?

Michael Wong: Can these writes communicate outside of a transaction?

Victor: Assumes the answer is *no*. If there is disagreement, we need to iron out the semantics first.

Michael Scott: so atomic read and atomic write behave like closed nested transactions, with all other atomic reads/writes behaving like atomic blocks if they interleave?

(all agree: atomics not meant to be form of inter-transaction communication)

Victor: then the semantics are easy. It's just implementation cost.

Maged: We can't increase cost outside of transactions

Michael Scott: That's the challenge.

Michael Wong: Making sure we aren't just talking about using atomics as shared counters.

Mike Spear: Should "memory_order_transactional" as a parameter to atomics be a "last resort" too?

Victor: wants to focus on not-relaxed. And he recalls limited use of memory_order_consume (per Hans), but not limited use of acquire/release and relaxed. They are used, e.g., in mutexes.

Michael Scott: Mike was proposing that memory_order_seq_cst wouldn't be the top of the lattice anymore, but should be "super duper transactions too".

Mike Spear: are shared pointers more or less likely to introduce bottlenecks than atomics?

Maged: acquire/release should have razor-thin overheads. (Victor: just a cache miss?). But if I atomically update x and y, how to prevent someone from seeing new X and old Y? If we don't have HTM, this can happen.

Victor: Inside of transaction, do acquire as it is, and defer release until outermost transaction.

Mike Spear: but that's still not atomic

Jens: Revisit claim of mini-transaction for each atomic access. The point to avoid is introduction of data race between update outside transaction and read/update inside of transaction.

Mike: but without HTM, how can we do that without increasing overhead for the nontransactional atomic?

Jens: We might be at a dead end. It looks like we can't put all the overhead on the transaction side of things. On Intel, atomics are just a single locked instruction. People's expectations are probably not that much less for shared_ptr, because it's a form of poor man's garbage collection.

Jens: Shared Pointer is not easier, because of the destructor.

Mike: we could say "mini transaction for all shared pointer operations" and "if the shared pointer has an unsafe destructor, then it's not transaction safe to use that shared pointer".

Michael Scott: Can a compiler see that the shared pointer has a safe destructor?

Mike: we've never discussed inheritance of transaction safety of virtual destructors, have we?

Maged: We could defer.

Jens: Defer could compromise atomicity of the destructor.

Mike: Not clear about the transaction safety of destructors.

Jens: Whether or not it is virtual is the key.

Michael Wong: If it's not virtual, then we don't carry the transaction safety of the destructor.

(discussion about inheritance, safety, and destructors)

Mike: Then the compiler can do it. And thus destructors are not a problem, and it really just reduces to "if we use transactions for all shared pointers, is the overhead too high"?

Jens: Thinks that works.

(Time up, time to adjourn)
- show quoted text -
None