Document Number: P0229R0
Date: 2012-02-12
Authors: Michael Wong
Project: Programming Language C++, SG5 Transactional Memory
Reply to: Michael Wong <fraggamuffin@gmail.com>

SG5: Transactional Memory (TM) Meeting Minutes 2015/11/02-2016/02/08

Contents

Minutes for 2015/11/02 SG5 Conference Call

Meeting minutes by Victor Luchangco

Minutes:

1.1 Roll call of participants

Attendees:
Michael Wong, Brent Hall, Torvald Riegel, Maged Michael, Victor Luchangco, Michael Scott

1.2 Adopt agenda

Adopted.

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

Already approved and published.

1.4 Review action items from previous meeting (5 min)

None.

2. Main issues (50 min)

2.1 Review TM for C meeting results

Presented by Michael Wong at Kona: very positive response. Perceived as one of the most exciting features. (Vote of 16 to 0, 1 abstention, to continue.)

Re: cancel: Don't put it in for now. (If it is in, use "cancel" rather than overloading "break".) And break should have commit semantics.

Re: optimized-for-synchronized: Also don't do for now: it's a quality of implementation issue, and possibly not worth the burden added to name-mangling. Could be added as a pragma if so desired by particular implementation.

Q: Can goto be used to exit a transaction? YES!

Big discussion about what should be allowed within atomics. For example, what about floating-point operation (because they have floating-point exceptions)? There are a number of tricky issues, and requires thought. Should think about this more. C says that signal handlers cannot assume anything about values other than sig_atomic_t types, so it's okay if we expose "intermediate states" of a transaction (or roll back). We just need to specify what happens for the sig_atomic_t values.

gcc has now implemented our C++ specification (except atomic_cancel)

2.2 Discuss Torvald's question on the reflector:
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/rzLypbHQB6w

Q: Why call std::terminate instead of std::abort when initialization of exception object in atomic_cancel blocks fails? (That is, std::terminate is specified in the first bullet of 15.2 when throwing an exception that supports cancellation within a atomic_cancel block, but elsewhere, std::abort is specified.)

We couldn't remember any reason why this difference should be there. It might be an oversight, but we should check with Jens.

2.2 Review on_commit proposal
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/RFA2j5-ojNU

We (Mike Spear, Michael Wong, Victor Luchangco) have a draft of this, but it was finished too late to add to the meeting (and there wouldn't have been time to discuss it at Kona anyway). So we should just revisit this issue and make sure we have a draft by Feb 12, the next submission deadline.

2.3 If time permits: retry discussion
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/qB1Ib__PFfc

This is led by Hans Boehm, so wait for him.

3. Any other business

No other business.

Next call : Nov 30

Minutes for 2015/11/30 SG5 Conference Call

Meeting minutes by Maged Michael

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

Agenda:

1. Opening and introductions

1.1 Roll call of participants

Mike Spear, Victor Luchangco, Maged, Michael Wong, Hans, Torvald,

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)

(Not verbatim. Paraphrased. --Maged)

2.1 Time for future meetings, and logistics

Check with Michael Scott about going back to 3pm ET in the new year.

2.2 Discuss Torvald's question on the reflector:
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/rzLypbHQB6w

M W to Torvald: Did we get an answer from Jens?
Torvald: Jens comment was that the suggestion is reasonable (abort instead of terminate)
M Spear: It is a larger issue. It should be std::abort not std::terminate
Victor: We should get a definite answer from Jens on whether this was intentional or an oversight

AI: Check with Jens on whether writing std::terminate instead of std::abort was intentional or not

2.3 Review on_commit proposal
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/RFA2j5-ojNU

M Spear: We have something adequate.

AI: (M Spear Victor) Follow up with the group with the proposal

2.4 If time permits: retry discussion
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/qB1Ib__PFfc

Hans & Victor: Retry is inconsistent with cond vars in some cases
Hans: Nesting is problematic with retry. E.g., communication inside transactions

M Spear: All of the uses of cond vars in parsec needed trivial code refactoring to use retry, also trivial to refactor to use the await mechanism (Stanford) which simpler than retry (awaits update of a specific variable).
Hans: This is orthogonal to the nesting issue
M Spear: Yes.
M Spear: In parsec there was no nesting with retry
Hans: Suggesting prohibiting nesting with retry
M Spear: Example with two bounded buffers that motivates composability of nested retry transactions.
Hans & M Spear: Some operations cannot complete without concurrency.
M Spear: That's a consequence of the use of transactions not retry.
M Wong: We'll need to continue this discussion.

3. Any other business

Note: Need to submit any new papers by the Feb 12 mailing deadline for the Jacksonville meeting.

MW: Checking on implementations: gcc, clang

MW: Suggests putting a position paper in the Feb 2016 meeting

MSpear: Is synchronized() viable by itself for 2017?
MWong: Yes
Hans: It might be. There might be issues with conditional synchronization.
Victor: We might not have experience more than we had before.
MSpear: Concerned that pushing something minimal might be counterproductive.
Victor: It is OK to wait for years of experience.
Torvald: Agrees
M Wong: There seems to be a consensus that we should wait for more experience. Upgrading a TS to standard might be OK in a minor release.
M Spear: We should have a solid TS by the time of the 2017 release.
M Wong: Will check if there is a Technical Report designation for reporting our experience with the TS
Hans and M Wong: We can check in Feb if there is external pressure to put TM in the 2017 standard.

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

4.2 Review action items (5 min)

AI: (M Wong or Torvald) Check with Jens on whether writing std::terminate instead of std::abort was intentional or not.
AI: (M Spear Victor) Follow up with the group with the proposal.

5. Closing process

5.1 Establish next agenda
5.2 Future meeting
Next call : Jan 25

Victor will host a call on Dec 14.

Minutes for 2015/12/14 SG5 Conference Call

Minutes by Mike Spear +Victor

> Hans, Michael Wong, Torvald, Mike Spear, Tatiana,Michael Scott, Jens Maurer, Victor, Maged

Hans, Michael Wong, Torvald, Tatiana,Michael Scott, Jens Maurer, Victor, Maged, Mike Spear

> Agenda:
>
> 1. Opening and introductions
>
> 1.1 Roll call of participants

Maged, Michael Scott, Mike, Victor, Hans

>
> 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)
>
> AI: (M Wong or Torvald) Check with Jens on whether writing std::terminate instead of std::abort was intentional or not

Nobody on the line is able to comment on this AI.

> AI: (M Spear or Victor) Follow up with the group with the proposal

Victor sent a new draft of transaction_defer to the group. Victor made many edits to the previous draft from Mike, which had been edited by Maged and Michael Wong.

There were a few terminology changes, such as use of "defer" instead of "delay", and "commit actions" instead of "oncommit actions"

> 2. Main issues (50 min)
>
> 2.1 Review on_commit proposal

> https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/RFA2j5-ojNU
> (also see email just sent a few minutes ago)

Some references are still required.

Mike asked if the comment in section 6 about adding this feature to std::mutex is worth retaining.

Victor suggests we focus on our TS and don't move out of scope. Michael Scott agrees, based on feeling that the main purpose is to defer what "can't be done in a transaction".

Mike mentioned that there is another reason for std::mutex, which is to delay operations until the lock is released so that the critical section is smaller (example: logging, or zeroing out memory).

General consensus that it's not worth our stepping out of the TM space.

Other features/requirements of deferred actions? If not, we will work on the examples a little bit more, after which the document should be ready for another review.

AI: Everyone should give feedback on this document, so that Victor and Mike can produce a final draft in time for the paper deadline on Feb 12.

Hans: Do we need TS wording in the document? It's not necessary, but could speed things up.

Victor: Not ready to commit to that: We want Jens around before we have conversations about wording.

Hans: A first attempt would help.

Victor and Mike will try to have draft of wording for the TS, in addition to a paper that we can present to the committee.

> 2.2 Retry discussion
> https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/qB1Ib__PFfc

Hans: less convinced that the problems he worries about are retry; rather, they are about communication among transactions, and this raises some questions.

Consider a program that uses a function call that logically behaves as a purely functional operation, but it does so by interacting with another thread (i.e., posting a request, awaiting a response). Could retry() make this easier? We can't call this function (call it f()) in a transactional or synchronized context, because it is not atomic. Do we need some facility to let the programmer write code that can't be executed in a transactional context (atomic or synchronized) due to inevitable deadlocks?

Victor: asked for clarification on the program behavior. Concern that the program is "bad code".

Hans: f() is reasonable in a different context. There is no way to say in the interface that it only works in nontransactional contexts.

Victor: if I knew when I wrote code that I needed to wait for communication, then what I should do is label the function transaction-unsafe. We allow that, though we can't check that it has been done correctly for those operations that are not unsafe by definition.

Hans: the concern is communication via atomic blocks.

Victor: but we know that it's transaction-unsafe. At that point, we could treat this as a warning from synchronized blocks.

Hans: agree, but it seems like two cases: one uses a helper thread and C++ atomics, and it just works from synchronized blocks. In the second, with atomic transactions, it doesn't work from synchronized or atomic blocks. How do we make the distinction at the interface level for f().

Victor: do we need to put transaction unsafeness into the type system? Victor leans towards warnings, not types/errors.

Hans: maybe we could dynamically test, and it seems there is a relation to commit handling, where we'd want to have a way to know if we are in a transaction or not.

Victor: agreed, we should have that feature. Any concerns?

Mike: if we had it, could the compiler accept a transaction_unsafe function that had code of the form "if (!in_transaction()) unsafe_thing();"

Michael: Clarified the meaning of the function and its impact on compiler analysis.

Victor: imagine a future where the compiler is allowed to take advantage of this.

Victor: let's follow up with Michael Wong about where to put this.
> 4.2 Review action items (5 min)

AI: (M Wong or Torvald) Check with Jens on whether writing std::terminate instead of std::abort was intentional or not (deferred from this meeting)

AI: Everyone should give feedback on this document, so that Victor and Mike can produce a final draft in time for the paper deadline on Feb 12.

AI: Victor follow up with Michael Wong about how to add an "in_transaction()" predicate to the TS.

<<Mike Spear departs>>

Completing the minutes inline below:
- show quoted text -
Hans: in_transaction is related to commit actions in that we could implement commit actions if we had an in_transaction function.

Victor: it's a bit tricky because we don't know about nesting. It might be possible, but the concepts seem logically orthogonal (at least each can stand without the other).
The question is whether it makes sense to have such a proposal just with in_transaction.

Hans: there's precedent for half-page papers.

Tentative agreement to have in_transaction as a separate proposal, but will ask Michael Wong for his opinion.

Hans: retry will retry only innermost transaction, giving us closed nesting semantics.

Victor: Yes, I think we should stick to closed nesting unless we have a clear case that we need to do otherwise (e.g., for malloc/free).

Hans: For cases where we have communication, marking transaction-unsafe seems fine. Does closed nesting make sense in all the other use cases?

Victor: open to there being other use cases, but they should be code we actually want to enable people to write. If a function requires communication to be correct, it should be labeled transaction-unsafe. Not doing so would be bad code.

AI: think about other use cases for retry with nested transactions. In particular, is closed nesting sufficient (for code we actually want people to write).

We should consider writing up a concrete proposal for retry (but no AI on it yet, since no one has volunteered to do it).

>
>
> >
> > 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
> >
> > 4.2 Review action items (5 min)
>
> AI: (M Wong or Torvald) Check with Jens on whether writing std::terminate instead of std::abort was intentional or not (deferred from this meeting)
>
> AI: Everyone should give feedback on this document, so that Victor and Mike can produce a final draft in time for the paper deadline on Feb 12.
>
> AI: Victor follow up with Michael Wong about how to add an "in_transaction()" predicate to the TS.
(This AI is modified to just check with Michael Wong about writing up a separate proposal to amend TS just for in_transaction?)

AI: think about other use cases for retry with nested transactions. In particular, is closed nesting sufficient (for code we actually want people to write).

Minutes for 2016/01/25 SG5 Conference Call

Minutes by Hans Boehm

MINUTES for 1/25/2016 SG5 Phone conference

Attendees

Hans Boehm
Brett Hall
Victor Luchangco
Jens Maurer
Maged Michael
Michael Scott
Michael Spear
Michael Wong

Approved minutes for publication.

Michael Wong: Should always call abort() instead of terminate(). Should keep this item, and get Jens to confirm this was the intent. [Jens joined later, see below.]

Michael Wong: Attendance at physical WG21 meetings currently no longer critical.

Victor: in_transaction proposal still needed.
On_commit could still used wording proposal.

Review in_transaction proposal. Do we allow transaction-unsafe constructs after testing that we're not in a transaction? Can we use transaction-unsafe constructs inside transaction-safe functions this way?

Michael Scott: What's your favorite use case?

Michael Spear: Perform retry only in transaction. Make decisions about handling exceptions.

Michael Scott: Static analysis required? Where's the limit? Use a special construct instead?

Victor: Must properly define accepted programs, so it can be relied on.

Do we need to distinguish synchronized blocks? Cancel vs. commit?

Can pass lambdas to library functions. Avoids the static analysis issues.

Discussion of whether the specific conditions should be encoded in function name, runtime parameter, or template parameter.

Jens: If you generate two function versions, depending on whether you're in a transaction, then it seems strange to make this a runtime check. Each version already knows the answer statically.

Jens: Do we really have good use cases?

Michael Scott: Preprocessor macro? Never mind.

Michael Spear: Has real example, but motivated by gcc non-compliance for strings in exceptions.

Brett: No real use cases in their code.

Victor: Brett's code also uses a somewhat different model.

Papers for Jacksonville: on_commit paper.

TM in C++17?

Previous discussion concluded that we should do this only if there is outside support?

Jens: Poll plenary for TM in C++17.

(AI) Michael Wong: Good idea. Michael will pursue.

(AI) Michael Spear to start thread on motivation for in_transaction.

Jens confirms we should change std::terminate() to abort(). Should also address other cases in which exception handling invokes std::terminate() from 15.5.1 .

Michael Scott: All terminate() calls in transactions should really call abort()?

Jens: Yes, that is the logical consequence, though not enthusiastic about it.

Michael Scott: Do we need to update all mentions of terminate()?

Jens: One bug, unclear we want to update all references.

(AI) Add this as a bug.

Action Items:

Michael Wong (2): Poll plenary on C++17 inclusion. Add bug for remainining terminate() calls in transactions.

Michael Spear: Start mailing list thread on in_transaction().

Minutes for 2016/02/08 SG5 Conference Call

Minutes by Michael Wong

1.1 Roll call of participants
Michael Scott, Maged, Mike Spear, Michael Wong, Brett Hall, Jens, Victor

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)

AI: Michael W to send note to C++ reflector asking for interest. Done, no response. But resent to all@lists.isocpp.org, parallel@lists.isocpp.org, ext@lists.isocpp.org
AI: M Spear or Victor Follow up with the group with the revised on_commit proposal, ready for publication. Michael W sent the number. and paper should be sent to John Spicer.

AI: M SPear to initiate use cases for in_transaction. Done. Thanks to Brett too.

2. Main issues (50 min)

2.1 Logistics on whether to propose TS for C++17
Group's feeling is no due to needing more user feedback, waiting for when GCC 6 starts to support it.
Unless others on the committee register a strong interest.
Look at any feedbacks

2.1 Review on_commit proposal
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/bS7u2S8RGQU
To be discussed offline

2.2 DIscuss in_transaction proposal from Dec 14 call
https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!topic/tm/fn73ZddwmoA
overload tx-safe and tx-unsafe version rather then having an if inside of the transactions
Jens: current status is cannot overload on tx-safety; just did not see the utility at the time
can do away with compile-time analysis that we had a few emails about
overloading is also compile-time analysis
allowing overloading in this case may make a few things more ambiguous
taking the address of a fn, one safe and one not safe, it does not disambiguate it, must decide when taking the address which variant is needed. So we dodged this issue by saying you can't have overloading

Victor: in another call, can have an if in_transaction predicate, but no guarantee about safety
can address implementation complexity but not whether motivation is suffiicient
Michael S: if cmplicated predicate compiler can analyze, predicate replies true only if in a transaction, then if you are a compiler, I must know I am not in a transaction, then have tx-unsafe code in there and it is fine.
Victor: then what went wrong is clear; now static checking is straight forward; if programmer is wrong that it is outside of tx, the overhead to write this not a big deal
Jens: has not made the language easier; don't have enough info to make decision; most prominent is logging within a transaction-safe tx, and postponing the output by means of on-commit handler
Victor, only want to write function once, and use it inside and outside tx (if inside tx, dont need to bother with the locks), so it is also about software engeinnering
Michael S: pass sync info in as a lambda
Jens: first user takes lock, and enters tx; second user does not take lock, and enters tx, now we have a race
Michael Spear: do we want 2 different versions with 2 names
memcpy example wants to fast as possible outside of tx, but inside tx just want it to be safe
Torvald memcpy just works in GCC, great job
want to write my source code once, and optimize it in 2 calling context
Why do we have this talk:
1. nice in C++ to give low and high level from Hans
2. the assert if in transaction, then call retry is something we want in future, things that can only be done inside a transaction, dont want a type on a function such that it can only be called inside tx context
Michael: I find that convincing
Victor: having impose overhead, dont want to have code that run when I know it is local, and when I know it is touching shared things
arrange the rules of synchronization not on the version where the compiler cant determine easily that it is local
dont want to think where I am when I use this function
Spear: Torvald saids this reduce to halting problem; library std:tm where u send a lambda to execute code;
Jens: there is core language effect when there is code inside tx; machine code has to have read/write sets
still need tx-safe annotations, and compiler has to generate code; now we switch into this other mode vs hiding in some library interface
we did right in balancing this
Agree this is our intuition
torvald does not want compiler to unparse expression to see if it is in tx or not
trying to put non-tx -safe code in a tx that is tx-safe is not the right way
agree
runtime knows inside a tx or not, so just ask it

Spear: with Torvald syntax, it would be lexically scoped block
a first level language feature using 2 new keywords

would have a block asserting you are not in tx

Jens: undecided on issue; postpone; lets see where it stands on pain point
Spear: if we have library fn that took a lambda then we would not have to change, and just do it in library
Jens: personally dont think lambda interface is pretty
spear: do we have solution for tx-safety for std functions
Jens: no; cannot tell 2 fns name the same
std functions is on our agenda, need to sched work on it; just telling impl to make it work

Do not think we have an in_transaction proposal at this point.

We wont make progress unless we have specific and have something written down.
doubt on whether we need it
also some doubt on implementation
overload resolution, lambda to library; or separate block option with another keyword
describe all options laid out in the papers
show use cases it solves
lists their know implementation +/-
Even a paper that dont know all the answers but record the debate and progress is useful
AI: Michael Spear+Victor to work on getting something to paper
Michael Scott. Michael Spear.

- show quoted text -
Next call. Feb 22