Document Number: P0685R0
Date: 2017-06-19
Authors: Michael Wong
Project: Programming Language C++, SG5 Transactional Memory
Reply to: Michael Wong <michael@codeplay.com>

SG5: Transactional Memory (TM) Meeting Minutes 2017/01/30-2017/06/05

Contents

Minutes for 2017/01/30 SG5 Conference Call

Meeting minutes by Victor
Participants: Michael Scott, Hans, Mike Spear, Victor, Maged
Adopted agenda, approved minutes.

>
> 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
>
> Do we wish to publish any papers on this or any other subject for the mailing?

Mike Spear (summary of status):
Should we try to support shared pointers or atomics first within transactions?
We could have a special kind of transactional shared pointer? We said we didn't want this if we could avoid it.
Could we have mini-transactions for all shared pointer operations? There are two issues.

Maged: Why not defer destruction?

Victor: I think when destruction occurs is guaranteed by the semantics.

Hans: Yes, destruction is guaranteed to occur at a particular time. More worried about destructor using data structures on the stack than referenced-counted shared pointers on the stack.

Mike Spear: May be better to consider the following case: imagine an execution service that can execute several tasks. It may happen in parallel outside in a transaction, but within a transaction, it would happen in some order. We use shared pointers to keep track of the tasks. When you come back from execution service, all shared pointers should be cleaned up?

Michael Scott: I think we agree that it isn't semantically correct to defer destruction. So what can we do to make normal shared pointer operations interoperate with shared pointer operations within transactions without making normal operations egregiously expensive?

Mike Spear: Two levels of answers:
(1) As long as transactions must be turned on explicitly, everything should be free as long as we don't turn them on.
(2) Is it acceptable to say, "As long as you don't touch this shared pointer with a transaction, it will be cheap. But after that, it may be expensive." Then we can use technology similar to "exploded locks". It should be possible to do this reasonably cheaply as long as the shared

pointers aren't used within transactions.

Hans: Java thin-lock stuff depends on safepoints, so may not translate to C++.

Mike: Are shared pointers always thread-safe?

Hans: They are thread-safe in the weak sense. But they can misbehave badly when there are data races to shared pointers. (This is a program bug, but it is particularly nasty.)

Mike Spear: I tend to conflate TM in C++ with gcc implementation. It may be with a less ambitious implementation, we could solve these problems more cheaply. If the TM implementation is either just synchronized blocks or HTM+fallback to single global lock. Then it just works with shared pointers. It's only integration with STM that's a problem.

Victor: But we may not have HTM available. So don't we need to support STM?

Mike Spear: But current STMs underperform single global lock implementations (single-threaded) by a factor of 2 or 3, at least, so we would need to have a lot of parallelism available before this is a problem.

Maged: Returning to the question of deferring destruction, what if we adopt a redo log approach, so that destruction happens within transaction, but their effects are deferred from being visible outside the transaction.

Mike Spear: Something about gcc implementation overwriting virtual table when destroying

Hans: Possible way out: In some TS, we have atomic_shared_ptr, and weak_shared_ptr, which could be used with atomics. We could repurpose those to use with transactions, as there may not be much additional overhead. But they are expensive enough that people might not want to use them anyway. (Can find by searching for atomic_shared_ptr.)

Victor: But this still doesn't make shared_ptr operation transaction-safe. If we resorted to having transactional shared pointers, then it makes sense to perhaps use these ones, but it doesn't allow code that uses ordinary shared_ptrs in transactions, which limits the applicability of transactions.

Michael Scott: I like Mike Spear's suggestion about inflation. Someone should look in to that.

Hans: May not be trivial because we don't have safepoints to leverage (as we do in Java).

Possible action item to look at this: Hans and Mike will try, but no promises.

Minutes for 2017/02/13 SG5 Conference Call

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

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

Agenda:

1. Opening and introductions

1.1 Roll call of participants

Maged, ,Hans, ,Michael Scott ,Mike Spear

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)

Hans and M Spear to look at shared_ptr implementations.

No progress yet.

2. Main issues (50 min)

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

Hans: The high-level question is can we make shred_ptr safe for transactions.

Hans: Will look at the shared_ptr implementation.

M. Spear: In gcc implementation seems to boil down to operating on an atomic integer.

Hans: Looking at he code, seems all the implementation is in a header file.

Hans: Not clear that we have a good idea to be backwards compatible. Although not a big concern for us, because we need to recompile.

MLS: We don't need to recompile everything if the linked code doesn't include TM.

Hans: The conclusion is that one problem is if pass an inflated transactional shared_ptr to a legacy library.

M Spear: We seem to be reaching the conclusion that only shared_ptr-s with template args with tm_safe d'tors are safe to use in transactions. But we can defer the ref count to the end of the transaction.

Hans: This more complicated by weak_ptr

Maged: It seems that we can't get away with keeping the implementations of non-TM shared_ptr and weak_ptr unchanged.

Hans: There are problems (undebuggable data races) in the current shared_ptr and weak_ptr. Maybe we have some leverage to add some cost for compatibility with TM but fix this problem.

M Spear: And this doesn't shouldn't require vendors to implement the whole of TM.

2.2 Do we wish to publish any papers on this or any other subject for the mailing?

2.3 Discuss defects if any work done since last call

Minutes for 2017/03/13 SG5 Conference Call

Minutes by Victor

Minutes from meeting on 13 March 2017:

Attendees: Michael Scott, Mike Spear, Victor, Hans

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

Michael Scott: How good is the gcc implementation of TM in C++? Are there any others?

Mike Spear: gcc implementation isn't great, and it doesn't seem like it's high priority to gcc to fix problems. No really good alternatives. gcc is TinySTM + quiescence (for privatization safety).

Mike Spear: project by grad students: if only have synchronized blocks, what language support would you need? I was thinking that synchronized blocks was just lock elision, but if you want any decent performance, you need to be able to determine when it's safe to execute as a transaction. (Otherwise, you always have to serialize all synchronized blocks.)

Victor: Mark Moir's "colors" for synchronized blocks was really intended to address this issue. But I pushed back about it, asking how it's really different from increasingly fine-grained locking.

Michael: could we have "summer of code" project?

Victor/Mike: we need well-defined projects

Hans: Lock elision (i.e., Intel's HLE) doesn't perform very well, but I don't know why. Is there some good explanation

Mike: Issue with HLE is that it has a single policy. For example, if you fail, it doesn't try again—it falls back immediately.

Michael Scott: it's compatible only with nonscalable critical sections.

Hans: Shouldn't that result in at worst 2x performance?

Michael: When someone fails, they take the lock and kill everyone doing a HW transaction.

Mike: Beginning/ending transaction are expensive (2 or more times the cost of CAS).

Hans: CAS or SC can be quite expensive (and unpredictable).

Michael Scott: Pushing things forward requires someone to work on this, and no one is doing this.

Mike Spear: I have a masters student doing the work previously mentioned, but only on synchronized blocks so far, no cancelation.

Victor: Haskell? Is there anything to learn?

Michael Scott: Not sure there is anything to learn, because transactional state and nontransactional state is statically separate, performance doesn't really matter because transactions essentially serialize, and that's okay.

Mike Spear: I agree with Michael: short of people using TM, there's not much we can do. So we need to make it more accessible.

Atomic vs. synchronized made us focus on cancelation, rather than what the programming issues are, of which cancelation is only one. Let's get people using speculation, under the hood.

Victor: Transactions should play well with others.

Michael Scott: One could have std:tm-lock-guard, which says here's a lock, but you might try to elide it.

Also, in Haskell, retry is really popular: it is what people use for condition synchronization.

Mike Spear: What about orElse?

Michael Scott: I think retry is used a lot, but orElse not so much.

Hans: We see less and less condition synchronization exposed to programmers because they tend to be embedded in lower-level code as libraries improve.

Michael: retry gets used in Haskell in places we don't think to use it in C++: scheduling is done by changing a flag that some thread is retrying on.

Spear: I will try to have more info about what we have been doing in LLVM (with his grad student) for the next meeting.

Next meeting is March 27.

Minutes for 2017/03/27 SG5 Conference Call

Minutes by Michael Wong

Hans, Jens, Michael W, Michael S, Victor, Maged

Agenda:

1. Opening and introductions

1.1 Roll call of participants

1.2 Adopt agenda

Yes with addition of 2.2 on Kona review

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)

A: llvm TM impl

1.5 Call schedules

April 10

April 24

May 8

May 22

June 5

June 19: Mailing deadline June 19

July 3:Victor away,

July 10 C++ Std meeting Toronto

2. Main issues (50 min)

2.2 review of the Kona meeting
pushed out integrated C++17 draft to be electronically voted
to be voted in time for Toronto, and forwarded for Geneva

Lots more corrections on parallel algo, but it is looking better

SG1 stuff is queued behind C++17 stuff

hazard ptrs and RCu, concurrent counters and queues, latches

executors heavily reviewed

discussion on thread local and cpu counter updates

Hans: SC is obtained with right usage restrictions instead of fences

Jens: not sure we need the same apis of counters

concerned there is enough interaction between lewg and its viewpoint on library design and concurrency viewpoint

Jens We have a DIS, out for vote by the NBs, mandatory 2 month translation period, before 3 month ballot, so have 5 month delay

requires immense throughout to get through 1400 pages

may not make it in time for Toronto

if there is no NO then DIS can be published

if there is any technical comment from DIS, then we need to go to ballot again, through an FDIS 2 month yes/no (no comment allowed)

can work on updates on C++20 in Toronto

can we take comments as DR

module did not make it to PDTS

coroutine made it PDTS

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

review of last call by Victor
lessons in Haskell, not as concerned with perf
Michael: what topic for future
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)
3. more papers?
4. Issue 1-4 paper updates to current TM spec
5. std library

Minutes for 2017/04/10 SG5 Conference Call

Minutes by Hans Boehm

1.1 Roll call of participants
Hans Boehm
Victor Luchangco
Jens Maurer
Michael Scott
Michael Spear
Michael Wong

New secretary rota: (Please correct if I didn't get this right)

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

1.5 Call schedules

April 10: future issues list

April 24

May 8

May 22

June 5

June 19: Mailing deadline June 19

July 3: July 10 C++ Std meeting Toronto

2. Main issues (50 min)

2.1 future issues list:

1. llvm synchronized blocks

Michael Spear:

Approach: Backend compiler support only. Pass on bitcode. Not a clang modification.

Michael Scott:

What about negative performance impact, even without front-end changes?

Michael Spear:

Shouldn't happen. Represent transaction body as lambda.

TM-safety represented as annotation.

Michael Scott:

Why not lock_guard-like construct?

Michael Spear:

1. Easier just to work on a lambda.
2. Difficult to implement correct lazy STM in gcc. Hard to tell whether stack variables are transaction-local. Longish example followed. Lambdas make it much easier to tell what's transaction-local. Lazy updates important for hybrid TM.

LLVM loses annotations from declaration in the bitcode. Leads to overhead.

Currently proof-of-concept, not ready for users.

Exclusively looking at synchronized blocks. 20 line runtime library if you don't really want to support TM. Can't support self-abort. Volatile access immediately makes transaction irrevocable.

Very strong arguments for annotating functions called from synchronized blocks. Currently clone lambda TM arguments, called functions, and annotated functions.

Determining whether there is a clone in another translation unit requires hash lookup and indirect call.

Annotation does not imply transaction-safety. Only whether a clone should be generated.

Michael Scott:

Could this turn into future simplified direction?
Could lambdas interoperate with future syntax extensions?

Michael Spear:
Yes. Restricts return from function surrounding transaction.

Victor:
What's lambda return type?

M Spear:
Currently void.

M Scott:
Exceptions?

M Spear:
Caught in library.

M Scott:
Lambda or std::function?

M. Spear:
Passing lambda as std::function. Not examined very carefully so far. Also has
C API.

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)

Jens:

Summary: Want a shared_ptr without performance disadvantage outside transactions,
but can correctly work with transactions. Something has to give. We've been
trying to understand what.

Need a paper describing constraints.

M. Wong:

Atomics inside atomic blocks?

Jens:

Fundamental question: Do atomic blocks stay atomic?

Hans:

Another issue: Preserving atomic lock-freedom for signal handler safety.

M Spear:

Minimal progress guarantees for hardware transactions would be be nice?

Can probably address this for lazy TMs by using a double-compare, single-swap to write back each word.

2. more smart ptrs?
3. more papers?
4. Issue 1-4 paper updates to current TM spec
5. std library

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

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

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 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)

5. Closing process

5.1 Establish next agenda

5.2 Future meeting
Next call:April 24 (tentatively cancelled)
May 8

Minutes for 2017/05/08 SG5 Conference Call

Minutes by Maged Michael

1.1 Roll call of participants

Victor, Maged, Michael Spear, Michael Wong

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)

None

1.5 Call schedules

April 10: future issues list

April 24: Discussed Future Issue #1

May 8: Discuss interaction with executors

May 22:Discuss Future Issue #2 on papers

June 5: Review any papers we write or Discuss Future Issue #3

June 19: Mailing deadline June 19

July 3: July 10 C++ Std meeting Toronto

2. Main issues (50 min)

2.1: Interaction with Executors

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

M Wong (describing the work on executors): Subgroup of SG1 combining the features of multiple proposals. Biweekly calls for discussions for about a year. Concerns in Kona: Rationale for executors. In response prepared an explainer document for p0443. In TM context, interaction of execution agents with synchronized and atomic blocks with their compound statements.

M Spear: Question about compilation. It seems that some environments will require compilers to generate multiple versions appropriate for different code paths. This may simplify things for TM.

M Wong: This is planned to be done statically.

M Spear: If executors are intended to be general purpose to encompass distributed shared memory, it would cover TM.

M Wong: Now the scope is limited.

M Wong: Do we want to add anything to the executor specification?

M Spear: One feature would be to have a way to specify that a function passed to an execution agent is transaction_safe.

Hans: Tx executions are orthogonal to other code.

M Spear: Can use executors to say try this transaction 5 times before falling back on a lock. How to use HTM if available.

M Wong: Tx safety can be one of execution properties

M Spear: There research proposals where HTM is used to sandbox code, so they don't want to fall back on locks.

M Wong: Do we want to align TM with executors?

Hans: There are executor issues that need to be discussed in SG1 about synchronization, e.g., locks, doing different things to do acquire locks. How to block depending on what kind of execution agents?

M Spear: There seems to be a lot of gotchas if someone want to pass data already locked to be unlocked by the execution agent or passing by value can mess with NUMA allocators.

Hans: In the case of TM we want the code to be executed synchronously.

AI: M Wong and M Spear write some points about the interaction of TM with executors.

2.2 future issues list:

1. llvm synchronized blocks

- show quoted text -
- show quoted text -

AI: M Wong and M Spear write some points about the interaction of TM with executors.

Minutes for 2017/05/22 SG5 Conference Call

Minutes by Michael Scott

1.1 Roll call of participants

Michael Scott, Michael Wong, Mike Spear, Victor Luchangco

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)

1.5 Call schedules

May 22: Continue any executor feedbacks; Discuss Future Issue #2 on papers

June 5: Review any papers we write or Discuss Future Issue #3

June 19: Mailing deadline June 19

July 3: July 10 C++ Std meeting Toronto

2. Main issues (50 min)

2.1: Interaction with Executors

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

Michael Wong and Mike Spear were hoping to get a chance to look at
these, but didn't.

Michael W. forwarded a slide deck introducing executors.
Most of the meeting was devoted to going through this deck and providing comments.

Executors serve as an in-between agent between specification of work and
its actual execution. Designed to enable heterogeneous computing in C++.
Attempting to bring together several previous proposals (Mysen, Hoberock
et al., Kohlhoff).
Generalization (big generalization) of things like async()
Can create one or many
Can pass args and optionally return results
Can run immediately, "post" for later execution, or "defer" for

later execution (by self, if already running on a suitable thread)
In short: specifies when, where, and how to run code.

Open question: how does all of this interact with locks?
Also: how uniform can the interface be? -- how much can be pulled into this framework?
Could transactions be re-cast as a style of executor?
Do executor functions need to be labeled as transaction safe/unsafe?

Michael Scott: More generally: can executor functions block?
(They can't in Java.)
Worried about deadlock, "viral" property propagation (in or beyond the type system) -- analogues of the nested monitor problem, or of the management of punctuated transactions.

Mike Spear: Another issue: if something is supposed to run on, say, a GPU, when and how is the GPU code produced?
Michael Wong: CodePlay is using fat binaries.

Victor: simple examples would be very helpful in the presentation -- including examples that ship to the GPU.

Mike Spear: composition seems like a big problem. Can the GPU call back to the CPU?

Michael Wong: data movement is also a problem.

Mike Spear: do executors subsume lock_guard?

2.2 future issues list:

1. llvm synchronized blocks

Mike Spear has been working on this (see 10 Apr. minutes).

[Ran out of time for the commented-out items below; preserving for future reference.]

Minutes for 2017/06/05 SG5 Conference Call

Minutes by Michael Spear

1.1 Roll call of participants

Victor, Michael Scott, Hans, Mike Spear, Michael Wong

1.2 Adopt agenda

Hans: should we also discuss a push to get technical specifications into C++20? Herb has not specifically mentioned TM, but it might be worth considering. There have been discussions about moving the concurrency TS forward, for example.

Decision: discuss before Executors.

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 W. hasn't prepared more about Executors, no other action items.

1.5 Call schedules

April 10: future issues list, Discussed Future Issue #1

April 24:No Call

May 8: Discuss interaction with executors

May 22:Continue any executor feedbacks;

June 5: Review any papers we write or Discuss Future Issue #2

June 19: Mailing deadline June 19

July 3: July 10 C++ Std meeting Toronto

2. Main issues (50 min)

2.0 Relationship between TS and C++20

V1 of concurrency TS had future extensions, atomic_shared_ptr, latches, barriers. There is a push to get all of them in, though there may be some push-back on futures. Hans is concerned that futures don't work well without executors, since you can't specify where the continuation runs. There is a possibility of blocking the executing thread to run the continuation, and there is difference of opinion about that special case.

Ranges, Networking, Coroutines, and Modules were all mentioned in a recent email from Herb. Also there was a question about adding Concepts and Concurrency from Herb.

M. Scott: What do we want?

Michael Wong: Executors can impact this a lot. There may be some procedural issues that need to be addressed, too.

If we want to move any part of SG5 into C++20, we will need to write a paper.

SG1 has also been discussing the idea of eliminating all of these different names for TSs. Maybe they should be merged into a single SG1 TS.

Hans: that probably doesn't affect TM, since it's a separate SG.

M. Scott: has been wondering if we should reconsider the whole API for TM. Instead of lexical blocks, should we pass lambdas.

M. Spear: two dimensions: functionality (synchronized vs. atomic) and syntax (lexically scoped blocks vs. lambdas)

Discussion: what does it mean to have a library API? The compiler infrastructure for instrumentation is important.

Victor: people seem to be more comfy using lambdas these days.

M. Scott: lambda benefits include that it is more uniform if executors ever happen; that many semantic subtleties become more obvious (capture, what is being touched by the atomic block of code; exceptions, which must be declared)

Hans: but you don't declare globals that are captured, which is a small limitation.

Victor: if you are used to using lambdas, then you know about capture, so making it more explicit.

M. Spear: return value of lambdas may be an issue, because right now you can break/return from a TM lexically-scoped block.

M. Wong: there was a question last call about how executors interact with locks, and whether executors can subsume a lock guard. Executors give a way to control the how, where, when of execution. Is it possible to create an executor that does locking, and another that does not do locking. There is an implicit memory access. Right now executors treat memory as a resource that the executor controls. The executor group may have decided that this isn't in their scope, since they are thinking about holistic execution contexts. This topic has not been fleshed out fully. If executors can replace lock_guard, it opens a role for TM, too. But the executor group isn't really well equipped to discuss how it interacts with locking.

Hans: still seems strange to view locking as an executor. But your lock implementation may vary based on the executor being used. For example, barriers have a similar problem, because barrier implementation depends on the executor being used, but it's not clear how you'd do that.

Mike: I'd prefer for us to make the decision about a lambda interface independent of the question of executors. If we go with it, it might align nicely with executors in the future, but the decision should stand on its own merits.

Hans: lambda-syntax seems to be the future for lots of these emerging constructs.

M. Wong: there was a question about similarity to Java executors. For example, there were questions about blockingness (Java's are nonblocking; not all C++ ones are).

M. Scott: clarification: a task run by an executor in Java is not allowed to block.

M. Wong: that isn't guaranteed of all C++ executors.

M. Scott: Java executors are much simpler than the C++ proposal. It simply says "instantiate an object and pass runnables to it, and it will run them; they aren't allowed to block, because the programmer doesn't know how many threads are assigned to the executor. But that's not enforced, it's just a rule." Then there are a few more pieces to the API, but nothing about "where does it run" or "what sort of synchronization does it use" or "express a continuation" in Java.

M. Wong: thinks that synchronization interacting with executors is either (a) something they want someone else to tackle, or (b) not related. He suspects it's (a).

Mike: Do the strengths of lambdas outweigh the costs? Benefits are clarity of capture and exceptions. Weaknesses are early return/break and extra allocations.

M. Wong: C committee for C parallel programming has been moving toward function-based approach, in part to be compatible with Cilk.

Mike: thinks that lambdas for synchronized blocks is a straightforward proposal: doesn't deal with the hard questions, but may be a bit prettier than the old way.

Hans: initiated a discussion about declaring things transaction-safe. For example, does a std::function need to have a way to capture transaction safety.

Mike: the issue is really about separate translation units.

M. Scott: transaction safety could be part of the function? Do we already need a std::tx_safe_function? Do we need it if we move to lambdas?

M. Spear: will there be too much allocation?

Hans: the lambda implementations seem to be getting better, it may not be an issue anymore, but that's just what he's been told, not what he's confirmed himself.

M. Wong: Will there be objections from people who can't implement it (e.g., GPU vendors)?

M. Scott: synchronized is already equivalent to single lock, so there should be a trivial correct implementation.

M. Wong: so synchronized blocks should be a problem for C++20. Then lambdas could be syntactic sugar.

M. Scott: but it's a big language, why add something if we'll take it out?

M. Spear: keyword is really the syntactic sugar.

M. Scott: trivial implementation of lambda interface, with no compiler support and a tiny library.

M. Spear: lambdas for synchronized is probably the minimum viable product to add to C++20.

M. Scott: but we don't want to be incompatible with executors.

M. Spear: if we move fast, will we help executors to take shape?

Hans: seems like TM may just not work with certain types of executors, e.g., GPU ones. But that's probably true for latches, mutexes...

M. Scott: or code that does I/O.

M. Wong: executors could change everything.

M. Spear: if we want a full proposal, we need to flesh out continuations vs. deferral; exceptions; and expressing where things can/can't run.

Hans: paper deadline is in two weeks.

M. Wong: maybe have a joint session with executor group? They want feedback.

M. Spear: wants a paper first, and then let them decide if they want to work with us.

M. Wong: meeting might be good, because both groups don't understand each other.

M. Spear: even if we can't get a paper in by the deadline, we need it before we have any discussion with the executor group. And we won't have that discussion within 2 weeks anyway.

Victor: agrees.

M. Wong: let's write the paper, then have the discussion. We may want to have that discussion long before the next meeting. After the mailing deadline and Toronto meeting might be the right time to have the discussion.

M. Spear: if we have the paper before Toronto, could we share it with executor group and discuss at Toronto?

M. Scott: when is the Toronto meeting?

M. Wong: July 10. Let's keep this going at next meeting.

<<End of Call>>