Has anyone tried transactional memory for C++?

Asked 16 years, 3 months ago Modified 6 years, 6 months ago

Viewed 6k times

11

I was checking out Intel's "whatif" site and their Transactional Memory compiler (each thread has to make atomic commits or rollback the system's memory, like a Database would).

It seems like a promising way to replace locks and mutexes but I can't find many testimonials. Does anyone here have any input?

c++ multithreading locking intel transactional-memory

Share

Improve this question

Follow

edited Sep 17, 2008 at 12:22

Mark Biek

151k ● 54 ● 158 ● 201

asked Sep 17, 2008 at 12:20

Robert Gould

69.7k ● 61 ● 191 ● 275

Is this question and its answers still current?
– Janus Troelsen Jun 13, 2017 at 11:46

@JanusTroelsen check out the available implementations in en.m.wikipedia.org/wiki/Transactional_memory – Adam Davis Jun 13, 2017 at 11:53

Related: realworldtech.com/haswell-tm for David Kanter's write-up of some under-the-hood details on how it's actually implemented on Intel CPUs. And also some neat stuff about transactional memory in general. – Peter Cordes Nov 22, 2017 at 11:03

5 Answers

Sorted by: Highest score (default)

8

I have not used Intel's compiler, however, Herb Sutter had some interesting comments on it...

From Sutter Speaks: The Future of Concurrency

Do you see a lot of interest in and usage of transactional memory, or is the concept too difficult for most developers to grasp?

It's not yet possible to answer who's using it because it hasn't been brought to market yet. Intel has a software transactional memory compiler prototype. But if the question is "Is it too hard for developers to use?" the answer is that I certainly hope not. The whole point is it's way easier than locks. It is the only major thing on the research horizon that holds out hope of greatly reducing our use of locks. It will never replace locks completely, but it's our only big hope to replacing them partially.

There are some limitations. In particular, some I/O is inherently not transactional—you can't take an atomic block that prompts the user for his name and read the name from the console, and just automatically abort and retry the block if it conflicts with another transaction; the user can tell the difference if you prompt him twice. Transactional memory is great for stuff that is only touching memory, though.

Every major hardware and software vendor I know of has multiple transactional memory tools in R&D. There are conferences and academic papers on theoretical answers to basic questions. We're not at the Model T stage yet where we can ship it out. You'll probably see early, limited prototypes where you can't do unbounded transactional memory—where you can only read and write, say, 100 memory locations. That's still very useful for enabling more lock-free algorithms, though.

Share Improve this answer

Follow

answered Sep 17, 2008 at 15:12

Brian Stewart

9,267 ● 11 ● 55 ● 66

About where it is too hard for developers: see the paper "Is Transactional Programming Actually Easier" by Rossbach et al: cs.utexas.edu/~rossbach/pubs/wddd09-rossbach.pdf – amit kumar Oct 28, 2011 at 15:02

I've built the combinatorial STM library on top of some functional programming ideas. It doesn't require any compiler support (except it uses C++17), doesn't bring a

new syntax. In general, it adopts the interface of the STM library from Haskell.

So, my library has several nice properties:

- Monadically combinatorial. Every transaction is a computation inside the custom monad named `STML`. You can combine monadic transactions into more big monadic transactions.
- Transactions are separated from data model. You construct your concurrent data model with transactional variables (`TVars`) and run transactions over it.
- There is `retry` combinator. It allows you to rerun the transaction. Very useful to build short and understandable transactions.
- There are different monadic combinators to express computations shortly.
- There is `Context`. Every computation should be run in some context, not in the global runtime. So you can have many different contexts if you need several independent STM clusters.
- The implementation is quite simple conceptually. At least, the reference implementation in Haskell is so, but I had to reinvent several approaches for C++ implementation due to the lack of a good support of Functional Programming.

The library shows very nice stability and robustness, even if we consider it experimental. Moreover, my approach

opens a lot of possibilities to improve the library by performance, features, comprehensiveness, etc.

To demonstrate its work, I've solved the `Dining Philosophers` task. You can find the code in the links below. Sample transaction:

```
STML<bool> takeFork(const TVar<Fork>& tFork)
{
    STML<bool> alreadyTaken = withTVar(tFork, isForkTa
    STML<Unit> takenByUs    = modifyTVar(tFork, setFor
    STML<bool> success      = sequence(takenByUs, pure
    STML<bool> fail         = pure(false);
    STML<bool> result       = ifThenElse(alreadyTaken,
    return result;
};
```

UPDATE I've wrote a tutorial, you can find it here.

- Dining Philosophers task
- My C++ STM library

Share Improve this answer Follow

edited May 26, 2018 at 19:55

answered Apr 2, 2018 at 16:41

Alexander Granin

983 ● 15 ● 20

4

Dr. Dobb's had an article on the concept last year: Transactional Programming by Calum Grant -- http://www.ddj.com/cpp/202802978

It includes some examples, comparisons, and conclusions using his example library.

Share Improve this answer

Follow

answered Sep 17, 2008 at 15:29

Kris Kumler

6,307 ● 3 ● 25 ● 27

1

Sun Microsystems have announced that they're releasing a new processor next year, codenamed Rock, that has hardware support for transactional memory. It will have some limitations, but it's a good first step that should make it easier for programmers to replace locks/mutexes with transactions **and** expect good performance out of it.

For an interesting talk on the subject, given by Mark Moir, one of the researchers at Sun working on Transactional Memory and Rock, check out this link.

For more information and announcements from Sun about Rock and Transactional Memory in general, this link.

The obligatory wikipedia entry :)

Finally, this link, at the University of Wisconsin-Madison, contains a bibliography of most of the research that has been and is being done about Transactional Memory, whether it's hardware related or software related.

Share Improve this answer

Follow

answered Sep 17, 2008 at 21:24

fuad
4,425 • 9 • 37 • 32

-2

In some cases I can see this as being useful and even necessary.

However, even if the processor has special instructions that make this process easier there is still a large overhead compared to a mutex or semaphore. Depending on how it's implemented it may also impact realtime performance (have to either stop interrupts, or prevent them from writing into your shared areas).

My expectation is that if this was implemented, it would only be needed for portions of a given memory space, though, and so the impact could be limited.

-Adam

Share Improve this answer

Follow

answered Sep 17, 2008 at 12:22

Adam Davis
93.5k • 60 • 271 • 333

This doesn't sound like transactional memory... What is the overhead compared with mutexes and semaphores?
– Andres Jaan Tack Nov 19, 2009 at 16:47

There are several places where the overhead is higher. One obvious example is rolling back a transaction. It also makes caching more difficult and carry more overhead, as everything has to be immediate write-back to memory. Transactional memory is a good idea for some applications, but it does impact system performance, and thus shouldn't be

deployed for every application and system. – Adam Davis Nov 19, 2009 at 17:13

1 Ah, but multiple restarts are the pathological case for transactions. How does this compare with the pathological case for locks (long blocking, bouncing between caches)? – Andres Jaan Tack Nov 19, 2009 at 17:27

Is this post still current? – Janus Troelsen Jun 13, 2017 at 11:45

1 @JanusTroelsen: Intel's current implementation (Broadwell/Skylake) doesn't block interrupts, instead it aborts the transaction. See realworldtech.com/haswell-tm for more about how it works under the hood. (AFAIK the basic design is the same in Skylake as Haswell, although HSW had bugs that required disabling it in a microcode udpate.) Basically it adds some extra bits to L1D cache, and aborts the transaction if anything tricky happens. – Peter Cordes Nov 22, 2017 at 11:01