Performance Effects of Lock Fallback in Best Effort HTM Systems

Matthew Gaudet, José Nelson Amaral, Guido Araújo
Subtleties of Transactional Memory Atomicity Semantics

Colin Blundell
University of Pennsylvania, blundell@cis.upenn.edu

E. Christopher Lewis
University of Pennsylvania, eclewis@cis.upenn.edu

Milo Martin

LogTM: Log-based Transactional Memory

Kevin E. Moore, Jayaram Bobba, Michelle J. Moravan, Mark D. Hill & David A. Wood
Department of Computer Sciences, University of Wisconsin–Madison
{kmoore, bobba, moravan, markhill, david}@cs.wisc.edu
http://www.cs.wisc.edu/multifacet

Sequential Specification of Transactional Memory Semantics

Michael L. Scott
Department of Computer Science
University of Rochester
scott@cs.rochester.edu

On the Performance of Contention Managers for Complex Transactional Memory Benchmarks

Mohammad Ansari, Christos Kotselidis, Mikel Luján, Chris Kirkham and Ian Watson
School of Computer Science, University of Manchester
Email: {ansari, kotselidis, mikel, chris, watson}@cs.manchester.ac.uk

Adaptive Software Transactional Memory and Runtime Support for Efficient Software Transactional Memory

Hung Chung, Hassan Chafi, Minh, Lance Hammond, Christos Kozyrakis, Kunle Olukotun

Yet, something interesting started happening over the last couple of years
IBM’s new transactional memory: make-or-break time for multithreaded revolution

At Hot Chips last week, IBM talked about BlueGene/Q, the processor powering ...
This level of reliability and security makes mainframes ideal for running mission-critical business processes where any downtime would be catastrophic. In 2012, 96 of the world's top 100 banks used IBM's mainframe technology.

Transactional Memory Architecture and Implementation for IBM System z

Christian Jacobi, Timothy Slegel
IBM Systems and Technology Group
Poughkeepsie, NY, USA
cjacobi@us.ibm.com, slegel@us.ibm.com

Dan Greiner
IBM Systems and Technology Group
San Jose, CA, USA
dgreiner@us.ibm.com
Transactional memory going mainstream with Intel Haswell

Transactional memory is a promising technique for making the development of ...

by Peter Bright - Feb 8 2012, 7:10pm MST

Quad core die shown above

Transistor count: 1.4 Billion

Die size: 177mm²

** Cache is shared across all 4 cores and processor graphics
TM HAS ESCAPED THE LAB
Is Parallel Programming Hard, And, If So, What Can You Do About It?

First Edition Release Candidate 1

Edited by:

Paul E. McKenney
Linux Technology Center
IBM Beaverton
paulmck@linux.vnet.ibm.com

January 5, 2014
Parallel Programming

- Work Partitioning
- Resource Partitioning
- Hardware Interaction
- Parallel Access Control
Parallel Access Control
Protect Shared Data
Imperative
Pessimistic
CAN WE HAVE OPTIMISM?
Optimistic Parallel Access Control

 TRANSACTIONAL MEMORY
Transactional Memory:
Architectural Support for Lock-Free Data Structures

Maurice Herlihy
Digital Equipment Corporation
Cambridge Research Laboratory
Cambridge MA 02139
herlihy@crl.dec.com

J. Eliot B. Moss
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
moss@cs.umass.edu
Transactions
Transactions

ACID:
- Atomicity
- Consistency
- Isolation
- Durability

Diagram from:
http://maxdb.sap.com/doc/7_7/44/
d776a368113ee3e10000000a114a6b/content.htm
atomic {
    // Do stuff.
    // Atomically.
}

Conflicts

Transaction 1

\[ b = a + 1 \]

Transaction 2

\[ a = 10 \]
Software TM

atomic {
    a[i] = b[c[i]];
}

stm_begin();
tmp_c = stm_load(&c[i]);
tmp_b = stm_load(&b[tmp_c]);
stm_store(tmp_b, &a[i]);
stm_commit();
Contention Manager

Forward Progress Guarantee

Contention Manager

TX 1

TX 2

TX 3
BEST EFFORT HARDWARE
TRANSACTIONAL MEMORY

Blue Gene Q's
Blue Gene/Q

- 1.6 Ghz
- 16 user accessible cores
- 32 MB L2 cache
  - With Speculation Support, enabled by multiversioning.

Best-Effort Transactional Memory Support.
Blue Gene/Q

Two TM Modes:

**Short Running Mode:**
- Evict speculatively written lines to L2.
- Cost: Longer latencies for read and writes after write.

**Long Running Mode:**
- Stores lines, speculatively read and written in L1
- Cost: Flushing L1 at transaction start.
BG/Q’s TM

```c
#pragma tm_atomic
{
    a[i] = b[ c[i] ];
}
```

- IBM XL C/C++/Fortran
- TM Runtime
- CNK
- BG/Q L2 Cache
FORWARD PROGRESS?
Forward Progress

- Contention Management done in HW
  - Limited control
- Transactions that are poor fits:
  - I/O, MMIO, Network writes
    - Anything that cannot be rolled back!

Solution: Serialization, and the Serialization Manager
Serialization

- Run transaction without speculation
  - To provide isolation guarantee, use a lock to ensure no overlap with other transactions.
When to serialize?

Serialization Manager:
Policy for serialization + any data required to inform policy.

Yoo et. al. 2013

Wang et. al. 2013

If the transactional execution is unsuccessful, then the lock may be explicitly acquired to ensure forward progress. The decision to acquire the lock explicitly is based on the number of times the transactional execution has been tried but failed; for our hardware and workloads,\textsuperscript{5} gave the best overall performance. To ensure correct interaction of the transactional execution with other threads that may or already has explicitly acquired the lock, the state of the lock is tested during the transactional execution.

RTM alone does not provide any progress guarantees for an application: a hardware transaction may keep retrying (and aborting) indefinitely. To ensure progress, we implement a spinlock-based software fallback path for every RTM transaction started. TSX will speculatively attempt to elide the spin locks but should any transactions abort, critical sections will be protected by acquiring the spin locks in software. Our implementation supports both multiple fine-grained spin locks and one global spin lock as the software fallback path.

In order to facilitate transaction retries, users can provide a parameter to specify and adjust the number of retries in RTM before falling back to software execution. The retry parameter combined with the hardware status registers [1] allow libTM to maximize the likelihood of transactions committing successfully.
Hypothesis:

Serialization Management is a large part of TM Performance
OUR APPROACH

1. Build multiple Serialization Managers across a range of possible designs.
2. Evaluate these Serialization Managers thoroughly, across a wide range of tunings.
3. Use both Long and Short running mode to understand how policies handle different behaviour of the underlying machine.
SERIALIZATION MANAGERS
We built many Serialization Managers...

Here I’ll be discussing three...
MaxRetry

Track retries per dynamic transaction
• Serialize if $N$ retries fail to allow the transaction to successfully commit speculatively.
Converting an STM Transaction Scheduler to a Serialization Manager: Use queuing to reduce contention

Each Thread tracks “Conflict Intensity”:

\[ CI_n = \alpha \times CI_{n-1} + (1 - \alpha) \times CC \]

- \( CC = 1 \) on abort
- \( CC = 0 \) on commit
The role of $\alpha$

History Control – or the rate of adaptation:

• High $\alpha$, more weight on history, slower adaptation.

• Low $\alpha$, more weight on currency, faster adaptation.

\[ CI_n = \alpha \times CI_{n-1} + (1 - \alpha) \times CC \]
LimitMeanST

More
Retries?

Less
Retries?
LimitMeanST

- For each static transaction compute the mean-execution time
  - Provide each dynamic transaction a budget, some multiple $M$ of this mean.
  - When the budget is exhausted, serialize
- Intention is to punish long-running outliers, and allow more retries for short aborts.
1. Benchmarks:
   • STAMP: TM benchmarks from Stanford
   • RMS-TM: TM benchmarks from machine learning area
2. Variance:
   • 5 Repetitions.
3. Measurement:
   • Absolute Speedup: Ratio of Parallel Execution time to the best sequential implementation
An ASF implementation may have a much higher capacity than the four architectural memory lines, but software cannot rely on any forward progress if it attempts to use more than four lines. In this case, software has to provide a fallback path to be taken in the event of capacity overflow, if the transactional execution is unsuccessful, then the lock may be explicitly acquired to ensure forward progress. The decision to acquire the lock explicitly is based on the number of times the transactional execution has been tried but failed; for our hardware and workloads, gave the best overall performance. To ensure correct interaction of the transactional execution with other threads that may or already has explicitly acquired the lock, the state of the lock acquired by the RTM executor needs be cleaned up to allow the transaction to commit successfully. In order to facilitate transaction retries, users can provide a parameter to specify and adjust the number of retries in RTM before falling back to software execution. The retry parameter combined with the hardware status registers [1] allow libTM to maximize the likelihood of transactions committing successfully.
Can other strategies perform better than MaxRetry?

LimitMeanST
Fraction of Best Absolute Speedup

Benchmark
- fluidanimate
- genome
- hmmcalibrate
- hmmsearch
- intruder
- kmeans
- ScalParC
- vacatio
- yada
COMPARING Serialization MANAGERS
1. Serialization Management is important to performance, sometimes dramatically!

2. In BE-HTM system evaluations, you **must** discuss serialization management.
QUESTIONS?
Credits

• Graphics from:
  – http://www.theregister.co.uk/Print/2012/08/31/ibm_power7_plus_processors/
  – http://paddyinba.blogspot.ca/2012/03/travel-hacking-tip-1-credit-cards.html
  – http://en.wahooart.com/@@/8LT54H-Piet-Mondrian-Cuadro-II
  – http://qualitybs.wordpress.com/2011/01/12/de-mens-als-vervuiler-van-de-cleanroom/
Credits

• Slide backgrounds from:
  – [http://unsplash.com](http://unsplash.com)
  – My personal photography
Absolute Speedup

Rollbacks

SerializationManager

LimitMeanST

MaxRetry