Cost of Concurrency in Hybrid Transactional Memory

Trevor Brown (University of Toronto)
Srivatsan Ravi (Purdue University)
Transactional Memory: a history

1993 ➔ Original proposal by Herlihy-Moss ('93) ➔ Exploit cache-coherence ➔ Optimistic synchronization: buffer speculative updates ➔ Software implementation by Shavit-Touitou ('95) ➔ "static" transactions ➔ "Dynamic" STM by Herlihy et al. ('03)

◆ Incremental validation cost ➔ TL2 by Dice et al. ('06), NOrec by Spear et al. ('06),...

◆ Mitigate validation cost ➔ HTM support: Intel Haswell, IBM Power8,..
◆ Different memory models and supported instructions ➔ Hardware limitations and "spurious" aborts ➔ Fallback to software transactions

1995-today ➔ Hardware TM ➔ Software TM ➔ Hybrid TM

Today
Transactional Memory: a history

**Hardware TM**

1993

**Software TM**

1995-today

**Hybrid TM**

Today

---

Cost of Concurrency in Hybrid TMs?
Hybrid Transactional Memory (HyTM) Model

- Transactions
- Data items
- Base objects

**Fast-path**
- Executed in hardware
- Exploit cache-coherence
- Spurious aborts
- Cache limitations
Hybrid Transactional Memory (HyTM) Model

Transactional operations

- Transactions
- Data items
- Base objects

Slow-path

HW primitives

- Executed in software
  - Reliable for large transactions
- Slower execution time
  - Harder to verify
Hybrid Transactional Memory (HyTM) Model

For fast-path transactions
- Operate on “cached” memory state
  - Direct on Power8
- Maintain TRACKING SET
  - Shared/exclusive mode
  - Capacity limit

- For slow-path transactions
- Operate directly on memory state

Direct access
Cached access
Hybrid Transactional Memory (HyTM) Model

Tracking set aborts in fast-path transactions
Automatic contention detection for cached accesses

WRITE to base object B (EXCLUSIVE mode)

Tracking set is invalidated and T2 must abort

Fast-path or slow-path

ACCESS base object B

Fast-path

T2

A2

T1
Hybrid Transactional Memory (HyTM) Model

Tracking set aborts in fast-path transactions
Automatic contention detection for cached accesses

READ to base object B (SHARED mode)

Fast-path

Tracking set is invalidated and T2 must abort

WRITE base object B

Fast-path or slow-path
Hybrid Transactional Memory (HyTM) Model

Committed fast-path transactions (only cached accesses) appear to execute atomically (single step)

Execution indistinguishable to T1 from an execution in which T2 does not participate
Instrumentation

- Fast-path transactions intuitively need code “instrumentation”
  - Detect overlap contention with slow-path
    - Global timestamp, ownership records, etc.
  - Instrumentation affects performance

T0

\[ \text{Slow-path} \]

\[ \text{W}(X,1) \quad \text{W}(Y,1) \]

\[ \text{Fast-path} \]

\[ \text{T1} \]

\[ \text{R}(X) \rightarrow ? \]

Partial commit of X and Y

T1 must access “meta-information” to detect contention with T0
Cost of concurrency in HyTM

Sequential
- Minimal concurrency
- $O(1)$ fast-path instrumentation

Progressive
- More concurrency
- $\Omega(m)$ fast-path instrumentation

Cost on slow-path transactions?
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions

(m-1) invisible reads of distinct data items \(X_1 \ldots X_{m-1}\)

Read of \(X_m\) must return the value 1 updated by \(T_m\)
Linear validation cost in HyTM

Progressive opaque HyTM + Invisible reads $\Rightarrow$ Linear time and space complexity for slow-path transactions

Slow-path

Fast-path

$T_0$ does not observe $T_m$ until the access of data item $X_m$

Read of $X_m$ by $T_0$ must return the value 1
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions

Slow-path

Fast-path

Write new value to data item $X_1$ and commit

Cannot return value 1--cycle in serialization
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions

Each fast-path transaction writes new values to data objects $X_1$ to $X_{m-1}$

$T_0$ is invisible to fast-path transactions $T_1 \ldots T_m$
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads $\Rightarrow$ Linear time and space complexity for slow-path transactions

Tracking set aborts: cannot contend on same memory location
Validation cost in HyTM

Progressive opaque HyTM+Invisible reads $\Rightarrow$ Linear time and space complexity for slow-path transactions

Read of $X_m$ must access $m-1$ distinct memory locations
Validation cost in HyTM

Progressive opaque HyTM+Invisible reads $\Rightarrow$ Linear time and space complexity for slow-path transactions

Transaction $T_0$ must take at least $\sum_{i=1}^{m-1} = \Omega(m^2)$ memory steps.
Cost of concurrency in HyTM

Sequential

- Minimal concurrency
- O(1) fast-path instrumentation + O(1) slow-path reads

Progressive

- More concurrency
- Fast-path transactions aborted by non-conflicting ones or linear fast-path instrumentation cost and linear slow-path steps
- Ω(m) fast-path instrumentation + Ω(m) slow-path steps per-read
Cost of concurrency in HyTM

Sequential

Minimal concurrency

O(1) fast-path instrumentation + O(1) slow-path reads

Progressive

More concurrency

Progressive STMs like TL2 circumvent the cost of validation for better performance, but impossible in progressive HyTMs!

Ω(m) fast-path instrumentation + Ω(m) slow-path steps per-read
## Cost of Concurrency in HyTM

<table>
<thead>
<tr>
<th></th>
<th>Algorithm 1</th>
<th>Algorithm 2</th>
<th>Transactional Lock Elision</th>
<th>Hybrid Norec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instrumentation in fast-path reads</strong></td>
<td>per-read</td>
<td>constant</td>
<td>constant</td>
<td>constant</td>
</tr>
<tr>
<td><strong>Instrumentation in fast-path writes</strong></td>
<td>per-write</td>
<td>per-write</td>
<td>constant</td>
<td>constant</td>
</tr>
<tr>
<td><strong>Validation in slow-path reads</strong></td>
<td>$\Omega(</td>
<td>Rset</td>
<td>)$</td>
<td>$\Omega(</td>
</tr>
<tr>
<td><strong>h/w-s/w concurrency</strong></td>
<td>prog</td>
<td>prog for slow-path readers</td>
<td>zero</td>
<td>Not prog; small contention window</td>
</tr>
<tr>
<td><strong>Direct accesses inside fast-path?</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Experimental setup

- Large-scale 2-socket Intel E7-4830 v3 with 12 cores per socket and 2 hyperthreads (HTs) per core, for a total of 48 threads

- Each core has a private 32KB L1 cache and 256KB L2 cache (which is shared between HTs on a core)
  - All cores on a socket share a 30MB L3 cache
  - Non-uniform memory architecture (NUMA)
  - 128GB of RAM, and runs Ubuntu 14.04 LTS.

- All code was compiled with the GNU C++ compiler (G++) 4.8.4 with build target x86_64-linux-gnu and compilation options
  -std=c++0x -O3 -mx32
Experimental methodology

- Six timed trials for several thread counts $n$
  - *Prefilling*: $n$ concurrent threads perform 50% Insert and 50% Delete operations on keys drawn uniformly randomly from $[0, 10^5)$ until the size of the tree converges to a steady state (containing approximately $10^5 / 2$ keys)
  - *Measuring*: Each thread performs $(U/2)\%$ Insert, $(U/2)\%$ Delete and $(100 - U)\%$ Search operations, on keys/values drawn uniformly from $[0, 10^5)$; $U=0,10,40$

- Plots for Binary Search Tree (BST) microbenchmark
  - With (without) one (any) thread performing *RangeIncrement* operations
    - Capacity aborts on fast-path
Cost of Concurrency in HyTM

0% updates: #threads vs. ops/microsec

Graphs showing the cost of concurrency for different algorithms and thread counts.
Cost of Concurrency in HyTM

10% updates: #threads vs. ops/microsec
Cost of Concurrency in HyTM

- **Read-only workloads**: costs purely down to fast-path
  - Algorithm 1 overhead due to linear instrumentation
- **Update workloads with** RangIncrement
  - TLE suffers due to global lock bottleneck
  - NUMA effects on update heavy workloads
    - From thread counts > 24
    - Hybrid noREC performs poorly to Algorithm 2 for same reasons as TLE

40% updates: #threads vs. ops/microsec
Circumventing the impossibilities?

- Middle-path approach? *ongoing work*
  - Almost uninstrumented “fast” fast-path
    - No concurrency with slow-path
    - Concurrent with middle-path
- Ongoing experiments on Intel Haswell and IBM Power8 (*STAMP* and data structure microbenchmarks)
  - Completely different memory models
    - Power8 allows “direct” accesses inside hardware
Transactional memory is here to stay?

• HyTM: an efficient “universal construction”?  
  ○ Start with a “base” HyTM with minimal instrumentation overhead, maximal concurrency and little global metadata bottleneck  
  ○ Dynamic implementation choices depending on workloads  
    ■ Multi-path approach  
  ○ Formal methods and verification techniques  
  ○ Impact of cache hierarchy, cache-size and memory model on HyTM performance