VIRTUES AND LIMITATIONS OF COMMODITY HARDWARE TRANSACTIONAL MEMORY

Nuno Diegues, Paolo Romano and Luís Rodrigues
The multi-core (r)evolution
The multi-core (r)evolution

Multi-cores are now ubiquitous
The multi-core (r)evolution

Multi-cores are now ubiquitous
The multi-core (r)evolution

Multi-cores are now ubiquitous
The multi-core (r)evolution

Multi-cores are now ubiquitous

Concurrent programming is complex

Shared Memory

CPU1  CPU2  CPU3  CPU4
The multi-core (r)evolution

Multi-cores are now ubiquitous

Concurrent programming is complex
The multi-core (r)evolution

Multi-cores are now ubiquitous

Concurrent programming is complex

Classic approach: Locking

Hard to get right:
- fine-grained locks
- deadlocks
- correctness
The multi-core (r)evolution

- Multi-cores are now ubiquitous
- Concurrent programming is complex
- Classic approach: Locking
  - Hard to get right:
    - fine-grained locks
    - deadlocks
    - correctness

- Transactional Memory abstraction
  - Transactional Memory System
  - Atomic access:
    - `atomic {
        withdraw(acc1,val);
        deposit(acc2,val);
    }

Programmer identifies atomic blocks
Runtime implements synchronization
TM is now available in commodity processors
TM is now available in commodity processors

- Intel: Haswell in desktops, laptops, tablets, servers…
- IBM: BG/Q, zEC12, Power8
TM is now available in commodity processors

- Intel: Haswell in desktops, laptops, tablets, servers…
- IBM: BG/Q, zEC12, Power8

Over 10 years of:
- Software implementations (STMs)
- Simulations of HTMs and HybridTMss
TM is now available in commodity processors

- Intel: Haswell in desktops, laptops, tablets, servers…
- IBM: BG/Q, zEC12, Power8

Over 10 years of:
- Software implementations (STMs)
- Simulations of HTMs and HybridTM}s

Where does commodity HTM stand in the big picture?
TM is now available in commodity processors

• Intel: Haswell in desktops, laptops, tablets, servers…
• IBM: BG/Q, zEC12, Power8

Over 10 years of:
• Software implementations (STMs)
• Simulations of HTMs and HybridTM

Where does commodity HTM stand in the big picture?

Our contribution: largest TM study to date
TM is now available in commodity processors

- Intel: Haswell in desktops, laptops, tablets, servers…
- IBM: BG/Q, zEC12, Power8

Over 10 years of:
- Software implementations (STMs)
- Simulations of HTMs and HybridTM

Where does commodity HTM stand in the big picture?

Our contribution: largest TM study to date

Framework with 4 STMs, Intel HTM, 2 HyTM and locking strategies;
Metrics for performance and power consumption;
10 benchmarks.
HTM: Intel Transactional Synchronization Extensions (TSX)
HTM: Intel Transactional Synchronization Extensions (TSX)

Widely available in millions of machines

Similar in nature to IBM’s HTMs
HTM: Intel Transactional Synchronization Extensions (TSX)
Widely available in millions of machines
Similar in nature to IBM’s HTMs

- L1 modified to be transactional
- Cache coherence detects conflicts eagerly
- Strong atomicity
HTM: Intel Transactional Synchronization Extensions (TSX)
HTM: Intel Transactional Synchronization Extensions (TSX)

CPU 1

xbegin

CPU 2
HTM: Intel Transactional Synchronization Extensions (TSX)

CPU 1

```
xbegin
read x: 0  // Set bit read on x cache line
```
HTM: Intel Transactional Synchronization Extensions (TSX)

CPU 1

xbegin
read x: 0  // Set bit read on x cache line

x: 0 -- r

CPU 2
HTM: Intel Transactional Synchronization Extensions (TSX)

CPU 1

xbegin
read x: 0 // Set bit read on x cache line
write y = 1 // Buffer write in L1 cache

CPU 2

x: 0 -- r
y: 1 -- w
HTM: Intel Transactional Synchronization Extensions (TSX)

```
xbegin
read x: 0   // Set bit read on x cache line
write y = 1  // Buffer write in L1 cache
xend       // Atomically clean bits and publish
```
HTM: Intel Transactional Synchronization Extensions (TSX)

**CPU 1**

```
xbegin
read x: 0       // Set bit read on x cache line
write y = 1     // Buffer write in L1 cache
xend            // Atomically clean bits and publish
```

**CPU 2**

```
xbegin
read y: 1
```

![Memory Bus Diagram](image)
Virtues and Limitations of HTM       PACT 2014

HTM: Intel Transactional Synchronization Extensions (TSX)

CPU 1

xbegin
read x: 0  // Set bit read on x cache line
write y = 1  // Buffer write in L1 cache
xend  // Atomically clean bits and publish

CPU 2

xbegin
read y: 1

Memory Bus

x: 0
y: 1

L1 Cache  CPU1  L2 Cache  L3 Cache

L1 Cache  CPU2  L2 Cache
HTM: Intel Transactional Synchronization Extensions (TSX)

CPU 1

\texttt{xbegin}
\texttt{read x: 0} // Set bit read on x cache line
\texttt{write y = 1} // Buffer write in L1 cache
\texttt{xend} // Atomically clean bits and publish

\texttt{write y = 2}

CPU 2

\texttt{xbegin}
\texttt{read y: 1}

\texttt{y: 1 -- r}
HTM: Intel Transactional Synchronization Extensions (TSX)

CPU 1

xbegin
read x: 0  // Set bit read on x cache line
write y = 1  // Buffer write in L1 cache
xend  // Atomically clean bits and publish

write y = 2

CPU 2

xbegin
read y: 1

y: 1 -- r
HTM: Intel Transactional Synchronization Extensions (TSX)

```
xbegin
read x: 0  // Set bit read on x cache line
write y = 1  // Buffer write in L1 cache
xend  // Atomically clean bits and publish

write y = 2
```

```
xbegin
read y: 1
write y = 2
xabort
```

Invalidation:
```
x: 0
y: 2
```

```
x: 0
y: 1 -- r
```

Snooped write invalidates tx read:
```
xbegin
read y: 1
```

```
x: 0
y: 2
```

```
x: 0
y: 1 -- r
```
In an ideal world...

\begin{verbatim}
xbegin
withdraw(acc1,val)
deposit(acc2,val)
xend
\end{verbatim}
In an ideal world...

Transactions may abort:
• because of contentiousion on same memory locations

\[
\begin{align*}
\text{xbegin} \\
\text{withdraw}(\text{acc1}, \text{val}) \\
\text{deposit}(\text{acc2}, \text{val}) \\
\text{xend}
\end{align*}
\]
In an ideal world...

Transactions may abort:

- because of contention on same memory locations

...and every transaction shall eventually succeed
...in practice: Best-Effort Nature

No progress guarantees:
...in practice: Best-Effort Nature

No progress guarantees:

• A transaction may \textit{always} abort
...in practice: Best-Effort Nature

No progress guarantees:

- A transaction may always abort

...due to a number of reasons:

- Forbidden instructions
- Capacity of caches
- Faults and signals
- Contending transactions, aborting each other
Restrictions of TSX
Restrictions of TSX

- Writes:
  - size of L1 cache: 32KB
  - non-negligible aborts for >8KB
  - cache associativity
Restrictions of TSX

• **Writes:**
  - size of L1 cache: 32KB
  - non-negligible aborts for >8KB
  - cache associativity

• **Reads:**
  - up to 4MB
  - overflow structure in L2 cache
  - presumed to be a Bloom-Filter [Eurosys14]
Restrictions of TSX

• **Writes:**
  - size of L1 cache: 32KB
  - non-negligible aborts for >8KB
  - cache associativity

• **Reads:**
  - up to 4MB
  - overflow structure in L2 cache
  - presumed to be a Bloom-Filter [Eurosyst14]

• **Interrupts:**
  - up to 1M cycles
  - Roughly 0.5 ms on a Haswell Xeon
Restrictions of TSX

• Writes:
  • size of L1 cache: 32KB
  • non-negligible aborts for >8KB
  • cache associativity

• Reads:
  • up to 4MB
  • overflow structure in L2 cache
  • presumed to be a Bloom-Filter [Eurosys14]

• Interrupts:
  • up to 1M cycles
  • Roughly 0.5 ms on a Haswell Xeon
TSX with a fall-back

**start:**

```c
int status = xbegin
if (status == ok)       // != ok when aborted
  if (fallback-in-use())
    xabort            // fall-back in use
  else goto code    // fast-path
if (shouldRetry())      // retry policy
goto start
else
  use-fallback()       // use fall-back
```

**code:**

```c
application logic
```

```c
if (inFastPath)
  xend            // fast-path
else
  quit-fallback() // fall-back
```
TSX with a fall-back: a single lock

```c
start:
int status = xbegin
if (status == ok)    // != ok when aborted
    if (isTaken(lock))
        xabort        // fall-back in use
    else goto code  // fast-path
else goto code       // fast-path
if (shouldRetry())   // retry policy
    goto start
else
    acquire(lock)  // use fall-back

code:
    application logic
if (inFastPath)      // fast-path
    xend
else
    release(lock)  // fall-back
```
Lesson #1

Tuning best-effort HTMs is extremely important

The hardware is only a part of the solution.
Lesson #1

Tuning best-effort HTMs is extremely important

The hardware is only a part of the solution.

Avoid HLE when possible
  • the fallback is triggered too often
  • cannot be tuned
Lesson #1

Tuning best-effort HTMs is extremely important

The hardware is only a part of the solution.

Avoid HLE when possible
- the fallback is triggered too often
- cannot be tuned

2x improvement by choosing the best configuration on average across all workloads.
Lesson #1: Tuning TSX

Which fall-back to use?
  • Lock implementation

When to take the fall-back?
  • Retry policy
  • Contention management
Lesson #1: Tuning TSX

Which fall-back to use?
- Lock implementation

When to take the fall-back?
- Retry policy
- Contention management

<table>
<thead>
<tr>
<th>Lock</th>
<th>Performance</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ticket</td>
<td>1.0</td>
<td>1.1</td>
</tr>
<tr>
<td>MCS</td>
<td>2.4</td>
<td>1.2</td>
</tr>
<tr>
<td>CLH</td>
<td>2.9</td>
<td>2.4</td>
</tr>
<tr>
<td>RW</td>
<td>14.2</td>
<td>17.4</td>
</tr>
<tr>
<td>TTAS</td>
<td>15.2</td>
<td>17.4</td>
</tr>
<tr>
<td>Spin</td>
<td>16.4</td>
<td>17.5</td>
</tr>
</tbody>
</table>

average across all benchmarks and thread counts
Lesson #1: Tuning TSX

Which fall-back to use?
- Lock implementation

When to take the fall-back?
- Retry policy
- Contention management

Avoid lemming effect [ASPLOS12]
- avalanche aborts that exhaust retry policy

Manage contention with auxiliary lock [PPOPP13]
- fallback lock creates spurious aborts

Retry policy using literature values [HPC13,HPCA14]
- give up on HTM after a threshold of aborts

<table>
<thead>
<tr>
<th>Lock</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Performance</td>
</tr>
<tr>
<td>Ticket</td>
<td>1.0</td>
</tr>
<tr>
<td>MCS</td>
<td>2.4</td>
</tr>
<tr>
<td>CLH</td>
<td>2.9</td>
</tr>
<tr>
<td>RW</td>
<td>14.2</td>
</tr>
<tr>
<td>TTAS</td>
<td>15.2</td>
</tr>
<tr>
<td>Spin</td>
<td>16.4</td>
</tr>
</tbody>
</table>

average across all benchmarks and thread counts
Software TM in the picture
Software TM in the picture

Source program

```java
int i = 0

...

atomic {
    i++
}
```
Software TM in the picture

Source program

```c
int i = 0
...
atomic {
    i++
}
```

Compiled program

```c
int i = 0
...
TM.begin-tx()
int tmp = TM.read(&i)
tmp++
TM.write(&i, tmp)
TM.end-tx()
```
Over 10 years of research on STM ➔ many prototypes and designs. We considered four state of the art implementations:

- **TL2**: commit-time locking, used in Intel paper for comparison with TSX
- **Norec**: single commit lock, least instrumentation overhead
- **TinySTM**: encounter-time locking
- **SwissTM**: lazy val for r/w and eager for w/w; novel contention manager
Fall-backs for best-effort HTM

Single global lock
  • as seen before
Fall-backs for best-effort HTM

Single global lock
  • as seen before

Fine-grained locks
  • possibly check more than one lock
  • requires programmer to define which and how many locks
  • or automatic lock inference techniques [Transact06,LCPC13]
Fall-backs for best-effort HTM

Single global lock
  • as seen before

Fine-grained locks
  • possibly check more than one lock
  • requires programmer to define which and how many locks
  • or automatic lock inference techniques [Transact06,LCPC13]

STMs
  • separate code paths
    • uninstrumented for fast path in HTM
    • instrumented reads and writes for STM
  • use HTM to boost STM commit [SPAA13]
  • NOrec and TL2 for HybridTMs
Fall-backs for best-effort HTM

Single global lock
  • as seen before

Fine-grained locks
  • possibly check more than one lock
  • requires programmer to define which and how many locks
  • or automatic lock inference techniques [Transact06,LCPC13]

STMs
  • separate code paths
    • uninstrumented for fast path in HTM
    • instrumented reads and writes for STM
  • use HTM to boost STM commit [SPAA13]
  • NOrec and TL2 for HybridTM

• tomorrow: Invyswell
Experimental settings

Synchronization techniques under comparison:

- **Locking**
  - GL
  - FL

- **HTM**
  - TSX-GL
  - TSX-FL

- **STM**
  - TL2
  - NOrec
  - SwissSTM
  - TinySTM

- **HyTM**
  - TSX-TL2
  - TSX-NOrec
Experimental settings

Target machine:
• Intel Haswell Xeon E3-1275v3 3.5GHz (3.9GHz Turbo)
• 4 cores, 8 hardware threads (via hyper-threading)

Standard metrics for evaluation:
• Time to complete benchmarks
  • presented as speedup
• Energy consumed (collected via Intel RAPL)
  • presented as relative energy
• The baseline for comparison is a sequential, non-synchronized execution

Benchmarks:
• 7 STAMP benchmarks
• Memcached [ASPLOS14]
• Concurrent data-structures
Lesson #2: HTM is not a silver bullet
Lesson #2: HTM is not a silver bullet

Tested 900 scenarios (STAMP+data structures)
Identified three categories of applications:
Lesson #2: HTM is not a silver bullet

Tested 900 scenarios (STAMP+data structures)
Identified three categories of applications:

1. Uncontested winner
   • concurrent data structures
   • STAMP: small, occasional transactions
Lesson #2: HTM is not a silver bullet

Tested 900 scenarios (STAMP+data structures)
Identified three categories of applications:

1. Uncontested winner
   • concurrent data structures
   • STAMP: small, occasional transactions

![Graph showing performance comparison between STX-GL, TSX-NOrec, and TinySTM across different thread counts.](image)
Lesson #2: HTM is not a silver bullet

Tested 900 scenarios (STAMP+data structures)
Identified three categories of applications:

1. Uncontested winner
   • concurrent data structures
   • STAMP: small, occasional transactions

2. Only better without hyper-threading
   • more competitive energy-wise
   • spurious aborts due to hw restrictions
Lesson #2: HTM is not a silver bullet

Tested 900 scenarios (STAMP+data structures)
Identified three categories of applications:

1. Uncontested winner
   • concurrent data structures
   • STAMP: small, occasional transactions

2. Only better without hyper-threading
   • more competitive energy-wise
   • spurious aborts due to hw restrictions
Lesson #2: HTM is not a silver bullet

Tested 900 scenarios (STAMP+data structures)
Identified three categories of applications:

1. Uncontested winner
   • concurrent data structures
   • STAMP: small, occasional transactions

2. Only better without hyper-threading
   • more competitive energy-wise
   • spurious aborts due to hw restrictions

3. Always worse than others
   • except for single-threaded execution
   • capacity aborts and quantum exhaustion
Lesson #2: HTM is not a silver bullet

Tested 900 scenarios (STAMP+data structures)
Identified three categories of applications:

1. Uncontested winner
   - concurrent data structures
   - STAMP: small, occasional transactions

2. Only better without hyper-threading
   - more competitive energy-wise
   - spurious aborts due to hw restrictions

3. Always worse than others
   - except for single-threaded execution
   - capacity aborts and quantum exhaustion
Lesson #3: STM is still very competitive
Lesson #3: STM is still very competitive

Most robust all-around solution
• albeit more power hungry than HTM-based approaches
Lesson #3: STM is still very competitive

Most robust all-around solution
• albeit more power hungry than HTM-based approaches

Considering the best STM (SwissTM would be similar)
Lesson #3: STM is still very competitive

Most robust all-around solution
• albeit more power hungry than HTM-based approaches

Considering the best STM (SwissTM would be similar)
• Worst at 1 thread

Relative time

All STAMP benchmarks

Relative energy
Lesson #3: STM is still very competitive

Most robust all-around solution
• albeit more power hungry than HTM-based approaches

Considering the best STM (SwissTM would be similar)
• Worst at 1 thread
• Turning point at 3 threads
Lesson #3: STM is still very competitive

Most robust all-around solution
• albeit more power hungry than HTM-based approaches

Considering the best STM (SwissTM would be similar)
• Worst at 1 thread
• Turning point at 3 threads
• 84% more performance, 46% more energy-efficiency (over TSX-GL)
Lesson #4: Fine-grained locking is not worth it
Lesson #4: Fine-grained locking is not worth it

HTM is able to achieve the same degree of parallelism
- TSX-FL checks more locks in each transaction on average
- tends to be worse than TSX-GL

![Graph showing Speedup and Relative Energy with fine-grained locks](image)
Lesson #4: Fine-grained locking is not worth it

HTM is able to achieve the same degree of parallelism
- TSX-FL checks more locks in each transaction on average
- tends to be worse than TSX-GL

Benchmarks include concurrent data-structures
- small transactions benefit HTM or FL approaches
Research Directions: HybridTMss
Research Directions: HybridTMs

None of the evaluated HybridTMs is ever the best approach
Research Directions: HybridTMs

None of the evaluated HybridTMs is ever the best approach
Research Directions: HybridTMs

None of the evaluated HybridTMs is ever the best approach
Research Directions: HybridTMs

None of the evaluated HybridTMs is ever the best approach

- Spurious aborts from fallback of STM with HTM
- More efficient algorithms exist with non-transactional operations
  - Not available on Intel TSX or IBM BG/Q
- Is it a requirement for efficient future HybridTMs?

![Graphs showing Speedup and Relative Energy for All STAMP benchmarks with different HybridTMs: TSX-GL, TSX-NOrec, and TinySTM.](image-url)
Research Directions: Compiler Instrumentation
Research Directions: Compiler Instrumentation

STMs (and HybridTM’s software path) used manual instrumentation.
Research Directions: Compiler Instrumentation

STMs (and HybridTM’s software path) used manual instrumentation.

What changes if we rely on the compiler?
Research Directions: Compiler Instrumentation

STMs (and HybridTM's software path) used manual instrumentation.

What changes if we rely on the compiler?  GCC 4.8
Research Directions: Compiler Instrumentation

STMs (and HybridTM’s software path) used manual instrumentation.

What changes if we rely on the compiler? GCC 4.8

• Read- and write-sets can increase up to 3x (in SSCA2)
• Conservative compiler instruments accesses that are clearly not shared
• Even simple static analysis (such as those in Clang) would improve
Research Directions: Compiler Instrumentation

STMs (and HybridTM’s software path) used manual instrumentation.

What changes if we rely on the compiler? GCC 4.8

- Read- and write-sets can increase up to 3x (in SSCA2)
- Conservative compiler instruments accesses that are clearly not shared
- Even simple static analysis (such as those in Clang) would improve

Also important for HTM:
- If non-transactional operations are available
- May reduce capacity aborts
Research Directions: Automatic HTM tuning
Research Directions: Automatic HTM tuning

We used the best tuning of TSX on average.
Research Directions: Automatic HTM tuning

We used the best tuning of TSX on average.

But what about the optimal for each case?
Research Directions: Automatic HTM tuning

We used the best tuning of TSX on average.

But what about the optimal for each case?

<table>
<thead>
<tr>
<th>Speedup %</th>
<th>Kmeans</th>
<th>SSCA2</th>
<th>Intruder</th>
<th>Vacation</th>
<th>Genome</th>
<th>Yada</th>
<th>Labyrinth</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 threads</td>
<td>12</td>
<td>7</td>
<td>20</td>
<td>36</td>
<td>12</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>8 threads</td>
<td>5</td>
<td>8</td>
<td>80</td>
<td>21</td>
<td>2</td>
<td>55</td>
<td>39</td>
</tr>
</tbody>
</table>
Research Directions: Automatic HTM tuning

We used the best tuning of TSX on average.

But what about the optimal for each case?

<table>
<thead>
<tr>
<th>Speedup %</th>
<th>Kmeans</th>
<th>SSCA2</th>
<th>Intruder</th>
<th>Vacation</th>
<th>Genome</th>
<th>Yada</th>
<th>Labyrinth</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 threads</td>
<td>12</td>
<td>7</td>
<td>20</td>
<td>36</td>
<td>12</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>8 threads</td>
<td>5</td>
<td>8</td>
<td>80</td>
<td>21</td>
<td>2</td>
<td>55</td>
<td>39</td>
</tr>
</tbody>
</table>

Technique for optimal tuning, focus only on performance:
  * Self-Tuning Intel TSX --- Best paper at USENIX ICAC’14
Summary
Summary

HTM is not a silver bullet
• Shines with short infrequent transactions and concurrent data structures
  • Great in energy efficiency and at low thread count
• Hyper-threading amplifies HTM’s inherent limitations
• HTM requires careful tuning of parameters governing the fallback:
  • automatic tuning is highly desirable to preserve ease of usage
Summary

HTM is not a silver bullet
- Shines with short infrequent transactions and concurrent data structures
  - Great in energy efficiency and at low thread count
- Hyper-threading amplifies HTM’s inherent limitations
- HTM requires careful tuning of parameters governing the fallback:
  - automatic tuning is highly desirable to preserve ease of usage

STM performs best on average
- …and with applications with complex transactions
- Its energy efficiency tends to be worse than HTM
- Compiler instrumentation has room for improvement
Summary

HTM is not a silver bullet
- Shines with short infrequent transactions and concurrent data structures
  - Great in energy efficiency and at low thread count
- Hyper-threading amplifies HTM’s inherent limitations
- HTM requires careful tuning of parameters governing the fallback:
  - automatic tuning is highly desirable to preserve ease of usage

STM performs best on average
- …and with applications with complex transactions
- Its energy efficiency tends to be worse than HTM
- Compiler instrumentation has room for improvement

HybridTMs are not there yet
- Need better support from hardware
- Can we do better without it?