Improved Single Global Lock Fallback for Best-effort Hardware Transactional Memory

TRANSACT 2014

Irina Calciu
Tatiana Shpeisman
Gilles Pokam
Maurice Herlihy
Multicore Performance Scaling

- Supralinear
- Linear
- Typical scaling
- No performance scaling

Goal: Better scaling.
Hardware Transactional Memory (HTM)

IBM’s Blue Gene/Q & System Z & Power Architecture

Intel’s Haswell TSX: RTM & HLE

Low overhead (cache based)
Haswell RTM

if (_xbegin() == _XBEGIN_STARTED)
  Speculate Execution
  _xend()
else
  Abort Handler

  Speculate Execution, without any locks
  Read and Write Sets
  Abort on memory conflict
Haswell RTM

if (_xbegin() == _XBEGIN_STARTED)
Speculate Execution
_xend()

_xbegin()
Read X → Add to Read Set
Write Y → Add to Write Set

_xbegin()

Add to Write Set ← Write X
Add to Write Set ← Write Y

COMMIT
_xend() → Make the change to Y visible

ABORT
_xbegin()

_xend()
Lock Elision

Atomic region executed as a *transaction* or *mutually exclusive on L*

Execute optimistically, without any locks

Track Read and Write Sets

Abort on memory conflict: rollback acquire lock
Lock Elision

Lock transfer latencies
Serialized execution

Time

T0 T1 T2 T3

T0 T1 T2 T3

Concurrent execution
No lock transfer latencies

Reducing lock instruction latencies insufficient

[Anand Tech]
Haswell RTM

Small & Medium Transactions

Overflow

Conflicts

Interrupts

Best-effort

Unsupported Instructions

Needs software fallback
Overview

- Best-effort Hardware Transactional Memory
- Lazy SGL
- Bloom Filter SGL

Description

Correctness

Results
Single Global Lock HyTM (simple and common)

Try_SPEC:

Wait until Lock is free
Transaction_Read(Lock)
If Lock is taken ABORT
Speculate critical section
End speculation

On_ABORT:
If try_lock(Lock)
Critical section
Release(Lock)
Else Try_SPEC

Does not abort!
Single Global Lock HyTM (simple and common)
Thread 1

Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)

Acquire(L)
CRITICAL SECTION (SW TXN)
Release(L)

Thread 2

Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)

Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)

Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)

Time

Execution Time 1
Lazy SGL

Try_SPEC:

Speculate critical section

Transactional_Read(Lock)

If Lock is taken ABORT

End speculation

On_ABORT:

If try_lock(Lock)

Critical section

Release(Lock)

Else Try_SPEC

Does not abort!
Lazy SGL

Legend:
X = ABORT

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Begin SW txn</td>
</tr>
<tr>
<td></td>
<td>Acquire L</td>
</tr>
<tr>
<td></td>
<td>Begin HW txn</td>
</tr>
<tr>
<td></td>
<td>Read L</td>
</tr>
<tr>
<td></td>
<td>Release L</td>
</tr>
<tr>
<td></td>
<td>End</td>
</tr>
<tr>
<td></td>
<td>HW txn</td>
</tr>
<tr>
<td></td>
<td>Begin HW txn</td>
</tr>
<tr>
<td></td>
<td>Read L</td>
</tr>
<tr>
<td></td>
<td>End</td>
</tr>
<tr>
<td></td>
<td>HW txn</td>
</tr>
<tr>
<td>(1)</td>
<td>Begin HW txn</td>
</tr>
<tr>
<td></td>
<td>Read L</td>
</tr>
<tr>
<td></td>
<td>End</td>
</tr>
<tr>
<td></td>
<td>HW txn</td>
</tr>
<tr>
<td>(2)</td>
<td>Begin HW txn</td>
</tr>
<tr>
<td></td>
<td>Read L</td>
</tr>
<tr>
<td></td>
<td>End</td>
</tr>
<tr>
<td></td>
<td>HW txn</td>
</tr>
<tr>
<td>(3)</td>
<td>Begin HW txn</td>
</tr>
<tr>
<td></td>
<td>Read L</td>
</tr>
<tr>
<td></td>
<td>End</td>
</tr>
<tr>
<td></td>
<td>HW txn</td>
</tr>
</tbody>
</table>

COMMIT
Overview

- Best-effort Hardware Transactional Memory
- Lazy SGL
- Bloom Filter SGL

Description

Correctness

Results
Transactional Memory Correctness

Transaction 1
SW
ORDER T2 AFTER T1
COMMIT

Transaction 2
HW
ORDER T2 BEFORE T1
COMMIT
Case 1:  
HW begins  
SW begins  
HW ends  
SW ends

X value:  
[ ]  
Correct: a  
Actual: b  
✗

Thread 1  
(SW)

Thread 2  
(HW)

Acquire Lock  
TXN_BEGIN

X = a  
...

Release Lock  
TXN_END

Check Lock  
ABORT

X = b  
...

Correct: a  
Actual: b  
✗
Case 2:
SW begins
HW begins
HW ends
SW ends

X value:
Correct: a
Actual: b

Thread 1
(SW)
Acquire Lock
... X = a
... TXN_BEGIN
... X = b
... TXN_END
Release Lock
Thread 2
(HW)

Time

Correct: a
Actual: b

ABORT

Check Lock
**Case 3:**
SW begins
HW begins
SW ends
HW ends

X value:
Correct: b
Actual: b

**Thread 1**
(SW)

Acquire Lock
TXN_BEGIN
...  
X = a
...  
Release Lock
TXN_END

**Thread 2**
(HW)

TXN_BEGIN
...  
X = b
...  
TXN_END

Check Lock

Correct: b
Actual: b

COMMIT
Case 4:
HW begins
SW begins
SW ends
HW ends

Thread 1
(SW)

Thread 2
(HW)

Acquire Lock
...
X = a
...
Release Lock

TXN_BEGIN

X = b
...

TXN_END

Thread 1
Thread 2

Time

X value:
Correct: b
Actual: b

Correct: b
Actual: b

Check
Lock

COMMIT
Hardware Sandboxing

Thread 1
(SW)

X = 5; Y = 6
Acquire Lock

... ++X
... ++Y
... Release Lock

Thread 2
(HW)

TXN_BEGIN

Z = 1/(Y-X)

TXN_END

Z = 1/0 !!!
Indirect Jumps

Thread 1 (SW)

X = 5; Y = 6
Acquire Lock
...
++X
...
++Y
...
Release Lock

Thread 2 (HW)

_xbegin
...
if (X == Y) *p = garbage
p()
...
if (lock) abort
_xend
Overview

• Best-effort Hardware Transactional Memory

  Description

  • Lazy SGL

  Correctness

  • Bloom Filter SGL

Results
Intruder (medium txns)

Better

Ssca2 (small txns)

Labyrinth (large txns)
Improved Lock Acquisition Rate

Vacation Low (medium txns)

Kmeans High (small txns)

Intruder (medium txns)

Labyrinth (large txns)

Better
No single thread overhead

Slowdown relative to sequential for 1 thread

- TL2
- SGL
- HLE
- E-SGL
- L-SGL
Overview

- Best-effort Hardware Transactional Memory
- Lazy SGL
- Bloom Filter SGL
Bloom Filters

• Efficient probabilistic data structure to compute fast set intersection

• Can admit false positives

• No false negatives

• Used in TM for Conflict Detection
Lazy SGL

Legend:
X = ABORT

Time
Case 1:
HW begins
SW begins
HW ends
SW ends

X value:

Correct: a
Actual: a

Thread 1
(SW)

Thread 2
(HW)

Acquire Lock

TXN_BEGIN

X = a

Release Lock

TXN_END

If BFs intersect: ABORT
Else: COMMIT

Time

Check BF
Case 2:
SW begins
HW begins
HW ends
SW ends

Thread 1
(SW)

Thread 2
(HW)

Acquire Lock
...
X = a
...

TXN_BEGIN
...
X = b
...

TXN_END

Check BF

If BF intersects:
ABORT
Else: COMMIT

X value:
Correct: a
Actual: b
Conclusions

• HTMs are becoming more available

• Best-effort – need software fallback

• Eager SGL
  • simple and fast fallback,
  • often preferred to more efficient solutions
Conclusions

- Lazy SGL
  - as simple as Eager SGL
  - more efficient

- Bloom Filter SGL
  - more accurate conflict detection
  - Slower

- Can be implemented directly in hardware