Making the Fast Case Common and the Uncommon Case Simple in Unbounded Transactional Memory

Colin Blundell (University of Pennsylvania)
Joe Devietti (University of Pennsylvania)
E Christopher Lewis (VMware, Inc.)
Milo M. K. Martin (University of Pennsylvania)
Overview

• Small transactions: no problem
  • Implement using local structures of bounded size
  • Simple/highly-concurrent/low-overhead
• Overflowed transactions: problem
  • Difficult to preserve all nice properties of bounded TM
  • Many papers in last several years
• Previous approaches: focus on concurrency
  + Sustain performance as overflows increase
    – Involve complex resource manipulation
• Our approach: decouple into two problems
  • Simple overflow handling: OneTM
  • Making overflows rare: Permissions-only cache
Background

• Transactional memory: the new hot thing
  • Interface: serialization
  • Implementation: optimistic parallelism

• Tasks of every TM
  • Conflict detection: was serializability violated?
  • Version management: how do we recover serializability?

• Bounded hardware TM implementation:
  • Conflict detection: extend cache coherence
  • Version management: many schemes
Running Example

L1 Cache

- Tags
- State
- Data

- L1 direct-mapped
- No L2
- Invalidation-based system
- b & d map to same L1 entry

Memory
Transactional Execution

L1 Cache

<table>
<thead>
<tr>
<th>Tags</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>M</td>
<td>56</td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>a:</th>
<th>b:</th>
<th>d:</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>25</td>
<td>17</td>
</tr>
</tbody>
</table>

Checkpoint
Conflicts Detection

L1 Cache

- Tags:
  - a: S
  - b: M

- State:
  - a: S
  - b: M

- Data:
  - a: 31
  - b: 56

Memory

- a: 31
- b: 25
- d: 17

Load a from P0 to L1 Cache.

a: read from P1.

Conflict detection is local.
Committing a Transaction

L1 Cache

<table>
<thead>
<tr>
<th>Tags</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>M</td>
<td>56</td>
</tr>
</tbody>
</table>

Memory

- a: 31
- b: 25
- d: 17

+ Commits are local
Version Management

L1 Cache

<table>
<thead>
<tr>
<th>Tags</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>M</td>
<td>56</td>
</tr>
</tbody>
</table>

Memory

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a:</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>b:</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>d:</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

+ Commits do not change
+ Log is not bounded
Aborting a Transaction

L1 Cache

<table>
<thead>
<tr>
<th>Tags</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>M</td>
<td>56</td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>a:</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>b:</td>
<td>25</td>
</tr>
<tr>
<td>d:</td>
<td>17</td>
</tr>
</tbody>
</table>

Log

P0

P1

checkpoint
b: read

OneTM - Blundell - ISCA 2007 [ 9 ]
The Catch: Overflows

Load d

L1 Cache

Tags | State | Data
--- | --- | ---
a | S | R | 31
b | M | W | 56 | 42

Memory

a: 31
b: 25
d: 17

Need another mechanism for conflict detection
Handling Overflows: Strawman

L1 Cache

<table>
<thead>
<tr>
<th>Tags</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>M</td>
<td>56</td>
</tr>
</tbody>
</table>

Memory

|    | a: 31 | b: 2 | d: 17 |

checkpoint

load d

Log

b: 56
Handling Overflows: Strawman

L1 Cache

Tags | State | Data
--- | --- | ---
a | S R | 31

Memory

- **a**: 31
- **b**: 26, 42
- **d**: 17

+ Preserved safety
The Catch to Handling Overflows

L1 Cache

<table>
<thead>
<tr>
<th>Tags</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S</td>
<td>31</td>
</tr>
<tr>
<td>d</td>
<td>S</td>
<td>17</td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>Tags</th>
<th>Data</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>42</td>
<td>W</td>
</tr>
<tr>
<td>d</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

Log

b: 56

Need metadata for all n processors
The Catch to Handling Overflows

L1 Cache

<table>
<thead>
<tr>
<th>Tags</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>R</td>
<td>17</td>
</tr>
<tr>
<td>d</td>
<td>S</td>
<td>17</td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>Data</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>W</td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

Log

b: 56

Need metadata for all n processors each SW thread

unbounded
The Catch to Handling Overflows

L1 Cache

Tags | State | Data
---|---|---
a | S R | 31
b | | d | S R | 17

Memory

a: 31
b: 26 42

dunded

How to detect conflicts efficiently?
How to commit efficiently?
How to (de)allocate metadata?

UTM, VTM, PTM, Bulk, LogTM(-SE),…

Log

b: 56
Rest of my talk: a different approach

- **Claim 1:** bounding concurrency of overflows simplifies implementation
  - Eases the problem of conflict detection
  - Removes the problem of dynamic metadata allocation
- **Is unbounded concurrency necessary?**
  - Depends on the frequency of overflows
- **Claim 2:** We can make overflows rare
- Take each claim in order
  - Claim 1: OneTM
  - Claim 2: Permissions-only cache
OneTM

- **Key idea:** one overflowed transaction at a time
  - On a per-application basis
  - Better name: HighlanderTM?
- Two implementations
  - **OneTM-Serialized:** all threads stall for overflow
  - **OneTM-Concurrent:** serialize only overflows
- Key mechanism: per-application *overflow bit*
  - Processors check to determine when to stall
  - Coherently cached in a special register
OneTM-Serialized

Fully Concurrent

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
</table>

OneTM-Serialized

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
</table>

No changes to bounded TM

Similar to original TCC, but:
- Maintain aborts
- Standard CC protocol

Non-trans

Bounded

Overflowed

Stalled

4-processor execution
No conflicts
OneTM-Serialized: Evaluation

Takeaway #1: If overflows are rare, serialization is sufficient.

Compare to TM that idealizes overflow handling.
First workload: SPLASH2.

8 processors
Simics + GEMS

Normalized Runtime

idealized overflows
OneTM-Serialized

Normalized Runtime

barnes cholesky ocean radix raytrace-base raytrace-opt volrend water
OneTM-Serialized: Evaluation

- \texttt{btree-<n>}: mix of updates & read scans (n\% read scans)
  - Performance worse as number of overflows increases
OneTM-Concurrent

<table>
<thead>
<tr>
<th>Fully Concurrent</th>
<th>OneTM-Serialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0   P1   P2   P3</td>
<td>P0   P1   P2   P3</td>
</tr>
</tbody>
</table>

- Non-trans
- Bounded
- Overflowed
- Stalled

4-processor execution
No conflicts
OneTM-Concurrent Conflict Detection

L1 Cache

<table>
<thead>
<tr>
<th>Tags</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>M</td>
<td>56</td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th></th>
<th>a: 31</th>
<th>b: 2</th>
<th>d: 17</th>
</tr>
</thead>
</table>

Log

b: 56

load d
OneTM-Concurrent Conflict Detection

L1 Cache

- Tags: a, b, d
- State: S, R
- Data: 31, 42

Memory

- a: 31
- b: 23, 42 (W)
- d: 17

Log

- b: 56

Checkpoint

load d

Preserved safety

- Added metadata

bounded
OneTM-Concurrent Commits

• **Problem:** actively clearing metadata is nasty
  • Commit is now a high-overhead operation
• **Solution:** lazy clearing of metadata
  • Mechanism: overflowed transaction ID’s
  • Block metadata extended to include ID’s
  • Current ID stored with overflow bit
  • **Key:** only one active ID (so, notion of a “current ID”)
• Changes
  + **Commit now cheap**
    – Widens datapath
    – Admits false conflicts (since ID’s are finite-length)
OneTM-Concurrent: Evaluation

- Performance better than OneTM-Serialized
- Still falls off ideal as overflows increase
The Permissions-Only Cache

Goal: avoid overflow
Sol’n: permissions-only cache
The Permissions-Only Cache

L1 Cache

Tags | State | Data
---- | ---- | ----
  a  | S R  | 31
  b  |      |    
  c  |      |    
  d  |      |    

PO Cache

Tags | State
---- | ----
  b  | E W 
  c  |    
  d  |    

Log

b: 56

d: read

Memory

a: 31
b: 242

Checkpoint

load d

P0

P1
The Permissions-Only Cache

L1 Cache

Tags | State | Data
--- | --- | ---
a | S R | 31
b | | 
d | S R | 17

PO Cache

Tags | State
--- | ---
b | E W

Memory

a: 31
b: 26 42
d: 17

Log

b: 56

Basically unchanged:
+ Conflict detection
+ Version management
+ Commits & aborts
The Permissions-Only Cache

- Two key features
  1. Accessed only on snoops and evictions
  2. Efficient encoding (sector cache)
- **Impact:** Extends overflow threshold
  - 4 KB PO cache: \( \sim 1 \text{ MB} \) data
  - 64 KB PO cache: \( \sim 16 \text{ MB} \) data
  - Store metadata in 4 MB L2 data lines: up to 1 GB data

**Takeaway #2:**
We can engineer systems for rare overflows
The Permissions-Only Cache: Evaluation

Add 4 KB permissions-only cache to OneTM
The Permissions-Only Cache: Evaluation

Overflows reduced to virtually nil
OneTM-Serialized + PO cache: a sweet spot?
Related Work

• Lots!
• Proposals with low-overhead overflow handling mechanisms
  • UTM/LTM, VTM, PTM, LogTM, …
  • Our scheme: PO cache reduces overflow, OneTM handles it simply
  • Many proposals enhanced by permissions-only cache
• Bounded HTM’s backed by software (HyTM, XTM, …)
  • Similar philosophy to ours (uncommon case simple)
  • Their schemes maintain concurrency but introduce overheads…
  • …OneTM-Concurrent sacrifices concurrency but has low overheads
  • Again, enhanced by permissions-only cache
• Signature-based TMs: conflict detection through finite-sized signatures (Bulk, LogTM-SE, …)
  + Signatures can be saved architecturally
  + Serialize gradually rather than abruptly
  – Still an unbounded number of signatures
Conclusions

- **OneTM**: make overflow handling simple
  - OneTM-Serialized: entry-point unbounded TM
  - OneTM-Concurrent: more robust to overflows
- **Permissions-only cache**: make overflows rare
  + Can engineer to keep overflow rate low for your workload
  + Enhances many prior unbounded TM proposals

**Combination**: TM that’s both fast and simple to implement
LogTM-SE

+ Very neat!
- Paging more complex than in OneTM
- Commit of a transaction that has migrated processors must trap to OS
  • Our hope for PO cache: overflow only on context switch
    • And there LogTM-SE loses directory filter…
  • Sticky state + OneTM-Serialized?
Hybrid Transactional Memories

• Similar philosophy to OneTM
• Our goal: make overflows so rare that it doesn’t really matter what you use for them
  • And then OneTM-Serialized is pretty simple…
• If overflows are frequent, need to handle them with high performance
  • Permissions-only cache + UTM/VTM/PTM?
• Spot in the middle for hybrid TM’s/OneTM-Concurrent
  • Occasional overflow: OneTM-Concurrent appealing
  • Tipping point where concurrency matters more than overheads…I don’t know where it is (need workloads)
Context Switching & Paging

• Context switching “just works”
  • OneTM-Serialized: overflowed bit persists
  • OneTM-Concurrent: metadata persists as well

• Paging during an overflowed transaction:
  • OneTM-Serialized: no problem
  • OneTM-Concurrent: page metadata (OS help)

• Paging during a bounded transaction:
  • Abort and transition to overflowed mode
Transitioning to Overflowed Mode

• OneTM-Serialized: just set the bit
  • Synchronize access
• OneTM-Concurrent: have to set metadata
  • Simple: abort and restart (what we simulate)
  • Higher-performance schemes are possible
    • Walk the cache
    • Overflow gradually
Summary

L1 Cache

Tags | State | Data
--- | --- | ---
a | S | 31
b | M | 56
   | W | 42

Memory

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
a: | 31 |
b: | 25 |
d: | 17 |

Overflow bit

Checkpoint

Log

b: 56
Summary

L1 Cache

Tags | State | Data
---|---|---
a | S R | 31
b | M W | 56 42

PO Cache

Tags | State
---|---

Memory

a: 31
b: 25
d: 17

Metadata
(for OneTM-Concurrent only)

overflow bit
checkpoint

PO Cache

Tags | State
---|---

Log

b: 56

Copyright © 2023 Penn University of Pennsylvania Architecture + Compilers Group

OneTM - Blundell - ISCA 2007 [40]
The Permissions-only Cache: Efficient Storage

- Sector cache to reduce tag overhead
- Now: (close to) 2 bits per data block
- 64-byte blocks: 256 to 1 compression ratio
- 4 KB metadata: 1 MB transactional data
- Even larger: metadata in L2 data lines
- Add bit to distinguish data/metadata
- 4 MB L2: 1 GB transactional data