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

Conflict Detection

Committing a Transaction

Version Management

+ Commits are local
+ Conflict detection is local

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

Memory

UTM, VTM, PTM, …

checkpoint

P0

P1

Load d

b: read

Log

Tags State Data

a: S R 31
b: M W 25 42
d: 17

Memory

a: 31
b: 25
d: 17

Log

b: 56

The Catch: Overflows

Memory

UTM, VTM, PTM, …

checkpoint

P0

P1

Load d

b: read

Log

Tags State Data

a: S R 31
b: M W 25 42
d: 17

State

S
R
M
W

Tags

S
R
M
W

Data

L1 Cache

Preserved safety

Handling Overflows: Strawman

Memory

UTM, VTM, PTM, …

checkpoint

P0

P1

Load d

b: read

Log

Tags State Data

a: S R 31
b: M W 25 42
d: 17

State

S
R
M
W

Tags

S
R
M
W

Data

L1 Cache

+ Preserved safety
The Catch to Handling Overflows

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: Evaluation

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

OneTM-Serialized: Evaluation

- **First workload:** SPLASH2
- **Second workload:** btree-n: mix of updates & read scans (n% read scans)
- Performance worse as number of overflows increases
OneTM-Concurrent

Fully Concurrent

OneTM-Serialized

Non-trans  Bounded  Overflowed  Stalled

4-processor execution
No conflicts

OneTM-Concurrent Conflict Detection

Load d  P0  checkpoint  b: read  P1

Tags  State  Data
a  S  R  31
b  M  W  30  42
d  37

Memory

+ 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
Sofn: permissions-only cache

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: ~1 MB data
  - 64 KB PO cache: ~16 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

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

<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>42</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>b: 59</td>
</tr>
</tbody>
</table>

<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>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metadata (for OneTM-Concurrent only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b: 31</td>
</tr>
<tr>
<td>d: 17</td>
</tr>
</tbody>
</table>
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

Tags

R W R W R W R W

Tags

R W R W R W R W