An Object-Aware Hardware Transactional Memory System

Behram Khan, Matthew Horsnell, Ian Rogers, Mikel Lujan, Andrew Dinn and Ian Watson

http://www.cs.manchester.ac.uk/apt/projects/TM
Why?

- In 2006 more than 60% of new applications were using managed runtime environments (JVM, CLR, …)
- In 2008 estimated to be more than 80%
- Current computer architectures only see a flat memory address space
- Communication infrastructure will be key in multi-core (many-core)
- We are interested in finding ways of reducing the demands on this infrastructure
In a Nutshell

- Similar to hardware support for paged virtual memory using virtually addressed caches and TLB
- Cache hierarchy addressing based on unique object identifiers
- Benefits:
  - Support of cache overflows of uncommitted data
  - Novel commit and conflict detection mechanisms
- Object granularity / Lazy versioning / Lazy conflict detection
Hardware Transactional Memory

- Buffer changes in hardware
- Exploit buffering already present in cache (TCC)
- Or create undo log and assume conflict is unlikely (LogTM)
- Resort to STM when buffers/logs overflow
- Detect conflicts at the cache line level
Problems

• TM proposals artificially work at granularities of existing memory systems
  • byte, short, int, long
  • cache line
  • page
• Often assume that conflicts are infrequent
  • rollback is expensive
• There is a lack of harmony between the world of the programmer (objects) and the hardware
Object-Aware Memory Architecture

Core

Load/Store Address

OID  Field

Pointer to Obj A

Object Translation Table (OTT)

Object A

Memory
Object-Aware Memory Architecture

- Core
- Load/Store Address
  - OID Field
  - OID Field Value
  - Object Cache
    - OID Field Value
    - Translation Buffer
      - OID Pointer to Obj A
      - OID Pointer to Obj X
  - Memory
    - OID Pointer to Obj A
    - Object A
  - Object Translation Table (OTT)
What’s been achieved

- Memory now addressed by objects
- Virtually addressed cache
- Large objects map to consecutive OIDs
- Using objects allows aspects of protocol to be concise whilst avoiding false sharing
- Indirection has been shown to aid garbage collection
  - Moving an object requires a single pointer update
Object-Aware Transactional Memory - during transaction

- Core

Load/Store Address

OID | Field
---|---

OID Field Value

OID Field Value

Object Cache

OID | Ptr X | Ptr X*
---|---|---

OID | Ptr A | Ptr A*
---|---|---

Translation Buffer

OID

Pointer to Obj A

Object Translation Table (OTT)

Object A

Memory

Object A*
Object-Aware Transactional Memory
- commit

Core

Load/Store Address

OID | Field
----|------

Object Cache

OID | Field | Value
----|------|------

Translation Buffer

OID

Object Translation Table (OTT)

OID

Pointer to Obj A

Object A

Object A*

Memory
Object-Aware Transactional Memory
- conflict

Transaction #1

<table>
<thead>
<tr>
<th>OID</th>
<th>Ptr X</th>
<th>Ptr X₁*</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>Ptr A</td>
<td>Ptr A₁*</td>
</tr>
</tbody>
</table>

Transaction Buffer

Object A₁*

Pointer to Obj A

(OTT)

Object A

Transaction #2

<table>
<thead>
<tr>
<th>OID</th>
<th>Ptr X</th>
<th>Ptr X₂*</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>Ptr A</td>
<td>Ptr A₂*</td>
</tr>
</tbody>
</table>

Translation Buffer

Object A₂*
Object-Aware Transactional Memory
Object-Aware Transactional Memory
- conflict detected by broadcast of OID

Transaction #1
Transaction Buffer

Translation Buffer

OID | Ptr X | Ptr X²*
Transaction #2

OID | Ptr A | Ptr A²*

Object A

(OTT)

Object A¹*

Memory

Object A²*
Simulated Object-Aware Transactional Memory

Core

O-Cache | TTB

Bus (Network)

L2 Cache

Translation Unit (T-Unit)

Memory
# Evaluation

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object size</td>
<td>128B.</td>
</tr>
<tr>
<td>L1 object cache</td>
<td>32KB, private, 4-way assoc, 32B line, 1-cycle access.</td>
</tr>
<tr>
<td>TTB</td>
<td>24KB, private, 4-way assoc, 12B lines, 1-cycle access.</td>
</tr>
<tr>
<td>Network</td>
<td>256-bit bus, split-transactions, pipelined, no coherence.</td>
</tr>
<tr>
<td>L2 cache</td>
<td>4MB, shared 32-way assoc, 32B lines, 16-cycles access.</td>
</tr>
<tr>
<td>TU</td>
<td>4MB, shared, 32-way assoc, 12B lines, 16-cycles access.</td>
</tr>
<tr>
<td>Memory</td>
<td>100-cycles off-chip access.</td>
</tr>
</tbody>
</table>

*Simulation parameters*
Normalized Execution Time

Execution Time Normalized To The Single-Core Time

- Idle
- Fetch
- Commit
- Violations
- Busy

Breakdown of Bus Traffic

Bus Utilization

- Idle
- Misc.
- L1-Writeback
- TTB Miss
- L1-Cache Miss
- Rd-broadcast
- L1-Cache Flush

Percentage Of Total Bus Cycles

What is Lee-TM?

- Realistic TM benchmark
- Benchmark based on Lee’s routing algorithm
  - Used in circuit routing
  - Each route is pair of points
  - Connect pair of points
    - Shortest distance possible
    - Can’t cross other routes
  - Connection on 3D array
    - Represents circuit board
## Evaluation

<table>
<thead>
<tr>
<th>Application</th>
<th>Readset/ Tx</th>
<th>Writeset/ Tx</th>
<th>Inst/ TX</th>
<th>Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>CoV</td>
<td>Mean</td>
<td>CoV</td>
</tr>
<tr>
<td>LeeTM</td>
<td>117.5</td>
<td>2.9</td>
<td>78.8</td>
<td>3.1</td>
</tr>
<tr>
<td>LeeTM-ER</td>
<td>15.8</td>
<td>1.4</td>
<td>78.8</td>
<td>3.1</td>
</tr>
</tbody>
</table>

### Lee profile

<table>
<thead>
<tr>
<th>Application</th>
<th>L1-Cache Flush</th>
<th>Oid-broadcast</th>
<th>L1-Cache Miss</th>
<th>TTB Miss</th>
<th>L1-Wrback</th>
<th>Misc</th>
<th>Idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeeTM</td>
<td>.8%</td>
<td>.5%</td>
<td>30.1%</td>
<td>17.4%</td>
<td>7%</td>
<td>4%</td>
<td>40%</td>
</tr>
<tr>
<td>LeeTM-ER</td>
<td>1.2%</td>
<td>1.3%</td>
<td>27.5%</td>
<td>33.9%</td>
<td>8%</td>
<td>3%</td>
<td>24%</td>
</tr>
</tbody>
</table>

### Bus Utilization
In a Nutshell

- Similar to hardware support for paged virtual memory using virtually addressed caches and TLB
- Cache hierarchy addressing based on unique object identifiers
- Benefits:
  - Support of cache overflows of uncommitted data
  - Novel commit and conflict detection mechanisms
- Object granularity / Lazy versioning / Lazy conflict detection
Summary

- **Object-Aware Hardware Transactional Memory**
  - Elegantly allows cache overflows of uncommitted data
  - Deal with frequent aborts (rollback overhead)
  - Reduce the demands on network-on-chip
    - OID broadcast accounts for less than 1.5% of the total bus traffic using LeeTM and LeeTM_ER benchmarks.
  - Concise handling of programmatic entities
  - Benefits to runtime and garbage collection
Backup Slides
Extensions

- Bitmaps within object
  - prevent false sharing within an object

- Self validation
  - Suspending (even migrating) transactions can be checked to see if cached object reference is the same as in the OTT

- Overflow
  - If caches and TTB overflow then parts can be written to memory - a bloom filter encoding OIDs can be used to conservatively approximate the TTB entries that have been spilled to memory
Speedup Results