Flexible Decoupled Transactional Memory Support

Arrvindh Shriraman
Sandhya Dwarkadas
Michael L. Scott
Department of Computer Science
UNIVERSITY of ROCHESTER
Transactions: Our Goal

- Lazy Txs (i.e., optimistic conflict resolution)
  - more concurrency

- SW coordinates conflict management
  - when (i.e., eagerly or lazily)
  - how (i.e., stalling, who aborts)

- Limitless Txs
  - Large: cache victimization and paging
  - Long: thread switches
Flexible Transactional Memory

- STM (e.g., RSTM)
  - all software approach

Execution Time

- **Versioning** (Isolation)
- **Validation** (Consistency check)
- **Bookkeeping** (Metadata ops.)
- **Application** (Useful Work)
Flexible Transactional Memory

STM (e.g., RSTM)
- all software approach

RTM [ISCA’ 07]
- new cache states help boundedtxs
- software handles large & long txs
Flexible Transactional Memory

- STM (e.g., RSTM)
  - all software approach

- RTM [ISCA’ 07]
  - new cache states help bounded txs
  - software handles large & long txs

- FlexTM [this paper]
  **Good Performance**
  - No per-location software metadata
  **Simple hardware**
  - No bulk arbiters like lazy HTMs
  **Allows software policy**
Decoupled Hardware Primitives (1/2)

- Separate interchangeable basic hardware ops. that can be coordinated by software

Why?
- Minimizes hardware state
  - small footprint, simplifies virtualization
  - reduces development time

- Software accessible
  - to build transactions & fine-tune policy decisions
  - to repurpose hardware for non-tx applications
Decoupled Hardware Primitives (2/2)

1. Data Isolation (delaying visibility of stores)
   - caches buffer speculative values, provide fast-commit
   - SW allocates overflow region & HW performs access

2. Access Summary (tracking locations accessed)
   - maintains list of locations read & written
   - check on coherence messages or local memory ops.

3. Conflict Summary (tracking data conflict events)
   - tracks conflict occurrence and type between processors

4. Alert-On-Update
   - monitor cache-blocks and trigger handlers
Outline

Preview

Data Isolation (aka. Lazy Versioning)
  – Lazy coherence
  – Overflow-Table

Conflict Management

FlexTM Software

Evaluation

Summary
Lazy Coherence (1/2): Approach

- **Lazy coherence:**
  - permit multiple readers & writers for a cache block
  - restore coherence for multiple lines simultaneously

- **Current Research (e.g., TCC, Bulk)**
  - bulk arbiters, bulk GetXs, bulk ops. on directory

- **Our approach: eager messages but lazy coherence**
  - look out for sharer conflicts in standard coherence msgs.
  - continue caching data, but use T-MESI states
  - simple bit-clear ops. convert T-MESI to MESI

No bulk messages or address ops.
Two new ‘T’ tagged states: TMI (T+M) and TI (T+I)
- TStores & TLoads denote speculative operations
  - ISA can include instructions or SW can tell HW the regions

- TMI buffers TStores
  - allows multiple writers and readers
  - no data response but threaten
    On commit, T+M => M
    On abort, T+M => T+I => I

- TI caches threatened TLoads
  - cache remotely TStored block
  - On commit/abort, T+I => I

+ cached locations are accessed directly
+ bounded txs perform in-place update
Overflow Table

Challenge: Where to put evicted TMI lines?
Solution: Per-thread hash table (in virtual memory)

Hardware controller
- fill table with TMI lines evicted from cache
- removes table entries when reloaded into cache
- performs look-aside transparently on L1 miss in parallel with L2
Outline

Preview

Data Isolation (aka. Lazy Versioning)

Conflict Management (flexible)
  - Access summary signatures
  - Conflict table
  - Alert-On-Update

FlexTM Software

Evaluation

Summary
Access Summary (1/2): Signatures

- **Signatures** [Bulk ISCA’06, LogTM-SE HPCA’07, SigTM ISCA’07]
  - Bloom filters to represent unbounded set of cache blocks
  - approx. representation with false positives

  **Cache block Addr.**

  ![Cache block Addr Diagram]

- Processor has two signatures:
  - $R_{\text{sig}} (W_{\text{sig}})$ summarizes locations TLoad (TStore)

**Conflict Detection:** Signatures snoop coherence messages
- responder detects conflict and overloads response
- requester picks response and resolves or notes conflict
Access Summary (2/2): Virtualization
[details in paper]

- Required to handle long running txs & tx pauses

**Challenge**: How to detect conflicts with suspended txs?

**Solution**: Read and Write summary signatures at the directory,
(note: does not affect cache hit critical path)

**Details:**
- merge suspended txns signature with summary sig.
- all L1 cache misses test signatures
- if miss, no further action necessary
- if hit, trap to software routine that mimics conflict HW
Conflict Tables: Tracking Conflicts

- Current HTMs detect and resolve at the same time
  - Eager HTM systems perform both on a conflict
  - Lazy HTM systems perform both at commit time

- Our approach: decouple detection from resolution
  - HW bitmaps record conflict event & expose to SW
  - SW decides when and how to resolve conflicts

- Per-core conflict bitmap
  - Core-P’s table
  - P’s read--remote write
  - P’s write--remote write
  - P’s write--remote read

Is there a conflict between P and core i? Ans: Yes (1) / No (0)
Conflict Tables: Operation

4 core machine

- Either processor can resolve conflict prior to commit
  - If eager, requester resolves conflict immediately
- Conflicter known, no central arbiter required

L2 Directory

A : M@C1
Conflict Tables: Operation

4 core machine

- TStore A
  - C0
    - $W_{\text{sig}}: \{\}$
    - $R_{\text{sig}}: \{\}$
    - W-W
  - C1
    - $W_{\text{sig}}: \{A\}$
    - $R_{\text{sig}}: \{\}$
    - W-W

- L2 Directory
  - A : M@C1

- Either processor can resolve conflict prior to commit
  - If eager, requester resolves conflict immediately
- Conflicter known, **no central arbiter** required
Either processor can resolve conflict prior to commit
  - If eager, requester resolves conflict immediately

Conflicter known, no central arbiter required
Alert-On-Update (AOU) [ISCA’07]

- Vector specific coherence or update events to the processor in the form of a lightweight event/interrupt
  - on invalidation (capacity eviction or coherence)
  - on access/update (local event)
Outline

- Preview
- Data Isolation (aka. Lazy Versioning)
- Conflict Management (flexible)
- FlexTM Software
  - FlexTM Transaction
  - Example
- Evaluation
- Summary
FlexTM Transaction (1/2)

- **Per-Tx descriptor**
  
  | TSW       | active / committed / aborted |
  | State     | running / suspended          |
  | CM<sub>PC</sub> | Abort<sub>PC</sub>          |

  handler for conflict table events | AOU events on TSW

- **FlexTM deploys**
  - **Signatures** for detecting and notifying conflicts
  - **Conflict Tables** for tracking and managing conflicts
  - **T-MESI** for in-cache buffering and OT for cache overflows
  - **AOU** for propagating abort events to remote txs.

- **FlexTM software**
  - checkpoints registers at Begin_Tx
  - manages conflicts; aborts remote tx by changing TSW
  - controls commit protocol routine
Lazy Transactions: Example

T1: Begin_Tx abort_pc1

T2: Begin_Tx abort_pc2

L2 Directory
Lazy Transactions: Example

**T1** Begin.Tx abort_pc1
ALD TSW0

**T2** Begin.Tx abort_pc2
ALD TSW1

L1: TSW0: AE

- L1
- C0: W
- TSW0: M@C0

L2 Directory

C1

- L1
- C1: W
- TSW1: M@C1
Lazy Transactions: Example

**T1**
- Begin_Tx
- abort_pc1
- ALD TSW0
- TSt A

**T2**
- Begin_Tx
- abort_pc2
- ALD TSW1

---

**L1**
- A: TMI
- TSW0: AE

**C0**
- \( W_{sig}:\{A\} \)  \( R_{sig}:\{\} \)
- W-W

**L1**
- TSW1: AE

**C1**
- \( W_{sig}:\{\} \)  \( R_{sig}:\{\} \)
- W-W

---

**L2 Directory**
- A : M@C0
- TSW0 : M@C0
- TSW1 : M@C1
Lazy Transactions: Example

**T1**
- Begin_Tx abort_pc1
- ALD TSW0
- TSt A
- TSt B

**T2**
- Begin_Tx abort_pc2
- ALD TSW1

---

<table>
<thead>
<tr>
<th>L1</th>
<th>C0</th>
<th>L1</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: TMI</td>
<td>W&lt;sub&gt;sig&lt;/sub&gt;:{A,B} R&lt;sub&gt;sig&lt;/sub&gt;:{}</td>
<td>TSW1: AE</td>
<td>W&lt;sub&gt;sig&lt;/sub&gt;:{} R&lt;sub&gt;sig&lt;/sub&gt;:{}</td>
</tr>
<tr>
<td>B: TMI</td>
<td>W-W</td>
<td></td>
<td>W-W</td>
</tr>
<tr>
<td>TSW0: AE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

A : M@C0
B : M@C0

L2 Directory

TSW0 : M@C0
TSW1 : M@C1
Lazy Transactions: Example

**T1**
- Begin.Tx
- abort_pc1
- ALD TSW0
- TSt A
- TSt B

**T2**
- Begin.Tx
- abort_pc2
- ALD TSW1
- TSt A

---

**L2 Directory**

- A: M@C0, C1
- B: M@C0

---

**C0**

- W sig: {A,B}
- R sig: {}

**C1**

- W sig: {A}
- R sig: {}

---

TSW0: M@C0

TSW1: M@C1
Lazy Transactions: Example

**T1**
- Begin.Tx
- abort_pc1
- ALD TSW0
- TSt A
- TSt B

**T2**
- Begin.Tx
- abort_pc2
- ALD TSW1
- TSt A
- TSt B

**L2 Directory**
- A : M@C0, C1
- B : M@C0, C1
- TSW0 : M@C0
- TSW1 : M@C1
Lazy Transactions: Example

**T1**
- Begin_Tx
- abort_pc1
- ALD TSW0
- TSt A
- TSt B

**T2**
- Begin_Tx
- abort_pc2
- ALD TSW1
- TSt A
- TSt B

Conflict & Commit protocol
- For-each i set in W-R or W-W
- CAS (Status[i], ACT, ABORT)

In software, decentralized, minimal overhead \( \propto \) No. of conflicting Txs
Lazy Transactions: Example

**T1**
- **Begin Tx abort_pc1**
- **ALD TSW0**
- **TSt A**
- **TSt B**

**Conflict & Commit protocol**
For-each i set in W-R or W-W
CAS (Status[i], ACT, ABORT)

**T2**
- **Begin Tx abort_pc2**
- **ALD TSW1**
- **TSt A**
- **TSt B**

**Conflict Handler!**

In software, decentralized, minimal overhead \(\propto\) No. of conflicting Txss

---

**L1**

<table>
<thead>
<tr>
<th>A: TMI</th>
<th>B: TMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSW0: AE</td>
<td>TSW1: M</td>
</tr>
</tbody>
</table>

**C0**

<table>
<thead>
<tr>
<th>W-R</th>
<th>W-W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**WSig:** \{A, B\}

**Rsig:** {}

**L2 Directory**

| A : M@C0, C1 |
| B : M@C0, C1 |

---

**L1**

<table>
<thead>
<tr>
<th>A: TMI</th>
<th>B: TMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSW0: AE</td>
<td>TSW1: M</td>
</tr>
</tbody>
</table>

**C1**

<table>
<thead>
<tr>
<th>W-R</th>
<th>W-W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**WSig:** \{A, B\}

**Rsig:** {}
Lazy Transactions: Example

T1: Begin_Tx abort_pc1
   ALD TSW0
   TSt A
   TSt B

Conflict & Commit protocol
For-each i set in W-R or W-W
   CAS (Status[i], ACT, ABORT)
   CAS-Commit Status[id]

T2: Begin_Tx abort_pc2
   ALD TSW1
   TSt A
   TSt B

Conflict Handler!

In software, decentralized, minimal overhead \( \propto \) No. of conflicting Txs
Outline

Preview

Data Isolation (aka. Lazy Versioning)

Conflict Management (flexible)

FlexTM Software

Evaluation
  - Speedup
  - Conflict resolution tradeoffs
  - Other results

Summary
Evaluation set-up

- Full system simulation, GEMS/SIMICS framework
  - 16 core CMP with shared L2
  - ORIGIN 2000 like coherence protocol
    (3 hop requests and silent evictions)

- Workloads
  - Data Structures: Hash, RBTREE, LFUCache, Graph
  - Applications: Scott’s Delaunay, STAMP*, STMBench7

- Runtime systems
  - CGL, FlexTM (HTM interface), RTM-F, RSTM, & TL2
  - Polka conflict manager

* - STAMP does not (yet?) interface with RTM-F and RSTM
FlexTM gains over RTM-F proportional to SW bookkeeping overheads
- software metadata management ~50% of tx latency

FlexTM gains over RSTM comparable to rigid policy HTMs
FlexTM is Fast (2/2)

- Kmeans-L and Genome performance gains lower
  - TL-2 per-access overheads low (i.e., high instructions / mem_access)

- Performance gains in Vacation higher
  - lower number of instructions per memory word accessed
Lazy mode aids progress

- Lazy provides more commits
- Exploits R-W sharing, allows reader & writers to commit in parallel

Eager causes cascaded stalls and aborts
- Lazy narrows conflict window
Mixed-mode can be better

STMBench7

- Long writer (~1ms) mixed with short readers (tens thousands cycles)
  - Pair-wise conflicts between writers, conflicts with multiple readers
- Eager doesn’t permit R-W sharing and reduces reader throughput
- Lazy permits W-W sharing, but wastes writer work on aborts

Best Policy: Eager-WW with Lazy-RW
Other Results

- **Area analysis** [in paper]
  - increase in core area small, OoO (0.6%), InO (3%)
  - minimal change to pipeline, most hardware on L1 miss

- **Comparison with Central-Arbiter HTM** [in paper]
  - broadcasts and central arbiters are an overkill
  - de-centralized SW commit is efficient & important

Non-Tx Applications

- **Watchpoints** [in TR-925]
  - Two memory monitoring primitives, AOU & Signatures
  - SW framework for detecting buffer overflows, memory leaks etc.
  - 15-50X speedup over binary instrumentation
Summary

- Decouple TM hardware components to
  - reduce HW complexity
  - enable deployment for varied purposes

- FlexTM
  - HW manages TM operations, SW manages policy
  - decentralized conflict and commit protocol in SW

- Conflict management
  - laziness is an important design requirement
  - provides best value when left under software control
Decouple TM hardware components to reduce HW complexity and enable deployment for varied purposes.

**FlexTM**
- HW manages TM operations, SW manages policy
- decentralized conflict and commit protocol in SW

**Conflict management**
- laziness is an important design requirement
- provides best value when left under software control

**Acknowledgments**
- Multifacet Research group, Wisconsin
- STAMP group, Stanford
- Transaction Benchmark group, EPFL
- Shan Lu, Opera group, Illinois

Questions?

[http://www.cs.rochester.edu/research/cosyn](http://www.cs.rochester.edu/research/cosyn)
[http://www.cs.rochester.edu/research/synchronization](http://www.cs.rochester.edu/research/synchronization)
FlexTM per-Core Hardware

Processor Context

Registers

Control Regs.

Read Signature

Write Signature

Handler PC
Flag register

Read & Write Access Summary

L1 Data Cache

Tag
Data
FlexTM per-Core Hardware

Processor Context

Registers

Control Regs.

Read Signature

Write Signature

R-W

W-R

W-W

Conflict Tables

\( N_{\text{core}} \text{ bits} \)

Handler PC

Flag register

Read & Write

Access Summary

Conflict Table

Tag

Data

L1 Data Cache
FlexTM per-Core Hardware

Processor Context
- Registers
  - Control Regs.
  - Read Signature
  - Write Signature
- Conflict Tables
  - R-W
  - W-R
  - W-W
- \(N_{core}\) bits

Handler PC
Flag register

Read & Write
Access Summary
Conflict Table
Alert-On-Update

L1 Data Cache
- A
- Tag
- Data
FlexTM per-Core Hardware

Processor Context

Registers
- Control Regs.
- Read Signature
- Write Signature

Conflict Tables
- R-W
- W-R
- W-W

N_{core} bits

Handler PC
Flag register

Data
Isolation

ASI
Base Address
Overflow Count
Overflow Sig.
Hash Param.
C/A

L1 Data Cache

Tag
Data

Overflow Table Controller

Read & Write
Access Summary

Alert-On-Update

L1 miss

Handler PC
Flag register

Data
Isolation

ASI
Base Address
Overflow Count
Overflow Sig.
Hash Param.
C/A

L1 Data Cache

Tag
Data

Overflow Table Controller

Read & Write
Access Summary

Alert-On-Update

L1 miss
FlexTM per-Core Hardware

Processor Context

Registers

Control Regs.

Read Signature

Write Signature

R-W

W-R

Conflict Tables

Read & Write Access Summary

Alert-On-Update

Data Isolation

ASI

Overflow Sig.

Base Address

Hash Param.

Overflow Count

C/A

Overflow Table Controller

Handler PC

Flag register

1 miss

T A

L1 Data Cache

Tag

Data

N_{core} bits
## FlexTM Area Complexity

<table>
<thead>
<tr>
<th></th>
<th>Core2</th>
<th>Power6</th>
<th>Niagara2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Orig. Core Area</strong></td>
<td>32mm²</td>
<td>53mm²</td>
<td>12mm²</td>
</tr>
<tr>
<td><strong>L1 area</strong></td>
<td>1.8mm²</td>
<td>2.6mm²</td>
<td>0.4mm²</td>
</tr>
<tr>
<td><strong>Signatures (2Kbit)</strong></td>
<td>0.10%</td>
<td>0.12%</td>
<td>2.1%</td>
</tr>
<tr>
<td><strong>Overflow Control</strong></td>
<td>0.5%</td>
<td>0.45%</td>
<td>0.3%</td>
</tr>
<tr>
<td><strong>%L1D area inc.</strong></td>
<td>0.35%</td>
<td>0.3%</td>
<td>3.9%</td>
</tr>
<tr>
<td><strong>% core area inc.</strong></td>
<td>0.61%</td>
<td>0.58%</td>
<td>2.5%</td>
</tr>
</tbody>
</table>

- **Effect on the processor core minimal**
  - OoO cores (~0.6\%), In-Order (~4%)  
- **Negligible effect on L1 latency**
  - Small area effects, data array is the critical path  
- **Signature effects noticeable only on Niagara2**
  - 8-way SMT needs 16 2Kbit signatures (4KB state)
FlexWatcher: Memory Bug Detection

- FlexTM HW provides two HW primitives for watching memory
  - AOU precisely monitors cache block aligned regions but is limited by cache size
  - Signatures provided unlimited monitoring but are vulnerable to false positives.

- Extended the ISA to support them as first-class entities
  - insert, member, read-index, activate, clear etc

- Developed a software bug detection tool
  - add required addresses to signatures
  - HW checks local & remote accesses against the signatures.
  - triggers SW trampoline on signature hits
  - handler disambiguates, if false positive return to execution
FlexWatcher Evaluation

- BugBench from illinois, set of real-life programs with known bugs.
- Bugs detected
  - Buffer Overflow
    Solution: Pad all heap allocated buffers with 64 bytes, watch padded locations
  - Memory Leak
    Solution: Monitor all heap allocated objects and update the address’s timestamp on access.
  - Invariant Violation:
    Solution: ALoad cache line of interested variable X. On AOU handler trigger assert program specific invariants.
FlexWatcher Performance

- Compared against **Discover**, popular SPARC binary instrumentation tool from **OpenSPARC**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Bug</th>
<th>FlexWatcher</th>
<th>Discover</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>BO</td>
<td>1.5X</td>
<td>75X</td>
</tr>
<tr>
<td>GZIP</td>
<td>BO</td>
<td>1.15X</td>
<td>17X</td>
</tr>
<tr>
<td>GZIP(^2)</td>
<td>IV</td>
<td>1.05X</td>
<td>N/A</td>
</tr>
<tr>
<td>Man</td>
<td>BO</td>
<td>1.80X</td>
<td>65X</td>
</tr>
<tr>
<td>Squid</td>
<td>ML</td>
<td>2.50X</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Execution time normalized to sequential thread performance

FlexWatcher overheads were estimated on the simulator
Discover overheads were estimated on a Sun T1000 server