Hardware Support for Transactional Memory

Sandhya Dwarkadas

Department of Computer Science
University of Rochester

HW support required for TM

- Isolation
  - Write visibly in place, log updates for rollback
  - Write invisibly in place
- Conflict detection
  - At the time of a conflicting write
  - At commit time
- Abort/Rollback and Restart
- Commit mechanism
  - Simple atomic operation
  - Atomic operation along with transactional updates

Hardware Proposals

- Early work – Transactional memory [ISCA'93], Oklahoma update [IEEE PDT'93]
- Speculative lock elision [MICRO'01]
- Transactional memory coherence and consistency [ISCA'04]
- Virtual Transactional Memory (VTM) [ISCA'05]
- LogTM [HPCA'06]
- Others that I will not cover …

Speculative Lock Elision [MICRO'01]

- Micro-architectural technique that dynamically removes lock-induced serialization
- Enables highly concurrent multi-threaded execution
- Hardware dynamically identifies synchronization operations, predicts and elides them
- Conventional coherence messages used to detect atomicity violations and recover

Examples of Potential Parallelism Masked By Dynamically Unnecessary Synchronization

a)

```
1. LOCK(locks->error_lock)
2. if (local_error > multi->err_multi)
3.       multi->err_multi = local_err;
4. UNLOCK(locks->error_lock)
```

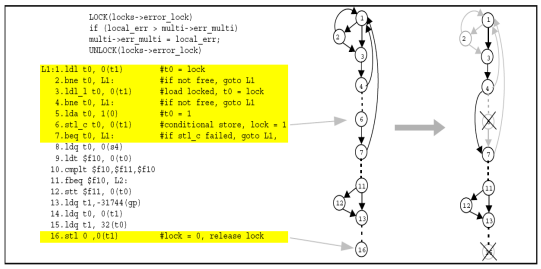
b)

Thread 1	Thread 2
LOCK(hash_tbl.lock)	
var = hash_tbl.lookup(X)	
if (!var)	
hash_tbl.add(X);	
UNLOCK(hash_tbl.lock)	
	LOCK(hash_tbl.lock)
	var = hash_tbl.lookup(Y)
	if (!var)
	hash_tbl->add(Y);
	UNLOCK(hash_tbl.lock)

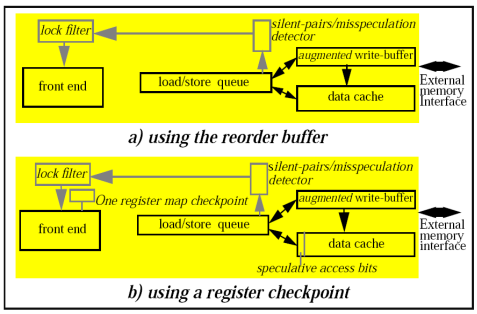
Enabling Concurrent Critical Sections

- Guarantee atomicity for all memory operations within a critical section
  - Data read within a critical section should be guarded from other writes
  - Data written within a critical section should not propagate till release

SLE by Identifying and Eliding Silent-Store Pairs



SLE Hardware



SLE Hardware support

- Initiating speculation
- Speculative register state
  - ROB – reads not allowed to retire
  - Register checkpointing – reads retire from ROB
- Speculative memory state
  - Caches track speculative reads
  - Speculative writes placed in write buffer
- Support for Abort/Restart
  - Cache coherence used to detect memory violations
- Instantaneous commit of data in write buffer on completion – lazily drained into cache
- Conflict resolution: newest write aborts old reads/writes

Hardware Proposals

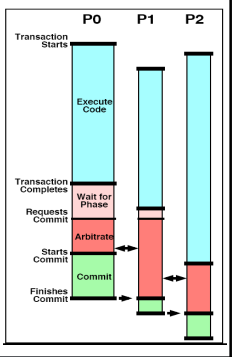
- Speculative lock elision [MICRO'01]
- Transactional memory coherence and consistency [ISCA'04]
- Virtual Transactional Memory (VTM) [ISCA'05]
- LogTM [HPCA'06]

Transactional Memory Coherence and Consistency [ISCA'04]

- Transactions are a first class entity
- TCC does not impose ordering amongst individual memory loads/stores.
- TCC imposes sequential ordering amongst transaction commits.
- Stores are buffered and kept within a processor node for the duration of a transaction.
- RAW dependencies cause abort and restart
- Anti-dependencies are handled by serializing commits

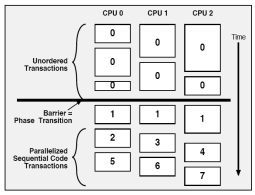
TCC Transactions

- Transactions execute in a phase cycle:
  - Execute->Wait->Request->Commit
- Speculatively execute code and buffer all reads/writes.
- Wait for commit permission Arbitrate with other CPUs
- Commit stores together

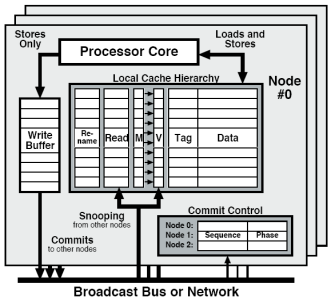


All Transactions All the Time

- Programmer orders transactions as necessary
  - Ordering implemented using phase and sequence numbers
  - Transactions allowed to complete only if in "oldest" phase



TCC Hardware



Read bits – Indicate speculatively read lines
Modified bits – Indicate speculatively modified lines,

Hardware Proposals

- Speculative lock elision [MICRO'01]
- Transactional memory coherence and consistency [ISCA'04]
- Virtual Transactional Memory (VTM) [ISCA'05]
- LogTM [HPCA'06]

Virtualized Transactional Memory [ISCA'05]

- Cache coherence for in-cache conflicts
- Additional data structures to track overflowed state
  - Transaction status word (XSW)
  - Global transaction address data table (XADT) for overflowed data
  - XADT filter to detect the absence of conflicts

VTM Highlights

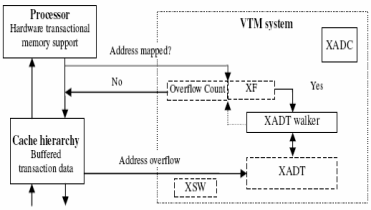


Figure 2 VTM overview. Each processor has its own VTM system machinery. The software-resident XADT and XF data structures are shown with dashed boxes, and the hardware structures are shown with solid boxes. The VTM machinery operates on the XF and XADT using cacheable operations. The XADC caches remapping translation information for overflowed blocks.

Hardware Proposals

- Speculative lock elision [MICRO'01]
- Transactional memory coherence and consistency [ISCA'04]
- Virtual Transactional Memory (VTM) [ISCA'05]
- LogTM [HPCA'06]

LogTM: Log-based Transactional Memory [HPCA'06]

- Isolation
  - Old values written to log in thread-private virtual memory
  - New values "in place" to make common commits fast
  - Also allows cache overflow & software abort handling

LogTM's Eager Conflict Detection

- LogTM (and most other hardware TMs)
  - Add per-processor transactional write (W) & read (R) bits
  - Setting W bit requires M state; setting R requires S or M
  - Lets coherence protocol detect transactional data conflicts
    - E.g., Writer seeks M copy, seeks S copies, & finds R bit set
- LogTM uses directory protocol

LogTM conflict detection

- Cache Overflows
  - Replacement sets directory in Sticky-M / Sticky-S
  - Commit fast due to leaving Sticky-M / Sticky-S unchanged
  - Check/clean-up on (potential) conflict
  - Chance of false positive
- In all cases
  - Requesting processor informed of conflict so that it can resolve it

Conflict Resolution

- Conflict Resolution
  - Can wait risking deadlock
  - Can abort risking livelock
  - Wait/abort transaction at requesting or responding proc?
- LogTM resolves conflicts at requesting processor
  - Requesting processor waits (using coherence nacks/retries)
  - Aborts if other processor is waiting (deadlock possible) & it is logically younger (using timestamps)

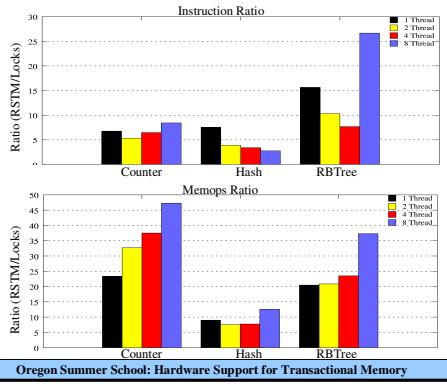
Challenges

- Support for I/O
- Exception thrown within transactions
- OS/Library calls invoked within transactions
- Transactions within the OS
- Nested Transactions
- Weak Vs Strong atomicity

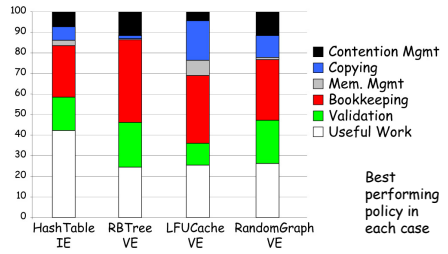
Hardware and Software TM

- Software
  - High runtime overhead
  - \+ Policy flexibility
    - conflict detection
    - contention management
    - non-transactional accesses
- Hardware
  - \+ Speed
  - Premature embedding of policy in silicon

RSTM Overhead [TRANSACT'06]



Single Thread STM Cost Breakdown



- All components other than memory mgmt significant
- (Copying overhead can be made arbitrarily large)

STM Performance Issues

- Memory management overhead
  - Garbage collection, object cloning/buffering of writes
  - Multiple pointer chasing required to access object data
- Validation overhead
  - Visible readers require 2N CASs to read N objects
  - Invisible readers need to perform bookkeeping and validation – $O(N^2)$ operation with N objects

RTM: HW-Assisted STM [TRANSACT'06]

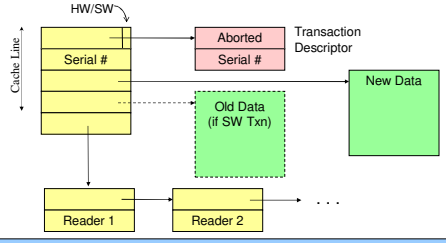
- Leave (almost) all policy in SW
  - don't constrain conflict detection, contention mgmt, non-transactional accesses, irreversible ops
- HW for in-place mutation
  - eliminate copying, memory mgmt
- HW for fast invalidation
  - eliminate validation overhead

Outline

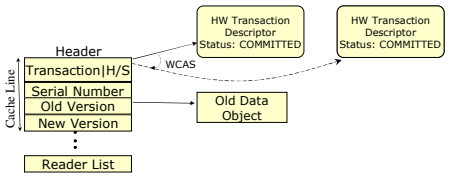
- RTM API and Software Metadata
- Support for isolation
  - TMESI coherence protocol with concurrent readers and writers
- Alert-on-invalidate
- Policy flexibility
- Preliminary results

RTM API and Object Metadata

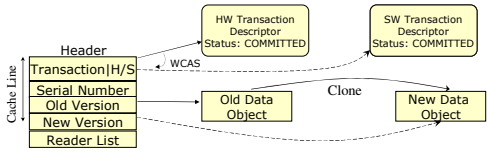
Threads define a set of objects as shared with associated metadata headers
Transactions involve
(1) Indicating start of transaction and registering abort-handler PC
(2) Opening object metadata before reading/writing object data
(3) Acquiring ownership of all objects that are written
(4) Switching status atomically to committed, if still active.



HW Ownership Acquisition



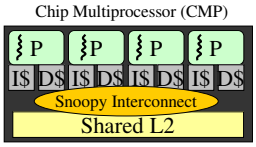
SW Ownership Acquisition



RTM Highlights

- Leave policy decisions in software
  - A multiple writer hardware coherence protocol (TMESI) to achieve isolation, along with lightweight commit and abort
  - Hardware conflict detection support and contention management controlled by software
- Eliminate the copying overhead
  - Employ caches as thread local buffers
- Minimize the validation overhead
  - Provide synchronous remote thread aborts
- Fall back to SW on context switch or overflow of cache resources

Prototype RTM TMESI



- Prototype system
  - CMP, 1 Thread/core
  - Private L1 caches
  - Shared L2 cache
  - Totally-Ordered network
- Additions to the base MESI coherence protocol
  - Transactional and abort-on-invalidate states

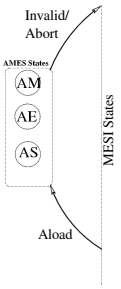
Outline

- RTM API and Software Metadata
- Support for isolation
  - TMESI coherence protocol with concurrent readers and writers
- Alert-on-invalidate
- Policy flexibility
- Preliminary results

Transactional (Speculative) Lines

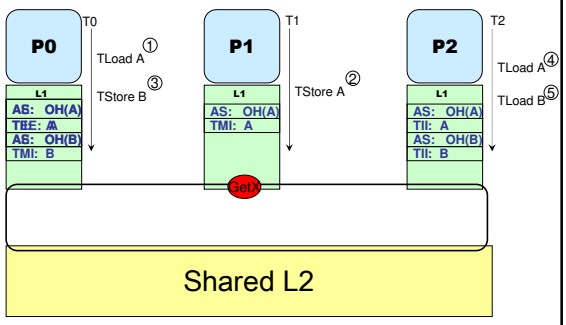
- TII lines isolate concurrent readers from transactional writers
  - can be dropped without aborting
  - allow caching, revert to I on exiting a transaction
- TStored (TMI) lines buffer transactional stores
  - must remain in cache (or txn falls back to SW)
  - revert to M on commit, I on abort
- Support R-W and W-W speculation (if SW wants)
- No extra transactional traffic; no global consensus in HW
  - commit is entirely local; SW responsible for correctness

Alert on Invalidate

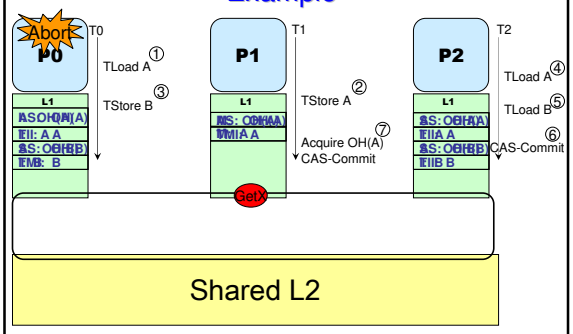


- A-tagged line invalidation aborts a transaction and jumps to a software handler
- Invalidation can be due to
  - Capacity: Abort since cache cannot track conflicts for object
  - Coherence: Remote potential writer/reader of the object cache line has acquired object ownership
- Transactional object headers when ALoaded in open() eliminate the need for
  - incremental validation
  - explicitly visible hardware readers
- Transaction descriptors are ALoaded by all transactions, allowing synchronous alerts

Example



Example



ISA Additions

- TStore — transactional (speculative) store
- ALoad, ARelease — alert if line is invalidated
- SetHandler — where to jump on abort
- CAS-Commit — if successful, revert T&A lines
- Abort — self-induced
- 2-C-4-S — if compare succeeds, swap 4 words (example, IA-64)

RTM Policy Flexibility

- Conflict detection
  - Eager (i.e., at open())
  - Lazy (i.e., at commit)
  - Mixed (i.e., eager write-write detection and lazy read-write detection)
- Flexible software contention managers
  - Contention managers arbitrate among conflicting transactions to decide the winner

Deferred Aborts

- Alert-on-invalidate allows synchronous aborts
  - Need to delay aborts inside RTM system code
  - Other potential uses – logging, profiling, debugging
  - Handler returns immediately and is reinvoked at the end of the NO_ABORT block

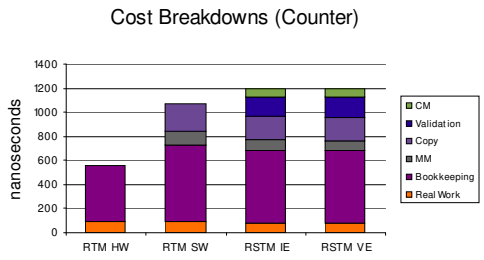
Transactions Unbounded in Space and Time

- Unbounded in space
  - Revert to software mode due to capacity or conflict issues
- Unbounded in time
  - Make scheduler aware of each thread's transaction descriptor that indicates
    - Transaction status
    - Transaction mode – hardware or software
    - RTM software status (details omitted)
  - Scheduler executes an abort instruction on every context switch to clear cache and modifies state for hardware transaction so it wakes up in handler

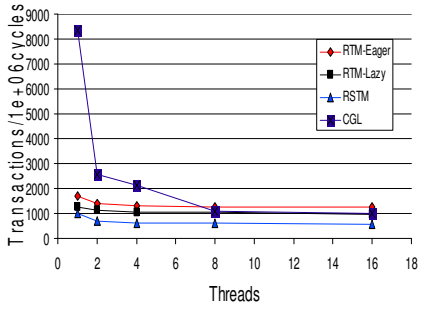
Simulation Framework

- Target Machine:16 way CMP running Solaris 9
  - 16 SPARC V9 processors
  - 1.2GHz in-order processors with ideal IPC=1
  - 64KB 4-way split L1 cache, latency=1 cycle
  - 8MB 12way L2 with 16 banks, latency=20cycle
  - 4-ary hierarchical tree
    - Broadcast address network and point-point data network
    - On-Chip link-latency=1cycle
  - 4GB main memory , 80 cycle access latency
  - Snoopy broadcast protocol
- Infrastructure
  - Virtutech Simics for full-system function
  - Multifacet GEMS [Martin et. al, CAN 2005] Ruby framework for memory system timing
  - Processor ISA extensions implemented using SIMICS magic no-ops
  - Coherence protocol developed using SLICC [Sorin et. al, TPDS 2002]

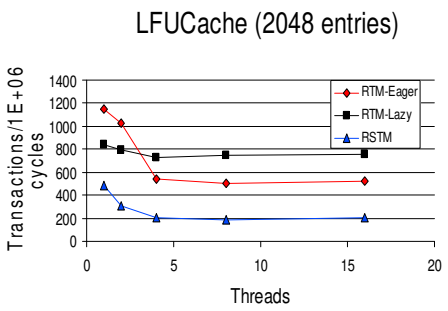
Cost Breakdowns



Shared Counter



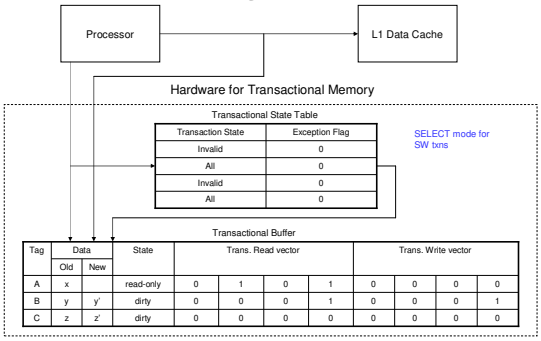
Lazy Versus Eager



Other Hybrid Proposals

- Sun's HyTM [Moir'05]
- Intel's Hybrid TM [PPoPP'06]

Intel's Hybrid TM



Conclusions

- Coherence protocol additions at the L1 cache allow
  - Transactional overhead reductions in copying and conflict detection in order to enforce isolation
  - Flexible policy decisions implemented in software that improve the scalability of the system
- Allowing software fallback permits transactions unbounded in space and time
- Additional features
  - Deferred aborts

Future Work

- A more thorough evaluation of the proposed architecture including
  - Effects of policy flexibility
- Extensions to multiple levels of sharing and to directory-based coherence protocols
- Incremental fallback to software for only those cache lines that don't fit in the cache