An Effective Hybrid Transactional Memory System with Strong Isolation Guarantees

Minh, Trautmann, Chung, McDonald, Bronson, Casper, Kozyrakis, Olukotun

Presented by Cynthia Sturton

5/5/08

Outline

- Software Transactional Memory
- Hardware Transactional Memory
- SigTM

Software Transactional Memory

- Lazy versioning
  - Global version clock
  - Write set buffer
- Lazy conflict detection
  - Lock associated with every word in memory
  - Bloom filter to maintain write set

Software Transactional Memory

High-level → Compiler → Low-level

```
ListNode n;
atomic {
  n = head;
  if (n != null) {

    head = head.next;

  }
}
```

```
ListNode n;
STMstart();
  n = STMread(&head);
  if (n != null) {
    ListNode t;
    t = STMread(&head.next);
    STMwrite(&head, t);
  }
STMcommit();
```

Software Transactional Memory - Start

- Checkpoint current execution environment
- Read global version clock value into RV

Software Transactional Memory – Read

- Check if in write set
- Check for conflicts with committed or committing transactions
  - Abort!
- Insert address into read set (FIFO)
- Load word from memory, return value to user

Software Transactional Memory - Write

- Check for conflict from committed or committing transactions
  - Abort!
- Insert address in Bloom filter for write set
- Insert address and data in write set

Software Transactional Memory - Commit

- Acquire locks for write set
- Atomically increment global clock
- Validate items in read set

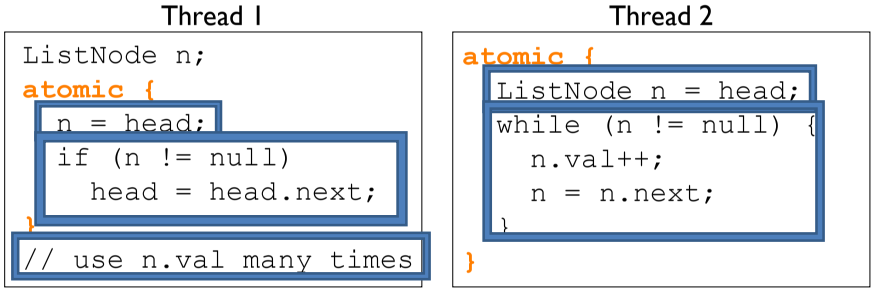
** Transaction Validated **

- Copy write set values to memory
- Release locks on write sets

Correctness in STM

- Strong Isolation
  - Data races
  - Privatization code
- Read sets not validated until commit

Strong Isolation



- Thread 1 can read partially committed transaction state of Thread 2

Hardware Transactional Memory

- Lazy versioning
  - Write set buffered in cache
  - W and R bits added to cache line hardware
- Eager conflict detection (reads & writes)
  - Cache coherency messages

Hardware Transactional Memory - Start

- Register checkpoint done by hardware

Hardware Transactional Memory - Read

- Cache hit:
  - Set R bit if W bit isn't already set
- Cache miss:
  - Request line in shared state
  - Set R bit

Hardware Transactional Memory - Write

- Cache miss:
  - Request line in shared state
- Cache hit:
  - If data is modified write back to underlying memory
- Write to cache and set W bit

Hardware Transactional Memory - Commit

- Acquire commit lock
- Acquire exclusive state on all lines in write set

** Transaction Validated **

- Reset W and R bits
- Release commit lock
- Modified data in cache can be read by others

Hardware Transactional Memory – Conflict Detection

- Conflict:
  - Process receives exclusive request for data in read set
  - Process receives any request for data in write set
- Generated by committing or non-transactional process
- Software abort handler invoked
  - Invalidate all cache lines in R and W set
  - Restore register checkpoint
- Forward progress – validated transaction cannot abort
- No starvation – starving transactions acquire commit lock at outset

SigTM

- Hardware – Software transactional memory hybrid
- Eager conflict detection (on read set)
  - Hardware signature (Bloom filter)
- Lazy versioning
  - Write set buffer in SW
- Strong isolation guarantees

SigTM - Start

- Take a checkpoint
- Enable read set signature lookups for exclusive coherence requests

SigTM - Read

- Check if address is in write set
- Insert address into read set signature
- Read word from memory

SigTM - Write

- Add address to write signature
- Update address and value in software write set

SigTM - Commit

- Enable coherence lookups in write set for all requests
- Acquire exclusive access for every address in write set
- Enable NACKs for requests in write set

** Transaction validated **

- Reset read set signature
- Store values from write set to memory
- Reset write set signature
- Disable NACKing

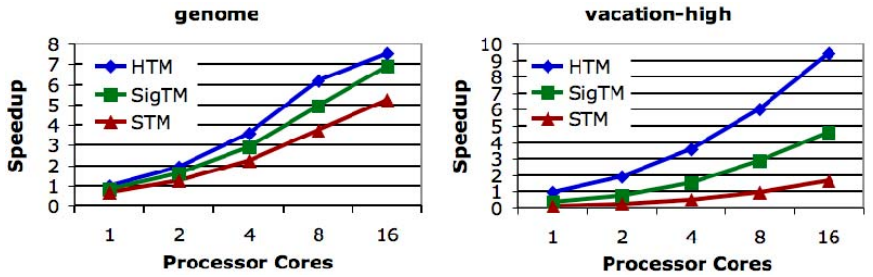
SigTM vs. STM

- Read barriers accelerated with read set signature
- No locking or timestamps
- Commit accelerated
  - Two traversals of write set
  - No read set validation
- Early conflict detection
  - False positives with read or write signatures?

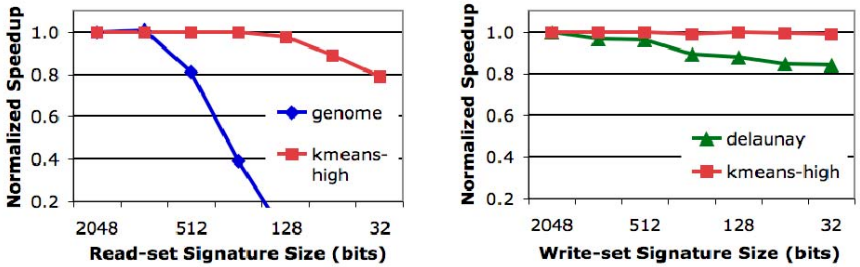
SigTM vs. HTM

- No hardware cache modification
- Flexible
- Nested transactions

Performance Evaluation



Accuracy of Read and Write Signatures



SigTM

STM vs. HTM

STM

- Maintenance and validation of read set.
- During commit – one read barrier and timestamp validation per word in read set.
- 3 traversals of write set in Validate and commit:
  - Acquire locks
  - Write to memory
  - Release locks
- Lazy conflict detection (at end of execution when validating read set) – wasted work on aborted transactions

HTM

- No additional instructions to maintain read/write set
- Read set validation occurs continuously
- One traversal of write set on commit
- Virtualization on cache overflow/associativity conflict → STM-like performance in that case
- False conflicts due to cache-line level granularity
- Strong isolation

Transactional Memory

- "Provide good performance with simple parallel code that frequently uses coarse-grain synchronization"
- Version management for transaction data
- Conflict detection as transactions execute concurrently
- SigTM:
  - Lazy versioning
  - Eager conflict detection (on reads)