Carnegie Mellon

MVCS-TM: Multiversion Conflict Serializable Hardware Transactional Memory

Ziqi Wang, Hao Wei
Computer Science Department
Carnegie Mellon University

Overview

Problem:

- Commercial implementations of Hardware Transactional Memory are overly restrictive. Only small transactions can be supported
- Conflict resolution mechanism using cache coherence is sufficient, but only low degrees of parallelism is achieved

Solution:

- HTM systems must overflow transactional states into DRAM to support large transactions
- Instead of relying on 2PL-style cache coherence for conflict detection, we leverage Optimistic Concurrency Control for better parallelism

Motivation

Stanford SI-TM (Snapshot-Isolation TM):

- Physical address space is multiversioned
- A special hardware device, the Multiversion Manager (MVM), is inserted between the L2 and shared LLC
- Transactional instructions access memory with a timestamp. Given a physical address and a timestamp, the MVM translates them into the physical address to the versioned storage
- At most four versions are supported

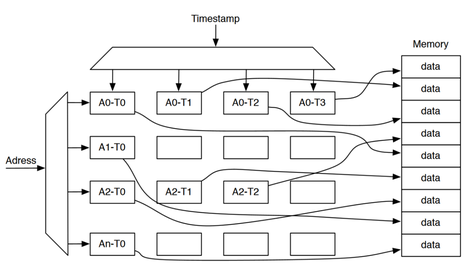

Figure 1. The Multiversion Manager (MVM)

SI-TM transaction commit and begin protocol:

- A global shared timestamp counter provides monotonically increasing source of timestamps
- On transaction begin, the counter is read as the begin timestamp of the transaction
- On transactional load, the processor uses the begin timestamp as the version to access MVM
- On transactional store, the speculative value is kept in the local cache
- On eviction of speculative cache lines, the processor writes them into MVM using a special, invisible version
- On transaction commit, the processor obtains a commit timestamp by atomically fetch-and-add the timestamp counter. It then checks whether cache lines in its write set have been overwritten by other transaction commits during its begin and commit. If not, commit succeeds, and all dirty lines are forced back to MVM.

Advantage:

- Support unbounded transactions; Fewer aborts

Disadvantage

- No support for serializability; Global lock on commit

Method

We extend SI-TM in the following aspects:

- Instead of a single shared timestamp counter, we propose a global transaction queue that holds the status and write set of all committed and committing transactions in the memory controller.
- Using bloom filters to approximate transaction's read and write sets. Membership testing and intersection is merely bitwise AND test
- Support fully conflict serializable semantics in addition to snapshot isolation. Users make choices based on the workload

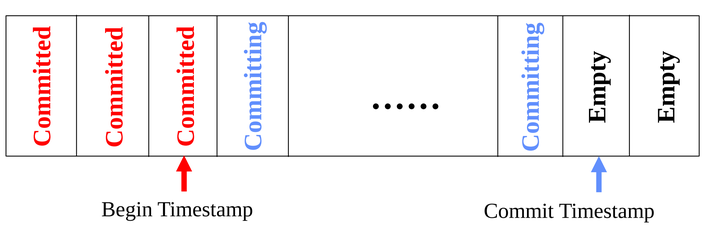

Figure 2. The Commit Queue and Timestamps

Protocol changes compared with SI-TM:

- On transaction begin, a begin timestamp is obtained as the version of the most recent committed transaction
- On transaction commit, a commit timestamp is obtained as the next empty slot's identifier
- Conflicts are detected by intersecting current transaction's read set with committed transactions' write sets between the begin and commit timestamp

Results

Implemented a prototype using zSim, and ran performance test using the STAMP benchmark

- 4, 8, 16 and 32 Threads

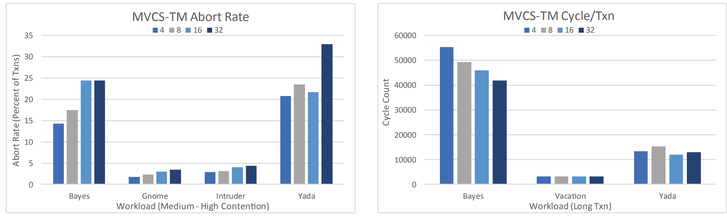

Figure 3. Abort Rate and Cycles per Transaction of MVCS-TM Using High Contention Workloads

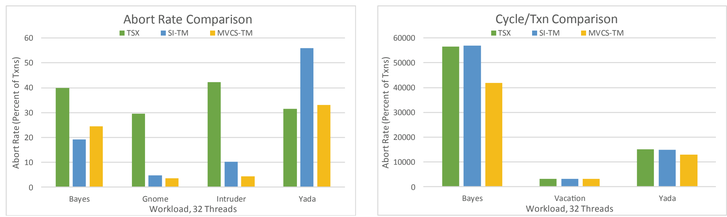

Figure 4. Comparison between Intel TSX, SI-TM and MVCS-TM on Abort Rate and Cycles per Transaction

Conclusion from the benchmark:

- Our implementation has little overhead compared with Intel TSX and SI-TM
- The abort rate decreases by using flexible ways of running transactions in either snapshot isolation mode or conflict serializable mode
- MVCS-TM scales with up to 32 worker threads

Future Work

- More detailed profiling of our simulations to determine typical transaction sizes, reasons for aborting, etc, to better understand the characteristic of MVCS-TM
- Support concurrent transaction commit as long as their write sets do not overlap
- Better support for nested transactions as well as interaction with non-transactional memory operations