Unbounded Transactional Memory

Paper by Ananian et al. of MIT CSAIL

Presented by Daniel

Outline

- Motivation
- UTM vs LTM
- UTM in detail
  - Processor changes
  - Transaction state data structure
  - Operational description
- LTM
  - changes required
  - description
- Simulation Results

Motivation

- Transactional memory is great, but currently saddled with hardware-imposed limitations.
- Transactional memory must allow arbitrary sized transactions to provide 'ease of programming'.
- Otherwise, TM can be as difficult to program with as locks, because programmers need to figure out how to break transactions up.

UTM vs LTM

- Unbounded Transactional Memory (UTM) is the first, more flexible but more complicated and hardware costly approach.
- Large Transactional Memory (LTM) is a less costly compromise, that still allows for transactions bigger than transactional cache, but no larger than physical memory.

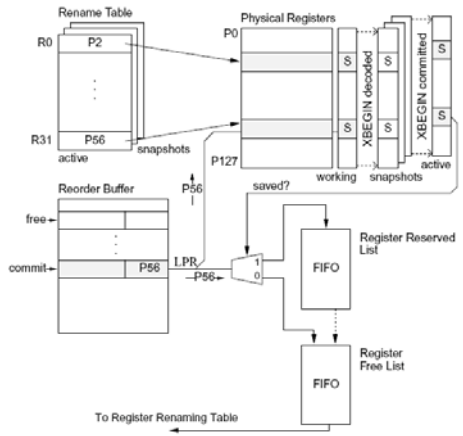
UTM processor changes

- Two new processor instructions: transaction being and transaction end
- XBEGIN pc Begins a transaction, incrementing the transaction counter and saving the abort handler located at 'pc.' Similar to a branch instruction.
- XEND Finishes the current transaction, atomically committing all data.

UTM processor changes (cont.)

- XBEGIN causes all current physical registers in use to be marked 'saved' and the register rename table is saved.
- Saved physical registers are moved to a register reserved list, not the free list, upon graduation.
- Because inner transactions are flattened, only one copy of architectural state is saved.

UTM processor changes (cont.)



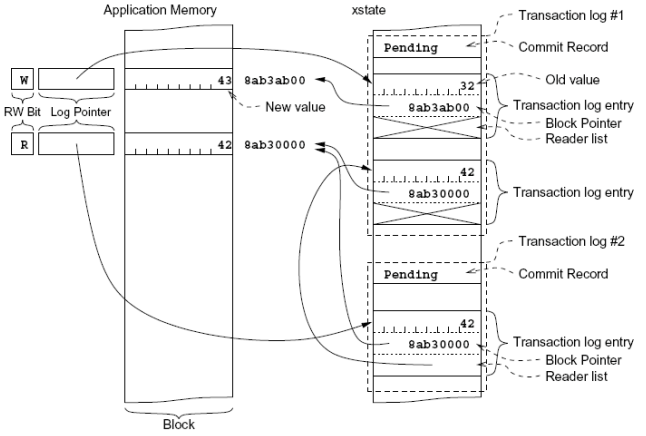
UTM transaction state data structure

- A single data structure, the xstate holds all transactional information, and is stored in main memory.
- xstate contains:
  - All transaction logs
  - For each block in memory (and each paged block), a log pointer and a read or write bit.
- Each active transaction gets a transaction log.
- Transaction log contains:
  - Pointer to commit record (pending, aborted, committed)
  - An array of log entries

UTM xstate (cont.)

- Each block of memory touched by a transaction gets a log entry.
- Log entries contain:
  - Pointer to the block of memory
  - The clean value
  - Pointer to the commit record
  - A linked list of all log entries in all transaction logs referring to this block

UTM's xstate



UTM description

- The status of each block of memory determined by following the log pointer, then following the commit record pointer.
- A commit consists of setting the commit record, then deleting all log pointers that are part of the transaction log.
- Because the speculative value is stored in memory, cleanup is only required for aborts, optimizing for the common case of success.

UTM description (cont.)

- When a transaction loads, it ensures that the block is not part of a transaction, or that the Read bit is set.
- When a transaction stores, it checks that this block belongs only to this transaction.
- In case of conflict, newer transactions are aborted.

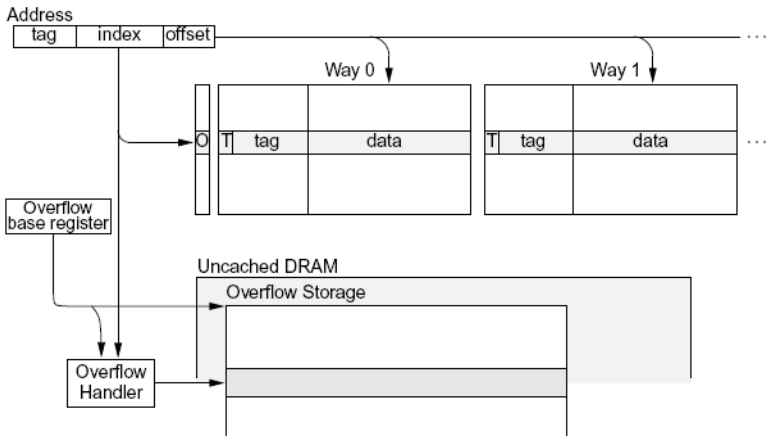
UTM description (cont.)

- UTM supports caching like in earlier HTM systems.
- Transaction state is only moved into xstate when there is overflow or a cache coherence conflict.
- UTM supports transactions as large as virtual memory, by paging the xstate out to disk and using global virtual addresses.

LTM

- Limited to transactions the size of physical memory.
- Transactions are aborted upon interrupts or thread migration.
- Only system changes required are to the cache and processor.

LTM modifications



LTM modifications (cont.)

•Each cache line now has a Transaction bit set when it is read or written during a transaction.

•If a cache line is evicted, an Overflow bit is set.

LTM description

- Once a transaction starts, all cache lines that are read or written cause the T bit to be set.
- If a cache line is accessed with the O bit, main memory is checked for that cache line.
- The cache coherency protocol detects conflicts as proceeds as per other HTM systems.
- To commit, clear all T bits and write overflowed data back.

Simulation Results

- LTM offers much better scaling than Conditional Load/Store locks. Overhead is less than 10%.
- Time dealing with overflow is insignificant.
- There are some applications that need huge transactional memory footprints.
- TM does increase concurrency by decreasing serial regions, including in the Linux kernel.