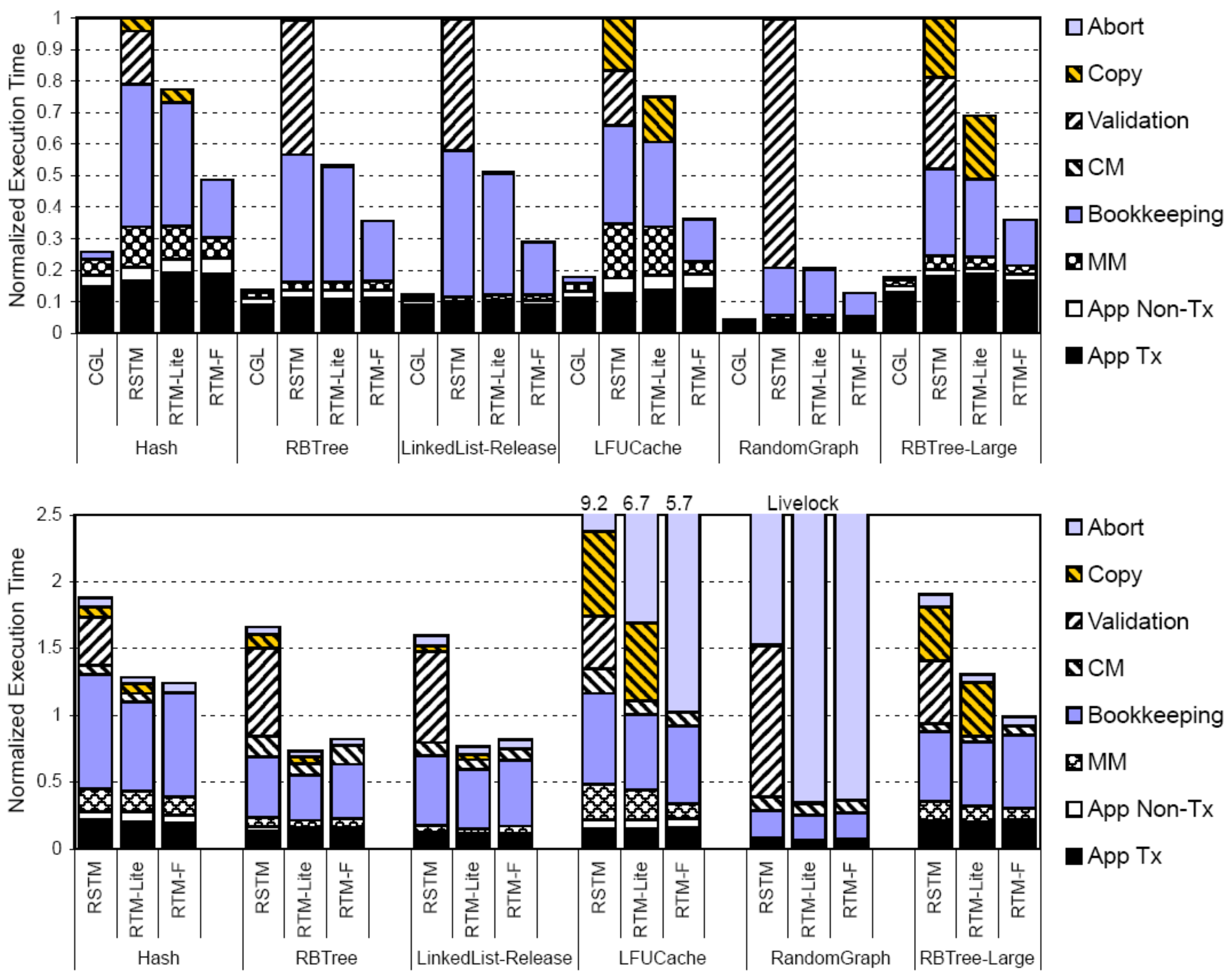
Lecture Notes: Wednesday, May 28, 2008

Hardware Assisted Software Transactional Memory

Rochester Transactional Memory

This lecture is based on: Shriraman, Spear, Hossain, Marathe, Dwarkadas & Scott: [An Integrated Hardware-Software Approach to Flexible Transactional Memory]()

Software Transactional Memory Overheads



Breakdown of per transaction latency for 1 thread (top) and 8 threads (bottom).

- Aborting (Abort): all costs incurred by an aborted transaction
- Copying Data (Copy): for logging and taking backups
- Validation: mainly validating the readset
- Contention Management (CM): what do to if there's a conflict
- Bookkeeping: maintaining the metadata and statistics
- Memory Management (MM): allocating and deallocating memory
- Useful Work (App): Non-Tx is the work done outside transactions, Tx is the work done in a transaction.

How can hardware/cache coherence help?

Eliminate the need for copying: utilize incoherence. Cache already buffers the data, which in essence is a copy of the original in main memory. This would effectively remove the copying overhead.

Detect conflicts between transactions: cache coherence already has the capability of detecting conflicts in cache lines, add a hook to it to utilize that information. This would effectively remove the validation overhead.

Show effects of tentatively written data at commit time: by allowing the transaction to only expose its modifications at commit time, some read-write sharing of the same cache line is possible. This would reduce some of the contention management and abort overhead.

Rochester's Solution

Alert-On-Update (AOU): can be used to selectively expose coherence events, and transfer control to an alert handler in case of a coherence violation. This is the scheme used to detect conflicts therefore eliminating the validation overhead.

AOU is also relatively simple to implement, as it is completely compatible with existing bus interfaces and protocols.

Programmable Data Isolation (PDI): can be used to selectively stop cache line modifications from being propagated until needed (i.e. committing a transaction). This is the scheme used to perform tentative writes therefore eliminating the need for explicit logging or taking backup copies. It also could allow read-write sharing if the reader(s) commit before the writer does.

PDI is relatively difficult to implement since it requires adding and modifying existing bus signals, as well as adding more cache line states to the cache coherence protocol.

Pros and Cons

Pros

Flexibility: instead of hardware forcing the transactional model and the use of its features limited to begin/commit transaction, the developer can choose how to utilize the hardware's features.

For example; contention management, eager/lazy updates are all controlled by the software as a matter of policy rather than being forced by the hardware making decisions that might not suit all cases.

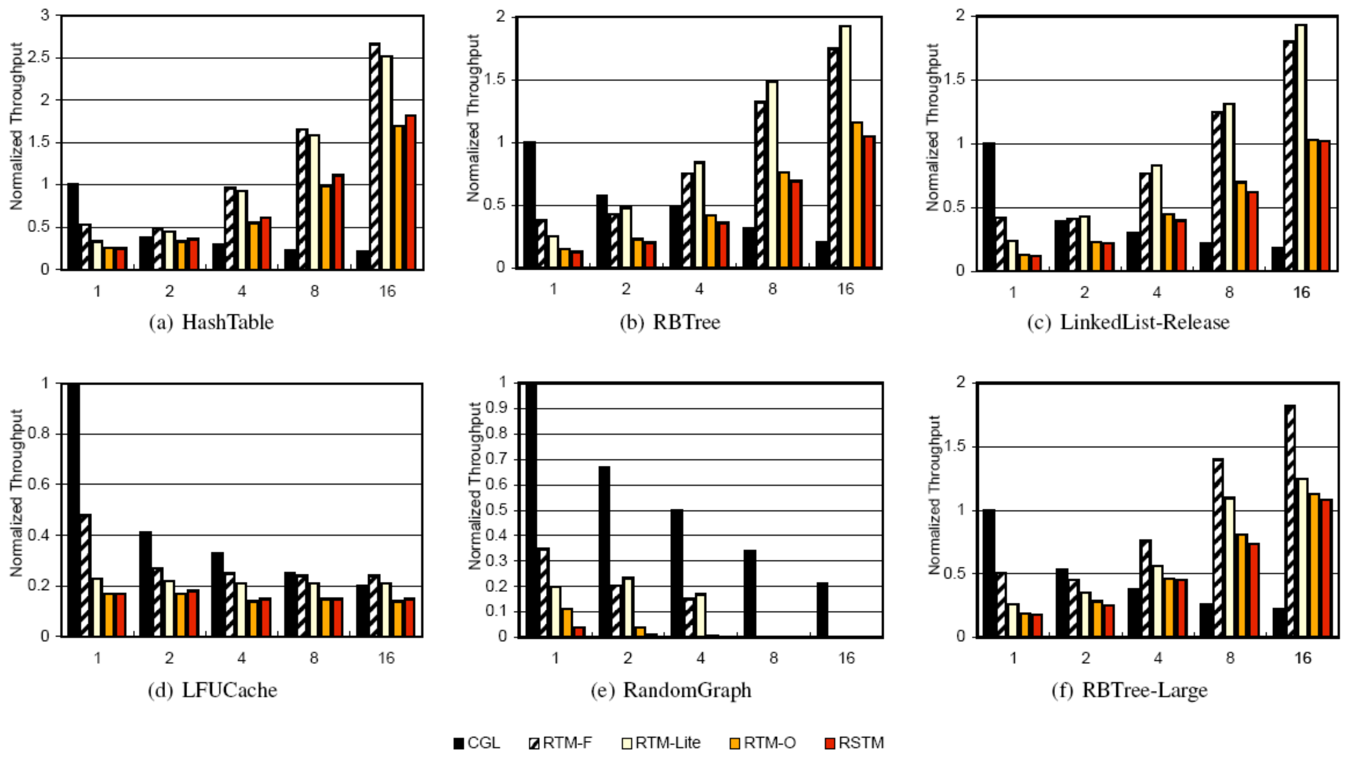
Moreover, by having direct access to these features, developers could utilize them in contexts other than Transactional Memory.

Improved performance: their results showed an average of 2x performance (throughput) improvement compared to the pure software RSTM.

Cons

Requires more hardware modifications than needed for best-effort hardware transactional memory.

In single threaded mode, RTM is still about twice as slow as coarse-grained locking (CGL). In the best case scenario, performance (throughput) caught up with CGL on a single processor at 8 processors, and in the worst case scenario even at 16 it couldn't catch up with CGL on a single processor (see figure below).



3: Throughput (transactions/10^6 cycles), normalized to 1-thread CGL. X-axis specifies the number of threads.

Question

Can you think of other uses for AOU and PDI in a scheme such as NZSTM?

References

Larus & Rajwar: Transactional Memory (Chapter 4.6)
Shriraman, Spear, Hossain, Marathe, Dwarkadas & Scott: An Integrated Hardware-Software Approach to Flexible Transactional Memory