Review of "Transactional Memory: Architectural Support for Lock-Free Data Structures"[1]

Riley Wood

April 4, 2016

Key Ideas

The protocol itself is relatively straightforward, and takes a page from the basic MESI cache coherence protocol. Essentially, all edits made to memory within a transaction are tentative, stored in temporary storage, and only committed if its read set (the data it read initially) has not been invalidated, i.e. written to by another transaction. The supporting architecture is also fairly simple to implement given that most existing cache coherence protocols are already adaptable to transactional memory. Herlihy and Moss make it a goal to avoid affecting the performance of non-transactional operations. To that end, they recommend each processor have two caches: a transactional cache and a regular cache. The fully-associative transactional cache supports the new commit and abort operations and is a temporary place to store uncommitted values.

Review

Doesn't the need for a commit introduce additional overhead? I mean the time it takes to copy the data from temp storage to main memory. This may be amortized by the reduction in memory accesses brought about by not having to access locks in memory. Adding a whole extra cache for each core is expensive, especially a fully-associative one. But it's made up for by the fact that the rest of the architecture support is already provided by cache coherence hardware. In the case where the transactional cache fills up, how can a larger one be emmulated in software? Where is the transactional metadata stored? If in main memory, I wish they described how. They also don't discuss how code that currently uses locks would port over. I suppose there is no reason why lock instructions would stop being supported, so maybe the two protocols could coexist.

Conclusions

Transactional memory seems very promising since it is an easy-to-understand, non-hardware-complex way of sharing memory in concurrent programs. There are some unanswered questions, such as how exactly a larger transactional cache can be emulated in software; certainly it will fill up, so how will transactional metadata be preserved further down the memory hierarchy? Their simulation results show marked improvement over existing solutions, which is promising. I would think the primary inhibitor to widespread adoption might be cost, since integrating a second, fully-associative–albeit small–cache can be expensive. There must be a reason why architects have only just now begun adopting this scheme. There is also the fact that, as I mentioned before, legacy lock code would need to be proven to work with transactional memory. Demonstrating that the cost of this new system is reasonable and that old code would be supported are both necessary steps to validating this new idea, in my opinion.

References

[1] M. Herlihy, J. Eliot, and B. Moss. Transactional memory: Architectural support for lock-free data structures. In *Computer Architecture, 1993., Proceedings of the 20th Annual International Symposium on*, pages 289–300, May 1993.