

TRAMP Position Statement and Prior Contributions

Colin Blundell Milo M. K. Martin

Department of Computer and Information Science
University of Pennsylvania
{*blundell,milom*}@cis.upenn.edu

Challenges and Future Directions

Workloads. In our view, developing transactional workloads is the most pressing challenge for the transactional memory research community. Such development will both give us insight into the practical usage of transactions and provide us with benchmarks to better quantitatively evaluate our proposals, allowing us to break free of microbenchmarks. One difficulty in such development is that for a suite of transactional memory benchmarks to become widely used, they must necessarily incorporate a lowest-common-denominator execution model for transactions that is widely accepted. Deciding on such a model is challenging and important in its own right.

Transactional Execution Models. A lowest-common-denominator transactional execution model should be simple enough to be suitable for programmers to informally reason about transactions, admit a wide range of implementations, and be complete enough to capture the formal semantics of transactions. Starting points for such models include “single-lock atomicity” [5] or global serialization (to capture strong atomicity). We conjecture that an important element of any such lowest-common-denominator execution model will be that *a serial implementation is a correct implementation*. This property is important for two reasons: (i) it separates correctness and performance, with concurrency squarely on the performance side, and (ii) it admits implementations ranging from a global lock on legacy hardware to a high-performance STM to a full-fledged HTM. It does, however, place a restriction on advanced features such as open nesting: these features can be used to enhance performance (for example, in building concurrent data structures), but the added concurrency that they allow cannot be necessary for the program to perform correctly. We believe that this restriction is natural: programs that rely on concurrently-executing transactions for correctness may break when run on systems that do coarse-grained conflict detection or serialize transactions to ensure forward progress in uncommon cases.

Prior Contributions

Our prior work has focused on both the semantics of memory transactions and on their implementation in hardware.

Semantics. In our work on transactional memory semantics, we’ve described how transparently transforming lock-based critical sections into transactions can prevent forward

progress [2, 3], introduced the terms strong and weak atomicity [2, 3], and discussed the forward progress issues when a programmer or runtime system combines transactions [3].

Hardware Implementation. Our work on hardware transactional memory implementation has strived to build a system that is practical to implement in hardware, but is fast in almost all cases [1]. We achieve this goal in two parts. First, we focus on making the fast case common by extending the reach of the now-standard bounded hardware implementations. A permissions-only cache, which efficiently tracks permissions—but not data—for blocks transactionally read and written, extends the size of bounded transactions. In the extreme, dynamically using the second-level cache data array to hold such information allows transactions to touch up to a gigabyte of data before overflowing. Second, on the rare cases that a transaction exceeds this bound, the transaction transitions to an unbounded execution mode that limits concurrency (only one unbounded transaction can be active at a time, although they may execute concurrently with other bounded transactions). This less-concurrent form of unbounded execution is simpler to implement, trading some concurrency for implementation simplicity. We’ve also explored supporting I/O and other system calls via partial serialization [4].

References

- [1] C. Blundell, J. Devietti, E. C. Lewis, and M. M. K. Martin. Making the Fast Case Common and the Uncommon Case Simple in Unbounded Transactional Memory. In *Proceedings of the 34rd Annual International Symposium on Computer Architecture*, June 2007.
- [2] C. Blundell, E. C. Lewis, and M. M. K. Martin. Deconstructing Transactional Semantics: The Subtleties of Atomicity. In *Proceedings of Workshop on Duplicating, Deconstructing, and Debunking*, June 2005.
- [3] C. Blundell, E. C. Lewis, and M. M. K. Martin. Subtleties of Transactional Memory Atomicity Semantics. *IEEE TCCA Computer Architecture Letters*, 5(2), Nov. 2006.
- [4] C. Blundell, E. C. Lewis, and M. M. K. Martin. Unrestricted Transactional Memory: Supporting I/O and System Calls within Transactions. Technical Report CIS-06-09, Department of Computer and Information Science, University of Pennsylvania, Apr. 2006.
- [5] J. R. Larus and R. Rajwar. *Transactional Memory*. Morgan and Claypool, 2007.

About the authors

Colin Blundell is a 4th-year PhD candidate in the Department of Computer and Information Science at the University of Pennsylvania. His research focuses on support up and down the stack for ameliorating the process of parallel programming. He received an ScB in math and computer science from Brown University in 2003.

Milo Martin is an assistant professor in the Department of Computer and Information Science at the University of Pennsylvania. His research interests include scalable microarchitectures, multiprocessor computer architectures, memory systems, and verification. Martin is a 2007 recipient of the NSF CAREER award. He received a PhD in computer science from the University of Wisconsin-Madison in 2003.