

Intel Position Paper

Ali-Reza Adl-Tabatabai and Rick Hudson

Session I: Programming Model and Language Design

The success of transactional memory is related to how quickly it can be placed in the hands of practitioners and how comfortable they are with it. Historically programmers have shown great resistance to moving to new languages or even to new software development environments. If we can successfully add transactional memory to current languages it will ease the acceptance of transactional memory and enable concurrent programming for the general practitioner.

Unfortunately there is a tension between providing transactions in present day languages with their legacy runtime and libraries. This tension is most acute when it comes to providing semantics that enforce isolation. Should the programmer be expected to identify all transactional regions where conflicts can occur as they do today with locks or should the programmer be allowed to identify a single block of code and expect that if wrapped with an atomic statement that the code would have strong isolation guarantees regardless of non-transactional access to common variable that might exist in other code. The answer to these questions will have an impact on the environment the user expects and the implementations that are practical.

To a lesser extent how we define transactions will always have an impact on implementations. For example definitions contain the concept of serializability which is by its very nature a global property requiring global synchronization. Is serializability really needed or will something less restrictive be just as good but allow for better scalability across what could be hundreds of cores on thousands or tens of thousands of machines. Can we relax the notion of serializability like we can with isolation and if so how? Another issue is whether we expand our definition of transactional memory to include transactional execution or simply the effects transactions have on memory. Put simply should operations such as a file write be considered as part of the transaction or should we train programmers to think of transactional memory as merely operation on memory.

Session II: Implementation Challenges

The current challenge is to design transactional memory systems with the right balance of hardware and software. On a related note while transactional memory promises to provide good scalability, single thread performance can not be ignored. A system that slows single threaded applications excessively will not be acceptable in the market place. It may well be that hardware will have a much more profound impact on single threaded performance than on scalability.

Session III: Systems Platform

Software development environments must leverage and innovate with TM, not just mimic historic approaches. The entire stack could use a rethinking once we have transactions, complicated non-blocking algorithms used inside the virtual machine, the OS, and our memory allocation routines might be able to be simplified without losing scalability. Debugging of concurrent programs could be rethought. If a transaction with a break point aborts, the debugger might preserve the read and write sets so that the programmer can deterministically step through the transactions, possibly multiple times, debugging the code. Read and write sets could also be preserved during tracing at a cost similar to tools that detect memory leaks and this information could be data mined to help the user reason about his program. It is time to start rethinking the entire software stack just as the hardware is being rethought support many core systems.

TM bibliography.

“Enforcing isolation and ordering in STM.” Tatiana Shpeisman, Vijay Menon, Ali-Reza Adl-Tabatabai, Steve Balensiefer, Dan Grossman, Richard Hudson, Kate Moore, Bratin Saha. ACM Conference on Programming Language Design and Implementation, San Diego, CA, June 2007.

“Open nesting in software transactional memory” *Yang Ni (Intel), Vijay Menon (Intel), Ali-Reza Adl-Tabatabai (Intel), Antony Hosking (Purdue University), Richard L. Hudson (Intel), Eliot Moss (University of Massachusetts at Amherst), Bratin Saha (Intel), and Tatiana Shpeisman (Intel) PPOPP 2007*

“Code generation and optimization for software transactional memory in an unmanaged environment” by Cheng Wang, Wei Yu, Youfeng Wu, Bratin Saha and Ali-Reza Adl-Tabatabai, CGO 2007

“Architectural support for software transactional memory” Bratin Saha, Ali-Reza Adl-Tabatabai, Quinn Jacobson December 2006 Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture MICRO '06

Q focus: computer architecture: “Unlocking concurrency” Ali-Reza Adl-Tabatabai, Christos Kozyrakis, Bratin Saha December 2006 Queue, Volume 4 Issue 10

“Compiler and runtime support for efficient software transactional memory” Ali-Reza Adl-Tabatabai, Brian T. Lewis, Vijay Menon, Brian R. Murphy, Bratin Saha, Tatiana Shpeisman June 2006 ACM SIGPLAN Notices , Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation PLDI '06, Volume 41 Issue 6

“McRT-Malloc: a scalable transactional memory allocator” Richard L. Hudson, Bratin Saha, Ali-Reza Adl-Tabatabai, Benjamin C. Hertzberg June 2006 Proceedings of the 2006 international symposium on Memory management ISMM '06 Publisher: ACM Press

“McRT-STM: a high performance software transactional memory system for a multi-core runtime” Bratin Saha, Ali-Reza Adl-Tabatabai, Richard L. Hudson, Chi Cao Minh, Benjamin Hertzberg March 2006 Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming PPOPP '06