

# An Integrated Hardware-Software Approach to Transactional Memory

Sean Lie, C. Scott Ananian

**Background:** Programming in a shared-memory environment often requires the use of atomic regions for program correctness. Traditionally, atomicity is achieved through critical sections protected by locks. Unfortunately, locks are very difficult to program with since they introduce problems such as deadlock and priority inversion. Locks also introduce a significant performance overhead since locking instructions are expensive and performing deadlock avoidance can be slow. In addition, locks are simply memory locations so there is an added space overhead associated with locking as well.

**Transactional Memory:** Transactional memory has been proposed as a general and flexible way to allow programs to read and modify disparate primary-memory locations atomically as a single operation, much as a database transaction can atomically modify many records on disk. Hardware transactional memory (HTM), such as the scheme proposed by Herlihy and Moss [2], supports atomicity through architectural means. On the other hand, software transactional memory (STM), such as the scheme proposed by Ananian and Rinard [1], supports atomicity through languages, compilers, and libraries. HTM can be implemented quite easily with some modifications to the processor cache and cache controllers however generally impose limitations on the size and length of transactions. STM does not suffer from these limitations but has a much higher overhead.

We propose an integrated hardware-software implementation of transactional memory called HSTM. HSTM can give us the best of both hardware and software transactions. It allows small and short transactions (the common case) to run in hardware with very low overhead. If the transaction does not fit in hardware, it is run using software. Therefore, large and long transactions run slower in software but are nevertheless possible and fully supported. Moreover, the high overhead can often be amortized over the common case and thus does not have a significant penalty on overall performance.

**Size and Length Limitations:** Transaction size is the number of memory locations that are accessed (both read and write) within a transaction. HTM schemes have a size limitation since they store a speculative version of all transactionally accessed memory in a cache or hardware buffer. The size of such hardware structures ultimately determines the largest transaction that be run using HTM.

Transaction length is the number of instructions in a transaction. HTM schemes do not have a hard limit on transaction length as they do on transaction size. However, very long transactions will generally fail since HTM does not support context switching. In a modern system, context switches are very common as they are used to accomplish tasks such as multi-threading, system calls, and TLB re-fills. Therefore, by not supporting context switching, HTM effectively sets a limit on the length of a transaction.

**HSTM - The Integrated Approach:** In HSTM transactions are always first run using in hardware using HTM. If the transaction aborts in HTM, it is retried in STM. Thus, if the transaction can successfully run in hardware, there is no additional overhead over HTM. If the transaction aborts in HTM, it must have not fit in hardware. In that case, the transaction will incur the overhead associated with STM and the original attempt in HTM. However, our preliminary study shows that the overhead is very similar to that of STM and much better than simply retrying in hardware.

We use a scheme very similar to Herlihy and Moss's HTM design for hardware transactions. All speculative transactional state is stored in the L2 cache of the processor. For the software transactions, we use the FLEX software transaction system developed by Ananian and Rinard for object-oriented languages such as Java. For HSTM to function correctly, hardware and software transactions must interact appropriately so that they both maintain atomicity. Thus, we modified both HTM and STM slightly so that conflicts between the two could be detected.

**Evaluation:** To evaluate the performance of HSTM, support for the modified HTM and STM schemes were implemented in the UVSIM software simulator. UVSIM simulates the MIPS R10000 in a multi-processor system similar to the SGI Origin 3000.

Our preliminary studies show that HSTM deals with the size limitation of HTM as expected. As shown in Figure 1, small transactions run exactly as they would in HTM. When transactions become too large to fit in hardware, they run with similar performance to STM. They are approximately 3 times slower than hardware transactions but they nevertheless still run.

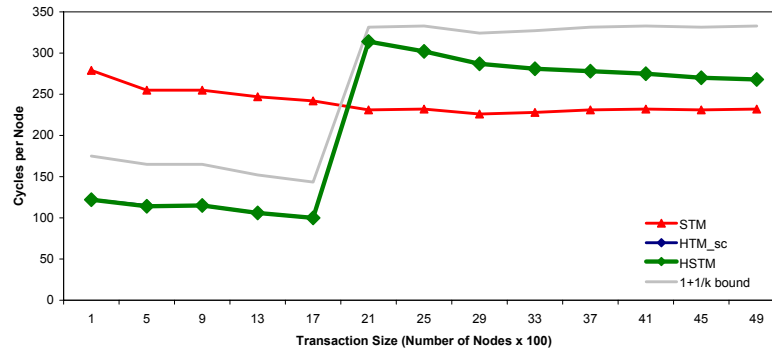


Figure 1: Performance of HSTM vs. transaction size. The application is a network push-relabel microbenchmark where the size of the transaction is varied. The HTM curve is not visible since it directly under the HSTM graph for less than 2100 nodes. The HTM curve stops at 1700 nodes since, after that point, transactions no longer fit in hardware. The  $1+1/k$  bound line shows the theoretical bound on HSTM performance.

HSTM performance on long transactions has similar preliminary results. As shown in Figure 2, short transactions run with performance similar to HTM. However, unlike size limitation, long transactions tend to still run in HTM so long as they eventually succeed after continuous retry. Thus, at 19000 nodes the HTM and STM curves cross. At that point, the hardware transaction needs to be retried so many times that the overhead surpasses that of STM. For such long transactions, HSTM tracks STM performance giving the best of both worlds.

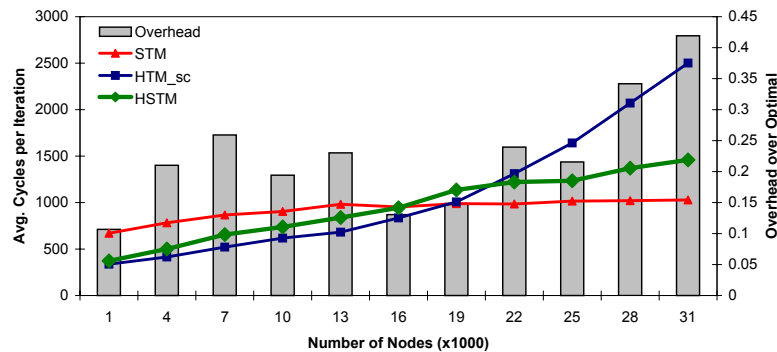


Figure 2: Performance of HSTM vs. transaction length. The application is a network push-relabel microbenchmark that accesses random locations in transactions. The length of the transaction is varied so that the longer the transaction, the more likely it will cause a context switch due to a TLB miss.

**Research Support:** This research is supported in part by the National Science Foundation Grant ACI-032497, in part by the Singapore-MIT Alliance, and in part by a grant provided by Silicon Graphics, Incorporated.

#### References:

- [1] C. Scott Ananian and Martin Rinard. Efficient software transactions for object-oriented languages. Technical Report Unpublished, MIT CSAIL, 2003.
- [2] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th Annual International Conference on Computer Architecture (ISCA)*, pages 289–300, San Diego, California, May 1993. ACM Press.