Hybrid TM Using NOrec STM *

Luke Dalessandro, Michael F. Spear,† and Michael L. Scott

University of Rochester
{luked,scott}@cs.rochester.edu

†Lehigh University
spear@cse.lehigh.edu

Transactional memory (TM) aims to simplify parallel programming by providing serializable memory transactions as an extension to the system's memory model. In the context of increasing numbers of on-chip processor cores, Sun's Rock processor [3] and AMD's proposed Advanced Synchronization Facility (ASF) [1] suggest that commercial support for *best-effort* hardware TM (HTM) may finally be forthcoming.

Best-effort HTMs impose implementation-defined limits on the length, size, or behavior of transactions. Rock transactions, for example, must not make function calls, overflow the write buffer, or suffer evictions of speculatively read cache lines. ASF transactions are guaranteed only four lines of speculatively written state, though more are likely in practice. Neither system supports transactions that span exceptions or interrupts.

Best-effort HTMs suffice for most concurrent data structure methods, but may fail repeatedly for larger, general-purpose transactions. *Hybrid* TM systems [5, 6] address this limitation by falling back to software TM (STM) when hardware limits are exceeded.

A hybrid system must arrange for hardware and software transactions to be aware of each others' activity. Because hardware transactions are typically isolated from nontransactional accesses (a property known as *strong atomicity* [2]), a hardware transaction can be expected to abort if a location it has accessed is written by a software transaction. Software transactions, however, are not usually strongly atomic; they detect conflicts based on *metadata* that identifies writers and (often) readers. Previous hybrid TM systems have therefore assumed that hardware transactions must include instructions that access metadata according to the STM protocol.

Kumar et al. [6] combine HTM with an object-cloning STM system. Hardware transactions modify the most recent version of an object in place, but software transactions create entirely new copies. Both kinds of transactions inspect and update object headers. Unfortunately, cloning introduces unacceptably high time and space overheads in many cases, and its data layout requirements are incompatible with systems languages like C. Recent STM research has moved to lower-overhead alternatives.

Damron et al. [5] combine HTM with a buffered-update, non-object-based STM that maintains a table of *ownership records* (orecs). Hardware transactions are instrumented with read and write barriers that inspect and update these orecs. This instrumentation significantly increases both the instruction count of hardware transactions and the size of their read sets, which in turn makes them more likely to fail due to capacity limits. Moreover metadata updates by software reader transactions, performed to alert hardware transactions of their existence, will induce cache misses in other software transactions and may abort concurrent hardware readers. Finally, subroutines called from both transactional and nontransactional contexts must be made available not only in native and STM versions, but in an HTM version as well.

We have devised [4] a low-overhead STM system ("NOrec") that we believe may constitute an ideal software fall-back in hybrid TM. Rather than a table of ownership records, NOrec maintains a single global *sequence lock*, which is used to trigger consistency checks and to serialize commits. Transactions log address/value pairs for read locations and check consistency by re-reading all locations and verifying that their values have not changed. This *value-based validation* eliminates the need for hardware transactions to modify metadata outside the begin- and end-transaction code. Crucially, hardware transactions in a NOrec hybrid do not require per-read or per-write barriers for communication with software-mode transactions.

For use in hybrid TM, we augment NOrec with a second word of global metadata. The original sequence lock continues to serve its purpose for software transactions, while a second lock serves to signal hardware transactions that a software writer is committing. Hardware transactions start by reading the second lock and then execute without further read or write instrumentation. As their last action before commit they read and increment the sequence lock, triggering validation in software transactions. Software transactions acquire both locks with a two-word hardware transaction at commit time, simultaneously signaling both software and hardware peers. The required read-modify-write sequence on the second lock will abort only those hardware transactions that are in the narrow window immediately prior to committing. We conjecture that the rate of such aborts will be very low in practice.

Experiments suggest [4] that the software-only version of NOrec combines surprisingly good scalability with lower overhead than any other STM system that supports concurrent writers. It also provides unusually clean semantics for programs that mix transactional and nontransactional access to the same data. The hybrid system described here adds minimal overhead to the hardware path. We plan to evaluate it on both Rock and ASF, with the expectation of developing a solid, general-purpose TM system suitable for production use on next-generation multicore chips.

References

[1] Advanced Micro Devices. Advanced Synchronization Facility: Proposed Architectural Specificaiton. Publication #45432, rev. 2.1, Mar. 2009. `developer.amd.com/assets/45432-ASF_Spec_2.1.pdf`.

[2] C. Blundell, E. C. Lewis, and M. M. K. Martin. Subtleties of Transactional Memory Atomicity Semantics. *IEEE Computer Architecture Letters*, 5(2), Nov. 2006.

[3] S. Chaudhry, R. Cypher, M. Ekman, M. Karlsson, A. Landin, and S. Yip. Rock: A High-Performance Sparc CMT Processor. *IEEE Micro*, 29(2):6–16, Mar.-Apr. 2009.

[4] L. Dalessandro, M. F. Spear, and M. L. Scott. NOrec: Streamlining STM by Abolishing Ownership Records. In *Proc. of the 15th ACM Symp. on Principles and Practice of Parallel Programming*, Jan. 2010.

[5] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum. Hybrid Transactional Memory. In *Proc. of the 12th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2006.

[6] S. Kumar, M. Chu, C. J. Hughes, P. Kundu, and A. Nguyen. Hybrid Transactional Memory. In *Proc. of the 11th ACM Symp. on Principles and Practice of Parallel Programming*, Mar. 2006.

* This work was supported in part by NSF grants CNS-0615139, CCF-0702505, and CSR-0720796; by equipment support from Sun and IBM; and by financial support from Intel and Microsoft.