Model checking transactional memory

John O'Leary
Intel

Bratin Saha
Intel

Mark R. Tuttle
Intel

Transactional memory is a programming abstraction for multi-threaded programming that provides thread synchronization without requiring the explicit use of locks. Locks are notoriously difficult to use correctly: They can lead to deadlock, priority inversion, and subtle errors, and they can impede compositionality. To the programmer, transactional memory has a very simple interface: Every block of code labeled "atomic" is executed as an "atomic transaction" without any interferences from other threads. To the implementer, transactional memory is an entre into the fascinating and subtle world of shared memory protocols.

Shared memory protocols are the way of the future. The programming model for the many-core chips coming from Intel and AMD is likely to be a shared memory model, but shared memory protocols are notoriously hard to get right, and their verification is not well supported by widely-available model checking implementations. For software protocol verification, explicit state model checkers like SPIN and Murphi appear to be the state of the art. Most explicit state model checkers, however, were written with a message-passing model of concurrency in mind, and not shared memory, which leads to two problems. First, shared memory protocols seem to bring explicit state model checker quickly to their knees, because, we suspect, the transition graph for shared memory protocols has a higher branching factor than message passing protocols. Second, models for most explicit state model checkers are written in a guarded-command style, which is a natural style for message passing but not for shared memory.

Our goal is to learn to model check shared memory algorithms in general, and transactional memory in particular. We take the Intel McRT STM [1] implementation of transactional memory as our example, which is itself an interesting, innovative, and subtle shared memory protocol. We choose SPIN [4] as our model checker because it is engineered for high performance, it is famous for its partial order reduction algorithm which has the effect of reducing the branching factor of the transition graph, and its input language, Promela, is a natural language for describing the sequential natural of each thread in a concurrent program. SPIN assumes processes communicate through message channels, but it does have global variables, and what is shared memory but a collection of globabl variables? The use of global variables, however, decimates the performance of the partial order reduction algorithm, which was the whole reason for choosing SPIN in the first place. It took significant engineering to model transactional memory efficiently in SPIN, but now, for example, we can model check the relatively large configuration of 2 threads each running a transaction consisting of 3 loads and stores — proving that every execution of every one of the 14,400 such programs is serializable — and we can do so in just over an hour and a half.

We are not the first paper to consider model checking transactional memory [2, 3]. What distinguishes our work, however, is our intention to allow both transactional and non-transactional loads and stores, meaning that load and stores are not required to appear within an "atomic" block. This means that we need to view the transactional model as an extension to the existing processor memory ordering model, which itself is already challenging to specify. Adding transactions adds new complexities: how do we understand the behavior when two threads access the same memory location, one in a transaction and one outside? This extension is work in progress, and we conclude our talk with a discussion of the issues arising from this extension.

References

[1] A. Adl-Tabatabai and et al. Compiler and runtime support for efficient software transactional memory. In *Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*, pages 26–37, June 2006.

[2] R. Alur, K. McMillan, and D. Peled. Model-checking of correctness conditions for concurrent objects. In *Eleventh Annual IEEE Symposium on Logic on Computer Science (LICS'96)*, pages 219—228, July 1996.

[3] A. Cohen, J. O'Leary, A. Pnueli, M. Tuttle, and L. Zuck. Verifying correctness of transactional memories. In M. Sheeran and J. Baumgartner, editors, *Seventh International Symposium on Formal Methods in Computer Aided Design (FMCAD'07)*, Nov. 2007.

[4] G. Holzman. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.