

Brief Announcement: Energy Implications of Multiprocessor Synchronization

Tali Moreshet*, R. Iris Bahar* and Maurice Herlihy†

*Brown University, Division of Engineering, Providence, RI 02912

†Brown University, Department of Computer Science, Providence, RI 02912

{tali,iris}@lems.brown.edu, mph@cs.brown.edu

Category: C.1.2 Computer Systems Organization Processor Architectures [Multiple Data Stream Architectures (Multiprocessors)]

Terms: Design

Keywords: energy aware, multiprocessor, transactional memory

Energy consumption is an increasingly important issue in multiprocessor design. The need for energy-aware systems is obvious for mobile systems, where low energy consumption translates to longer battery life, but it is also important for desktop and server systems, where high energy consumption complicates power supply and cooling. While energy consumption in uniprocessors has been the focus of a substantial body of research, energy consumption in multiprocessors has received less attention. This issue is becoming increasingly important as multiprocessor architectures migrate from high-end platforms into everyday platforms such as desktops, laptops, and servers.

In this paper we examine the energy implications of how multiprocessors synchronize concurrent access to memory. Whether synchronization is a substantial part of multiprocessor energy consumption is application-dependent. Nevertheless, the trends are clear. The widespread shift to multithreaded and multicore architectures is driven by two trends: Moore's law continues to provide more and more transistors, hence more and more threads and cores, while clock speed is hardly increasing at all. These trends imply that if applications are going to benefit from advancing technology, then they will have to become more and more parallel, and synchronization costs will become more and more significant.

The conventional way for applications to synchronize is by *locking*. Nevertheless, conventional synchronization techniques based on locks have substantial limitations [2]. Coarse-grained locks simply do not scale. Threads block one another even when they do not really interfere, and the lock itself causes memory contention. Fine-grained locks are more scalable, but they introduce substantial software engineering problems. Locks are also vulnerable to thread failures and delays.

Transactional memory [3] is a synchronization architecture that addresses these limitations. A transaction is a finite sequence of memory reads and writes executed by a single thread. Transactions are *atomic* and serializable. Hardware transactional memory proposals exploit hardware mechanisms such as speculative execution and on-chip caching. As a transaction executes, it caches the memory locations it reads or writes, marking them as transactional. A data conflict occurs if another thread accesses a transactionally cached location, and at least one access is a write. The native hardware cache coherence detects data conflicts. A transaction commits if it completes without encountering a conflict.

Here, we evaluate the energy/performance tradeoffs associated with these synchronization architectures. Overall, our tests suggest

that transactional memory is a promising approach to low-power synchronization. It appears to consume substantially less energy when synchronization conflict levels are low. To control energy costs at higher levels of conflict, we proposed a novel *serial execution mode* in which transactions are adaptively serialized at the hardware level. This technique decreases energy consumption, but sometimes (not always) decreases transaction throughput. Our results provide evidence that even at higher levels of conflict, transactions consume substantially less energy than locks. Moreover, switching to serial execution in the presence of conflicts may save additional energy.

It is important to note that the specific energy consumption numbers for locks and transactional memory are sensitive to several parameters: system configuration, conflict scenarios, type of locks being used, and the transactional memory hardware. To account for these parameters, we studied a variety of configurations and scenarios as described below. We considered a multiprocessor system consisting of several chips sharing an off-chip memory, as well as a multicore architecture in which processors communicate through a shared on-chip L2 cache. To account for various conflict scenarios, we used SPLASH-2 benchmarks, displaying very low levels of conflict, as well as our own microbenchmark that provides tunable levels of conflict. In addition, to provide a longer computation, we ran two variations of the OO7J benchmark [7]. We considered different types of locks, including unoptimized Pthread library locks and more energy-efficient, scalable, user-level MCS [5] and CLH spin locks [4] memory to have an energy advantage over locks in all of these variations. Finally, while we did not consider different types of transactional memory hardware, it was found that the large majority of transactions access a data set small enough to fit in a small cache [1], which fits our transactional memory hardware. Recent proposals of hybrid transactional memory [6] allow for transactions that do not fit in the cache to overflow to virtual memory. Additional hardware or software will incur a significant overhead in cases of large transactions. In these cases, locks are very likely to be more energy and performance efficient than transactional memory. Further study of these transactional memory systems is required to provide a more accurate appreciation of their energy consumption.

REFERENCES

- [1] J. W. Chung et al. The common case transactional behavior of multithreaded programs. In *HPCA*, pages 271–282, February 2006.
- [2] M. Herlihy, V. Luchangeo, M. Moir, and W. Scherer. Software transactional memory for dynamic-sized data structures. In *PODC*, July 2003.
- [3] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. In *ISCA*, May 1993.
- [4] P. S. Magnusson, A. Landin, and E. Hagersten. Queue locks on cache coherent multiprocessors. In *IPPS*, pages 165 – 171, April 1994.
- [5] J. M. Mellor-Crummey and M. L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM TOCS*, 9(1):21–65, 1991.
- [6] K. E. Moore, J. Bobba, M. J. Moravan, M. D. Hill, and D. A. Wood. LogTM: Log-based Transactional Memory. In *HPCA*, pages 258–269, February 2006.
- [7] A. Welc, S. Jagannathan, and A. L. Hosking. Transactional monitors for concurrent objects. In *ECOOP*, June 2004.