# Computer Architecture Reading Group Notes

Date: 1/22/04
Discussion Leader: Christos
Notes: Kelly
Topic: Multithreading
Papers:

1. R. Saavedra-Barrera, D. Culler, T. von Eicken. "Analysis of Multithreaded Architectures for Parallel Computing." Proceedings of the 2nd *Symposium on Parallel Algorithms and Architectures (SPAA)*, Crete, Greece, July 1990.
2. Ravi Rajwar, James R. Goodman. "Transactional lock-free execution of lock-based programs." *ASPLOS* 2002: 5-17.

## Administrative

Meeting time moved to Thursdays, 5-6pm, in Gates 392.

## "Analysis of Multithreaded Architectures for Parallel Computing"

Summary: This paper presents an analytical model for multithreading; no simulation results are presented. It examines the impact of remote reference latency (L), number of interleaved threads (N), context-switch latency (C), and the interval between context switches due to remote references (R). They show that a relatively small number of threads are needed to reach saturation at which the addition of threads does not improve performance. They also examine the impact of multithreading on cache behavior, showing that cache performance degrades as the number of threads becomes large.

Discussion:

1) How do the numbers they use compare to current numbers?
    a. R would be equivalent to an L2 cache miss latency because most L1 cache misses can be tolerated mostly.
    b. Their estimate for R would be small today.
2) In this paper, they obtain speedups around 2-3. However, numbers reported for SMT (which has up to 8 threads) are much smaller (closer to 1). Why?
    a. SMT covers small latencies, not long latencies.
    b. The goal of SMT is saturate resource usage.
    c. The bottleneck is the number of concurrent L2 cache misses supported by architecture.
    d. An additional note was that you can write a "performance virus" for SMT machines. For example, your program could cause the TLB to be flushed, causing simultaneously executing applications' performance to suffer.
3) What are some limitations or extensions of this work?
    a. Do not account for beneficial or conflicting sharing of cache.
    b. Could extend this model to handle multi-level memory hierarchies.
    c. Could extend this to include network model.

4) What happens when you incorporate both multithreading and multiprocessing into the same environment?
    a. From Vicky: "James Laudon did his thesis on interleaved multithreading design. His 1992 ISCA paper (Architectural and Implementation Tradeoffs in the Design of Multiple-Context Processors) described the multiprocessor simulation results (with SPLASH). The main point was to show the improvement of interleaved multithreading over blocked multithreading, but it did show the performance of multiple-context and single-context multiprocessor (like DASH)."
    b. How does the use of a shared L2 cache change? We discussed how graphics processors have up to 150 thread contexts.
    c. In general, if you start with simple cores, adding lots of multithreading hardware may not be worth it. However, if you start with a more complex core (4-way issue), adding another thread context may be easy. (Christos likes 2 contexts.)

**"Transactional lock-free execution of lock-based programs"**

Summary: This paper provides a hardware solution to improve performance of multithreaded applications using locks for synchronization without further burdening the programmer (for example, requiring the programmer to use finer grained locks). The authors improve upon an earlier approach called SLE which speculatively executes and atomically commits lock-based critical sections when there are no data conflicts. Transaction Lock Removal handles the remaining cases when conflicts occur; an order is imposed on lock requests. Consequently, the appearance of transactions is provided and starvation and live-lock are prevented.

Discussion:

1) What is the problem?
    a. Performance versus correctness.
2) What is their approach?
    a. Assume use of SLE hardware
    b. Add timestamps for ordering
3) How do they detect transactions?
    a. 1st time you execute you get lock but subsequent invocations you don't.
4) How does it work with multiple threads versus multiple processors?
    a. If you get switched out, you have to kill speculative instructions and rollback.
5) It is odd to implement speculation in hardware and then still require the programmer to get locking correct. Why not just give the programmer transactions?
6) It seems like this approach would be most useful for applications with coarse-grain locking. Why didn't they study that?
    a. They tuck away the results for a coarse-grained version of Mp3D despite getting good results?
7) Why did they use a deferral system instead of NACKS? They spend time explaining how to deal with deadlock because of using deferrals. Why bother?
8) It doesn't work with I/O.

9) In general, we thought the paper presented a good idea but had a yucky implementation. Additionally, people felt that the way of solving this problem was to use a programming language that understood locks or transactions.