Computer Architecture Reading Group Notes

Date: 4/22/04
Discussion Leader: Njuguna Njoroge
Topic: More on Transactions

Papers

1. Ravi Rajwar, James R. Goodman. Transactional lock-free execution of lock-based programs. ASPLOS 2002: 5-17
2. Jose F. Martinez and Josep Torrellas. Speculative Synchronization: Applying Thread-Level Speculation to Explicitly Parallel Applications. ASPLOS 2002.

Administrative
Think about future topics for discussion.
Next week's topic is power in servers.

Transactional Lock-Free Execution of Lock-Based Programs

Summary
For additional summary and discussion see notes from 1/22/04.
This paper proposes a hardware solution to improve multithreaded performance on programs using locks, without changing the programming model or further burdening the programmer with the need for fine grained locks. The paper builds on an earlier approach called Speculative Lock Elision (SLE), which speculatively executes beyond lock acquires and atomically commits critical sections when no conflicts occur. The new work, Transactional Lock Removal (TLR) uses time stamps to resolve conflicts and prevent starvation.

Discussion

- The approach relies on the cache coherence protocol to detect conflicts. Will it work in a hierarchical system in which some of the coherence information does not travel over the main system bus?
  - TLR will still be applicable for communication over the system wide bus (at the highest levels of the hierarchy).
  - It will probably add additional overhead to the already complex cache controllers.
  - The paper has ignored the issue of cache controller occupancy.
- How does the system handle rollover in timestamps?

- Does TLR improve programmability?
  - Programmers can continue to use legacy binaries.
  - The programmer can be good performance from locking that is more coarse grained
  - It may encourage the programmer to be overly conservative.
- What are the advantages/disadvantages of TLR over the TCC system proposed by Hammond et. al. in the paper from last week?
  - Programming Model
    - TLR allows the programmers to use legacy binaries and does not require the programmer to learn a new programming model, however, the existing programming model is hard to use correctly
    - TCC proposes a new model which the authors propose would be simpler to use.
  - Reliance on Cache Coherence Protocol
    - TLR relies on a correct cache coherence protocol, a difficult problem to solve.
    - TCC eliminates the need for a coherence protocol and and requires correct action be taken only at transaction boundaries
  - Software Complexity
    - TLR transactions are always of critical section size and resolve only atomic dependencies
    - TCC requires the programmer have some knowledge of appropriate transactions size, however it does provide a mechanism for ordered and partially ordered transactions

Speculative Synchronization: Applying Thread-Level Speculation to Explicitly Parallel Applications

Summary

This paper proposes using thread level speculation to speculate across synchronization primitive such as lock acquires and barriers. When a conflict occurs the system rolls back to the synchronization point. Forward progress is ensured by the presence of a safe thread. The approach does not require additional programmer effort.

Discussion

- Does this encourage the programmer to lock huge sections?

  - No, in this case the cache will overflow and the system will fall back on the safe thread.

- What is the safe thread
  - On a lock, it is the thread which actually acquires the lock.
  - On a barrier, it is the last thread through the barrier
  - On a flag, it is the producer thread
  - When all of a threads predecessors complete successfully the thread becomes safe and commits its memory state. If it is squashed by a conflict before this point all of its memory state is discarded.
- Where do the threads roll back to?
  - To the last synchronization point.
- The approach works by adding speculative bits to the cache lines and one extra line for the cache to hold the speculative lock or other sync primitive. How does it distinguish between multiple speculative threads on a single processor?
  - It doesn't, all speculative state is flushed.
- Is this technique reasonable if the application already has fine-grained locking?
  - It adds overhead on every synchronization primitive as well as overhead at the checkpoints.
- Does the fact that barriers contribute most to improvements in Figure 6 imply that the barrier instructions were unnecessary?
- How do the results in Figure 5 for percent execution time account for wasted speculative execution in IPC figures?
  - In general, squashed instructions and time spend in spin locks should not be counted in IPC measurements.
  - Also, comparisons should be make to the optimal code without the additional software overhead from the new approach.
  - In this case, the authors have avoided these issues by comparing actual execution time.
- How does this approach compare to the TCC system proposed by Hammond et. al. in the paper from last week?

- This approach may generate many small speculative threads to handle the case in which multiple objects are synchronized. This can be avoided by the programmer.
- TCC allows the programmer to put as many critical sections as are necessary into a single transaction, dealing more gracefully with multiple object transactions.