18-742
Lecture 8

Synchronization

Locks & Barriers

Spring 2005
Prof. Babak Falsafi
http://www.ece.cmu.edu/~ece742

Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Reinhardt, Smith, and Singh of University of Illinois, Carnegie Mellon University, University of Wisconsin, Duke University, University of Michigan, and Princeton University.

Announcements

Project proposal due on Friday
Homework 4 due on Monday

Performance of Locks

- Contested vs. Uncontested
- Test&set is good with no contention
- Array based (Queue) is best with high contention
- *Reactive Synchronization* by Lim & Agarwal
  - Choose lock implementation based on contention

Point-to-Point Event Synchronization

- Often use normal variables as flags

```
a = f(x);              while (flag == 0);
flag = 1;              b = g(a);
```

- If we know a before hand

```
a = f(x)               while (a == 0);
                       b = g(a);
```

- Assumes Sequential Consistency!!
- Full/Empty Bits
  - Set on Write
  - Cleared on Read
  - Can't write if set, can't read if clear

Transactional Memory

Lock-free data structures

Encapsulate all accesses in transactions

- **First class primitive supported by the hardware**
- **Example: begin, commit, or abort**

Transactional Memory:

- **Serializability**
- **Atomicity**
- **Do these requirements sound familiar?**

Transactional Memory: Why?

From the programming perspective, to avoid

- **Priority inversion**
- **Convoying**
- **Deadlock**

From the performance perspective

- **Avoid grabbing the lock**
- **No contention => no lock latency**
- **But, what happens when contention is high?**

Transactional Memory: Implementation

Hairy!

- **Need a Memory System/Cache that can**
  - **"flash" commit**
  - **"flash" invalidate**
  - **How?**
- **Must snoop to check access from other processors**
  - **Upon contention => abort**
- **The paper requires a special cache**
  - **Why?**
  - **How would you do it?**

SLE: Implementing Simple Transactions in a Modern OoO Core

Assumes annotated lock accesses

- **Lock acquire => "begin transaction"**
  - **Drop the RMW access (or LL/SC primitives) to the lock**
  - **Speculate within ROB & store buffer as in branch speculation**
- **Lock release => "commit transaction"**
  - **Drop the store to the lock**
- **Abort transaction**
  - **If external coherence requests for "speculatively-accessed" cache blocks**
    - **Data read must not be written by others**
    - **Data written must not be read or written by others**
- **How do you avoid the Beavis scenario?**

SLE Motivation

Placing locks is a hard problem

Let programs place locks conservatively & frequently

Let speculation elide the locks

Advantages:

- **Multiple procs can be in critical section simultaneously accessing separate data!**
- **Avoid RMW and stores to lock addresses**

SLE Implementation

Checkpoint register map tables

- Recall branch prediction in MIPS R10K

Memory values kept in store buffer

Head of ROB stalls until transaction commits

Disadvantages with implementation?

- ROB scalability
- Store buffer scalability
- Commit b/w
- Livelock?

How would you improve this implementation?

Experimental Methodology *

- **Measure performance for CMP, SMP, and DSM systems**
- **TSO Memory Consistency model**
- **Benchmarks simulated on SimpleMP, execution-driven simulator for running multithreaded binaries:**

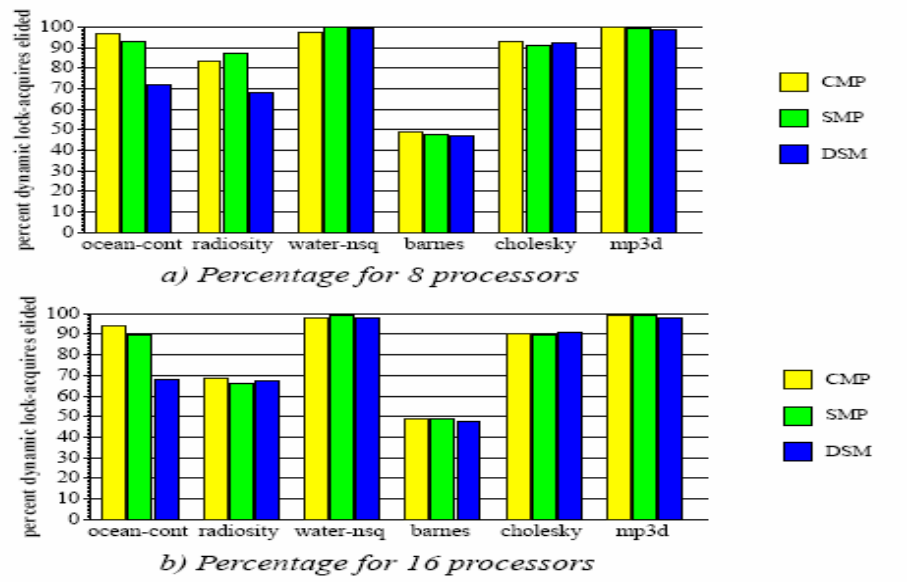
Application	Type of simulation	Inputs
Barnes	**N-Body**	**4k bodies**
Cholesky	**Matrix factoring**	**tk14.O**
Mp3D	**Rarefied field flow**	**24000 mols, 25 iter.**
Radiosity	**3D Rendering**	**-room, batch mode**
Water-Nsq	**Water molecules**	**512 mols, 3 iter.**
Ocean-cont	**Hydrodynamics**	**x130**

*** Slides from Eric Chung**

Configurations

All configurations	**1 Ghz, 128-entry ROB, 64-entry LSQ, 16-entry I-fetch queue, Out-of-order issue/commit of 8 inst. per cycle**
CMP	**Sun Gigaplane-type MOESI protocol between L1s, split transaction. Address bus: broadcast network**
SMP	**Sun Gigaplane-type MOESI protocol between L2s, split transaction. Address bus: broadcast network.**
DSM	**SGI Origin-2000-type MESI protocol between L2s. Directory: full mapped**

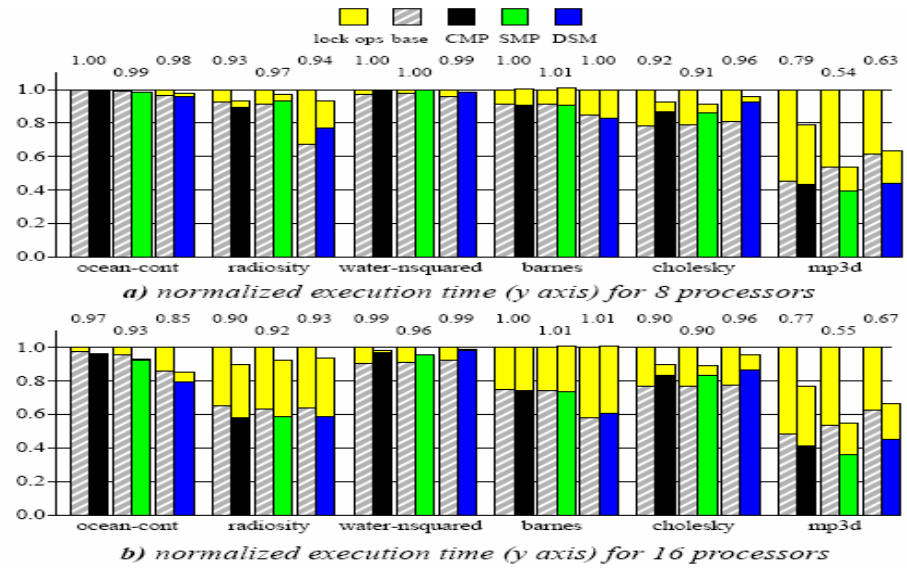
Percentage of dynamic locks elided



a) Percentage for 8 processors

b) Percentage for 16 processors

Execution Time



a) normalized execution time (y axis) for 8 processors

b) normalized execution time (y axis) for 16 processors

Implementing a Centralized Barrier

```
BARRIER(bar_name, p) {
    LOCK(bar_name.lock);
    if (bar_name.counter = 0)
      bar_name.flag = 0;
    bar_name.counter++;
    UNLOCK(bar_name.lock);
    if (bar_name.counter == p) {
      bar_name.counter = 0;
      bar_name.flag = 1;
    }
    else
      while(bar_name.flag = 0) {};        /* busy wait */
  }
```

- **Does this work?**

Barrier With Sense Reversal

```
BARRIER(bar_name, p) {
    local_sense = !(local_sense);              /* toggle private state */
    LOCK(bar_name.lock);
    bar_name.counter++;
    UNLOCK(bar_name.lock);
    if (bar_name.counter == p) {
      bar_name.counter = 0;
      bar_name.flag = local_sense;
    }
    else
      while(bar_name.flag != local_sense) {};    /* busy wait */
  }
```

Synchronization Algorithms

- **Tournament Barriers, SW Combining Tree**