18-742

Lecture 6

Symmetric Multiprocessors

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 posted on the web
Homework 3 posted due on Friday

Seminar @ Pitt:

Single Chip Multiprocessors: The Next Wave of Computer Architecture

Guri Sohi
University of Wisconsin
Friday, February 4, 2005
10:30am SENSQ 5317

Project Description

Dates:

- Proposal due Feb. 11th
- Proposal meetings the week of 14th
- Status report due Mar. 2nd
- Status meetings the week of 14th
- Final report/poster due Apr. 29th

Grading (30% of final class grade):

- Proposal 5 points
- Status report 5 points
- Final report 80 points
- Poster 10 points
- Total out of 100 points

Project Description (Cont.)

Proposal:

- Read related work first (ACM/IEEE conf. online links)
- Talk to me or Brian if you have questions

Infrastructure: SimFlex (www.ece.cmu.edu/~simflex)

- Scaffold simulator
- "Tools" button on the class web page for info
- Sophisticated tool/get started right away

Topics:

- List of eight topics
- Open to other topics

Readings

Chapter 5

Reader 3:

- **Alain Kägi, Doug Burger, and Jim Goodman *Efficient Synchronization: Let Them Eat QOLB*, Proc. 24th International Symposium on Computer Architecture (ISCA 24), June, 1997.**
- **M. Herlihy and J. E. B. Moss, *Transactional Memory: Architectural Support for Lock-Free Data Structures*, ISCA 1993.**
- **R. Rajwar and J. R. Goodman, *Speculative Lock Elision: Enabling Highly-Concurrent Multithreaded Execution*, Micro 2001.**
- **Alan Charlesworth, *StarFire: Extending the SMP Envelope*, IEEE Micro, Jan. 1998.**

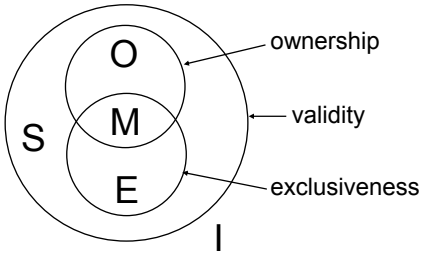
4-State (MESI) Invalidation Protocol

- **Often called the Illinois protocol**
- **Modified (dirty)**
- **Exclusive (clean unshared) only copy, not dirty**
- **Shared**
- **Invalid**
- **Requires shared signal to detect if other caches have a copy of block**
- **Cache Flush for cache-to-cache transfers**
  - **Only one can do it though**
- **What does state diagram look like?**

More Generally: MOESI

- **M - Modified (dirty)**
- **O - Owned (dirty but shared) WHY?**
- **E - Exclusive (clean unshared) only copy, not dirty**
- **S - Shared**
- **I - Invalid**

- **Variants**
  - **MSI**
  - **MESI**
  - **MOSI**
  - **MOESI**



Tradeoffs in Protocol Design

- **New State Transitions**
- **What Bus Transactions**
- **Cache block size**
- **Workload dependence**
- **Compute bandwidth, miss rates, from state transitions**

Computing Bandwidth

- Why bandwidth?
- How do I compute it?
- Monitor State Transitions
  - tells me bus transactions
  - I know how many bytes each bus transaction requires

MESI State Transitions and Bandwidth

FROM/TO	NP	I	E	S	M
NP	--	--	BusRd 6+64	BusRd 6+64	BusRdX 6+64
I	--	--	BusRd 6+64	BusRd 6+64	BusRdX 6+64
E	--	--	--	--	--
S	--	--	NA	--	BusUpgr 6
M	BusWB 6 + 64	BusWB 6+64	NA	BusWB 6 + 64	--

Bandwidth of MSI vs. MESI

- 200 MIPS/MFLOPS processor
  - use with measured state transition counts to obtain transitions/sec
- Compute state transitions/sec
- Compute bus transactions/sec
- Compute bytes/sec
- What is BW savings of MESI over MSI?
- Difference between protocols is Exclusive State
  - Add BusUpgr for E->M transtion
- Result is very small benefit!
  - Small number of E->M transitions
  - Only 6 bytes on bus

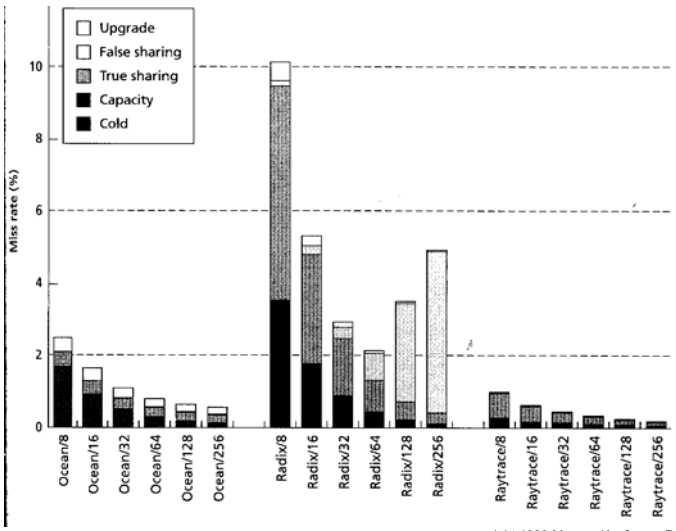
MSI BusUpgrd vs. BusRdX

- MSI S->M Transition Issues BusUpgrd
  - could have block invalidated while waiting for BusUpgrd response
  - adds complexity to detect this
- Instead just issue BusRdX
  - from MESI put BusRdX in E->M and S->M
- Result is 10% to 20% Improvement
  - application dependent

Cache Block Size

- **Block size is unit of transfer and of coherence**
  - **Doesn't have to be, could have coherence smaller [Goodman]**
- **Uniprocessor 3C's**
  - **(Compulsory, Capacity, Conflict)**
- **SM adds Coherence Miss Type**
  - **True Sharing miss fetches data written by another processor**
  - **False Sharing miss results from independent data in same coherence block**
- **Increasing block size**
  - **Usually fewer 3C misses but more bandwidth**
  - **Usually more false sharing misses**
- **P.S. on increasing cache size**
  - **Usually fewer capacity/conflict misses (& compulsory don't matter)**
  - **No effect on true/false "coherence" misses (so may dominate)**

Cache Block Size: Miss Rate



Invalidate vs. Update

- **Pattern 1:**

```
for i = 1 to k
   P1(write, x);        // one write before reads
   P2--PN-1(read, x);
end for i
```

- **Pattern 2:**

```
for i = 1 to k
   for j = 1 to m
      P1(write, x);   // many writes before reads
   end for j
   P2(read, x);
end for i
```


Invalidate vs. Update, cont.

- **Pattern 1 (one write before reads)**
  - N = 16, M = 10, K = 10
  - Update
    - » Iteration 1: N regular cache misses (70 bytes)
    - » Remaining iterations: update per iteration (14 bytes; 6 cntrl, 8 data)
  - Total Update Traffic = 16*70 + 9*14 = 1246 bytes
    - » book assumes 10 updates instead of 9…
  - Invalidate
    - » Iteration 1: N regular cache misses (70 bytes)
    - » Remaining: P1 generates upgrade (6), 15 others Read miss (70)
  - Total Invalidate Traffic = 16*70 + 9*6 + 15*9*17 = 10,624 bytes
- **Pattern 2 (many writes before reads)**
  - Update = 1400 bytes
  - Invalidate = 824 bytes

Invalidate vs. Update, cont.

- What about real workloads?
  - Update can generate too much traffic
  - Must limit (e.g., competitive snooping)
- Current Assessment
  - Update very hard to implement correctly (c.f., consistency discussion coming next)
  - Rarely done
- Future Assessment
  - May be same as current or
  - Chip multiprocessors may revive update protocols
    - » More intra-chip bandwidth
    - » Easier to have predictable timing paths?

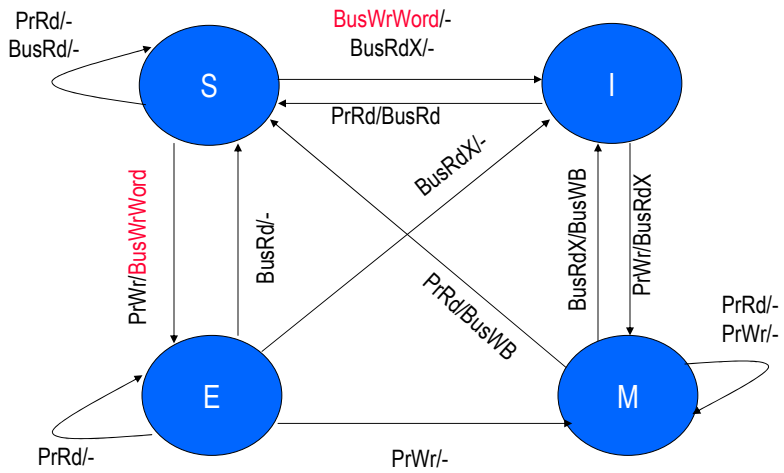
Qualitative Sharing Patterns

- [Weber & Gupta, ASPLOS3]
- Read-Only
- Migratory Objects
  - Maniputalated by one processor at a time
  - Often protected by a lock
  - Usually a write causes only a single invalidation
- Synchronization Objects
  - Often more processors imply more invalidations
- Mostly Read
  - More processors imply more invalidations, but writes are rare
- Frequently Read/Written
  - More processors imply more invalidations

Write-Once Protocol

- First bus-based coherence protocol proposed
- Designed to run on uniproc. bus: Multibus
- Read miss:
  - If block Dirty, supply block, writeback to memory
  - If block Valid, read from memory
- Write hit:
  - On Dirty and Reserved, proceed, on Valid write through word
- Write miss:
  - If block Dirty, the block comes directly from cacher
  - If block Valid, the block comes from memory

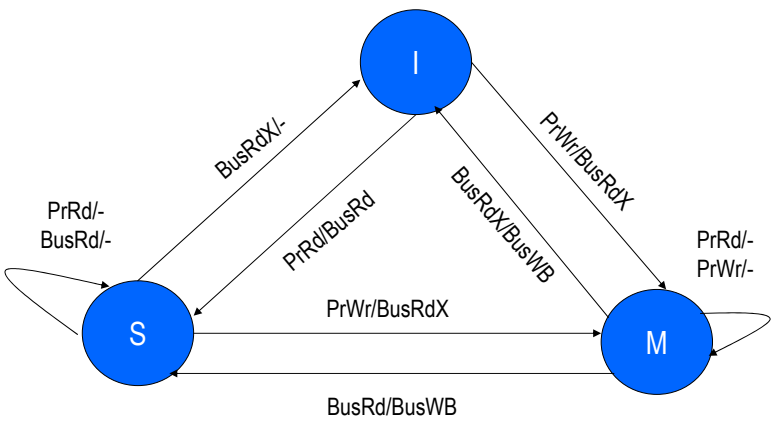
Write-Once Diagram [Goodman]



Synapse Protocol

- First bus-based product
- Write bits in memory to obviate inhibit line
- Read miss:
  - If block Dirty, stop and writeback to memory, then read
  - If block Shared, read from memory
- Write hit:
  - On Dirty, proceed, on Shared reads block exclusively
- Write miss:
  - As in Read miss, block always comes from memory

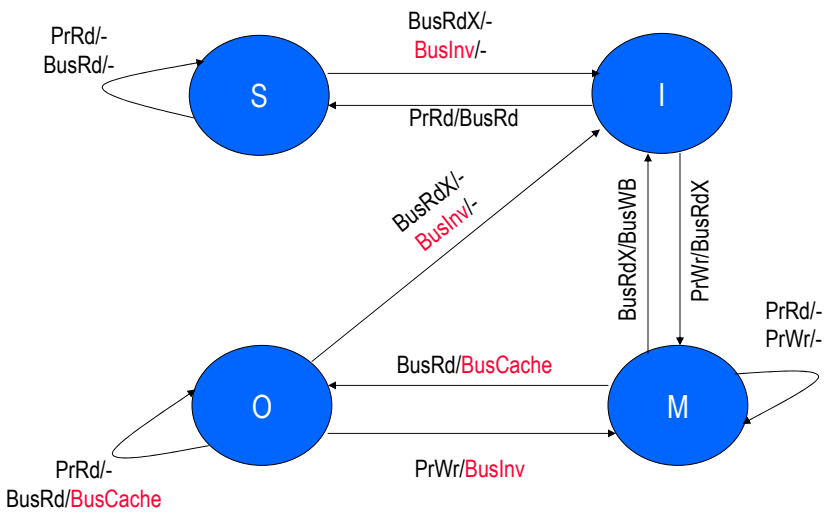
Synapse Diagram



Berkeley Protocol

- **Multiprocessor Workstation (SPUR)**
- **Uses "O" state to optimize cache-to-cache transfers**
- **Read miss:**
  - **If block Dirty, transfer cache-to-cache**
  - **If block Shared, read from memory,**
  - **If block Shared-Dirty read from owner**
- **Write hit:**
  - **On Dirty, proceed, on Shared-Dirty Invalidate**
- **Write miss:**
  - **Same as Read miss**

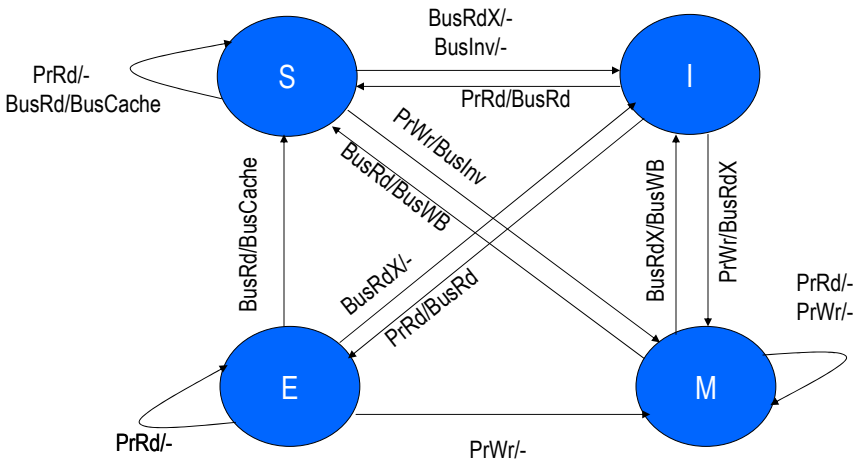
Berkeley Diagram



Illinois Protocol

- **Implemented in SGI multiprocessors in 1980's**
- **Missed data always comes from caches, bus SharedLine**
- **Read miss:**
  - **If block Dirty, transfer cache-to-cache, and write back**
  - **If bloc Shared, Exclusive transfer cache-to-cache**
- **Write hit:**
  - **On Dirty and Exclusive, proceed, on Shared, invalidate**
- **Write miss:**
  - **Same as Read miss**

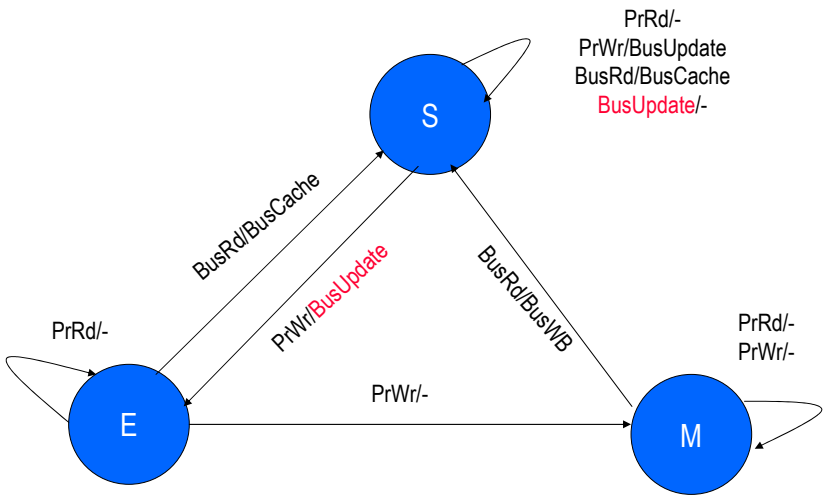
Illinois Diagram



DEC Firefly Protocol

- **Write-update protocol**
  - **Valid-Exclusive using bus SharedLine**
- **Read miss:**
  - **If block Dirty, cache-to-cache transfer, write back**
  - **If block Exclusive, Shared, cache-to-cache transfer**
- **Write hit:**
  - **On Dirty and Exclusive, proceed, on Shared update**
- **Write miss:**
  - **Same as Read miss**
  - **Check SharedLine to update state accordingly**

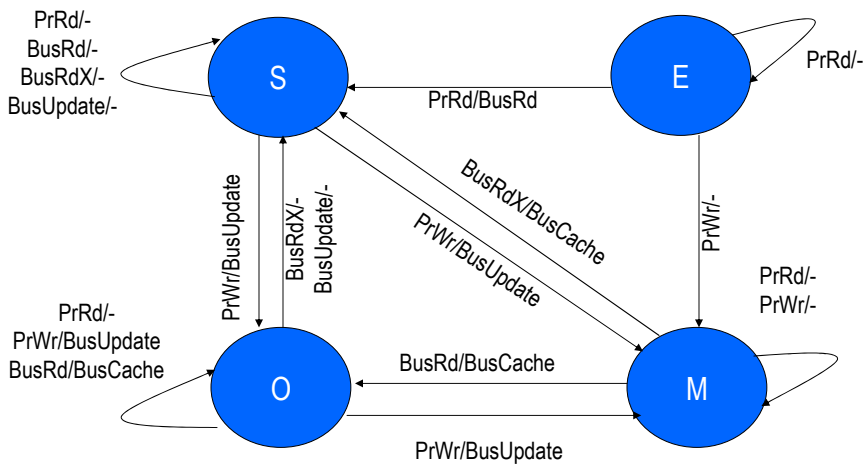
DEC Firefly Diagram



Xerox Dragon Protocol

- **Write-update protocol**
  - **Exclusive & Shared-Dirty states**
- **Read miss:**
  - **If block Dirty, Shared-Dirty, cache-to-cache transfer**
  - **If block Exclusive, Shared, data comes from memory**
- **Write hit:**
  - **On Dirty, Exclusive, proceed, on Shared, Shared-Dirty update**
- **Write miss:**
  - **Same as Read miss**
  - **If SharedLine set, must update**

Xerox Dragon Diagram



What did Archibald & Baer Find Out?

Private data:

- **Want to save on write latency (Exclusive State)**
- **E.g., Dragon, Firefly and Illinois**

Shared data:

- **Only equal-behavior case -> Read hits**
- **Small fractions of shared data -> Write update wins**
  - **Basically has similar cons as write through**
- **Ownership schemes improve perf. by keeping dirty data in caches**