CSE 490/590 Computer Architecture

Directory-Based Caches I

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last time…

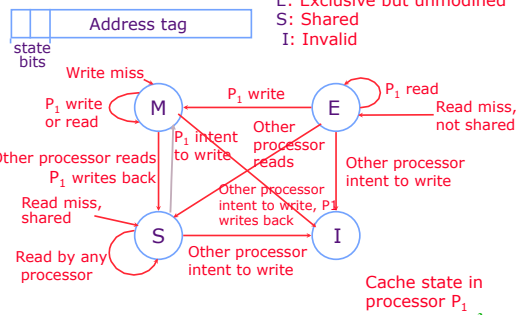
- Snoopy Cache Coherence Protocol
  - MSI & MESI

MESI: An Enhanced MSI protocol

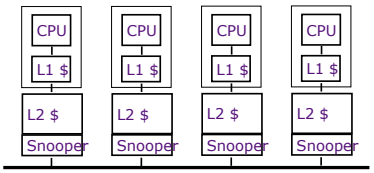
increased performance for private data

Each cache line has a tag

M: Modified Exclusive
E: Exclusive but unmodified
S: Shared
I: Invalid



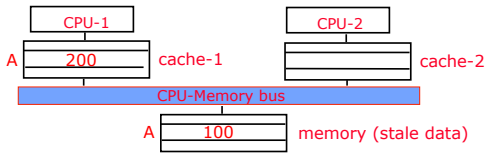
Optimized Snoop with Level-2 Caches



- Processors often have two-level caches
  - small L1, large L2 (usually both on chip now)
- *Inclusion property:* entries in L1 must be in L2
  invalidation in L2 ⇒ invalidation in L1
- Snooping on L2 does not affect CPU-L1 bandwidth

What problem could occur?

Intervention



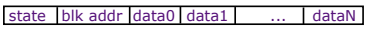
When a read-miss for A occurs in cache-2, a read request for A is placed on the bus

- Cache-1 needs to supply & change its state to shared
- The memory may respond to the request also!

Does memory know it has stale data?

Cache-1 needs to intervene through memory controller to supply correct data to cache-2

False Sharing



A cache block contains more than one word

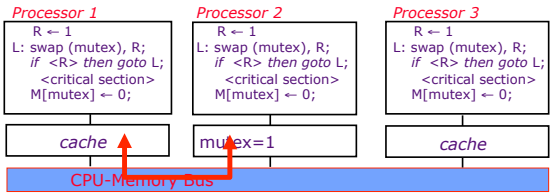
Cache-coherence is done at the block-level and not word-level

Suppose M_1 writes $word_i$ and M_2 writes $word_k$ and both words have the same block address.

What can happen?

Synchronization and Caches:

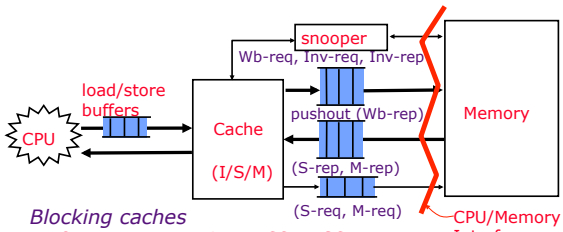
Performance Issues



Cache-coherence protocols will cause mutex to *ping-pong* between P1's and P2's caches.

Ping-ponging can be reduced by first reading the mutex location *(non-atomically)* and executing a swap only if it is found to be zero.

Out-of-Order Loads/Stores & CC



Blocking caches
One request at a time + CC ⇒ SC

Non-blocking caches
Multiple requests (different addresses) concurrently + CC
⇒ Relaxed memory models

CC ensures that all processors observe the same order of loads and stores to an address

Performance of Symmetric Shared-Memory Multiprocessors

Cache performance is combination of:

1. Uniprocessor cache miss traffic
2. Traffic caused by communication
   - Results in invalidations and subsequent cache misses
- Adds 4th C: *coherence miss*
  - Joins Compulsory, Capacity, Conflict
  - (Sometimes called a *Communication* miss)

Coherency Misses

1. True sharing misses arise from the communication of data through the cache coherence mechanism
   - Invalidates due to 1st write to shared block
   - Reads by another CPU of modified block in different cache
   - Miss would still occur if block size were 1 word
2. False sharing misses when a block is invalidated because some word in the block, other than the one being read, is written into
   - Invalidation does not cause a new value to be communicated, but only causes an extra cache miss
   - Block is shared, but no word in block is actually shared
     ⇒ miss would not occur if block size were 1 word

CSE 490/590 Administrivia

- Keyboards available for pickup at my office
- Project 2: less than 2 weeks left (Deadline 5/2)
  - Will have demo sessions
- No class on 5/2 (finish the project!)
- Final exam: Thursday 5/5, 11:45pm – 2:45pm
- Project 2 + Final = 55%

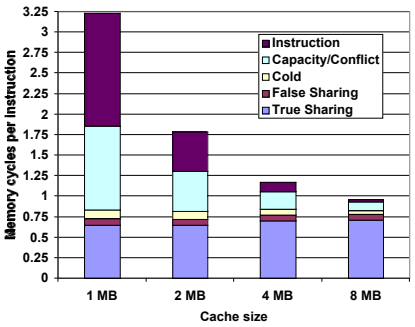
Example: True v. False Sharing v. Hit?

- **Assume x1 and x2 in same cache block. P1 and P2 both read x1 and x2 before.**

Time	P1	P2	True, False, Hit? Why?
1	Write x1		True miss; invalidate x1 in P2
2		Read x2	False miss; x1 irrelevant to P2
3	Write x1		False miss; x1 irrelevant to P2
4		Write x2	False miss; x1 irrelevant to P2
5	Read x2		True miss; invalidate x2 in P1

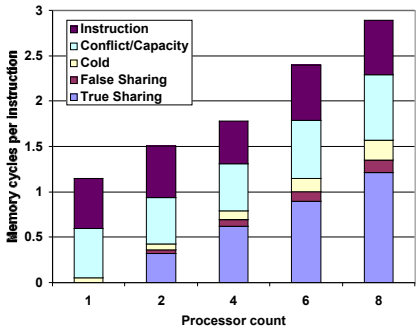
MP Performance 4 Processor Commercial Workload: OLTP, Decision Support (Database), Search Engine

- True sharing and false sharing unchanged going from 1 MB to 8 MB (L3 cache)
- Uniprocessor cache misses improve with cache size increase (Instruction, Capacity/Conflict, Compulsory)



MP Performance 2MB Cache Commercial Workload: OLTP, Decision Support (Database), Search Engine

- True sharing, false sharing increase going from 1 to 8 CPUs



A Cache Coherent System Must:

- Provide set of states, state transition diagram, and actions
- Manage coherence protocol
  - (0) Determine when to invoke coherence protocol
  - (a) Find info about state of address in other caches to determine action
    - » whether need to communicate with other cached copies
  - (b) Locate the other copies
  - (c) Communicate with those copies (invalidate/update)
- (0) is done the same way on all systems
  - state of the line is maintained in the cache
  - protocol is invoked if an "access fault" occurs on the line
- Different approaches distinguished by (a) to (c)

Bus-based Coherence

- All of (a), (b), (c) done through broadcast on bus
  - faulting processor sends out a "search"
  - others respond to the search probe and take necessary action
- Could do it in scalable network too
  - broadcast to all processors, and let them respond
- Conceptually simple, but broadcast doesn't scale with number of processors, P
  - on bus, bus bandwidth doesn't scale
  - on scalable network, every fault leads to at least P network transactions
- Scalable coherence:
  - can have same cache states and state transition diagram
  - different mechanisms to manage protocol

Scalable Approach: Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252