CMSC 411
Computer Systems Architecture
Lecture 21
Multiprocessors 3

Outline

- Review
- Coherence
- Write Consistency
- Administrivia
- Snooping
- Building Blocks
- Snooping protocols and examples
- Coherence traffic and performance on MP
- Directory-based protocols and examples
- Conclusion
COHERENCE COMPLICATIONS & PERFORMANCE FOR SNOOPING

Complications: Write Races

- Cannot update cache until bus is obtained
  - Otherwise, another processor may get bus first, and then write the same cache block!
- Two step process:
  - Arbitrate for bus
  - Place miss on bus and complete operation
Complications: Write Races (cont.)

- If miss occurs to block while waiting for bus
  - Handle miss (invalidate may be needed) and then restart
- Split transaction bus:
  - Bus transaction is not atomic:
    can have multiple outstanding transactions for a block
  - Multiple misses can interleave, allowing two caches to grab block in the Exclusive state
    » Must track and prevent multiple misses for one block

Limitations in Symmetric Shared-Memory Multiprocessors and Snooping Protocols

- Single memory to accommodate all CPUs
  - Multiple memory banks
- Bus must support both coherence traffic & normal memory traffic
  - Multiple buses or interconnection networks (cross bar or small point-to-point)
- Example - AMD Opteron:
  - Memory connected directly to each dual-core chip
  - Point-to-point connections for up to 4 chips
  - Remote memory and local memory latency are similar, allowing OS to view Opteron as uniform memory access (UMA) computer
Performance of Symmetric Shared-Memory MPs

• Cache performance is combination of
  – Uniprocessor cache miss traffic
  – Traffic caused by communication
    » Results in invalidations and subsequent cache misses

• 4th C: coherency miss
  – Joins Compulsory, Capacity, Conflict

Coherency Misses

• 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

• 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
**Example: True v. False Sharing v. Hit?**

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

<table>
<thead>
<tr>
<th>Time</th>
<th>P1</th>
<th>P2</th>
<th>True, False, Hit? Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write x1</td>
<td></td>
<td>True miss; invalidate x1 in P2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Read x2</td>
<td>False miss; x1 irrelevant to P2</td>
</tr>
<tr>
<td>3</td>
<td>Write x1</td>
<td></td>
<td>False miss; x1 irrelevant to P2</td>
</tr>
<tr>
<td>4</td>
<td>Write x2</td>
<td></td>
<td>False miss; x1 irrelevant to P2</td>
</tr>
<tr>
<td>5</td>
<td>Read x2</td>
<td></td>
<td>True miss; invalidate x2 in P1</td>
</tr>
</tbody>
</table>

**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

### Outline

- Review
- Coherence
- Write Consistency
- Administrivia
- Snooping
- Building Blocks
- Snooping protocols and examples
- Coherence traffic and Performance on MP
- Directory-based protocols and examples
- Conclusion
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 other copies 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 p (# of processors)
  - On bus, bus bandwidth doesn’t scale
  - On scalable network, every fault leads to at least p network transactions
Scalable Approach: Directories

- Scalable coherence uses
  - Same cache states and state transition diagram...
  - But different mechanisms to manage protocol
- 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

Basic Operation of Directory

- $k$ processors.
- With each cache-block in memory:
  - $k$ presence-bits, 1 dirty-bit
- With each cache-block in cache:
  - 1 valid bit, and 1 dirty (owner) bit

- Read from main memory by processor $i$:
  - If dirty-bit OFF then { read from main memory; turn $p[i]$ ON; }
  - if dirty-bit ON then { recall line from dirty proc (cache state to shared); update memory; turn dirty-bit OFF; turn $p[i]$ ON; supply recalled data to $i$; }

- Write to main memory by processor $i$:
  - If dirty-bit OFF then { supply data to $i$; send invalidations to all caches that have the block; turn dirty-bit ON; turn $p[i]$ ON; ... }
  - ...
**Picture To Keep In Mind**

- Cost structure
  - cache ~2 cycles
  - local memory ~80 cycles
  - remote memory ~200 cycles

**Directory Protocol**

- Similar to Snoopy Protocol: Three states
  - **Shared**: ≥ 1 processors have data, memory up-to-date
  - **Uncached** (no processor has it; not valid in any cache)
  - **Exclusive**: 1 processor (owner) has data; memory out-of-date

- In addition to cache state, must track which processors have data when in the shared state (usually bit vector, 1 if processor has copy)

- Keep it simple(r):
  - Writes to non-exclusive data
    - write miss
  - Processor blocks until access completes
  - Assume messages received and acted upon in order sent
Directory Protocol 2

- No bus and don’t want to broadcast:
  - Interconnect no longer single arbitration point
  - All messages have explicit responses
- Terms: typically 3 processors involved
  - Local node where a request originates
  - Home node where the memory location of an address resides
  - Remote node has a copy of a cache block, whether exclusive or shared
- Example messages on next slide
  - $P =$ processor number, $A =$ address

State Transition Diagram for One Block

- States identical to snoopy case; transactions very similar.
- Transitions caused by read misses, write misses, invalidates, data fetch requests
- Generates read miss & write miss msg to home directory.
- Write misses that were broadcast on the bus for snooping $\Rightarrow$ explicit invalidate & data fetch requests.
- Note: on a write, a cache block is bigger, so need to read the full cache block – write allocate
**CPU - Cache State Machine**

- State machine for CPU requests for each memory block
- Invalid state if in memory

**State Transition Diagram for Directory**

- Same states & structure as the transition diagram for an individual cache
- 2 actions: update of directory state & send messages to satisfy requests
- Tracks all copies of memory block
- Also indicates an action that updates the sharing set, Sharers, as well as sending a message
**Directory State Machine**

- State machine for Directory requests for each memory block
- Uncached state if in memory

**Uncached**

Data Write Back:
Sharers = {}
(Write back block)

Write Miss:
Sharers = {P};
send Data Value Reply msg to remote cache

Read Miss:
Sharers = {P};
send Data Value Reply

**Shared (read only)**

Write Miss:
Sharers = {P};
send Invalidate to Sharers;
then Sharers = {P};
send Data Value Reply msg

Read miss:
Sharers += {P};

**Exclusive (read/write)**

Read miss:
Sharers += {P};
send Data Value Reply
(b) Write miss to a block with two sharers

<table>
<thead>
<tr>
<th>Requestor</th>
<th>Directory node</th>
<th>Reply with owner identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Read request to directory</td>
<td>Directory node</td>
<td>2. Reply with owner identity</td>
</tr>
<tr>
<td>3. Read msg to memory</td>
<td>Directory</td>
<td>3a. Read msg to sharer</td>
</tr>
<tr>
<td>4a. Data Reply</td>
<td>Directory</td>
<td>3b. Data Reply</td>
</tr>
<tr>
<td>4b. Revision message to directory</td>
<td>Directory</td>
<td>3c. Revision message to sharer</td>
</tr>
</tbody>
</table>

(a) Read miss to a block in dirty state

<table>
<thead>
<tr>
<th>Requestor</th>
<th>Directory node</th>
<th>Reply with sharer's identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Read request to directory</td>
<td>Directory node</td>
<td>2. Reply with sharer's identity</td>
</tr>
<tr>
<td>3a. Read msg to sharer</td>
<td>Directory</td>
<td>3b. Data Reply</td>
</tr>
<tr>
<td>3c. Data Reply</td>
<td>Directory</td>
<td>3d. Data Reply</td>
</tr>
</tbody>
</table>

(b) Write miss to a block with two sharers
A Popular Middle Ground

- Two-level “hierarchy”
- Individual nodes are multiprocessors, connected non-hierarchically
  - e.g. mesh of SMPs
- Coherence across nodes is directory-based
  - directory keeps track of nodes, not individual processors
- Coherence within nodes is snooping or directory
  - orthogonal, but needs a good interface of functionality
- SMP on a chip directory + snoop?

And in Conclusion …

- Caches contain all information on state of cached memory blocks
- Snooping cache over shared medium for smaller multiprocessors
  - By invalidating other cached copies on write
- Sharing cached data ⇒
  - Coherence (values returned by a read)
  - Consistency (when a written value will be returned by a read)
- Snooping and Directory Protocols similar
  - Bus makes snooping easier because of broadcast
  - Snooping ⇒ uniform memory access
- Directory has extra data structure
  - Keep track of state of all cache blocks
- Distributing directory
  ⇒ scalable shared address space multiprocessor
  ⇒ cache coherent, non-uniform memory access