EE 660: Computer Architecture
Directory Cache Coherence

Yao Zheng
Department of Electrical Engineering
University of Hawai‘i at Mānoa

Based on the slides of Prof. David Wentzlaff
Coherency Misses

1. **True sharing misses** arise from the communication of data through the cache coherence mechanism
   - Invalidates due to 1\textsuperscript{st} 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
     \[\Rightarrow\]  miss would not occur if block size were 1 word
Example: True v. False Sharing v. Hit?

• Assume $x_1$ and $x_2$ in same cache block.
P1 and P2 both read $x_1$ and $x_2$ 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 $x_1$ in P2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Read x2</td>
<td>False miss; $x_1$ irrelevant to P2</td>
</tr>
<tr>
<td>3</td>
<td>Write x1</td>
<td></td>
<td>False miss; $x_1$ irrelevant to P2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Write x2</td>
<td>False miss; $x_1$ irrelevant to P2</td>
</tr>
<tr>
<td>5</td>
<td>Read x2</td>
<td></td>
<td>True miss; invalidate $x_2$ 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
Directory Coherence Motivation

• Snoopy protocols require every cache miss to broadcast
  – Requires large bus bandwidth, \( O(N) \)
  – Requires large cache snooping bandwidth, \( O(N^2) \) aggregate

• Directory protocols enable further scaling
  – Directory can track all caches holding a memory block and use point-to-point messages to maintain coherence
  – Communication done via scalable point-to-point interconnect
Directory Cache Coherence

Interconnection Network

Directory Controller

DRAM Bank
Distributed Shared Memory

Interconnection Network
Non-Uniform Memory Access (NUMA)

• Latency to access memory is different depending on node accessing it and address being accessed

• NUMA does not necessarily imply ccNUMA
Multicore Chips in a Multi-chip System
Address to Home Directory

High Order Bits Determine Home (Directory)

- OS can control placement in NUMA
- Homes can become hotspots

Low Order Bits Determine Home (Directory)

- OS loses control on placement
- Load balanced well
## Basic Full-Map Directory

<table>
<thead>
<tr>
<th>State</th>
<th>Sharers/Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1011001100011</td>
</tr>
<tr>
<td>U</td>
<td>xxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>U</td>
<td>xxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>E</td>
<td>0001000000000</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**State:** State that the directory believes cache line is in {Shared, Uncached, Exclusive, (Pending)}

**Sharers/Owner:** If State == Shared:
Bit vector of all of the nodes in system that have line in shared state.
If State == Modified:
Denotes which node has line in cache exclusively
Cache State Transition Diagram

*The MSI protocol*

**Each cache line has state bits**

<table>
<thead>
<tr>
<th>State Bits</th>
<th>Address Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>M: Modified</td>
<td></td>
</tr>
<tr>
<td>S: Shared</td>
<td></td>
</tr>
<tr>
<td>I: Invalid</td>
<td></td>
</tr>
</tbody>
</table>

Write miss (P1 gets line from memory)

Other processor reads (P1 writes back)

Read miss (P1 gets line from memory)

Read by any processor

Other processor intent to write

Other processor intent to write (P1 writes back)

P1 reads or writes

Cache state in processor P1
Cache State Transition Diagram

For Directory Coherence

- **M**: Modified
- **S**: Shared
- **I**: Invalid

Write miss, P1 Send Write Miss Message (Waits for reply before transition)

Receive Read Miss Message (P1 writes back)

Read miss, P1 Send Read Miss Message (Waits for reply before transition)

Read by P1

P1 reads or writes

Invalidate Message (P1 writes back, Reply after)

Cache state in processor P1
Cache State Transition Diagram

For Directory Coherence

M: Modified
S: Shared
I: Invalid

Write miss, P1 Send Write Miss Message (Wait for reply before transition)

Receive Read Miss Message (P1 writes back)

Read miss, P1 Send Read Miss Message (Wait for reply before transition)

Read by P1

P1 reads or writes
Writeback (Notify Directory)

Invalidate Message (P1 writes back, Reply after)

Invalidate Message (Reply)

Notify Directory

Cache state in processor P1

15
Directory State Transition Diagram

U: Uncached
S: Shared
E: Exclusive

Read Miss P: Fetch from E Node, Data Value Reply
Sharers = \{P\}

Write Miss P: Fetch/Invalidate From E Node, Data Value Reply
Sharers = \{P\}

Data Write-Back P: Sharers = {}
# Message Types

<table>
<thead>
<tr>
<th>Message type</th>
<th>Source</th>
<th>Destination</th>
<th>Message contents</th>
<th>Function of this message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
<td>Node P has a read miss at address A; request data and make P a read sharer.</td>
</tr>
<tr>
<td>Write miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
<td>Node P has a write miss at address A; request data and make P the exclusive owner.</td>
</tr>
<tr>
<td>Invalidate</td>
<td>Local cache</td>
<td>Home directory</td>
<td>A</td>
<td>Request to send invalidates to all remote caches that are caching the block at address A.</td>
</tr>
<tr>
<td>Invalidate</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
<td>Invalidate a shared copy of data at address A.</td>
</tr>
<tr>
<td>Fetch</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
<td>Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.</td>
</tr>
<tr>
<td>Fetch/invalidate</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
<td>Fetch the block at address A and send it to its home directory; invalidate the block in the cache.</td>
</tr>
<tr>
<td>Data value reply</td>
<td>Home directory</td>
<td>Local cache</td>
<td>D</td>
<td>Return a data value from the home memory.</td>
</tr>
<tr>
<td>Data write-back</td>
<td>Remote cache</td>
<td>Home directory</td>
<td>A, D</td>
<td>Write-back a data value for address A.</td>
</tr>
</tbody>
</table>
Multiple Logical Communication Channels Needed

• Responses queued behind requests can lead to deadlock

• Many different message types, need to determine which message type can create more messages

• Segregate flows onto different logical/physical channels
Memory Ordering Point

• Just like in bus based snooping protocol, need to guarantee that state transitions are atomic
• Directory used as ordering point
  – Whichever message reaches home directory first wins
  – Other requests on same cache line given negative acknowledgement (NACK)
• NACK causes retry from other node
• Forward progress guarantee needed
  – After node acquires line, need to commit at least one memory operation before transitioning invalidating line
Scalability of Directory Sharer List Storage

- **Full-Map Directory (Bit per cache)**

- **Limited Pointer (Keep list of base-2 encoded sharers)**

- **LimitLess (Software assistance)**
Beyond Simple Directory Coherence

• On-chip coherence (Leverage fast on-chip communications to speed up or simplify protocol)
• Cache Only Memory Architectures (COMA)
• Large scale directory systems (Scalability of directory messages and sharer list storage)
SGI UV 1000 (Origin Descendant)

Maximum Memory: 16TB
Maximum Processors: 256
Maximum Cores: 2560
Topology: 2D Torus
TILE64Pro

- 64 Cores
- 2D Mesh
- 4 Memory Controllers
- 3 Memory Networks
  - Divide different flows of traffic
- Each node can be a home

![Diagram of TILE64Pro](image)
Beyond EE 660

• Computer Architecture Research
  – International Symposium on Computer Architecture (ISCA)
  – International Symposium on Microarchitecture (MICRO)
  – Architectural Support for Programming Languages and Operating Systems (ASPLOS)
  – International Symposium on High Performance Computer Architecture (HPCA)
• Build some chips / FPGA
• Parallel Computer Architecture
Acknowledgements

• These slides contain material developed and copyright by:
  – Arvind (MIT)
  – Krste Asanovic (MIT/UCB)
  – Joel Emer (Intel/MIT)
  – James Hoe (CMU)
  – John Kubiatowicz (UCB)
  – David Patterson (UCB)
  – Christopher Batten (Cornell)

• MIT material derived from course 6.823
• UCB material derived from course CS252 & CS152
• Cornell material derived from course ECE 4750
Copyright © 2013 David Wentzlaff