EC 513
Computer Architecture

Cache Coherence - Snoopy Cache Coherence

Prof. Michel A. Kinsky
Memory Consistency in SMPs

CPU-1

A 100

Cache-1

CPU-Memory bus

CPU-2

A 100

Cache-2

Memory

A 100
Memory Consistency in SMPs

- Suppose CPU-1 updates $A$ to 200
  - Write-back: memory and cache-2 have stale values
  - Write-through: cache-2 has a stale value
Suppose CPU-1 updates A to 200
- Do these stale values matter?
- What is the view of shared memory for programming?
Write-through Caches & SC

<table>
<thead>
<tr>
<th>Prog T1</th>
<th>Cache-1</th>
<th>Memory</th>
<th>Cache-2</th>
<th>Prog T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST X, 1</td>
<td>X = 1</td>
<td>X = 0</td>
<td>Y = Y'</td>
<td>LD Y, R1</td>
</tr>
<tr>
<td>ST Y', 11</td>
<td>Y = 11</td>
<td>Y = 10</td>
<td>X = X'</td>
<td>ST Y', R1</td>
</tr>
</tbody>
</table>

- **T1 is executed**
- **Cache-1 writes back Y**
- **T2 executed**
- **Cache-1 writes back X**
### Write-through Caches & SC

<table>
<thead>
<tr>
<th>Prog T1</th>
<th>Cache-1</th>
<th>Memory</th>
<th>Cache-2</th>
<th>Prog T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST X, 1</td>
<td>X = 1</td>
<td>X = 0</td>
<td>Y = 11</td>
<td>LD Y, R1</td>
</tr>
<tr>
<td>ST Y, 11</td>
<td>Y = 11</td>
<td>Y = 10</td>
<td>X' =</td>
<td>ST Y', R1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X' =</td>
<td>Y' =</td>
<td>LD X, R2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X =</td>
<td>X' =</td>
<td>ST X', R2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y =</td>
<td>X' =</td>
<td></td>
</tr>
</tbody>
</table>

- **T1 is executed**
  - Cache-1 writes back Y
- **T2 executed**
  - Cache-1 writes back X
  - Cache-2 writes back X' & Y'
Write-through Caches & SC

- T1 is executed

- T2 executed

- Write-through caches don’t preserve sequential consistency either
Maintaining Sequential Consistency

- **Motivation:** We can do without locks -- SC is sufficient for writing producer-consumer and mutual exclusion codes (e.g., Dekker)
- **Problem:** Multiple copies of a location in various caches can cause SC to break down.
- **Hardware support is required such that**
  - Only one processor at a time has write permission for a location
  - No processor can load a stale copy of the location after a write
  - Cache coherence protocols
A System with Multiple Caches

- Modern systems often have hierarchical caches
- Each cache has exactly one parent but can have zero or more children
- Only a parent and its children can communicate directly
A System with Multiple Caches

- Inclusion property is maintained between a parent and its children, i.e.,
  - a in \( L_i \) implies that a in \( L_{i+1} \)
Cache Coherence Protocols for SC

- **Write request:**
  - The address is invalidated in all other caches before the write is performed, or
  - The address is updated in all other caches after the write is performed

- **Read request:**
  - If a dirty copy is found in some cache, a write-back is performed before the memory is read
  - We will focus on Invalidation protocols as opposed to Update protocols
DMA stands for Direct Memory Access

Either Cache or DMA can be the Bus Master and effect transfers

Page transfers occur while the Processor is running
Problems with Parallel I/O

- Memory $\rightarrow$ Disk: Physical memory may be stale if cache copy is dirty
- Disk $\rightarrow$ Memory: Cache may have data corresponding to the memory
Snoopy Cache Goodman 1983

- Idea: have the cache watch (or snoop upon) DMA transfers, and then “do the right thing”
- Snoopy cache tags are dual-ported

Diagram:
- Proc.
- Tags and State
- Data (lines)
- Cache

Arrows:
- A
- R/W
- D

Legend:
- Used to drive Memory Bus when Cache is Bus Master
- Snoopy read port attached to Memory Bus
# Snoopy Cache Actions

<table>
<thead>
<tr>
<th>Observed Bus Cycle</th>
<th>Cache State</th>
<th>Cache Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Cycle, i.e.,</td>
<td>Address not cached</td>
<td>No action</td>
</tr>
<tr>
<td>Memory → Disk</td>
<td>Cached, unmodified</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td>Cached, modified</td>
<td>Cache intervenes</td>
</tr>
<tr>
<td>Write Cycle, i.e.,</td>
<td>Address not cached</td>
<td>No action</td>
</tr>
<tr>
<td>Disk → Memory</td>
<td>Cached, unmodified</td>
<td>Cache purges its copy</td>
</tr>
<tr>
<td></td>
<td>Cached, modified</td>
<td>???</td>
</tr>
</tbody>
</table>
Use snoopy mechanism to keep all processors’ view of memory coherent
Cache State Transition Diagram

- The MSI protocol

- Each cache line has a tag
  - Address tag
  - State bits

- Cache state in processor $P_1$
  - $M$: Modified
  - $S$: Shared
  - $I$: Invalid

- Write miss
- Read miss
- Read by any processor

- Other processor reads $P_1$ writes back
- $P_1$ intends to write
- $P_1$ reads or writes

Department of Electrical & Computer Engineering
Cache State Transition Diagram

- The MSI protocol

Each cache line has a tag

Address tag

<table>
<thead>
<tr>
<th>state bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

M: Modified
S: Shared
I: Invalid

P₁ reads or writes
Write miss

P₁ intends to write
Other processor intends to write
P₁ writes back

Read miss

Read by any processor

Other processor intends to write

Cache state in processor P₁
Cache State Transition Diagram

- The MSI protocol

Each cache line has a tag

<table>
<thead>
<tr>
<th>state bits</th>
<th>Address tag</th>
</tr>
</thead>
</table>

M: Modified
S: Shared
I: Invalid

Cache state in processor $P_1$
2 Processor Example

- **P₁ reads**
- **P₁ writes**
- **P₂ reads**
- **P₂ writes**
- **P₁ reads**
- **P₁ writes**
- **P₂ writes**
- **P₁ writes**

**Diagram:**
- Node S: Read miss
- Node I: P₂ intends to write
- Node M: P₁ reads or writes, Write miss
- Arrows:
  - P₁ reads, P₁ writes back
  - P₂ reads, P₂ writes back

**Description:**
- P₁ intends to write
- P₂ reads or writes
- P₂ writes back
- P₁ writes back
- P₂ reads back
- P₁ reads back

*Department of Electrical & Computer Engineering*
2 Processor Example

- **P₁ reads**
- **P₁ writes**
- **P₂ reads**
- **P₂ writes**
- **P₁ reads**
- **P₁ writes**
- **P₂ reads**
- **P₂ writes**
- **P₁ writes**
Observation

- If a line is in the M state then no other cache can have a copy of the line!
  - Memory stays coherent, multiple differing copies cannot exist
Next Class

- Cache Coherence - Directory Cache Coherence