CIS 429/529
Cache Coherence Protocols
Sequential Consistency

Prof. Michel A. Kinsy
Memory Consistency in SMPs

CPU-1

A 100

cache-1

CPU-2

A 100

cache-2

CPU-Memory bus

A 100

memory
• Suppose CPU-1 updates A to 200
  ‣ write-back: memory and cache-2 have stale values
  ‣ write-through: cache-2 has a stale value
Memory Consistency in SMPs

Suppose CPU-1 updates A to 200

- Do these stale values matter?
- What is the view of shared memory for programming?
Write-back Caches & SC

• T1 is executed
  prog T1
  ST X, 1
  ST Y, 11

  X = 1
  Y = 11

• cache-1 writes back Y

  cache-1
  X = 1
  Y = 11

  cache-2
  Y = 11
  Y’ = 11
  X = 0
  X’ = 0

• T2 executed

  prog T2
  LD Y, R1
  ST Y’, R1
  LD X, R2
  ST X’, R2

  X = 1
  Y = 11
  X’ = 1
  Y’ = 11

• cache-1 writes back X

  cache-1
  X = 1
  Y = 11
  X’ = 1
  Y’ = 11

• cache-2 writes back X’ & Y’

  cache-2
  Y = 11
  Y’ = 11
  X = 0
  X’ = 0

  incoherent
Write-through Caches & SC

- **Write-through Caches don’t preserve sequential consistency either**

<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 = 0</td>
<td>Y = 10</td>
<td>Y = 11</td>
<td>X = 1</td>
</tr>
<tr>
<td>ST Y, 1</td>
<td>Y = 10</td>
<td>X = 0</td>
<td>X = 1</td>
<td>Y = 11</td>
</tr>
</tbody>
</table>

- **T1 is executed**

<table>
<thead>
<tr>
<th>cache-1</th>
<th>memory</th>
<th>cache-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 1</td>
<td>Y = 11</td>
<td>Y = 11</td>
</tr>
<tr>
<td>Y = 11</td>
<td>X = 11</td>
<td>X = 0</td>
</tr>
</tbody>
</table>

- **T2 executed**

<table>
<thead>
<tr>
<th>cache-1</th>
<th>memory</th>
<th>cache-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 1</td>
<td>Y = 11</td>
<td>Y = 11</td>
</tr>
<tr>
<td>Y = 11</td>
<td>X = 0</td>
<td>X = 0</td>
</tr>
</tbody>
</table>
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.

- DMA stands for Direct Memory Access
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
## Snoopy Cache Actions

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

Other processor reads
P₁ writes back

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
2 Processor Example

P₁ reads
P₁ writes
P₂ reads
P₂ writes
P₁ reads
P₁ writes

P₂ reads, P₁ writes back
P₁ reads or writes
Write miss
P₂ reads
P₂ writes back
P₁ reads
P₂ writes

P₁ reads, P₂ writes back
P₂ reads or writes
Write miss
P₁ reads
P₂ writes back
P₂ reads
P₂ writes

P₁ reads
P₂ reads
P₁ writes
P₂ writes

Read miss
P₁ reads, P₁ writes back
P₁ reads
P₁ writes

Read miss
P₂ reads, P₂ writes back
P₂ reads
P₂ writes

Read miss
P₁ reads, P₂ writes back
P₂ reads
P₂ writes

P₂ reads
P₂ writes
P₁ reads
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