6.823 SPring15 Quiz 3 Review (L15-L19)

TA: Po-An Tsai, Hsin-Jung Yang
Quiz 3 is about multi-core system

• How they communicate
  – NOC
    • Underlining communication links

  – Coherence
    • Same memory location

  – Sequential consistency
    • Different memory location
NoC

• Network-on-chip is about how elements (cores, cache banks, memory controller, I/O controller etc.) communicate with each other

• Important when you have a large system
Topology

• How different nodes connect to each other
  – Ring
  – Mesh/Torus
  – Tree

• Important properties
  – Diameter
  – Avg. distance
  – Bisection bandwidth
  – Links (overhead)
Flow control

• How messages are forwarded from src to dst

• Buffered/ Bufferless
  – Wormhole is the most common one
  – but there is head-of-line blocking problem
Router architecture

• Handout 10
## Router architecture

<table>
<thead>
<tr>
<th>Field</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Global state</td>
<td>Either idle (I), routing (R), waiting for an output VC (V), active (A), or waiting for credits (C).</td>
</tr>
<tr>
<td>R</td>
<td>Route</td>
<td>After routing is completed for a packet, this field holds the output port selected for the packet.</td>
</tr>
<tr>
<td>O</td>
<td>Output VC</td>
<td>After virtual-channel allocation is completed for a packet, this field holds the output virtual channel of port R assigned to the packet.</td>
</tr>
<tr>
<td>P</td>
<td>Pointers</td>
<td>Flit head and tail pointers into the input buffer.</td>
</tr>
</tbody>
</table>
Router architecture

• Handout 10

• Pipeline stages: RC->VA->SA->ST
Routing

- What is the path between src and dst

- Choose a path so that the message can arrive faster

- Choose a path to ensure there is no deadlock/livelock
Routing: Deadlock

Routing diagram showing packet 1, packet 2, packet 3, and packet 4 in a network.
Turn Model

The eight possible turns and cycles in a 2D mesh

Only four turns are allowed in the XY routing algorithm
Cache Coherence

• Coherence concerns reads/writes to a single memory location

• Coherence Rules
  – **Write propagation:** Writes eventually become visible to all processors
    • Write-invalidates protocol & write-update protocol
  – **Write serialization:** Writes to the same location are serialized (all processors see them in the same order)
    • Snoopy-based protocol and directory-based protocol
MSI Coherence States

- **Modified (M):** The cache has the exclusive copy of the line with read and write permissions.
- **Shared (S):** The cache has a shared, read-only copy of the line. Other caches may also have read-only copies.
- **Invalid (I):** Data is not present in this cache but can be present in other caches.
Snoopy-based Protocol

All caches observe (snoop) each other’s actions through a shared bus
MSI Snoopy-based Protocol

Processor-initiated transitions

Bus-initiated transitions

Actions

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Read (PrRd)</td>
</tr>
<tr>
<td>Processor Write (PrWr)</td>
</tr>
<tr>
<td>Bus Read (BusRd)</td>
</tr>
<tr>
<td>Bus Read Exclusive (BusRdX)</td>
</tr>
<tr>
<td>Bus Writeback (BusWB)</td>
</tr>
</tbody>
</table>
MSI Snoopy-based Protocol Example

LD 0xA
MSI Snoopy-based Protocol Example

BusRd 0xA

Memory

Cache 0
Tag  State  Data

Core 0

Cache 1
Tag  State  Data

Core 1

Shared Bus

LD 0xA
MSI Snoopy-based Protocol Example

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>2</td>
</tr>
</tbody>
</table>

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
</table>

Core 0

Core 1

LD 0xA
MSI Snoopy-based Protocol Example

<table>
<thead>
<tr>
<th>Cache 0</th>
<th>Cache 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>State</td>
</tr>
<tr>
<td>0xA</td>
<td>S</td>
</tr>
</tbody>
</table>

Core 0

Core 1

Memory

Shared Bus

ST 0xA
MSI Snoopy-based Protocol Example
MSI Snoopy-based Protocol Example

<table>
<thead>
<tr>
<th>Cache 0</th>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0xA</td>
<td>I</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache 1</th>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0xA</td>
<td>M</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 0

Core 1

Memory

Shared Bus

ST 0xA
MSI Snoopy-based Protocol Example

Memory

Shared Bus

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>M</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 0

Core 1

ST 0xA
MSI Snoopy-based Protocol Example

Memory

Core 0

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>2</td>
</tr>
</tbody>
</table>

Core 1

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>M</td>
<td>3</td>
</tr>
</tbody>
</table>

Shared Bus

LD 0xA
MSI Snoopy-based Protocol Example

BusRd 0xA

Memory

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>2</td>
</tr>
</tbody>
</table>

Core 0

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>M</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1

LD 0xA
MSI Snoopy-based Protocol Example

Memory

BusWB 0xA, 3

Shared Bus

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>2</td>
</tr>
</tbody>
</table>

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 0

Core 1

LD 0xA
MSI Snoopy-based Protocol Example

LD 0xA
Directory-based Protocol

• Better scalability: no need to broadcast

• Directory tracks possible sharers (in sharer set)
  – Memory-based directories v.s. directory caches
  – Full bit vectors, coarse-grain bit-vectors, limited pointers, or bloom filters...

• Directory serves as the ordering point: can use unordered network
Directory-based Protocol

• Better scalability: no need to broadcast
• Directory tracks possible sharers (in sharer set)
  – Memory-based directories v.s. directory caches
  – Full bit vectors, coarse-grain bit-vectors, limited pointers, or bloom filters...
• Directory serves as the ordering point: can use unordered network
MSI Directory-based Protocol

• **Cache State**
  - Modified (M), Shared (S), Invalid (I)
  - Transient states (I→S, I→M, and S→M):
    The cache has sent an upgrade request to the directory and is waiting for the response.
MSI Directory-based Protocol

• **Directory State**
  For each memory address, the directory maintains its coherence state and a sharer set:
  
  – **Coherence State:**
    
    • **Uncached (Un):** No cache has a valid copy.
    • **Shared (Sh):** One or more caches are in the S state.
    • **Exclusive (Ex):** One of the caches is in the M state.
    • **Transient states (Ex→Sh, Ex→Un, and Sh→Un):** The directory has sent a request to one or multiple caches and is waiting for the cache response(s).
  
  – **Sharer Set:** Contains the IDs of the caches with shared or exclusive permissions for that memory location.
Synchronization Primitives

• Why?
  – Modifying a shared variable must be atomic
  – Can be used to implement locks

• Examples of synchronization primitives:
  – Test-and-set: TST rs, Imm(rt):
    rs ← Mem[Imm+rt]
    if (Mem[Imm+rt] == 0)
      Mem[Imm+rt] ← 1

TST is useful to implement locks.
M[Imm+rt] == 1: Someone holds the lock.
M[Imm+rt] == 0: Lock is free.
Synchronization Primitives

- **Examples of synchronization primitives:**
  - **Test-and-set:** TST rs, Imm(rt)
  - **Compare-and-Swap:** CAS old, new, Imm(base):
    
    ```
    if (old == Mem[Imm+base])
    Mem[Imm+base] ← new
    else
    old ← Mem[Imm+base]
    ```

CAS can also be used to implement locks.

Please read Handout 14.
Synchronization Primitives

• Examples of synchronization primitives:
  – Test-and-set: TST rs, Imm(rt)
  – Compare-and-Swap: CAS old, new, Imm(base)
  – Load-reserve/Store-conditional (Most Efficient)

Load-reserve R, (a):
  <flag, adr> ← <1, a>;
  R ← M[a];

Store-conditional (a), R:
  if <flag, adr> == <1, a>
  then cancel other procs’ reservation on a;
  M[a] ← <R>;
  status ← succeed;
  else status ← fail;
Consistency

• Coherence:
  – Concerns reads/writes to a single memory location

• Consistency:
  – Concerns reads/writes to multiple memory locations
Why Consistency Matters

*Initial memory contents*

- a: 0
- flag: 0

**Processor 1**
- Store (a), 10;
- Store (flag), 1;

**Processor 2**
- L: Load r1, (flag);
- if r1 == 0 goto L;
- Load r2, (a);

- What value does r2 hold after both processors finish running this code?

  It depends on the order in which processor 2 observes processor 1’s stores!
  
  10 if Store (flag) > Store (a); 0 or 10 otherwise
Memory Consistency Models

• **Sequential Consistency:**
  – Program order maintained
  – Loads and stores are atomic

• **Weaker (Relaxed) Consistency:**
  – Total Store Order, Partial Store Order, Relaxed Memory Order...
  – Use memory fences to enforce load/store ordering:
    • \( \text{Fence}_{\text{ww}}, \text{Fence}_{\text{wr}}, \text{Fence}_{\text{rw}}, \text{Fence}_{\text{rr}} \)
The end

Good luck!! 😊