Cache Coherence:
Snooping Protocol, Directory Protocol

Some of these slides courtesy of
David Patterson and David Culler

Coherent Memory

- Caches need to behave in a way that causes reads and writes to behave 'as expected.'

- Intuitively, A read sees the most recent value written to it (regardless of which processor it comes from).

- Writes to the same location are serialized, I.e. two writes to the same location X are seen in the same order by all processors.
Coherent Memory

- Formally, coherence means it is possible to construct a total serial ordering of all reads and writes such that
  1. Operations issued by a given processor occur in the order that they were issued by that processor.
  2. The value returned by each read operation is the value written by the last write in that serial order.

SMP Architecture

- Memory: centralized with uniform access time ("uma") and bus interconnect
- Examples: Sun Enterprise 5000, SGI Challenge, Intel SystemPro

![Diagram of SMP Architecture]

Main memory

I/O System

One or more levels of cache

One or more levels of cache

One or more levels of cache

One or more levels of cache

Processor

Processor

Processor

Processor
Example Cache Coherence Problem

- Processors see different values for \( u \) after event 3
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value when
  - Processes accessing main memory may see very stale value
- Unacceptable to programs, and frequent!

SMP Cache Coherency Solution

- **Snooping Solution (Snoopy Bus):**
  - Send all requests for data to all processors
  - Processors snoop to see if they have a copy and respond accordingly
  - Requires broadcast, since caching information is at processors
  - Works well with bus (natural broadcast medium)
  - Dominates for small scale machines (most of the market)
Snoopy Cache-Coherence Protocols

- Bus is a shared broadcast medium
- Cache Controller “snoops” all transactions on the shared bus
  - relevant transaction if for a block it contains
  - take action to ensure coherence
    » invalidate, update, or supply value
  - depends on state of the block and the protocol

Basic Snoopy Protocols

- **Write Invalidate** Protocol (for write through and write back)
  - Multiple readers, single writer
  - Write to shared data: an invalidate is sent to all caches which snoop and invalidate any copies
  - Read Miss:
    » Write-through: memory is always up-to-date
    » Write-back: snoop in caches to find most recent copy
- **Write Broadcast** Protocol (typically for write through):
  - Write to shared data: broadcast on bus, processors snoop, and update any copies
  - Read miss: memory is always up-to-date
- **Write serialization**: **bus** serializes requests!
  - Bus is single point of arbitration
An Example Snoopy Protocol

- Invalidation protocol, write-back cache
- Each block of memory is in one state:
  - Clean in all caches and up-to-date in memory (Shared)
  - OR Dirty in exactly one cache (Exclusive)
  - OR Not in any caches
- Each cache block is in one state (track these):
  - Shared : block can be read
  - OR Exclusive : cache has only copy, its writeable, and dirty
  - OR Invalid : block contains no data
- Read misses: cause all caches to snoop bus
- Writes to clean line are treated as misses

Example 1: Snoopy Protocol for Write-through, Write-No-Allocate

- Invalidation protocol, write-back cache
- Each cache block is in one of two states:
  - Valid : block can be read or written
  - Invalid : block contains stale data or no data

- Notation: Event/Action
  Event can be from Processor (PrRd, PrWr)
  or observed on the bus (BusWr)
  Action can be nil or action on the bus (BusRd, BusWr)
Example 1: Snoopy coherence (cont.)

Example 2: Snoopy Protocol for Write-back Cache

- Invalidation protocol, write-back cache
- Each cache block is in one of three states:
  - **Shared**: block can be read
  - **Exclusive**: cache has only copy, writeable and dirty
  - **Invalid**: block contains no data
Snoopy-Cache State Machine-I

- State machine for CPU requests for each cache block

Invalid → CPU Read → Shared (read/only)
- Place read miss on bus

CPU Write
- Place Write Miss on bus

CPU read hit
- CPU write hit

Exclusive (read/write)

CPU Read hit

Snoopy-Cache State Machine-II

- State machine for bus requests for each cache block
- Appendix E gives details of bus requests

Invalid → Write miss for this block → Shared (read/only)

Write miss for this block
- Write Back Block; (abort memory access)

Exclusive (read/write)

Read miss for this block
- Write Back Block; (abort memory access)
Snoopy-Cache State Machine-III

- State machine for CPU requests for each cache block and for bus requests for each cache block

Cache Block State

- CPU Read hit
- CPU Write hit
- CPU read miss
- CPU Write miss
- Place read miss on bus
- Place Write Miss on bus
- Write miss for this block
- Write Back Block; (abort memory access)
- CPU Read
- CPU Write
- Place Write

Snooping Cache Variations

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive</td>
<td>Owned</td>
<td>Private Dirty</td>
<td>Modified</td>
</tr>
<tr>
<td>Shared</td>
<td>Exclusive</td>
<td>Shared Clean</td>
<td>(private,!=Memory)</td>
</tr>
<tr>
<td>Invalid</td>
<td>Shared</td>
<td>Shared</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Invalid</td>
<td>(shared,=Memory)</td>
</tr>
</tbody>
</table>

Owner can update via bus invalidate operation
Owner must write back when replaced in cache

If read sourced from memory, then Private Clean
If read sourced from other cache, then Shared
Can write in cache if held private clean or dirty
Distributed Memory Machines

- Separate Memory per Processor
- Local or Remote access via memory controller
- Alternative: directory per cache that tracks state of every block in every cache
  - Which caches have a copies of block, dirty vs. clean, ...
- Prevent directory as bottleneck:
  distribute directory entries with memory, each keeping track of which Procs have copies of their blocks

Distributed Memory Cache Coherency Solution

- Directory-Based Schemes
  - Keep track of what is being shared in 1 centralized place (logically)
  - Distributed memory => distributed directory for scalability (avoids bottlenecks)
  - Send point-to-point requests to processors via network
  - Scales better than Snooping
  - Actually existed BEFORE Snooping-based schemes
Distributed Directory MPs

- 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

- 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

Directory Protocol Messages

<table>
<thead>
<tr>
<th>Message type</th>
<th>Source</th>
<th>Destination</th>
<th>Msg Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td>Write miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td>Invalidate</td>
<td>Home directory</td>
<td>Remote caches</td>
<td>A</td>
</tr>
<tr>
<td>Fetch</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td>Fetch/Invalidate</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td>Data value reply</td>
<td>Home directory</td>
<td>Local cache</td>
<td>Data</td>
</tr>
<tr>
<td>Data write-back</td>
<td>Remote cache</td>
<td>Home directory</td>
<td>A, Data</td>
</tr>
</tbody>
</table>
State Transition Diagram for an Individual Cache Block in a Directory Based System

- 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 => explicit invalidate & data fetch requests.
- Note: on a write, a cache block is bigger, so read to read the full cache.

CPU - Cache State Machine

- State machine for CPU requests for each memory block.
- Invalid state if in memory.
- CPU read hit:
- CPU write hit:
- Fetch/Invalidate:
  Send Data Write Back message to home directory.
- CPU read miss:
  Send Read Miss message to home directory.
- CPU write miss:
  Send Data Write Back message and Write Miss to home directory.
- CPU read miss:
  Send Data Write Back message to home directory.
- CPU write miss:
  Send Data Write Back message and Write Miss to home directory.
**State Transition Diagram for the Directory**

- Same states & structure as the transition diagram for an individual cache
- 2 actions: update of directory state & send msgs 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

[Diagram showing state transitions between Uncached, Shared, and Exclusive states with actions associated with each transition.]
Example Directory Protocol

- Message sent to directory causes two actions:
  - Update the directory
  - More messages to satisfy request

- Block is in Uncached state: the copy in memory is the current value; only possible requests for that block are:
  - Read miss: requesting processor sent data from memory & requestor made only sharing node; state of block made Shared.
  - Write miss: requesting processor is sent the value & becomes the Sharing node. The block is made Exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner.

- Block is Shared => the memory value is up-to-date:
  - Read miss: requesting processor is sent back the data from memory & requesting processor is added to the sharing set.
  - Write miss: requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, & Sharers is set to identity of requesting processor. The state of the block is made Exclusive.

- Block is Exclusive: current value of the block is held in the cache of the processor identified by the set Sharers (the owner) => three possible directory requests:
  - Read miss: owner processor sent data fetch message, causing state of block in owner's cache to transition to Shared and causes owner to send data to directory, where it is written to memory & sent back to requesting processor. Identity of requesting processor is added to set Sharers, which still contains the identity of the processor that was the owner (since it still has a readable copy). State is shared.
  - Data write-back: owner processor is replacing the block and hence must write it back, making memory copy up-to-date (the home directory essentially becomes the owner), the block is now Uncached, and the Sharer set is empty.
  - Write miss: block has a new owner. A message is sent to old owner causing the cache to send the value of the block to the directory from which it is sent to the requesting processor, which becomes the owner.