Cache Coherence

Suvinay Subramnian

6.823 Spring 2016
Cache Coherence Problem

» Many parallel programs communicate through shared memory
  - Shared memory is easier for programmers

» Problem: If multiple processors cache the same block, how do they ensure “correct” view of the data?
## Cache Coherence Problem

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>addi r1, accts, r3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ld 0(r3), r4</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>blt r4, r2, 6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>sub r4, r2, r4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>st r4, 0(r3)</td>
<td>400</td>
</tr>
<tr>
<td>5</td>
<td>call spew_cash</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>addi r1, accts, r3</td>
<td>400</td>
</tr>
<tr>
<td>1</td>
<td>ld 0(r3), r4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>blt r4, r2, 6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>sub r4, r2, r4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>st r4, 0(r3)</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>call spew_cash</td>
<td></td>
</tr>
</tbody>
</table>

Two processors, no caches: No problem
Cache Coherence Problem

Two processors, write-back caches: Problem!
Cache Coherence Responsibility

» Software
  - What if ISA could provide FLUSH instruction?

» Hardware
  - Simplifies software’s job
  - Common in today’s systems
Cache Coherence Strategies

Two rules:
1. Write propagation: Writes eventually become visible to other processors
2. Write serialization: Writes to same location are serialized

» Invalidation-based:
  On a write, all other caches with copies are invalidated.

» Update-based:
  On a write, all other caches with copies are updated.
Cache Coherence Strategies

» Snoopy coherence protocol
   All caches observe other caches’ actions through a shared bus(-like) interconnect.

» Directory-based coherence protocol
   A coherence directory tracks contents of caches and sends (and receives) messages to maintain coherence.

What are the tradeoffs?
Valid-Invalid Snooping Protocol

» Simple rules
- Allows multiple readers, must write through to bus
- Write-through, no write-allocate
- All caches monitor ("snoop") bus traffic
Supporting Write-Back Caches

Key idea: Add notion of “ownership”

» Mutual exclusion – when “owner” has only replica of a cache block, it may update it freely

» Sharing – multiple readers ok, but they may not write without gaining ownership
MSI Protocol

» Three states per cache-line
  - Invalid (I): Cache does not have a copy
  - Shared (S): Cache has read-only copy; clean
  - Modified (M): Cache has only copy; writable; (potentially) dirty

» Processor Actions: Read (PrRd), Write (PrWr)
» Bus Actions: BusRd, BusRdEx, BusWB
MSI Protocol

Processor-initiated transitions

Bus-initiated transitions

### Actions

<table>
<thead>
<tr>
<th>Action</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: Optimizations

» Problem: MSI suffers from frequent read-upgrade sequences
  - Leads to two bus transactions even for private blocks.

» Solution: Add exclusive (E) state
  - E: Only one copy; writable; clean
  - Cores silently transition to M on PrWr to indicate dirty
MSI Optimizations

» Problem: MESI must write-back to memory on M→S transitions
  - Why? Because protocol allows silent evicts when in S state, which might cause dirty data to be lost.
  - Write-backs may be a waste of bandwidth; eg: producer-consumer scenarios

» Solution: Add owned (O) state
  - O: shared, but dirty; only one owner; entered on M→S downgrade
  - Owner responsible for write-back on eviction
Directory Coherence Protocol

» In addition to cache states, directory maintains its coherence state, and sharer set
  - Essentially, extending memory (or next cache level) to track caching information

» States:
  - Uncached (Un): No cache has a valid copy
  - Shared (S): One or more caches in S state
  - Exclusive (Ex): One of the caches in M state

» Sharer Set:
  - Tracks which caches hold the line
Snoopy, Directory Optimizations

» Snoopy:
  - Split-transaction buses

» Directory:
  - Centralized vs distributed directory
  - Efficient sharer representation
    • Bit-vectors; limited-pointers; bloom filters etc.
Cache Coherence Races

» Problem: Transactions are not atomic

» Transient states, protocol events to handle races

» Examples...
Murphi

» Formal state verification of finite state machines

» State-space exploration: explores all reachable states

» Uses symmetry to canonicalize redundant states
Murphi Language

» Rules: Transitions between states
» Invariants and asserts: Capture protocol correctness
» Scalarsets, multi-sets: Capture symmetry
Principles

» Think of sending and receiving messages as separate events.

» At each step, think what new requests can occur
  - Messages overtaking other messages

» Two messages in the same direction implies a race
  - Consider both deliver orders
  - Often, only one node knows how to resolve a race
    (might send other nodes msgs suitably)