Cache Coherence (Continued)

Miguel Gomez-Garica

6.823 Fall 2022

Adapted from prior course offerings
Cache Coherence

» Two *necessary* conditions:
   1. Write propagation: Writes *eventually* become visible to other processors
   2. Write serialization: All processors observe writes to one location appear to happen in a consistent order

» MSI protocol provides a *sufficient* condition via single-writer multi-reader policy
   - Only one cache may have write permission at any given point in time
   - Multiple caches can have read-only permission at a given point in time
Write-back caches: MSI

» Three stable 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-initiated actions:
  - Read: needs to upgrade permission to S
  - Write: needs to upgrade permission to M
  - Evict: relinquish permissions (caused by access to a different cache line)
MSI directory states

» Uncached (Un): No cache has a valid copy

» Shared (Sh): One or more caches in S state. Must track sharers.

» Exclusive (Ex): One of the caches in M state. Must track owner.

» Does the directory need transient states?
  - Yes on downgrades/invalidations, to guarantee serialization
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- S
- I

<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>Shared Request (ShReq)</td>
</tr>
<tr>
<td>Exclusive Request (ExReq)</td>
</tr>
</tbody>
</table>
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

**Actions**

- Processor Read (PrRd)
- Processor Write (PrWr)
- Shared Request (ShReq)
- Exclusive Request (ExReq)
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- Processor Read (PrRd)
- Processor Write (PrWr)
- Shared Request (ShReq)
- Exclusive Request (ExReq)
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- Processor Read (PrRd)
- Processor Write (PrWr)
- Shared Request (ShReq)
- Exclusive Request (ExReq)
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- Processor Read (PrRd)
- Processor Write (PrWr)
- Shared Request (ShReq)
- Exclusive Request (ExReq)
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- **M**
  - PrRd / --
  - PrWr / --

- **S**
  - PrWr / ExReq
  - PrRd / --

- **I**
  - PrRd / ShReq

### 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>Shared Request (ShReq)</td>
</tr>
<tr>
<td>Exclusive Request (ExReq)</td>
</tr>
</tbody>
</table>
MSI Protocol: Caches (2/3)

Transitions initiated by directory requests:

<table>
<thead>
<tr>
<th>M</th>
<th>S</th>
<th>I</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidation Request (InvReq)</td>
</tr>
<tr>
<td>Downgrade Request (DownReq)</td>
</tr>
<tr>
<td>Invalidation Response (InvResp)</td>
</tr>
<tr>
<td>Downgrade Response (DownResp)</td>
</tr>
</tbody>
</table>
Transitions initiated by directory requests:

Actions

<table>
<thead>
<tr>
<th>Invalidation Request (InvReq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downgrade Request (DownReq)</td>
</tr>
<tr>
<td>Invalidation Response (InvResp)</td>
</tr>
<tr>
<td>Downgrade Response (DownResp)</td>
</tr>
</tbody>
</table>
Transitions initiated by directory requests:

- **M**
  - DownReq / DownResp (with data)
  - InvReq / InvResp (with data)

- **S**
- **I**

**Actions**

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidation Request (InvReq)</td>
</tr>
<tr>
<td>Downgrade Request (DownReq)</td>
</tr>
<tr>
<td>Invalidation Response (InvResp)</td>
</tr>
<tr>
<td>Downgrade Response (DownResp)</td>
</tr>
</tbody>
</table>
MSI Protocol: Caches (2/3)

Transitions initiated by directory requests:

- **M** (Memcached)
- **S** (Secondary Cache)
- **I** (Invalidation)

**Actions**

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidation Request (InvReq)</td>
</tr>
<tr>
<td>Downgrade Request (DownReq)</td>
</tr>
<tr>
<td>Invalidation Response (InvResp)</td>
</tr>
<tr>
<td>Downgrade Response (DownResp)</td>
</tr>
</tbody>
</table>
Transitions initiated by evictions:

- M
- S
- I

Actions

- Writeback Request (WbReq)
MSI Protocol: Caches (3/3)

Transitions initiated by evictions:

- **M** → **S**
  - Eviction / WbReq (with data)

**Actions**
- Writeback Request (WbReq)
MSI Protocol: Caches (3/3)

Transitions initiated by evictions:

- Eviction / WbReq (with data)
- Eviction / WbReq (without data)

Actions
Writeback Request (WbReq)
MSI Protocol: Caches

- Transitions initiated by processor accesses
- Transitions initiated by directory requests
- Transitions initiated by evictions
Transitions initiated by data requests:

- Ex
- Sh
- Un
MSI Protocol: Directory (1/2)

Transitions initiated by data requests:

- **Ex**

- **Sh**
  - \(\text{ShReq} / \text{Sharers} = \text{Sharers} + \{P\} ; \text{ShResp}\)

- **Un**
  - \(\text{ShReq} / \text{Sharers} = \{P\} ; \text{ShResp}\)
MSI Protocol: Directory (1/2)

Transitions initiated by data requests:

ExReq / Sharers = \{P\}; ExResp

Ex

Sh

ShReq / Sharers = Sharers + \{P\}; ShResp

ShReq / Sharers = \{P\}; ShResp

Un
MSI Protocol: Directory (1/2)

Transitions initiated by data requests:

- `ExReq / Sharers = {P}; ExResp`
- `ExReq / Inv(Sharers – {P}); Sharers = {P}; ExResp`
- `ShReq / Sharers = Sharers + {P}; ShResp`
- `ShReq / Sharers = {P}; ShResp`
Transitions initiated by data requests:

- **ExReq / Sharers = {P}; ExResp**
- **ShReq / Down(Sharer); Sharers = Sharer + {P}; ShResp**
- **ExReq / Inv(Sharers – {P}); Sharers = {P}; ExResp**
- **ShReq / Sharers = Sharers + {P}; ShResp**
- **ShReq / Sharers = {P}; ShResp**
Transitions initiated by writeback requests:

- Ex
- Sh
- Un
Transitions initiated by writeback requests:

Ex

WbReq / Sharers = {}; WbResp

Sh

Un
MSI Protocol: Directory (2/2)

Transitions initiated by writeback requests:

- **Ex**: WbReq / Sharers = {}; WbResp
- **Sh**: WbReq && |Sharers| > 1 /
  Sharers = Sharers - {P}; WbResp
- **Un**: 

October 25, 2021
Transitions initiated by writeback requests:

- **Ex**
  
  $\text{WbReq} \land \text{Sharers} = \{\}; \text{WbResp}$

- **Sh**
  
  $\text{WbReq} \land |\text{Sharers}| > 1$
  
  $\text{Sharers} = \text{Sharers} - \{\text{P}\}; \text{WbResp}$

- **Un**
  
  $\text{WbReq} \land |\text{Sharers}| == 1$
  
  $\text{Sharers} = \{\}; \text{WbResp}$
Optimizations

» Problem: Frequent read-upgrade sequences
  - private read-modify-write
  - Requires two bus transactions even for private blocks

» Solution: Add Exclusive (E) state
  - E: Only one copy, writable, and clean
  - Core silently updates to M upon a write to indicate dirty line.
Optimizations

» Problem: Writeback to memory upon M->S downgrade
  - Sometimes wastes bandwidth e.g. producer-consumer scenarios
  - S implicitly assumes line is clean, allowing silent evictions.

» Solution: Add Owner (O) state
  - O: Multiple copies, read-only, and dirty. Also responsible for writing back the data
  - Core enters O upon a downgrade.
Lab Task: MSI Coherence Protocol

» Implement with Murphi description language
  - Rules: Define transitions between states
  - Invariants and asserts: Capture protocol correctness

» Murphi verifier
  - Explores reachable states until it finds:
    • A violation of an invariant or assertion, or
    • A state with no possible transitions (deadlock), or
    • It has explored all reachable states and found no errors.
  - Exploits symmetry to reduce redundant states
Races

» Occur when there are multiple messages/requests in flight concerning a single cache line.

» Try to minimize the opportunity for races by waiting for previous messages before sending new ones.

» Multiple processors may concurrently initiate conflicting requests.
   - L13-25 shows one case

» If network may deliver messages out of order, the protocol must handle this. For example:
   - The directory has two messages in flight to one private cache.
   - One processor/cache has two messages in flight to the directory.

» 3-hop protocol may require you to add more handling for additional races.
Tips

» Feel free to add to or rename states and messages.
» Get a 4-hop protocol working first, before attempting 3-hop.
» Get your protocol working with ProcCount set to 2 before handling the 3-processor case.
» Write more of assertions and/or invariants.
   - Add assertions/invariants about your transient states.