Cache Coherence

Silvina Hanono Wachman
Computer Science & Artificial Intelligence Lab
M.I.T.
Cache Coherence Avoids Stale Data

- Multicores have **multiple private caches** for performance
- Need to provide the illusion of a single shared memory
- Problem:

![Diagram showing cache coherence in a multicore system](image-url)
Cache Coherence Avoids Stale Data

- Multicores have **multiple private caches** for performance
- Need to provide the illusion of a single shared memory

**Problem:**

1. **LD 0xA → 2**
Cache Coherence Avoids Stale Data

- Multicores have **multiple private caches** for performance
- Need to provide the illusion of a single shared memory
- Problem:

1. LD 0xA → 2
2. ST 3 → 0xA
Cache Coherence Avoids Stale Data

- Multicores have multiple private caches for performance
- Need to provide the illusion of a single shared memory
- Problem:

1. LD 0xA → 2
2. ST 3 → 0xA
3. LD 0xA → 2 (stale!)
Cache Coherence Avoids Stale Data

- Multicores have multiple private caches for performance
- Need to provide the illusion of a single shared memory
- Problem:

  ![Cache Diagram](image)

  - **1** LD $0xA \rightarrow 2$
  - **2** ST $3 \rightarrow 0xA$
  - **3** LD $0xA \rightarrow 2$ (stale!)

- Solution: A cache coherence protocol controls cache contents to avoid stale lines
  - e.g., invalidate core 0’s copy of A before letting core 2 write to it
Implementing Cache Coherence

- Coherence protocols must enforce two rules:
  - **Write propagation**: Writes eventually become visible to all processors
  - **Write serialization**: Writes to the same location are serialized (all processors see them in the same order)
Implementing Cache Coherence

- Coherence protocols must enforce two rules:
  - **Write propagation**: Writes eventually become visible to all processors
  - **Write serialization**: Writes to the same location are serialized (all processors see them in the same order)

- How to ensure write propagation?
  - **Write-invalidate protocols**: Invalidate all other cached copies before performing the write
  - **Write-update protocols**: Update all other cached copies after performing the write
Implementing Cache Coherence

- Coherence protocols must enforce two rules:
  - **Write propagation**: Writes eventually become visible to all processors
  - **Write serialization**: Writes to the same location are serialized (all processors see them in the same order)

- How to ensure write propagation?
  - **Write-invalidate protocols**: Invalidate all other cached copies before performing the write
  - **Write-update protocols**: Update all other cached copies after performing the write

- How to ensure write serialization?
  - **Snooping-based protocols**: All caches observe each other’s actions through a shared bus
  - **Directory-based protocols**: A coherence directory tracks contents of private caches and serializes requests
Caches watch (snoop on) bus to keep all processors’ view of memory coherent
Snooping-Based Coherence

- Bus provides serialization point
  - Broadcast, totally **ordered**
  - Each cache controller “snoops” all bus transactions
  - Controller updates state of cache in response to processor and snoop events and generates bus transactions

![Diagram of processor cache interaction](image)
Snooping-Based Coherence

- Bus provides serialization point
  - Broadcast, totally *ordered*
  - Each cache controller “snoops” all bus transactions
  - Controller updates state of cache in response to processor and snoop events and generates bus transactions

- Snoopy protocol (FSM)
  - State-transition diagram
  - Actions

**Diagram:**
- Processor ld/st
- Cache
- State, Tag, Data
- Snoop (observed bus transaction)
A Simple Protocol: Valid/Invalid (VI)

- Assume write-through caches

### Actions
- Processor Read (PrRd)
- Processor Write (PrWr)
- Bus Read (BusRd)
- Bus Write (BusWr)
Valid/Invalid Example

Main Memory

Cache

Core 0

Cache

Core 1
Valid/Invalid Example

1. LD 0xA
Valid/Invalid Example

Main Memory

BusRd 0xA

Cache

Core 0

1 LD 0xA

Cache

Core 1
Valid/Invalid Example

BusRd 0xA

Main Memory

Cache

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

Core 0

1

LD 0xA

Cache

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

Core 1
Valid/Invalid Example

1. LD 0xA
Valid/Invalid Example

1. LD 0xA

2. LD 0xA
Valid/Invalid Example

[Diagram showing the bus read (BusRd 0xA) between Main Memory and Cores 0 and 1. Core 0 has a cache entry for 0xA with state V and data 2, and Core 1 has an empty cache entry.]

1. LD 0xA
2. LD 0xA
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
Valid/Invalid Example

1. LD 0xA

Core 0

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

Core 1

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

2. LD 0xA

BusRd 0xA

Main Memory

Additional loads satisfied locally, without BusRd
Valid/Invalid Example

1. LD 0xA

2. LD 0xA

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

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>V</td>
<td>2</td>
</tr>
</tbody>
</table>
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
Valid/Invalid Example

BusWr 0xA, 3

Core 0

1. LD 0xA
2. ST 0xA
3. ST 0xA

Core 1

LD 0xA
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. LD 0xA
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. LD 0xA
Valid/Invalid Example

BusRd 0xA

Core 0
1. LD 0xA
2. ST 0xA

Core 1
3. LD 0xA
4. LD 0xA

Main Memory
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. LD 0xA

VI Problems?
Valid/Invalid Example

VI Problems? Every write updates main memory
Every write requires broadcast & snoop
Modified/Shared/Invalid (MSI) Protocol

- Allows writeback caches + satisfies writes locally

**Actions**

- Processor Read (PrRd)
- Processor Write (PrWr)
- Bus Read (BusRd)
- Bus Read Exclusive (BusRdX)
- Bus Writeback (BusWB)
MSI Example

Main Memory

Cache

Tag | State | Data
--- | --- | ---

Core 0

Cache

Tag | State | Data
--- | --- | ---

Core 1
MSI Example

Core 0

1 LD 0xA
MSI Example

![Diagram of memory system with BUSRd 0xA, Cache, Main Memory, Core 0, and Core 1.]

1. LD 0xA
MSI Example

1. LD 0xA
MSI Example

1. LD 0xA
MSI Example

1. LD 0xA

2. LD 0xA
MSI Example

BusRd 0xA

Core 0

1. LD 0xA

Cache

<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>

Core 1

2. LD 0xA

Cache

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MSI Example

1. LD 0xA

2. LD 0xA
MSI Example

1. LD 0xA

2. LD 0xA

Additional loads satisfied locally, without BusRd (like in VI)
MSI Example

1. LD 0xA

2. LD 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
MSI Example

Core 0

1. LD 0xA

2. ST 0xA

Core 1

2. LD 0xA

1. ST 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
Additional loads and stores from core 0 satisfied locally, without bus transactions (unlike in VI)
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA

Core 0

Core 1

Main Memory

BusRdX 0xA

Cache

<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>

Cache

<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>
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
MSI Example

Core 0
1. LD 0xA
2. ST 0xA

Core 1
3. LD 0xA
4. ST 0xA

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

Cache
<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>

BusWB 0xA, 3
BusRdX 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
### Cache Interventions

- MSI lets caches serve writes without updating memory, so main memory can have stale data
  - Core 0’s cache needs to supply data
  - But main memory may also respond!
- Cache must override response from main memory
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
5. LD 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
5. LD 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
5. LD 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
5. LD 0xA
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
5. LD 0xA
MSI Optimizations: Exclusive State

- Observation: Doing read-modify-write sequences on private data is common
  - What’s the problem with MSI?
MSI Optimizations: Exclusive State

- Observation: Doing read-modify-write sequences on private data is common
  - What’s the problem with MSI?

- Solution: E state (exclusive, clean)
  - If no other sharers, a read acquires line in E instead of S
  - Writes silently cause E→M (exclusive, dirty)
MESI: An Enhanced MSI protocol
increased performance for private read-write data

Each cache line has a tag

<table>
<thead>
<tr>
<th>State</th>
<th>Address tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>

M: Modified Exclusive
E: Exclusive, unmodified
S: Shared
I: Invalid
Directory-Based Coherence

- Route all coherence transactions through a directory
  - Tracks contents of private caches → No broadcasts
  - Serves as ordering point for conflicting requests → Unordered networks
A cache line contains more than one word, and cache coherence is done at line granularity.
A cache line contains more than one word, and cache coherence is done at line granularity.

| state | line addr | word0 | word1 | ... | wordN |

Suppose $P_1$ writes $\text{word}_i$ and $P_2$ writes $\text{word}_k$ and both words have the same line address.

What can happen?
Cache Coherence and False Sharing

Performance Issue #1

- A cache line contains more than one word, and cache coherence is done at line granularity

| state | line addr | word0 | word1 | ... | wordN |

- Suppose $P_1$ writes $\text{word}_i$ and $P_2$ writes $\text{word}_k$ and both words have the same line address
- What can happen?

  The line may be invalidated (ping-pong) many times unnecessarily because addresses are in the same line.
Thank you!

Next lecture: Putting it all together