Cache Coherence

- No lecture or recitation next week
- Don’t forget Lab 7 checkoff

Happy Thanksgiving!
Modern microprocessors usually have 2 to 8 cores where each core has a private cache for performance.
Multicores

Modern microprocessors usually have 2 to 8 cores where each core has a *private cache* for performance.
- Cores can be used cooperatively to speed up an application.
- Cores communicate with each other via memory.
Cache Coherence Avoids Stale Data

- Need to provide the illusion of a single shared memory even though multicores have multiple private caches
- Problem:

```
Main Memory

Cache
Cache
Cache
Cache

Core 0
Core 1
Core 2
Core 3
```
Cache Coherence Avoids Stale Data

- Need to provide the illusion of a **single shared memory** even though multicores have multiple private caches
- Problem:

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

- Need to provide the illusion of a single shared memory even though multicores have multiple private caches
- Problem:

```
[0xA] = 2
[0xA] = 3
```

1. LD 0xA \(\rightarrow\) 2
2. ST 3 \(\rightarrow\) 0xA
Cache Coherence Avoids Stale Data

- Need to provide the illusion of a single shared memory even though multicores have multiple private caches

- Problem:

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

- Need to provide the illusion of a **single shared memory** even though multicores have multiple private caches
- **Problem:**

![Diagram](https://example.com/diagram.png)

1. **LD 0xA → 2**
2. **ST 3 → 0xA**
3. **LD 0xA → 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
In a *coherent memory* all loads and stores can be placed in a global order

- multiple copies of an address in various caches can cause this property to be violated
Maintaining Coherence

- In a *coherent memory* all loads and stores can be placed in a global order
  - multiple copies of an address in various caches can cause this property to be violated

- This property can be ensured if:
Maintaining Coherence

- In a *coherent memory* all loads and stores can be placed in a global order
  - multiple copies of an address in various caches can cause this property to be violated

- This property can be ensured if:
  - Only one cache at a time has the write permission for an address
Maintaining Coherence

- In a *coherent memory* all loads and stores can be placed in a global order
  - multiple copies of an address in various caches can cause this property to be violated

- This property can be ensured if:
  - Only one cache at a time has the write permission for an address
  - No cache can have a stale copy of the data after a write to the address has been performed
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
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

![State-table and Processor actions diagram]
A Simple Protocol: Valid/Invalid (VI)

- **Assume write-through caches**

**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 Write (BusWr)</td>
</tr>
</tbody>
</table>
Valid/Invalid Example

Main Memory

Cache

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

Core 0

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

BusRd 0xA

Main Memory

Cache

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

Core 0

Core 1

1 LD 0xA
Valid/Invalid Example

BusRd 0xA

Main Memory

Cache
- Tag: 0xA
- State: V
- Data: 2

Core 0

Core 1

1 LD 0xA

Tag | State | Data
--- | --- | ---
0xA | V | 2
Valid/Invalid Example

1. LD 0xA
Valid/Invalid Example

1. LD 0xA

2. LD 0xA
Valid/Invalid Example

1. LD 0xA

2. LD 0xA
Valid/Invalid Example

Core 0

1. LD 0xA

Core 1

2. LD 0xA

Main Memory

BusRd 0xA
Valid/Invalid Example

1. LD 0xA
2. LD 0xA

Additional loads satisfied locally, without BusRd
Valid/Invalid Example

Core 0

1. LD 0xA

Core 1

2. 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

Cache

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

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

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>V</td>
<td>3</td>
</tr>
<tr>
<td>0xA</td>
<td>I</td>
<td>2</td>
</tr>
</tbody>
</table>
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

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

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

- Each line in each cache maintains MSI state:
  I - cache doesn’t contain the address
  S - cache has the address but so may other caches; hence it can only be read
  M - only this cache has the address; hence it can be read and written
  - any other cache that had this address got invalidated
VI Drawbacks: Every write updates main memory, and every write requires broadcast & snoop
MSI Protocol FSM

- VI Drawbacks: Every write updates main memory, and every write requires broadcast & snoop
- MSI: Allows writeback caches + satisfies writes locally
VI Drawbacks: Every write updates main memory, and every write requires broadcast & snoop

MSI: Allows writeback caches + satisfies writes locally
MSI Example

![MSI Example Diagram](image-url)
MSI Example

1. LD 0xA
MSI Example

BusRd 0xA

Main Memory

Cache

Core 0

1 LD 0xA

Cache

Core 1
MSI Example

1. LD 0xA
MSI Example

1. LD 0xA
MSI Example

1. LD 0xA

<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>

2. LD 0xA

<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
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

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

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

Core 0

Core 1

BusRdX 0xA
MSI Example

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
### MSI Example

**Main Memory**

**BusWB 0xA, 3**

**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>

**Core 0**

1. LD 0xA

**Core 1**

2. LD 0xA

3. ST 0xA

4. ST 0xA
MSI Example

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

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

Main Memory

<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>
<tr>
<td>0xA</td>
<td>M</td>
<td>10</td>
</tr>
</tbody>
</table>

Core 0

Core 1
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

![MSI Diagram]

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

What’s the problem with MSI?

2 bus transactions for every read-modify-write of private data.
MSI Optimizations: Exclusive State

- Observation: Doing read-modify-write sequences on private data is common
  - What’s the problem with MSI?
    
    2 bus transactions for every read-modify-write of private data.

- 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)
Each cache line has a tag

M: Modified Exclusive
E: Exclusive, unmodified
S: Shared
I: Invalid

Address tag

State bits

PrWr / --
PrRd / --

BusRd / BusWB

PrWr / BusRdX

BusRd / BusRdX

PrRd / BusRdX

PrRd / BusRd

PrRd / BusRd if other sharers

PrRd if no other sharers

BusRdX / --

BusRdX / --

BusRdX / BusWB
Directory-Based Coherence

- Route all coherence transactions through a directory
  - Tracks contents of private caches \(\rightarrow\) No broadcasts
  - Serves as ordering point for conflicting requests \(\rightarrow\) Unordered networks
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 |
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?
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: Branch Prediction