ECE 454
Computer Systems Programming
Performance Implications of Parallel Architectures

Ashvin Goel
ECE Dept, University of Toronto
http://www.eecg.toronto.edu/~ashvin
We know that we need parallelization.

But will more parallelization always yield better performance?

We will study the reasons for this behavior.
Topics

- Cache coherence
- Performance of memory operations
- Implications for software design
Cache Coherence
Modern Shared Memory Parallel Architectures

- Provide several processing elements (cores or processors)
- Provide shared memory
  - Any processor can directly reference any memory location
  - Communication occurs implicitly through loads and stores
- Cores have private caches to improve performance
Cache Coherence Problem

• With multiple cores, data is cached in multiple locations, so how do you ensure consistency?
Example 1: Coherence Problem

Thread A: Store X=3

Thread B: Load X

X = ?

Cache

Tag | Data
---|---
X  | 2

Cache

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

Processor

Shared Memory (X=2)
Example 2: Coherence Problem

Thread A: Cache

Thread B: Load X

Shared Memory (X=2)
Example 2: Coherence Problem

Thread A:

Thread B:

Load X

Read

Shared Memory (X=2)
Example 2: Coherence Problem

Thread A:

Thread B:

Load X

Read

Fill

Shared Memory (X=2)
Example 2: Coherence Problem

Thread A: Store X=3

Thread B:

Shared Memory (X=2)
Example 2: Coherence Problem

Thread A: Store X=3
Thread B:

Inconsistency!

Shared Memory (X=2)
Cache Coherence Definition

- **Cache (or memory) coherence**: the behavior of the system is equivalent to there being only a single copy of the data except for the performance benefit of the cache. [Gray and Cheriton 83]

- Cache coherence ensures that all processors have a consistent view of a single memory location (e.g., X)
  - All reads and writes to X can be put on a timeline (total order) that respects the program order of each thread

- Ensuring coherence requires hardware support
  - Called coherence protocol
MSI Coherence Protocol

- Add three (exclusive) states to each cache line:
  - **Invalid** – data is not cached
  - **Modified** – core has written to the cache line
    - Cache line is inconsistent with primary storage
    - Cache line is not shared with other cores
  - **Shared** – core has read from the cache line
    - Cache line is consistent with primary storage
    - Cache line may be shared with other cores
MSI Coherence Protocol

Thread A:

- Processor
- Cache
- State: Invalid
- Tag: -
- Data: -

Thread B:

- Processor
- Cache
- State: Invalid
- Tag: -
- Data: -

Shared Memory (X=2)
MSI Coherence Protocol

Thread A:

Thread B:

Load X

Read

Shared Memory (X=2)
MSI Coherence Protocol

Thread A:

Thread B:

Load X

Read

Fill

Shared Memory (X=2)
Thread A: Store X=3

Thread B: Load X

Invalidation

Shared Memory (X=2)

invalidates all other copies
** MSI Coherence Protocol **

**Thread A:** Store X=3

**Thread B:** Load X

Invalidates all other copies
**MSI Coherence Protocol**

**Thread A:** Store X=3

**Thread B:** Load X

---

**Processor**

**Cache**

**State**

- Modified

**Tag**

- X

**Data**

- 3

Invalidation

**Shared Memory (X,2 (stale))**
Problem with MSI

- If a core reads a value that is not cached on any core and then writes to the value, then two cache coherence requests are generated
  - A request to read the value (required)
  - A request to write the value (unnecessary invalidation request sent because the MSI protocol doesn’t know that no one else has a copy)
MESI (aka Illinois) Protocol

- Four (exclusive) states of each cache line:
  - **Invalid** – data is not cached
  - **Modified** – core has written to the cache line
    - Cache line is inconsistent with primary storage
    - Cache line is not shared with other cores
  - **Shared** – core has read from the cache line
    - Cache line is consistent with primary storage
    - Cache line **may be shared** with other cores
  - **Exclusive**: core has read from the cache line
    - Cache line is consistent with primary storage
    - Cache line **is not shared** by other cores
    - **Write to Exclusive state does not generate invalidation request**
An attempt to write to a block that is in Invalid state is called a **write miss**

- Must cache the block in Exclusive state **before** writing to it
- Generates a **read-exclusive** (or read for ownership) request
  - A read request with intent to write to the memory address
  - If other caches have copy of data, they send it, invalidate their copy
  - Completes when there are no more valid copies
- Can then perform the write and enter the **modified** state
  - This step doesn’t require invalidation request
MESI Examples

- Example 1: **load** on one core followed by **load** on another core
- Example 2: **load** on one core followed by **store** on another core
- Example 3: **store** on one core followed by **load** on another core
Example 1: MESI Coherence

Thread A:

```
Processor
Cache
State: Invalid
Tag: -
Data: -
```

Thread B:

```
Processor
Cache
State: Invalid
Tag: -
Data: -
```

Shared Memory (X=2)
Example 1: MESI Coherence

Thread A:
Load X

Thread B:

Shared Memory (X=2)
Example 1: MESI Coherence

Thread A:

Load X

Thread B:

Read

Shared Memory (X=2)
Example 1: MESI Coherence

Thread A:

Load X

- Processor
- Cache
- State: Excl
- Tag: X
- Data: 2

Thread B:

- Load X
  - Read Fill
  - Processor
  - Cache
  - State: Invalid
  - Tag: -
  - Data: -

Shared Memory (X=2)
Example 1: Mesi Coherence

Thread A:
- Load X
- Cache
  - State: Excl
  - Tag: X
  - Data: 2

Thread B:
- Load X
- Shared Memory (X=2)
- Cache
  - State: Invalid
  - Tag: -
  - Data: -
Example 1: MESI Coherence

Thread A:

- Load X

Processor

Cache

State  Tag  Data
Excl    X     2

Thread B:

- Load X

Processor

Cache

State  Tag  Data
Invalid  -    -

Read

Shared Memory (X=2)
Example 1: MESI Coherence

Thread A:
- Load X
- Data: Share X 2

Thread B:
- Load X
- Read
- Fill
- Notify Shared

Shared Memory (X=2)
Example 2: MESI Coherence

Thread A:
Thread B:

Shared Memory (X=2)
Example 2: MESI Coherence

Thread A: Thread B:

Load X

Shared Memory (X=2)

Thread A:

Thread B:

Processor

Cache

State  Tag  Data
Invalid - -

Processor

Cache

State  Tag  Data
Invalid - -

Read
Example 2: MESI Coherence

Thread A:

- Load X

Thread B:

- Read

Shared Memory (X=2)

Processor

Cache

State  Tag  Data
Invalid  -  -

State  Tag  Data
Excl.  X  2

Load X

Read

Fill
Example 2: MESI Coherence

Thread A: Store X=3
Thread B: Load X

Shared Memory (X=2)

read-exclusive invalidates all other copies
Example 2: MESI Coherence

Thread A:
- Store X=3

Thread B:
- Load X

Shared Memory (X=2)

read-exclusive invalidates all other copies
Example 2: Mesi Coherence

Thread A:
Store X=3

Thread B:
Load X

the state ‘dirty’ implies exclusiveness
Example 3: Mesi Coherence

Thread A:

Thread B:

Shared Memory (X=2)
Example 3: MESI Coherence

Thread A: Store X=5

Thread B: 

Shared Memory (X=2)
Example 3: MESI Coherence

Thread A:
- Store X=5

Thread B:
- Read

- Exclusive

Shared Memory (X=2)
Example 3: MESI Coherence

Thread A: Store X=5

Thread B:

**Processor**

**Cache**

<table>
<thead>
<tr>
<th>State</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dirty</td>
<td>X</td>
<td>5</td>
</tr>
</tbody>
</table>

Fill

**Processor**

**Cache**

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

Read-Exclusive

**Shared Memory (X=out-of-date)**
Example 3: Mesi Coherence

Thread A: Store X=5

Thread B: Load X

Shared Memory (X, 2 (out of date))
Example 3: MESI Coherence

Thread A: Store $X=5$

Thread B: Load $X$

Shared Memory ($X=\text{out-of-date}$)
Example 3: MESI Coherence

Thread A:
Store X=5

Thread B:
Load X

Shared Memory (X=out-of-date)
Example 3: MESI Coherence

Thread A:
- Store X=5

Thread B:
- Load X

Shared Memory (X=5)
Example 3: MESI Coherence

Thread A:
Store X=5

Thread B:
Load X

Shared Memory (X=5)
# MESI Permitted States, Transitions

<table>
<thead>
<tr>
<th>Local Event</th>
<th>Initial State</th>
<th>Local</th>
<th>Message</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read miss</td>
<td>I</td>
<td>I → (S, E)</td>
<td>READ</td>
<td>(S, E) → S</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M → S + WB</td>
</tr>
<tr>
<td>Read hit</td>
<td>S, E, M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write miss</td>
<td>I</td>
<td>I → M</td>
<td>READEX</td>
<td>(S,E) → I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M → I + WB</td>
</tr>
<tr>
<td>Write hit</td>
<td>S</td>
<td>S → M</td>
<td>INVALIDATE</td>
<td>S → I</td>
</tr>
<tr>
<td></td>
<td>E, M</td>
<td>E → M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance of Memory Operations
Local Caches and Memory Latencies

- Cost of accessing memory
  - Best case
    - Data is cached locally: L1 < 10 cycles (remember this)
  - Worst case
    - Data is accessed from DRAM: 136 – 355 cycles (remember this)

<table>
<thead>
<tr>
<th></th>
<th>Opteron</th>
<th>Xeon</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>L2</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>LLC</td>
<td>40</td>
<td>44</td>
</tr>
<tr>
<td>RAM</td>
<td>136</td>
<td>355</td>
</tr>
</tbody>
</table>
Interconnect Between Sockets

Cross-sockets communication can be 2-hops

(a) AMD Opteron
(b) Intel Xeon
### Latency of Remote Access: Read (cycles)

<table>
<thead>
<tr>
<th>System</th>
<th>Opteron</th>
<th>Xeon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hops</td>
<td>one hop</td>
</tr>
<tr>
<td>State</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modified</td>
<td>same die</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>one hop</td>
<td></td>
</tr>
<tr>
<td>Exclusive</td>
<td>same die</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>one hop</td>
<td></td>
</tr>
<tr>
<td>Shared</td>
<td>same die</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>one hop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>two hops</td>
<td></td>
</tr>
</tbody>
</table>

**State** is the MESI state of a cache line in a remote cache
- Local cache line state is invalid

**Cross-socket communication is expensive!**
- Xeon: cross-socket latency is 4-7.5 larger than within socket
- Opteron: cross-socket latency even larger than RAM
Latency of Remote Access: Write (cycles)

- **State** is the MESI state of a cache line in a remote cache
  - Local cache line state is invalid

- **Cross-socket communication is expensive!**
  - Comparable or more expensive than DRAM accesses

---

**Table:**

<table>
<thead>
<tr>
<th>System</th>
<th>Opteron</th>
<th>Xeon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>one hop</td>
<td>two hops</td>
</tr>
<tr>
<td>Modified</td>
<td>83</td>
<td>191</td>
</tr>
<tr>
<td>Exclusive</td>
<td>83</td>
<td>191</td>
</tr>
<tr>
<td>Shared</td>
<td>246</td>
<td>286</td>
</tr>
</tbody>
</table>
Implications for Software Design
False Sharing

Thread A: Store X=5

Thread B:

```
<table>
<thead>
<tr>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
</tr>
<tr>
<td>State</td>
</tr>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>Data</td>
</tr>
</tbody>
</table>

Read-Exclusive

Shared Memory (X=0,Y=0)

X and Y are on the same cache line
False Sharing

Thread A:

Store X=5

Processor

Cache

State  Tag  Data
Dirty  X,Y  5,0

Fill

Thread B:

Read-Exclusive

Processor

Cache

State  Tag  Data
-,,-  -,,-

Shared Memory (X,Y=out-of-date)
False Sharing

Thread A:
Store X=5

Thread B:
Store Y=2

Invalidation

Shared Memory (X,Y=out-of-date)
False Sharing

Thread A: Store X=5

Thread B: Store Y=2

Update

Invalidation

Read-Exclusive

Shared Memory (X=5, Y=0)
False Sharing

Thread A:

- Store X = 5

Thread B:

- Store Y = 2

Shared Memory (X,Y = out-of-date)
False Sharing

Thread A:
- Store X = 5

Thread B:
- Store Y = 2

Shared Memory (X,Y = out-of-date)
False Sharing

Thread A:
while(1)
  Store X=5

Thread B:
while(1)
  Store Y=2

X,Y cache line will ping-pong back & forth
False Sharing Summary

- False sharing
  - Threads on different cores access unrelated objects
  - Objects are located in same cache block
  - Block will ping-pong between caches on different cores

- Avoid false sharing by careful data arrangement
  - Ensure that unrelated elements are mapped to separate blocks
  - E.g., insert padding (unused data) between shared items
  - Whose responsibility is it? compiler? malloc? programmer?
Implications for Programmers

- Cache coherence is expensive (more than you thought)
  - Avoid unnecessary sharing (e.g., false sharing)

- Crossing processors/sockets is a killer
  - Can be slower than running the same program on single core!
  - Pthreads provides CPU affinity mask
    - Pin cooperative threads on cores within the same die

- Next, we will see other implications of modern architectures on software design