CSC2/458 Parallel and Distributed Systems
Parallel Memory Systems: Coherence

Sreepathi Pai
February 06, 2018

URCS
Outline

Introduction to Parallel Memory Systems

Memory Systems in Parallel Processors

Coherence

Implementations in Hardware

Implications for Parallel Programs
Introduction to Parallel Memory Systems

Memory Systems in Parallel Processors

Coherence

Implementations in Hardware

Implications for Parallel Programs
• Memory is accessed through loads/stores
• Memory contents have addresses
• Smallest unit of access varies across machines
  • Usually 8 bits (i.e. 1 byte)
• Some machines have other correctness constraints
  • e.g. alignment
  • x86 has very few correctness constraints
Memory in most machines today

- Memory hierarchy
- Multiple levels of *cache* memory
  - pron. *cash*
- Multiple loads/stores can be in flight at same time
  - Called memory-level parallelism (MLP)
  - Stall-on-use, not stall-on-issue
Caches

- Caches are faster than main memory
  - Closer to pipeline
  - Smaller than main memory
  - Usually, SRAM instead of DRAM (main memory)

- Caches contain copies of data in main memory
  - If address requested by load/store exists in cache: *hit*
  - If address does not exist in cache: *miss*

- Addresses that hit can be satisfied from the cache
- Addresses that miss are forwarded to next level of memory hierarchy
  - Forwarding continues until found
  - What is the last level?
Internal Organization of Caches

- Unit of data organization in caches: Line
  - Line size may vary from 64 to 128 bytes across CPU models
  - Each line is a non-overlapping chunk of main memory
- Each line in cache contains:
  - State (e.g. valid or invalid)
  - Tag (part of address)
  - Data

<table>
<thead>
<tr>
<th>state</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>valid</td>
<td>0xdae</td>
<td>hello world</td>
</tr>
</tbody>
</table>
Outline

Introduction to Parallel Memory Systems

Memory Systems in Parallel Processors

Coherence

Implementations in Hardware

Implications for Parallel Programs
Symmetric Multiprocessors (SMPs)

- Each processor occupies one “socket”, and are identical
- All processors share same main memory
  - Known as Shared Memory Multiprocessors
- Not all levels of hierarchy are shared
  - Caches are private
- Not shown: interconnect between processors
- Superseded by Chip Multiprocessors
Chip Multiprocessors (CMPs)

- Each socket contains multiple cores (all identical)
- All processors share same main memory
- Some levels of cache can also be shared
  - Some still private
Reads and writes in xMPs - reads/reads

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Reads and writes in xMPs - reads/reads

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Reads and writes in xMPs - read/writes

processor/core

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oxdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

processor/core

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oxdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oxdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Reads and writes in xMPs - write/write

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**WRITE Z to 0xdae...**

**WRITE Y to 0xdae...**

<table>
<thead>
<tr>
<th>cc</th>
<th>tags</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xdae...</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Memory**

| 0xdae... | X    |      |
Outline

Introduction to Parallel Memory Systems

Memory Systems in Parallel Processors

Coherence

Implementations in Hardware

Implications for Parallel Programs
The problem of coherence

- Multiple copies of the same address exist in the memory hierarchy
- How do we keep all the copies the same?
- How do we resolve ordering of writes to the same address?

Usually resolved through a coherence protocol.
Coherence Protocols

- Can be transparent (in hardware)
- You might need to implement one in software
  - if you’re creating your own caches
- Basic idea: every read and write needs to participate in a “coherence protocol”
  - Usually a finite state machine (FSM)
  - Each line in the cache has a state associated with it
- Reads, writes and cache evictions in the coherence domain may change the line’s coherence state
  - Coherence domain can consist other CPUs, I/O devices, etc.
- State determines which actions (reads/writes/evictions) are valid
  - Validity conditions?
States in MESI

- Modified: line contains modified data
- Exclusive: line is not shared
- Shared: line is shared read-only
- Invalid: line contains no data
Solid lines – actions by this processor

Dashed lines – actions by other processor

Many transitions not shown – e.g. INVALID from SHARED, EXCLUSIVE on cache line replacement
The MESI Protocol (Simplified)

- Every cache line begins in INVALID state
- On a read, the cache line is put into:
  - EXCLUSIVE: if it was read from memory
  - SHARED: if it was read from another copy
- On a write, line is moved to MODIFIED state
  - If it was previously SHARED, all other copies are INVALIDATED
  - It will eventually be written back to memory
MESI protocol test

- How many concurrent readers does MESI allow?
- How many concurrent writers does MESI allow?
• How does the MESI protocol order concurrent writers?
Outline

Introduction to Parallel Memory Systems

Memory Systems in Parallel Processors

Coherence

Implementations in Hardware

Implications for Parallel Programs
Snoop Protocols

- Requires a shared bus among all processors
- All requests to read/write are broadcast on the bus
- All processors “snoop”/listen to memory requests
- If a processor has a copy in EXCLUSIVE/SHARED/MODIFIED state:
  - It responds with a copy of its data
  - Moves its line to SHARED
- Processors broadcast “INVALIDATE” to all processors before writing
  - Must wait for acknowledgements
Directory-based Protocols

- Requires a shared structure called “directory”
- Directory tracks contents of every cache in the system
  - Addresses only
- Caches talk to directory only
- Directory send messages only to caches that contain affected data
- Used in systems with large number of processors
  - $> 8$
- Implementation need NOT be a centralized structure
Summary of Cache Coherence

- Reads and writes to shared data involve communication with other processors
- Expensive
- Possible Serialization bottleneck
Introduction to Parallel Memory Systems

Memory Systems in Parallel Processors

Coherence

Implementations in Hardware

Implications for Parallel Programs
Variables that are read/written by multiple threads are called shared variables.
```c
int *a;

T0
while(*a < 1000) {
    *a = *a + 1;
}

T1
while(1) {
    printf("%d\n", *a);
}
```
volatile int *a;

T0
while(*a < 1000)
  *a = *a + 1;

T1
while(1)
  printf("%d\n", *a);
int sums[NTHREADS];

    Tx

for(...)
    sums[x] += a[...];
s\text{ums}[] \text{ occupies a single cache line.}
• No thread shares data with another thread
• However, thread data resides within the same cache line
• Coherence operates at cache-line granularity
• Every write to the cache line will potentially be serialized
Summary

- Memory locations may be stored in registers by compiler
  - Will not participate in cache coherence
- Data layout may cause inadvertent conflicts with each other
  - One solution: Privatize and then merge