Last Time in Lecture 12

• Reviewed store policies and cache read/write policies
  • Write through vs. write back
  • Write allocate vs. write no allocate

• Shared memory multiprocessor cache coherence
  • Snoopy protocols: MSI, MESI
  • Intervention
  • False Sharing
Review: Cache Coherence vs. Memory Consistency

For a shared memory machine, the memory consistency model defines the architecturally visible behavior of its memory system. Consistency definitions provide rules about loads and stores (or memory reads and writes) and how they act upon memory. As part of supporting a memory consistency model, many machines also provide cache coherence protocols that ensure that multiple cached copies of data are kept up-to-date.

~”A Primer on Memory Consistency and Cache Coherence”, D. J. Sorin, M. D. Hill, and D. A. Wood
Cache Coherence: Directory Protocol
Scalable Approach: Directories

- Every memory line has associated directory information
  - keeps track of copies of cached lines and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions

- Many alternatives for organizing directory information
Assumptions: Reliable network, FIFO message delivery between any given source-destination pair.
Cache States

For each cache line, there are 4 possible states:

- C-invalid (= Nothing): The accessed data is not resident in the cache.
- C-shared (= Sh): The accessed data is resident in the cache, and possibly also cached at other sites. The data in memory is valid.
- C-modified (= Ex): The accessed data is exclusively resident in this cache, and has been modified. Memory does not have the most up-to-date data.
- C-transient (= Pending): The accessed data is in a transient state (for example, the site has just issued a protocol request, but has not received the corresponding protocol reply).
Home directory states

- For each memory line, there are 4 possible states:
  - R(dir): The memory line is shared by the sites specified in dir (dir is a set of sites). The data in memory is valid in this state. If dir is empty (i.e., dir = ε), the memory line is not cached by any site.
  - W(id): The memory line is exclusively cached at site id, and has been modified at that site. Memory does not have the most up-to-date data.
  - TR(dir): The memory line is in a transient state waiting for the acknowledgements to the invalidation requests that the home site has issued.
  - TW(id): The memory line is in a transient state waiting for a line exclusively cached at site id (i.e., in C-modified state) to make the memory line at the home site up-to-date.
# Directory Protocol Messages

<table>
<thead>
<tr>
<th>Message type</th>
<th>Source</th>
<th>Destination</th>
<th>Msg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Processor P reads data at address A; send data and make P a read sharer</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Processor P writes data at address A; send data and make P the exclusive owner</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalidate</td>
<td>Home directory</td>
<td>Remote caches</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Invalidate a shared copy at address A.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Fetch the block at address A and send it to its home directory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch/Invalidate</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Fetch the block at address A and send it to its home directory; invalidate the block in the cache</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data value reply</td>
<td>Home directory</td>
<td>Local cache</td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Return a data value from the home memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data write-back</td>
<td>Remote cache</td>
<td>Home directory</td>
<td>A, Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Write-back a data value for address A</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example Directory Protocol

- Message sent to directory causes two actions:
  - Update the directory
  - More messages to satisfy request

- Block is in **Uncached** state: the copy in memory is the current value; only possible requests for that block are:
  - **Read miss**: requesting processor sent data from memory & requestor made only sharing node; state of block made Shared.
  - **Write miss**: requesting processor is sent the value & becomes the Sharing node. The block is made Exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner.

- Block is **Shared** => the memory value is up-to-date:
  - **Read miss**: requesting processor is sent back the data from memory & requesting processor is added to the sharing set.
  - **Write miss**: requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, & Sharers is set to identity of requesting processor. The state of the block is made Exclusive.
Example Directory Protocol

• Block is **Exclusive**: current value of the block is held in the cache of the processor identified by the set Sharers (the owner) => three possible directory requests:
  
  – **Read miss**: owner processor sent data fetch message, which causes state of block in owner’s cache to transition to Shared and causes owner to send data to directory, where it is written to memory & sent back to requesting processor. Identity of requesting processor is added to set Sharers, which still contains the identity of the processor that was the owner (since it still has a readable copy).
  
  – **Data write-back**: owner processor is replacing the block and hence must write it back. This makes the memory copy up-to-date (the home directory essentially becomes the owner), the block is now uncached, and the Sharer set is empty.
  
  – **Write miss**: block has a new owner. A message is sent to old owner causing the cache to send the value of the block to the directory from which it is sent to the requesting processor, which becomes the new owner. Sharers is set to identity of new owner, and state of block is made Exclusive.

Dave Patterson,
CS252, Fall 1996
## Example

<table>
<thead>
<tr>
<th>Step</th>
<th>P1</th>
<th>P2</th>
<th>Bus</th>
<th>Directory</th>
<th>Memo</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: Write 10 to A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1: Read A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Read A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Write 20 to A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Write 40 to A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A1 and A2 map to the same cache block
## Example

<table>
<thead>
<tr>
<th>Step</th>
<th>( P1 )</th>
<th>( P2 )</th>
<th>Bus</th>
<th>Directory</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P1: ) Write 10 to A1</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
<td>WrMs</td>
<td>P1</td>
</tr>
<tr>
<td>( P1: ) Read A1</td>
<td></td>
<td>DaRp</td>
<td></td>
<td>P1</td>
<td>A1</td>
</tr>
<tr>
<td>( P2: ) Read A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P2: ) Write 20 to A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P2: ) Write 40 to A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A1 and A2 map to the same cache block
## Example

<table>
<thead>
<tr>
<th>Step</th>
<th>P1</th>
<th>P2</th>
<th>Bus</th>
<th>Directory</th>
<th>Memo</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: Write 10 to A1</td>
<td></td>
<td></td>
<td></td>
<td>WrMs P1 A1</td>
<td>Ex {P1}</td>
</tr>
<tr>
<td></td>
<td>Excl. A1 10</td>
<td></td>
<td></td>
<td>DaR P1 A1 0</td>
<td></td>
</tr>
<tr>
<td>P1: Read A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Read A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Write 20 to A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Write 40 to A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A1 and A2 map to the same cache block

Dave Patterson,
CS252, Fall 1996

DAP.F96 45
Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: Write 10 to A1</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>WrMs</td>
<td>P1</td>
<td>A1</td>
<td>A1</td>
<td>Ex</td>
</tr>
<tr>
<td>P1: Read A1</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>DaRp</td>
<td>P1</td>
<td>A1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>P2: Read A1</td>
<td>Shar.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>RdMs</td>
<td>P2</td>
<td>A1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Write 20 to A1</td>
<td>Shar.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>Ftch</td>
<td>P1</td>
<td>A1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DaRp</td>
<td>P2</td>
<td>A1</td>
<td>10</td>
<td>A1</td>
</tr>
<tr>
<td>P2: Write 40 to A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

A1 and A2 map to the same cache block

Dave Patterson, CS252, Fall 1996
### Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: Read A1</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Read A1</td>
<td>Shar.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>RdMs</td>
<td>P2</td>
<td>A1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Write 40 to A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A1 Excl.</td>
<td>{P2}</td>
<td></td>
</tr>
</tbody>
</table>

A1 and A2 map to the same cache block

Dave Patterson, CS252, Fall 1996

DAP.F96 47
### Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: Write 10 to A1</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>WrMs</td>
<td>P1</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>P1: Read A1</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: Read A1</td>
<td>Shar.</td>
<td>A1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>RdMs</td>
<td>P2</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>P2: Write 20 to A1</td>
<td>Excl.</td>
<td>A1</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>WrMs</td>
<td>P2</td>
<td>A1</td>
<td>10</td>
</tr>
<tr>
<td>P2: Write 40 to A2</td>
<td>Excl.</td>
<td>A2</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td>WrBk</td>
<td>P2</td>
<td>A1</td>
<td>0</td>
</tr>
</tbody>
</table>

A1 and A2 map to the same cache block

Dave Patterson,
CS252, Fall 1996

DAP.F96 48
Read miss, to uncached or shared line

1. Load request at head of CPU->Cache queue.
2. Load misses in cache.
3. Send ShReq message to directory.
4. Message received at directory controller.
5. Access state and directory for line. Line’s state is R, with zero or more sharers.
6. Update directory by setting bit for new processor sharer.
7. Send ShRep message with contents of cache line.
8. ShRep arrives at cache.
9. Update cache tag and data and return load data to CPU.
Write miss, to read shared line

1. Store request at head of CPU->Cache queue.
2. Store misses in cache.
3. Send ExReq message to directory.
4. ExReq message received at directory controller.
5. Access state and directory for line. Line’s state is R, with some set of sharers.
6. Send one InvReq message to each sharer.
7. InvReq arrives at cache.
8. Invalidation cache line. Send InvRep to directory.
10. When no more sharers, send ExRep to cache.
11. ExRep arrives at cache.
12. Update cache tag and data, then store data from CPU.

Multiple sharers
Concurrency Management

- Protocol would be easy to design if only one transaction in flight across entire system
- But, want greater throughput and don’t want to have to coordinate across entire system
- Great complexity in managing multiple outstanding concurrent transactions to cache lines
  - Can have multiple requests in flight to same cache line!
Multithreading: Intro to MT and SMT
Multithreading

- Difficult to continue to extract instruction-level parallelism (ILP) from a single sequential thread of control
- Many workloads can make use of thread-level parallelism (TLP)
  - TLP from multiprogramming (run independent sequential jobs)
  - TLP from multithreaded applications (run one job faster using parallel threads)
- Multithreading uses TLP to improve utilization of a single processor
Multithreading

How can we guarantee no dependencies between instructions in a pipeline?
One way is to interleave execution of instructions from different program threads on same pipeline

Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

T1: LD x1, 0(x2)
T2: ADD x7, x1, x4
T3: XORI x5, x4, 12
T4: SD 0(x7), x5

Prior instruction in a thread always completes write-back before next instruction in same thread reads register file
CDC 6600 Peripheral Processors
(Cray, 1964)

- First multithreaded hardware
- 10 “virtual” I/O processors
- Fixed interleave on simple pipeline
- Pipeline has 100ns cycle time
- Each virtual processor executes one instruction every 1000ns
- Accumulator-based instruction set to reduce processor state
Simple Multithreaded Pipeline

- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

Thread select

PC

+1

2

I$

IR

GPR1

X

Y

D$
Multithreading Costs

- Each thread requires its own user state
  - PC
  - GPRs

- Also, needs its own system state
  - Virtual-memory page-table-base register
  - Exception-handling registers

- Other overheads:
  - Additional cache/TLB conflicts from competing threads
  - (or add larger cache/TLB capacity)
  - More OS overhead to schedule more threads (where do all these threads come from?)
Thread Scheduling Policies

- **Fixed interleave** *(CDC 6600 PPUs, 1964)*
  - Each of N threads executes one instruction every N cycles
  - If thread not ready to go in its slot, insert pipeline bubble

- **Software-controlled interleave** *(TI ASC PPUs, 1971)*
  - OS allocates S pipeline slots amongst N threads
  - Hardware performs fixed interleave over S slots, executing whichever thread is in that slot

- **Hardware-controlled thread scheduling** *(HEP, 1982)*
  - Hardware keeps track of which threads are ready to go
  - Picks next thread to execute based on hardware priority scheme
Issue Slots:
Vertical vs. Horizontal Waste


Figure 1: Empty issue slots can be defined as either vertical waste or horizontal waste. Vertical waste is introduced when the processor issues no instructions in a cycle, horizontal waste when not all issue slots can be filled in a cycle. Superscalar execution (as opposed to single-issue execution) both introduces horizontal waste and increases the amount of vertical waste.
Simultaneous Multithreading (SMT) for OoO Superscalars

- Techniques presented so far have all been “vertical” multithreading where each pipeline stage works on one thread at a time
- SMT uses fine-grain control already present inside an OoO superscalar to allow instructions from multiple threads to enter execution on same clock cycle. Gives better utilization of machine resources.
For most apps, most execution units lie idle in an OoO superscalar

Superscalar Machine Efficiency

- **Issue width**
  - **Instruction issue**
  - **Completely idle cycle** (vertical waste)
  - **Partially filled cycle, i.e., IPC < 4** (horizontal waste)
Vertical Multithreading

Cycle-by-cycle interleaving removes vertical waste, but leaves some horizontal waste.
What is the effect of splitting into multiple processors?
- reduces horizontal waste,
- leaves some vertical waste, and
- puts upper limit on peak throughput of each thread.
Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]

- Interleave multiple threads to multiple issue slots with no restrictions
SMT adaptation to parallelism type

For regions with high thread-level parallelism (TLP) entire machine width is shared by all threads

For regions with low thread-level parallelism (TLP) entire machine width is available for instruction-level parallelism (ILP)
Multithreaded Design Discussion

- Want to build a multithreaded processor, how should each component be changed and what are the tradeoffs?
  - L1 caches (instruction and data)
  - L2 caches
  - Branch predictor
  - TLB
  - Physical register file
Summary: Multithreaded Categories

- **Superscalar**
- **Fine-Grained**
- **Coarse-Grained**
- **Multiprocessing**
- **Simultaneous Multithreading**

- **Thread 1**
- **Thread 2**
- **Thread 3**
- **Thread 4**
- **Thread 5**
- **Idle slot**
Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Krste Asanovic (UCB)
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)