

CMSC 411
Computer Systems Architecture
Lecture 19
Multiprocessors 1

Outline

- Coherence
- Write Consistency
- Snooping
- Building Blocks
- Snooping protocols and examples
- Coherence traffic and Performance on MP
- Directory-based protocols and examples
- Conclusion

CMSC 411 - 19 (taken from Patterson, Subraman, others)

2

Challenges of Parallel Processing

1. Application parallelism \Rightarrow primarily via new algorithms that have better parallel performance
 2. Long remote latency impact \Rightarrow both by architect and by the programmer
- For example, reduce frequency of remote accesses either by
 - Caching shared data (HW)
 - Restructuring the data layout to make more accesses local (SW)
 - Start now on HW to help latency via caches

CMSC 411 - 19 (taken from Patterson, Subraman, others)

3

Symmetric Shared-Memory Architectures

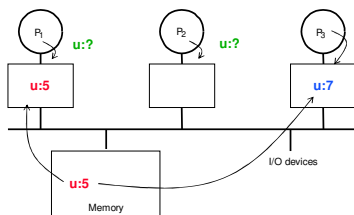
- From multiple boards on a shared bus to multiple processors inside a single chip
- Caches both
 - Private data that is used by a single processor
 - Shared data that is used by multiple processors
- Caching shared data reduces
 - Latency to shared data,
 - Memory bandwidth for shared data, and
 - Interconnect bandwidth

\Rightarrow But **cache coherence problem**

CMSC 411 - 19 (taken from Patterson, Subraman, others)

4

Example Cache Coherence Problem



- Processors see different values for **u** after P_3 's write
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value when
 - » Processes accessing main memory may see **stale** value
- Unacceptable for programming, and it happens frequently!

CMSC 411 - 19 (taken from Patterson, Subraman, others)

5

Example

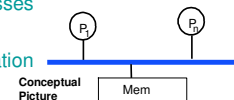
```

P1 _____ P2 _____
/*Assume initial value of A and flag is 0*/
A = 1;           while (flag == 0); /*spin idly*/
flag = 1;        print A;
  
```

- Intuition not guaranteed by coherence
- Expect memory to respect order between accesses to *different* locations issued by a given process
 - To preserve orders among accesses to same location by different processes

• Coherence is not enough!

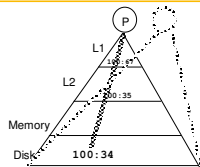
– Pertains only to single location



CMSC 411 - 19 (taken from Patterson, Subraman, others)

6

Intuitive Shared Memory Model



- **Reading an address should return the last value written to that address**
 - Easy in uniprocessors, except for I/O

- Too vague and simplistic; 2 issues
 1. **Coherence** defines **values** returned by a read
 2. **Consistency** determines **when** a written value will be returned by a read
- Coherence defines behavior to **same location**, Consistency defines behavior to **other locations**

CMSC 411 - 21 (lecture 10a: Peterson, Sussman, others)

7

Defining Coherent Memory System

1. **Preserve Program Order**: A read by processor P from location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P
2. **Coherent view of memory**: Read by a processor to location X that follows a write by **another processor** to X returns the written value if the read and write **are sufficiently separated in time** and no other writes to X occur between the two accesses
3. **Write serialization**: 2 writes to same location by any 2 processors are seen in the same order by **all** processors
 - For example, if the values 1 and then 2 are written to a location by different processors (or same), processors can never read the value of the location as 2 and then later read it as 1

CMSC 411 - 21 (lecture 10a: Peterson, Sussman, others)

8

Write Consistency

- For now assume
 1. A write does not complete (and allow the next write to occur) until all processors have seen the effect of that write
 2. The processor does not change the order of any write with respect to any other memory access
 - ⇒ If a processor writes location A then location B, any processor that sees the new value of B must also see the new value of A
- These restrictions allow the processor to reorder reads, but forces the processor to finish writes in program order

CMSC 411 - 21 (lecture 10a: Peterson, Sussman, others)

9

Basic Schemes for Enforcing Coherence

- Program running on multiple processors will normally have copies of the same data in several caches
 - Unlike I/O, where it is rare
- Rather than trying to avoid sharing in SW, SMPs use a HW protocol to maintain coherent caches
 - Migration and Replication key to performance on shared data

CMSC 411 - 21 (lecture 10a: Peterson, Sussman, others)

10

Basic Schemes for Enforcing Coherence

- **Migration** - data can be moved to a local cache and used there in a transparent fashion
 - Reduces both latency to access shared data that is allocated remotely and bandwidth demand on the shared memory
- **Replication** – for reading shared data simultaneously, since caches make a copy of data in local cache
 - Reduces both latency of access and contention for reading shared data

CMSC 411 - 21 (lecture 10a: Peterson, Sussman, others)

11

Outline

- Review
- Coherence
- Write Consistency
- Snooping
- Building Blocks
- Snooping protocols and examples
- Coherence traffic and Performance on MP
- Directory-based protocols and examples

CMSC 411 - 21 (lecture 10a: Peterson, Sussman, others)

12

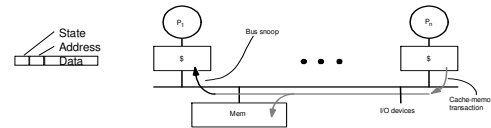
2 Classes of Cache Coherence Protocols

1. **Directory based** — Sharing status of a block of physical memory is kept in just one location, the **directory**
2. **Snooping** — Every cache with a copy of data block also has a copy of sharing status of block, but no centralized state is kept
 - All caches are accessible via some broadcast medium (a bus or switch)
 - All cache controllers monitor or **snoop** on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access

CMSC 411 - 21 (lecture from Patterson, Subraman, others)

13

Snoopy Cache-Coherence Protocols

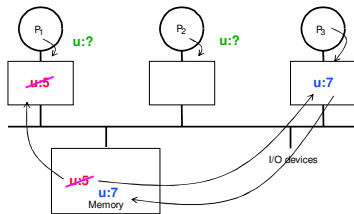


- Cache Controller “**snoops**” all transactions on the shared medium (bus or switch)
 - **Relevant transaction** if for a block it contains
 - Take action to ensure coherence
 - » invalidate, update, or supply value
 - Depends on state of the block and the protocol
- Either get exclusive access before write via write invalidate or update **all** copies on write

CMSC 411 - 21 (lecture from Patterson, Subraman, others)

14

Example: Write-through Invalidate



- Write update uses more broadcast medium BW
⇒ all recent MPUs use write invalidate

CMSC 411 - 21 (lecture from Patterson, Subraman, others)

15

Architectural Building Blocks

- Cache block state transition diagram
 - Finite state machine (FSM) specifying how disposition of block changes
 - » invalid, valid, exclusive
- Broadcast Medium Transactions (e.g., bus)
 - Fundamental system design abstraction
 - Logically single set of wires connect several devices
 - Protocol: arbitration, command/addr, data
 - **Every device observes every transaction**

CMSC 411 - 21 (lecture from Patterson, Subraman, others)

16

Architectural Building Blocks (cont.)

- Broadcast medium enforces serialization of read or write accesses ⇒ Write serialization
 - 1st processor to get medium invalidates others copies
 - Implies cannot complete write until it obtains bus
 - All coherence schemes require serializing accesses to same cache block
- Also need to find up-to-date copy of cache block

CMSC 411 - 21 (lecture from Patterson, Subraman, others)

17

Locate up-to-date copy of data

- Write-through (WT)
 - Get up-to-date copy from memory
 - Simpler if enough memory bandwidth
- Write-back (WB)
 - Harder, since most recent copy can be in a cache

CMSC 411 - 21 (lecture from Patterson, Subraman, others)

18

Locate up-to-date copy of data (cont.)

- Can use same snooping mechanism
 1. Snoop every address placed on the bus
 2. If a processor has dirty copy of requested cache block, it provides it in response to a read request and aborts the memory access
 - Complexity comes from retrieving cache block from cache, which can take longer than retrieving it from memory
- Write-back needs lower memory bandwidth
 - ⇒ Support larger numbers of faster processors
 - ⇒ Most multiprocessors use write-back

CMSC 411 - 21 (taken from Patterson, Sussman, others)

19

Cache Resources for WB Snooping

- Normal cache tags can be used for snooping
- Valid bit per block makes invalidation easy
- Read misses easy since rely on snooping
- Writes ⇒ Need to know whether any other copies of the block are cached
 - No other copies ⇒ No need to place write on bus for WB
 - Other copies ⇒ Need to place invalidate on bus

CMSC 411 - 21 (taken from Patterson, Sussman, others)

20

Cache Resources for WB Snooping

- To track whether a cache block is shared, add extra state bit associated with each cache block, like valid bit and dirty bit, **shared** bit
 - Write to Shared block ⇒ Need to place invalidate on bus and mark cache block as private (if an option)
 - No further invalidations will be sent for that block
 - This processor called **owner** of cache block
 - Owner then changes state from shared to unshared (or exclusive)

CMSC 411 - 21 (taken from Patterson, Sussman, others)

21

Cache behavior in response to bus

- **Every** bus transaction must check cache-address tags
 - Could potentially interfere with processor cache accesses
- A way to reduce interference is to duplicate tags
 - One set for caches access, one set for bus accesses

CMSC 411 - 21 (taken from Patterson, Sussman, others)

22

Cache behavior in response to bus (cont.)

- Another way to reduce interference is to use L2 tags
 - Since L2 less heavily used than L1
 - ⇒ Every entry in L1 cache must be present in the L2 cache, called the **inclusion property**
 - If Snoop gets a hit in L2 cache, then it must arbitrate for the L1 cache to update the state and possibly retrieve the data, which usually requires a stall of the processor

CMSC 411 - 21 (taken from Patterson, Sussman, others)

23

Example Snooping Protocol

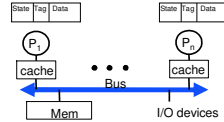
- Snooping coherence protocol is usually implemented by incorporating a finite-state controller in each node (cache)
- Logically, think of a separate controller associated with each cache block
 - That is, snooping operations or cache requests for different blocks can proceed independently
- In real implementations, a single controller allows multiple operations to distinct blocks to proceed in interleaved fashion
 - Meaning one operation may be initiated before another is completed, even though only one cache access or one bus access is allowed at a time

CMSC 411 - 21 (taken from Patterson, Sussman, others)

24

Write-through Invalidate Protocol

- 2 states per block in each cache
 - As in uniprocessor
 - State of a block is a p-vector of states
 - Hardware state bits associated with blocks that are in the cache
 - Other blocks can be seen as being in invalid (not-present) state in that cache
- Writes invalidate all other cache copies
 - Can have multiple simultaneous readers of block, but write invalidates them



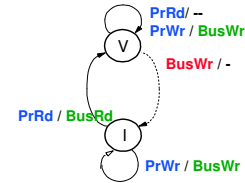
CMSC 411 - 22 (lecture from Patterson, Sussman, others)

25

The Example

Format: event / action

PrRd: Processor Read
PrWr: Processor Write
BusRd: Bus Read
BusWr: Bus Write



Single-writer, multiple-reader (SWMR)

CMSC 411 - 22 (lecture from Patterson, Sussman, others)

26

Is Example 2-state Protocol Coherent?

- Memory is **coherent** iff:
 - A read by P to location X that follows a write by P to X, w/ no intervening writes, returns written value.
 - A read by P to location X that follows a write by Q to X, w/ no intervening writes, and the read and write sufficiently separated, returns written value.
- Writes to same location are serialized
 - writes to same location by distinct processors seen in same order by all other processors

CMSC 411 - 22 (lecture from Patterson, Sussman, others)

27

Is Example 2-state Protocol Coherent?

- Assume bus transactions and memory ops atomic, and a one-level cache
 - All phases of one bus transaction complete before next one starts
 - Processor waits for memory operation to complete before issuing next
 - With one-level cache, assume invalidations applied during bus transaction

CMSC 411 - 22 (lecture from Patterson, Sussman, others)

28

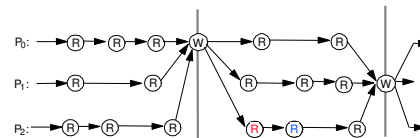
Is Example 2-state Protocol Coherent?

- Processors only observe state of memory through reads...
- Writes only observable by other processors if on bus...
- All writes go to bus! (in this example protocol, not all others)
 - Writes **serialized** by order in which they appear on bus (bus order)
 - Invalidations applied to caches in bus order
- How to insert reads in this order?
 - Important since processors see writes through reads, so determines whether write serialization is satisfied
 - But read hits may happen independently and do not appear on bus or enter directly in bus order

CMSC 411 - 22 (lecture from Patterson, Sussman, others)

29

Ordering



- Writes establish a partial order
- Doesn't constrain ordering of reads, though shared-medium (bus) will order read misses too
 - Any order among reads between writes is fine
- Writes serialized, reads and writes not interchanged, so coherent!

CMSC 411 - 22 (lecture from Patterson, Sussman, others)

30