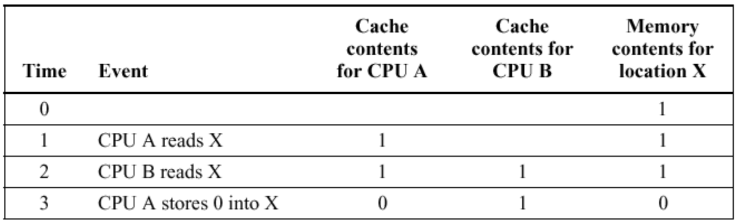
Multiprocessor Cache Coherency

CS448

What is Cache Coherence?

- Two processors can have two different values for the same memory location



Time	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 into X	0	1	0

Write Through Cache

Terminology

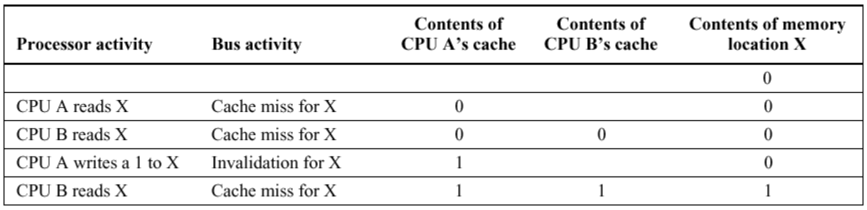
- Coherence
  - – Defines what values can be returned by a read
  - – Coherent if:
    - If P writes to X then reads X, with no writes to X by other processors, then the value read should be that written by P
    - If P writes to X and then another processor reads from X, if read/write sufficiently separated then the value read should be that written by P
    - Writes to the same location are serialized; two writes to the same location by any two processors are seen in the same order by all processors
- Consistency
  - – Determines when a written value will be returned by a read, we'll need to define a memory consistency model
    - For now assume all processors see effect of all writes before a write completes

Techniques to Enforce Coherence

- Directory-based
  - – Centralized Directory holds the status of sharing a block of physical memory
  - – Used with DSM machines; scales to larger processor counts but higher overhead than snooping
- Snooping
  - – No centralized directory
  - – Each cache "snoops" or listens to maintain coherency among caches
  - – Used with CSM machines using a bus

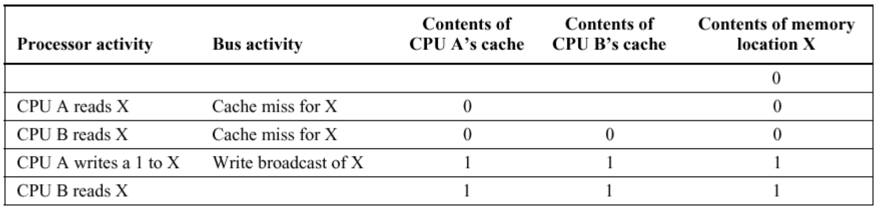
Snooping Protocols for Coherency

- Write Invalidate: When one processor writes, invalidate all copies of this data that may be in other caches – Write Back Cache



Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

- Write Broadcast: When one processor writes, broadcast the value and update any copies that may be in other caches



Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Write broadcast of X	1	1	1
CPU B reads X		1	1	1

Performance Differences

- Multiple writes to the same word
  - Multiple broadcasts using update protocol
  - Only one initial invalidation using invalidate protocol
- Multiword Cache Bocks
  - Invalidation works on cache blocks
  - Update must work on individual bytes
- Delay between writing a word and reading it
  - Less in write update, data immediately into reader's cache
  - Higher in invalidate, reader must cache miss and go to memory to fetch the data

- Usually Write Invalidate is used; less bandwidth

Implementing Invalidation

- Bus-based scheme
  - – Processor to invalidate acquires the bus
  - – Broadcasts the address to invalidate
  - – All other processors continuously snoop on the bus watching the addresses
    - If an address is invalidated that matches an address in its cache, then the corresponding data is invalidated
  - – Serialization of the bus forces serialization of access automatically

Implementing Invalidation

- Write-through cache
  - – To locate a data item when a cache miss occurs, just go to memory (since memory will contain the most up-to-date value in a write-through cache)
- Write-back cache
  - – What problem do we have reading in data on a cache miss when all processors use write-back caches?

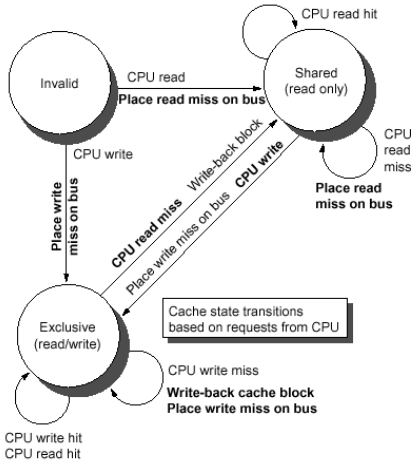
Implementing Invalidation

- Write-Back Cache
  - May need to find the most recent value of a data item in some other processor’s cache, not in memory
  - We can do this using the same snooping scheme for cache misses and writes
    - Each processor snoops every address placed on the bus when a read is requested from memory
    - If a processor has a dirty copy of the requested cache block (i.e. one we wrote to and is hence updated), provide that cache block to the requestor and abort the memory access
- Since write-back caches generate lower memory requirements, they are preferred in multiprocessors despite increased complexity

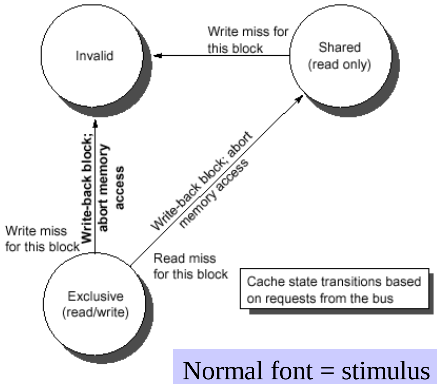
Invalidation with Write-Back Cache

- Snooping
  - Can use normal cache **valid**, **dirty** bits to invalidate or determine if we have the most updated copy
  - Add an extra **state** bit to indicate if a block is shared
    - Write to a block in the shared state generates an invalidation on the bus, marks the block as private
    - Writes to a block in the private state don’t generate invalidations since it should already be invalidated elsewhere
    - Move to the shared state when another processor has a read miss (tries to read this block from memory)
- Example protocol
  - Each cache uses a finite-state transition diagram to determine the proper state and action

Write-Invalidate Write-Back Cache Coherence Protocol



Action by CPU owner of cache


Invalid
Write miss for this block
Shared (read only)
Write-back block; abort memory access
Write-back block; abort memory access
Write miss for this block
Read miss for this block
Exclusive (read/write)
Cache state transitions based on requests from the bus

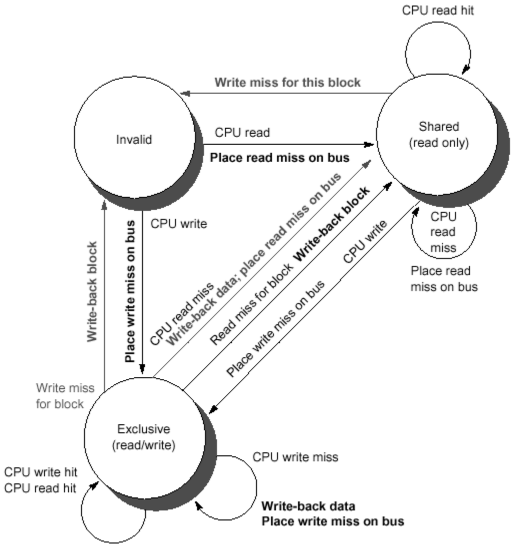
Normal font = stimulus
Bold font = action

Action from Bus

Explanation of Previous Slide

- Left side: state transitions based on actions of the CPU associated with the cache
- Right side: state transitions based on actions of other CPU's placed on the bus
- Example
  - CPU 1 starts in invalid; places read miss, reads block X, goes to shared state
  - CPU 1 re-reads block X, these are read hits
  - CPU 2 reads block X, places read miss, reads block X, goes to shared state
  - CPU 1 writes block X, always places write miss, moves to exclusive state
    - CPU 2 using right side reads write-miss and moves to Invalid state
  - CPU 1 writes or reads block X, stays in exclusive state
  - CPU 2 reads block X, places read miss
    - CPU 1 using right side gets read miss and moves to shared mode, supplies correct memory block to CPU 2
    - CPU 2 moves to shared mode

Merged State Transition Diagram



In practice, we'll have a single state transition diagram with both types of stimulus merged together

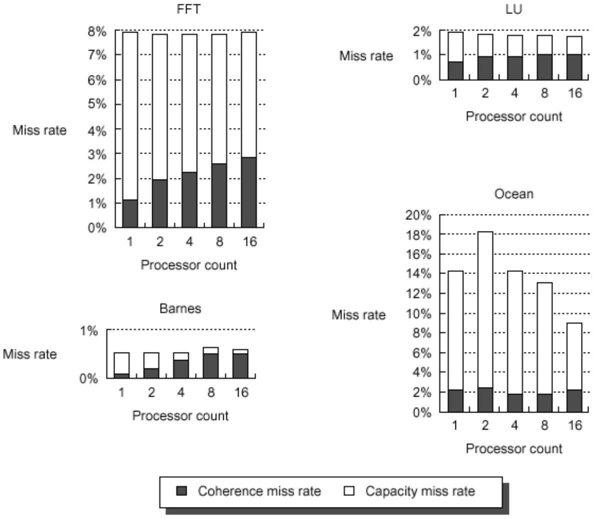
Functionally the same as the split diagram on the previous slide

Protocol somewhat simplified from those in use today

Performance of Snooping Coherence Protocols

- Use the four parallel programs described earlier as a benchmark
  - Split cache misses into two sets
    - Coherence Misses – misses due to cache invalidation
    - Capacity Misses – actually capacity, compulsory and conflict misses, but most of these are capacity. "Normal" cache misses from a uniprocessor

Miss Rate vs. Processor Count

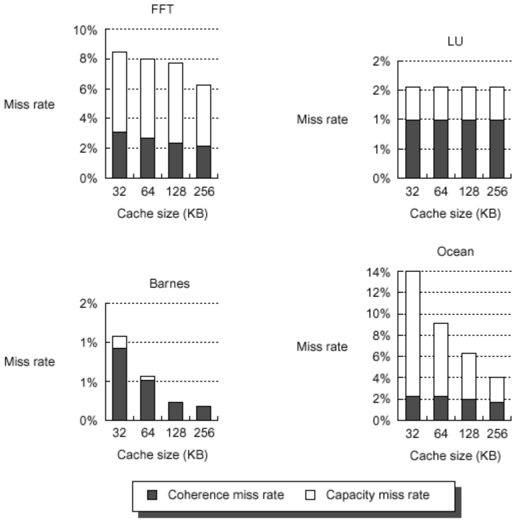


Coherence Miss Rate goes up with processor count, more communication

Overall miss rate slightly down, due to more cache as we add more processors

High-communication app would be bad

Miss Rate vs. Cache Size



Fixed processor count = 16

Miss rate drops as cache size increased, but varies on the application

Other variations possible: block size, set-associative cache, etc. Behaves similarly to the uniprocessor case

Distributed Shared Memory Architectures

- Snooping protocol not so efficient on most DSM machines
  - Snooping requires a broadcast mechanism, which is easy to do on a bus
  - Most DSM systems don't have a bus but a more complex system interconnect (LAN, mesh, hypercube, etc.) so broadcast becomes a much more expensive operation
- One solution:
  - Prevent coherency by marking shared data as uncacheable
  - Private data can still be cached
  - For shared data, we must always access through memory
  - Simple to implement, but can slow things down if programs are not written with this model in mind
    - Access to remote memory can be quite slow

DSM Coherency

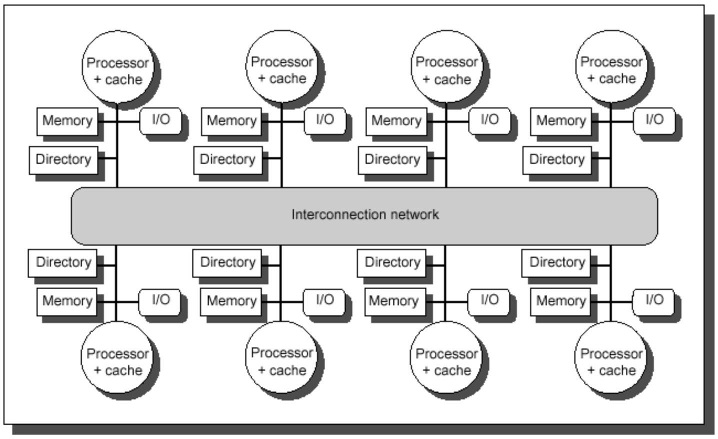
- Another solution: software-based coherency
  - Possible but slow and conservative, every block that might be shared treated as if it is shared
- Most popular alternative: Directory Protocol
  - Directory keeps the state of every block that may be cached
  - Information in the directory includes which caches have copies of the block, whether it is dirty, shared, etc.
  - Centralized version of snooping – this directory is always in the same location
  - Memory requirements for the directory are proportional to the number of memory blocks * number of processors

Directory Protocol

- Each directory must track the following states for its cache blocks
  - Shared?
    - If shared, what processors are sharing this block?
    - This prevents broadcast if we need to invalidate those blocks, instead we can send a message to only these specific processors
  - Uncached?
    - Set if no processor has a copy of the cache block
  - Exclusive?
    - Exactly one processor has a copy of the cache block and has written to it, so memory copy is out of date
    - The processor of the exclusive block is called the **owner** of the block
- Very similar to the snooping protocol, but the directory tracks of who has what data

19

Directory Protocol for DSM



Directory added to each node to implement cache coherence
Each directory tracks caching for memory in its node

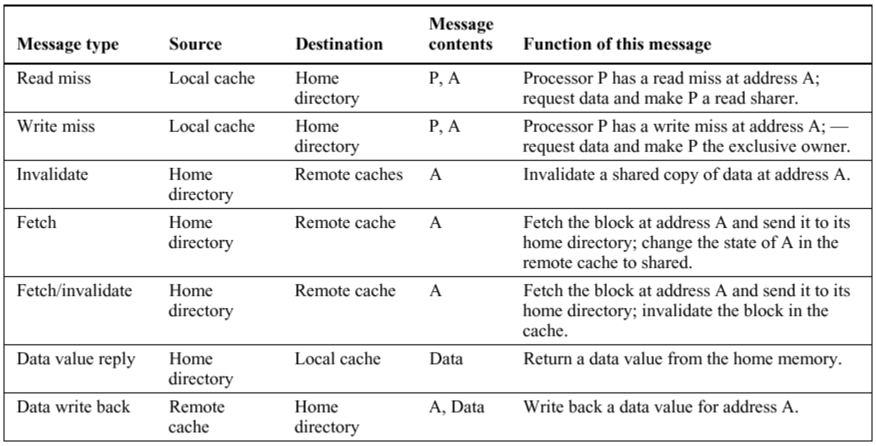
20

Directory Protocol Terminology

- Local node
  - Node where a request originates
- Home node
  - Node where the memory location and directory entry reside
  - Could be the local node as well
- Remote node
  - Node that has a copy of a cache block
  - Might be exclusive or shared

- Nodes will pass messages to one another; messages will move a directory between states in a transition diagram, just like with the snooping protocol

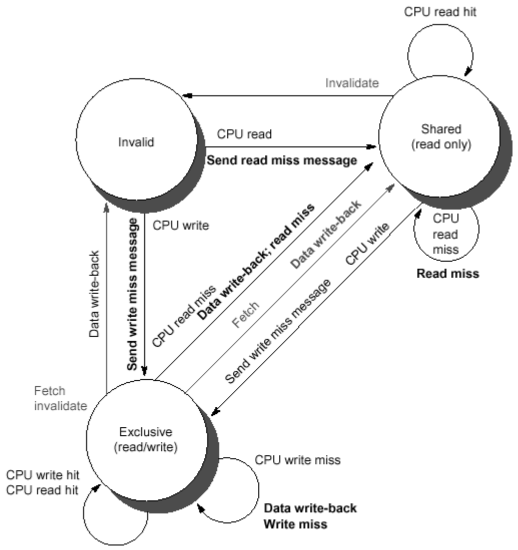
Directory Protocol Messages



Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Processor P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Processor P has a write miss at address A; — request data and make P the exclusive owner.
Invalidate	Home directory	Remote caches	A	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch/invalidate	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
Data value reply	Home directory	Local cache	Data	Return a data value from the home memory.
Data write back	Remote cache	Home directory	A, Data	Write back a data value for address A.

1-2 : Miss requests by local cache to home
3-5 : Home sends to remote cache when home needs to satisfy request
6 : Home sends requested data to local cache
7 : Block replaced, needs to be written back to home or fetch requested

Directory-Based State Transition Diagram

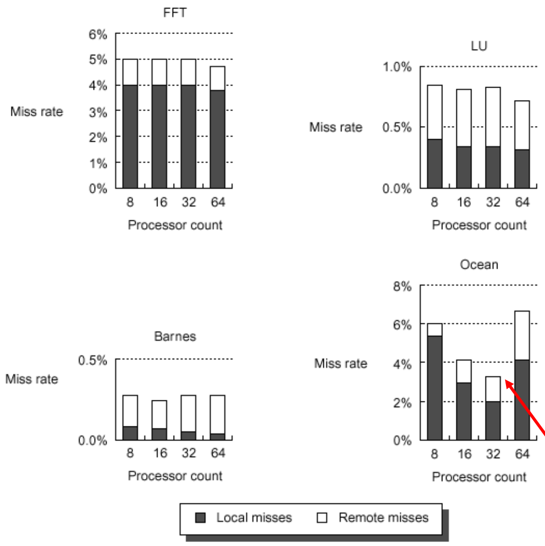


Same as the snooping protocol diagram, except we are sending explicit messages instead of putting data on a common bus

Directory-Based Performance

- Use same parallel programs as for the snooping protocol for our benchmark
- Miss rate broken into two categories
  - Local misses
  - Remote misses
    - Remote misses are much more expensive than local misses
      - Longer read latencies to traverse interconnect
      - Will want to have bigger caches to avoid the latencies
    - Mostly will be coherence misses

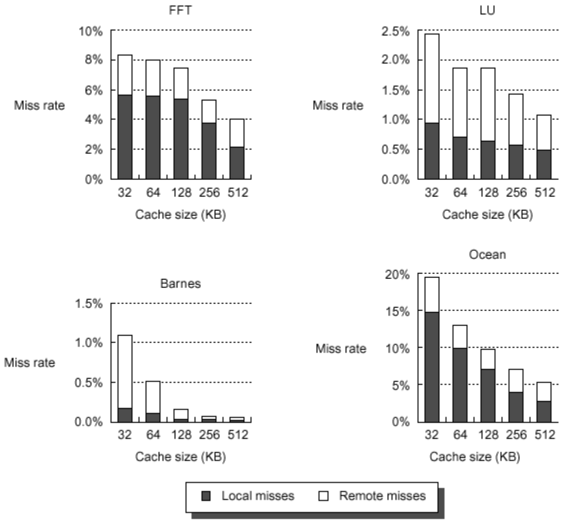
Miss Rate vs. Num Processors



As with snooping caches, remote misses increases somewhat as processor count increases

Anomaly here

Miss Rate vs. Cache Size



P fixed at 64

Miss rates decrease as cache size grows, as you would expect!

Plateau varies with the application

Summary

- Coherence protocols may be needed for correct program behavior
- Most common protocol is write-back cache, write invalidation
- Can use snooping or directory based mechanism to implement coherence
- Coherence requests become more important in programs that are less optimized
  - Optimized programs will access most data locally and have fewer requests
  - Exactly how the cache miss rates affect CPU performance depends on the memory system, interconnect, latency, bandwidth, etc.