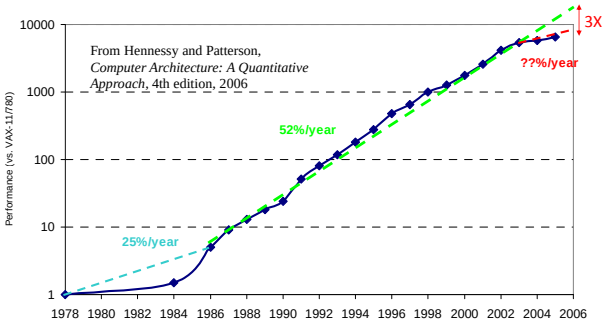
Multiprocessor Systems

55:132/22C:160

Spring2011

1

Uniprocessor Performance (SPECint)



- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

2

The Trend Toward Multiprocessing

- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"

Paul Otellini, President, Intel (2005)

Manufacturer/Year	AMD/' 05	Intel/' 06	IBM/' 04	Sun/' 05
Processors/chip	2	2	2	8
Threads/Processor	1	2	2	4
Threads/chip	2	4	4	32

3

Other Factors ⇒ Multiprocessors

- Growth in data-intensive applications
  - Data bases, file servers, …
- Growing interest in servers, server perf.
- Increasing desktop perf. less important
  - Outside of graphics
- Improved understanding in how to use multiprocessors effectively
  - Especially server where significant natural TLP
- Advantage of leveraging design investment by replication
  - Rather than unique design

4

Flynn's Taxonomy

M.J. Flynn, "Very High-Speed Computers", *Proc. of the IEEE*, V 54, 1900-1909, Dec. 1966.

- Flynn classified by data and control streams in 1966

Single Instruction Single Data (SISD) **(Uniprocessor)**	**Single Instruction Multiple Data SIMD** **(single PC: Vector, CM-2)**
Multiple Instruction Single Data (MISD) **(????)**	**Multiple Instruction Multiple Data MIMD** **(Clusters, SMP servers)**

- SIMD ⇒ Data Level Parallelism (Vector processing)
- MIMD ⇒ Thread Level Parallelism
- MIMD popular because
  - Flexible: N pgms and 1 multithreaded pgm
  - Cost-effective: same MPU in desktop & MIMD

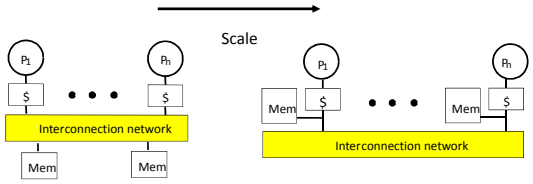
5

Back to Basics

- "A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast."
- Parallel Architecture = Computer Architecture + Communication Architecture
- Two Types of multiprocessors:
  1. Centralized Memory Multiprocessor
     - < few dozen processor chips (and < 100 cores) in 2006
     - Small enough to share single, centralized memory
  2. Physically Distributed-Memory multiprocessor
     - Larger number chips and cores than 1.
     - BW demands ⇒ Memory distributed among processors

6

Centralized vs. Distributed Memory



7

Centralized Memory Multiprocessor

- Also called symmetric multiprocessors (SMPs) because single main memory has a symmetric relationship to all processors
- Large caches ⇒ single memory can satisfy memory demands of small number of processors
- Can scale to a few dozen processors by using a switch and by using many memory banks
- Although scaling beyond that is technically conceivable, it becomes less attractive as the number of processors sharing centralized memory increases

8

Distributed Memory Multiprocessor

- Also called Non-uniform Memory Access time (NUMA) Multiprocessor
- Pro: Cost-effective way to scale memory bandwidth
  - If most accesses are to local memory
- Pro: Reduces latency of local memory accesses
- Con: Communicating data between processors more complex
- Con: Must change software to take advantage of increased memory BW

9

Models for Communication and Memory Architecture

1. Communication occurs by explicitly passing messages among the processors: message-passing multiprocessors
2. Communication occurs through a shared address space (via loads and stores): shared memory multiprocessors either
   - UMA (Uniform Memory Access time) for shared address, centralized memory MP
   - NUMA (Non Uniform Memory Access time multiprocessor) for shared address, distributed memory MP

10

Challenges of Parallel Processing

- First challenge: How much of the program inherently sequential (non-parallelizable)?
- To achieve an 80X speedup from 100 processors, what fraction of original program can be sequential?

  a. 10%

  b. 5%

  c. 1%

  d. <1%

11

Amdahl's Law Answers

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{parallel}}}{\text{Speedup}_{\text{parallel}}}}$$

$$80 = \frac{1}{(1 - \text{Fraction}_{\text{parallel}}) + \frac{\text{Fraction}_{\text{parallel}}}{100}}$$

$$80 \times ((1 - \text{Fraction}_{\text{parallel}}) + \frac{\text{Fraction}_{\text{parallel}}}{100}) = 1$$

$$79 = 80 \times \text{Fraction}_{\text{parallel}} - 0.8 \times \text{Fraction}_{\text{parallel}}$$

$$\text{Fraction}_{\text{parallel}} = 79 / 79.2 = 99.75\%$$

12

Challenges of Parallel Processing

- Second challenge: long latency to remote memory
- Suppose 32 CPU MP, 2GHz, 200 ns remote memory, all local accesses hit memory hierarchy and base CPI is 0.6. (Remote access = 200/0.5 = 400 clock cycles.)
- What is performance impact if 0.2% instructions involve remote access?
  a. 1.5X
  b. 2.0X
  c. 2.5X

13

CPI Equation

- CPI = Base CPI +
  Remote request rate x Remote request cost
- CPI = 0.6 + 0.2% x 400 = 0.6 + 0.8 = 1.4
- Overall Performance slowdown due to 0.2% remote memory accesses:
  1.4/0.6 = 2.33

14

Challenges of Parallel Processing

1. Application parallelism ⇒ primarily via new algorithms that have better parallel performance
2. Long remote latency impact ⇒ 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)
- Today's lecture on HW to help latency via caches

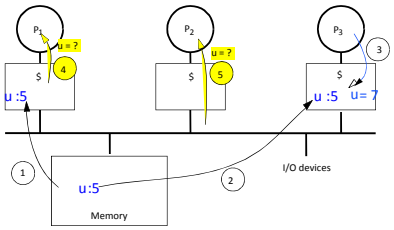
15

Symmetric Shared-Memory Architectures

- From multiple boards on a shared bus to multiple processors inside a single chip
- Caches both:
  - Private data are used by a single processor
  - Shared data are used by multiple processors
- Caching shared data
  ⇒ reduces latency to shared data, memory bandwidth for shared data, and interconnect bandwidth
  ⇒ cache coherence problem

16

Example Cache Coherence Problem



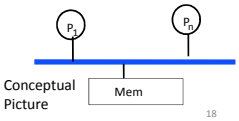
- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on order in which cache flushes occur
  - Processes accessing main memory may see very stale value
- Unacceptable for programming, and its frequent!

17

Example

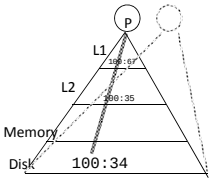
P_1	P_2
/*Assume initial value of A and flag is 0*/	
`A = 1;`	`while (flag == 0); /*spin idly*/`
`flag = 1;`	`print A;`

- 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



18

Intuitive 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

19

Defining Coherent Memory System

1. Preserve Program Order: A read by processor P to 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: Two writes to same location by any two processors are seen in the same order by all processors
   - If not, a processor could keep value 1 since saw as last write
   - For example, if the values 1 and then 2 are written to a location, processors can never read the value of the location as 2 and then later read it as 1

20

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 followed by 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

21

Basic Schemes for Enforcing Coherence

- Program on multiple processors will normally have copies of the same data in several caches
  - Unlike I/O, where its rare
- Rather than trying to avoid sharing in SW, SMPs use a HW protocol to maintain coherent caches
- Migration and Replication key to performance of shared data
  - 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 shared data being simultaneously read, since caches make a copy of data in local cache
    - Reduces both latency of access and contention for read shared data

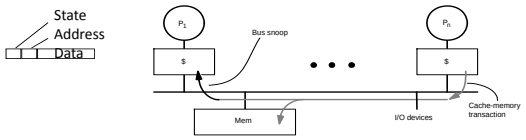
22

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 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

23

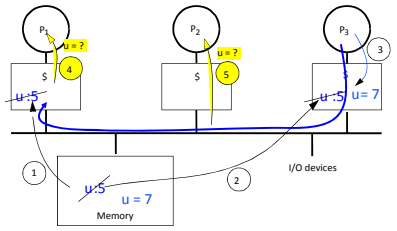
Snoopy Cache-Coherence Protocols



- 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

24

Example: Write-thru Invalidate



- Must invalidate before step 3
- Write update uses more broadcast medium BW
  ⇒ all recent MPUs use write invalidate

25

Architectural Building Blocks

- Cache block state transition diagram
  - FSM specifying how disposition of block changes
    - invalid, valid, dirty
- 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
- 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

26

Locate up-to-date copy of data

- Write-through: get up-to-date copy from memory
  - Write through simpler if enough memory BW
- Write-back is harder
  - Most recent copy can be in a cache
- 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 from retrieving cache block from a processor 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

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 if 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

28

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
  - 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)

29

Cache behavior in response to bus

- Every bus transaction must check the 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
- 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

30

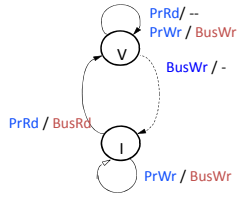
Example Protocol

- Snooping coherence protocol is usually implemented by incorporating a finite-state controller in each node
- 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 implementations, a single controller allows multiple operations to distinct blocks to proceed in interleaved fashion
  - that is, one operation may be initiated before another is completed, even through only one cache access or one bus access is allowed at time

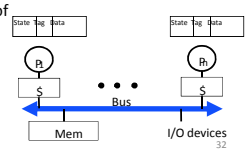
31

Write-through Invalidate Protocol

- Two 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



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



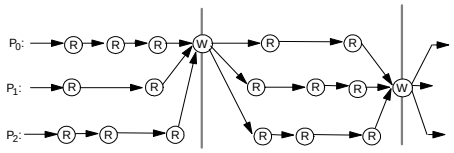
32

Is Two-state Protocol Coherent?

- Processor only observes state of memory system by issuing memory operations
- Assume bus transactions and memory operations are 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
- All writes go to bus + atomicity
  - 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
- Let's understand other ordering issues

33

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, as long as in program order

34

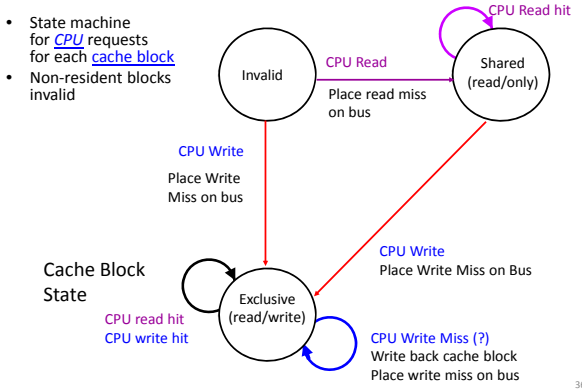
Example Write Back Snoopy Protocol

- Invalidation protocol, write-back cache
  - Snoops every address on bus
  - If it has a dirty copy of requested block, provides that block in response to the read request and aborts the memory access
- Each memory block is in one state:
  - Clean in all caches and up-to-date in memory (Shared)
  - OR Dirty in exactly one cache (Exclusive)
  - OR Not in any caches
- Each cache block is in one state (track these):
  - Shared : block can be read
  - OR Exclusive : cache has only copy, its writeable, and dirty
  - OR Invalid : block contains no data (in uniprocessor cache too)
- Read misses: cause all caches to snoop bus
- Writes to clean blocks are treated as misses

35

Write-Back State Machine - CPU

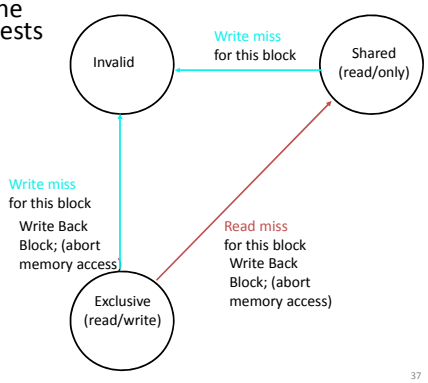
- State machine for *CPU* requests for each cache block
- Non-resident blocks invalid



36

Write-Back State Machine- Bus request

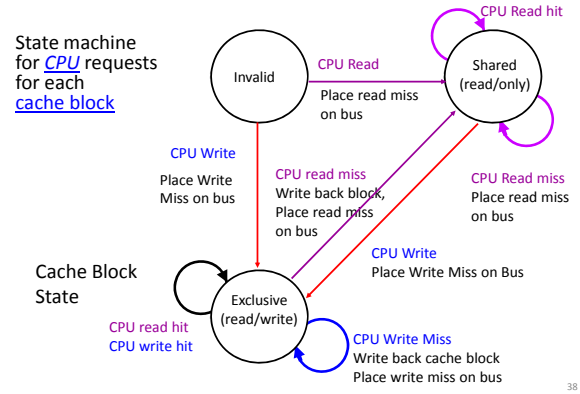
- State machine for *bus* requests for each cache block



37

Block-replacement

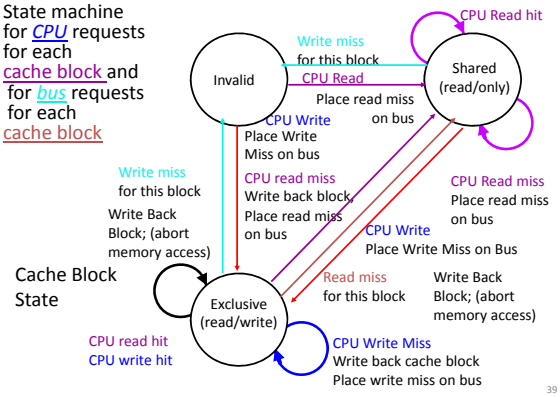
- State machine for *CPU* requests for each cache block



38

Write-back State Machine-III

- State machine for *CPU* requests for each cache block and for *bus* requests for each cache block



39

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory Value
P1 Write 10 to A1												
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block, initial cache state is invalid

40

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2							WrMs	P2	A2		A1	10
				Excl.	A2	40	WrBk	P2	A1	20	A1	20

Assumes A1 and A2 map to same cache block, but A1 != A2

45

Implementation Complications

- Write Races:
  - Cannot update cache until bus is obtained
    - Otherwise, another processor may get bus first, and then write the same cache block!
  - Two step process:
    - Arbitrate for bus
    - Place miss on bus and complete operation
  - If miss occurs to block while waiting for bus, handle miss (invalidate may be needed) and then restart.
  - Split transaction bus:
    - Bus transaction is not atomic: can have multiple outstanding transactions for a block
    - Multiple misses can interleave, allowing two caches to grab block in the Exclusive state
    - Must track and prevent multiple misses for one block
- Must support interventions and invalidations

46

Limitations in Symmetric Shared-Memory Multiprocessors and Snooping Protocols

- Single memory accommodate all CPUs
  ⇒ Multiple memory banks
- Bus-based multiprocessor, bus must support both coherence traffic & normal memory traffic

⇒ Multiple buses or interconnection networks (cross bar or small point-to-point)

- Opteron
  - Memory connected directly to each dual-core chip
  - Point-to-point connections for up to 4 chips
  - Remote memory and local memory latency are similar, allowing OS Opteron as UMA computer

47

Performance of Symmetric Shared-Memory Multiprocessors

- Cache performance is combination of
  1. Uniprocessor cache miss traffic
  2. Traffic caused by communication
- Results in invalidations and subsequent cache misses
- 4th C: *coherence miss*
  - Joins Compulsory, Capacity, Conflict

48

Coherency Misses

1. True sharing misses arise from the communication of data through the cache coherence mechanism
   - Invalidates due to 1st write to shared block
   - Reads by another CPU of modified block in different cache
   - Miss would still occur if block size were 1 word
2. False sharing misses when a block is invalidated because some word in the block, other than the one being read, is written into
   - Invalidation does not cause a new value to be communicated, but only causes an extra cache miss
   - Block is shared, but no word in block is actually shared ⇒ miss would not occur if block size were 1 word

49

Example: True v. False Sharing v. Hit?

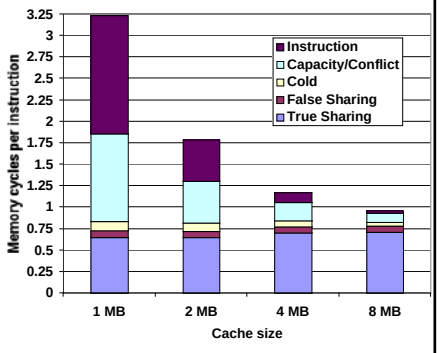
- Assume x1 and x2 in same cache block. P1 and P2 both read x1 and x2 before.

Time	P1	P2	True, False, Hit? Why?
1	Write x1		True miss; invalidate x1 in P2
2		Read x2	False miss; x1 irrelevant to P2
3	Write x1		False miss; x1 irrelevant to P2
4		Write x2	False miss; x1 irrelevant to P2
5	Read x2		True miss; invalidate x2 in P1

50

MP Performance: 4 Processor Commercial Workload: OLTP, Decision Support (Database), Search Engine

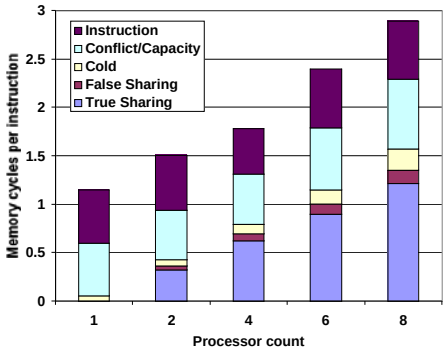
- True sharing and false sharing unchanged going from 1 MB to 8 MB (L3 cache)
- Uniprocessor cache misses improve with cache size increase (Instruction, Capacity/Conflict, Compulsory)



51

MP Performance 2MB Cache Commercial Workload: OLTP, Decision Support (Database), Search Engine

- True sharing, false sharing increase going from 1 to 8 CPUs



52

A Cache Coherent System Must:

- Provide set of states, state transition diagram, and actions
- Manage coherence protocol
  - (0) Determine when to invoke coherence protocol
  - (a) Find info about state of block in other caches to determine action
    - whether need to communicate with other cached copies
  - (b) Locate the other copies
  - (c) Communicate with those copies (invalidate/update)
- (0) is done the same way on all systems
  - state of the line is maintained in the cache
  - protocol is invoked if an "access fault" occurs on the line
- Different approaches distinguished by (a) to (c)

53

Bus-based Coherence

- All of (a), (b), (c) done through broadcast on bus
  - faulting processor sends out a "search"
  - others respond to the search probe and take necessary action
- Could do it in scalable network too
  - broadcast to all processors, and let them respond
- Conceptually simple, but broadcast doesn't scale with p
  - on bus, bus bandwidth doesn't scale
  - on scalable network, every fault leads to at least p network transactions
- Scalable coherence:
  - can have same cache states and state transition diagram
  - different mechanisms to manage protocol

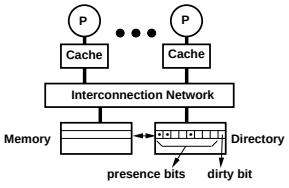
54

Scalable Approach: Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks 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

55

Basic Operation of Directory



- k processors.
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

- Read from main memory by processor i:

  If dirty-bit OFF then { read from main memory; turn p[i] ON; }

  if dirty-bit ON then { recall line from dirty proc (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}

- Write to main memory by processor i:
  - If dirty-bit OFF then { supply data to i; send invalidations to all caches that have the block; turn dirty-bit ON; turn p[i] ON; ... }
  - ...

56

Directory Protocol

- Similar to Snoopy Protocol: Three states
  - Shared: ≥ 1 processors have data, memory up-to-date
  - Uncached (no processor hasit; not valid in any cache)
  - Exclusive: 1 processor (owner) has data; memory out-of-date
- In addition to cache state, must track which processors have data when in the shared state (usually bit vector, 1 if processor has copy)
- Keep it simple(r):
  - Writes to non-exclusive data => write miss
  - Processor blocks until access completes
  - Assume messages received and acted upon in order sent

57

Directory Protocol

- No bus and don't want to broadcast:
  - interconnect no longer single arbitration point
  - all messages have explicit responses
- Terms: typically 3 processors involved
  - Local node where a request originates
  - Home node where the memory location of an address resides
  - Remote node has a copy of a cache block, whether exclusive or shared
- Example messages on next slide: P = processor number, A = address

58

Directory Protocol Messages (Fig 4.22)

Message type	*Source*	*Destination*	*Msg Content*
Read miss	Local cache	Home directory	P, A
– *Processor P reads data at address A; make P a read sharer and request data*			
Write miss	Local cache	Home directory	P, A
– *Processor P has a write miss at address A; make P the exclusive owner and request data*			
Invalidate	Home directory	Remote caches	A
– *Invalidate a shared copy 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 (read miss response)*			
Data write back	Remote cache	Home directory	A, Data
– *Write back a data value for address A (invalidate response)*			

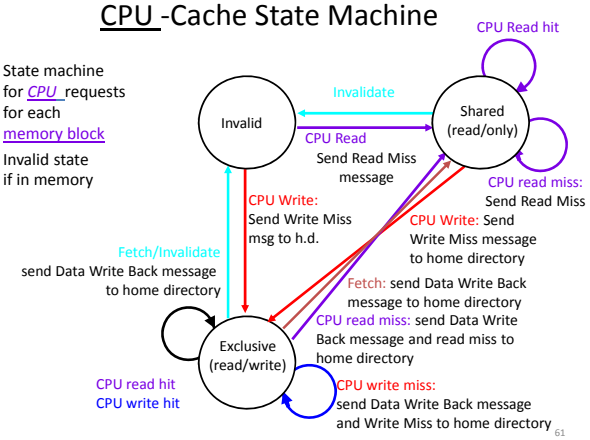
State Transition Diagram for One Cache Block in Directory Based System

- States identical to snoopy case; transactions very similar.
- Transitions caused by read misses, write misses, invalidates, data fetch requests
- Generates read miss & write miss msg to home directory.
- Write misses that were broadcast on the bus for snooping => explicit invalidate & data fetch requests.
- Note: on a write, a cache block is bigger, so need to read the full cache block

60

CPU -Cache State Machine

- State machine for *CPU* requests for each memory block
- Invalid state if in memory



61

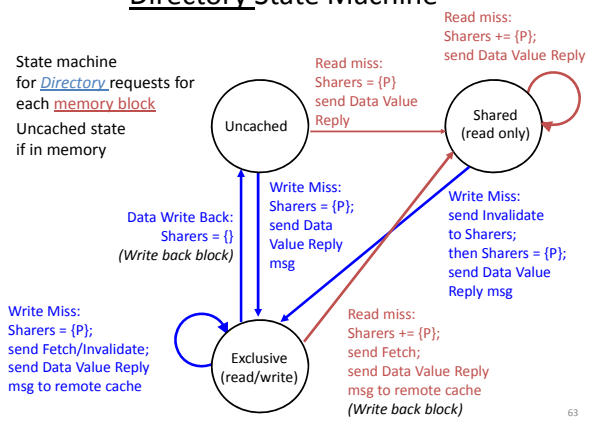
State Transition Diagram for Directory

- Same states & structure as the transition diagram for an individual cache
- 2 actions: update of directory state & send messages to satisfy requests
- Tracks all copies of memory block
- Also indicates an action that updates the sharing set, Sharers, as well as sending a message

62

Directory State Machine

- State machine for *Directory* requests for each memory block
- Uncached state if in memory



63

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.

64

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, causing 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). State is shared.
  - Data write-back: owner processor is replacing the block and hence must write it back, making 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.

65

Example

Processor 1 | Processor 2 | Interconnect | Directory | Memory

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Directory Addr	Directory State	Directory {Procs}	Memory Value
P1 Write 10 to A1														
P1: Read A1														
P2: Read A1														
P2: Write 20 to A1														
P2: Write 40 to A2														

A1 and A2 map to the same cache block

66

Example

Processor 1 | Processor 2 | Interconnect | Directory | Memory

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Directory Addr	Directory State	Directory {Procs}	Memory Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1		A1	Ex	{P1}	
P1: Read A1							DaRp	P1	A1	0				
P2: Read A1														
P2: Write 20 to A1														
P2: Write 40 to A2														

A1 and A2 map to the same cache block

67

Example

Processor 1 | Processor 2 | Interconnect | Directory | Memory

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Directory Addr	Directory State	Directory {Procs}	Memory Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1		A1	Ex	{P1}	
P1: Read A1	Excl.	A1	10				DaRp	P1	A1	0				
P2: Read A1														
P2: Write 20 to A1														
P2: Write 40 to A2														

A1 and A2 map to the same cache block

68

Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
step	P1 State	Addr	Value	P2 State	Addr	Value	Bus Action	Proc.	Addr	Value	Addr	State	{Procs}	Memory Value
P1 Write 10 to A1							WrMs	P1	A1		A1	Ex	{P1}	
	Excl.	A1	10				DaRp	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1				Shar.	A1		RdMs	P2	A1					
	Shar.	A1	10				Ftch	P1	A1	10				10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	{P1,P2}	10
P2: Write 20 to A1														10
														10
P2: Write 40 to A2														10

Write Back

A1 and A2 map to the same cache block

Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
step	P1 State	Addr	Value	P2 State	Addr	Value	Bus Action	Proc.	Addr	Value	Addr	State	{Procs}	Memory Value
P1 Write 10 to A1							WrMs	P1	A1		A1	Ex	{P1}	
	Excl.	A1	10				DaRp	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1				Shar.	A1		RdMs	P2	A1					
	Shar.	A1	10				Ftch	P1	A1	10				10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	{P1,P2}	10
P2: Write 20 to A1				Excl.	A1	20	WrMs	P2	A1					10
	Inv.						Inval.	P1	A1		A1	Excl.	{P2}	10
P2: Write 40 to A2														10
														-

A1 and A2 map to the same cache block

Example

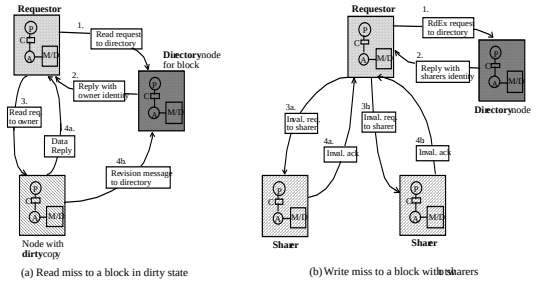
	Processor 1			Processor 2			Interconnect				Directory			Memory
step	P1 State	Addr	Value	P2 State	Addr	Value	Bus Action	Proc.	Addr	Value	Addr	State	{Procs}	Memory Value
P1: Write 10 to A1							WrMs	P1	A1		A1	Ex	{P1}	
	Excl.	A1	10				DaRp	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1				Shar.	A1		RdMs	P2	A1					
	Shar.	A1	10				Ftch	P1	A1	10				10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	{P1,P2}	10
P2: Write 20 to A1				Excl.	A1	20	WrMs	P2	A1					10
	Inv.						Inval.	P1	A1		A1	Excl.	{P2}	10
P2: Write 40 to A2							WrMs	P2	A2		A2	Excl.	{P2}	0
							WrBk	P2	A1	20	A1	Unca.	{}	20
				Excl.	A2	40	DaRp	P2	A2	0	A2	Excl.	{P2}	0

A1 and A2 map to the same cache block

Implementing a Directory

- We assume operations atomic, but they are not; reality is much harder; must avoid deadlock when run out of bufffers in network (see Appendix E)
- Optimizations:
  - read miss or write miss in Exclusive: send data directly to requestor from owner vs. 1st to memory and then from memory to requestor

Basic Directory Transactions

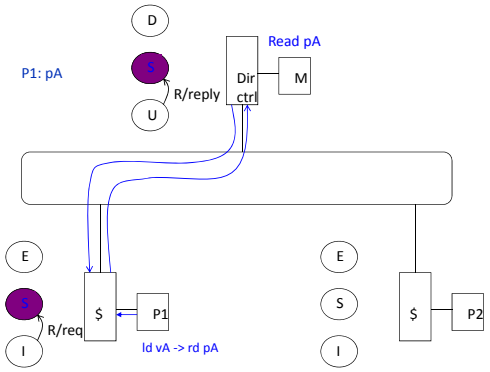


(a) Read miss to a block in dirty state

(b) Write miss to a block with two sharers

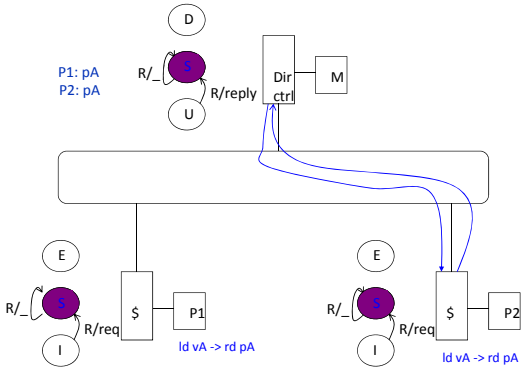
73

Example Directory Protocol (1st Read)



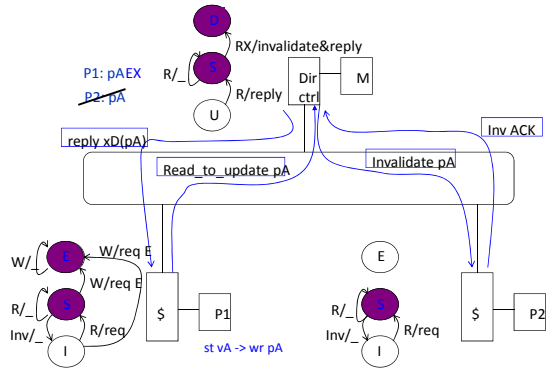
74

Example Directory Protocol (Read Share)



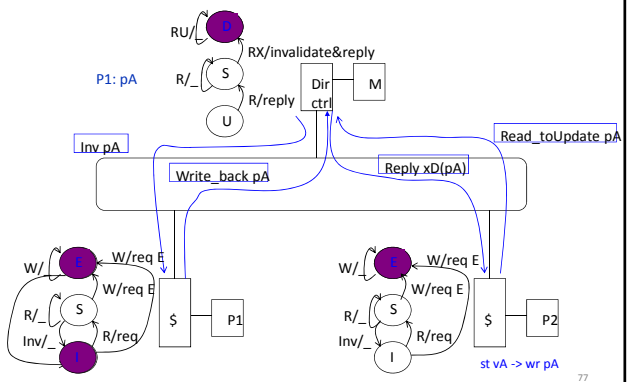
75

Example Directory Protocol (Wr to shared)



76

Example Directory Protocol (Wr to Ex)



77

A Popular Middle Ground

- Two-level "hierarchy"
- Individual nodes are multiprocessors, connected non-hierachically
  - e.g. mesh of SMPs
- Coherence across nodes is directory-based
  - directory keeps track of nodes, not individual processors
- Coherence within nodes is snooping or directory
  - orthogonal, but needs a good interface of functionality
- SMP on a chip directory + snoop?

78

And in Conclusion …

- Caches contain all information on state of cached memory blocks
- Snooping cache over shared medium for smaller MP by invalidating other cached copies on write
- Sharing cached data ⇒ Coherence (values returned by a read), Consistency (when a written value will be returned by a read)
- Snooping and Directory Protocols similar; bus makes snooping easier because of broadcast (snooping => uniform memory access)
- Directory has extra data structure to keep track of state of all cache blocks
- Distributing directory => scalable shared address multiprocessor
  => Cache coherent, Non uniform memory access

79