COEN-4730/EECE-5730 Computer Architecture

Lecture 8
Thread Level Parallelism and Coherence
(Ch.5)

Cristinel Ababei
***D**ept. of Electrical and Computer Engineering*
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

Credits: Slides adapted from presentations of Sudeep Pasricha and others: Kubiatowicz, Patterson, Mutlu, Elsevier

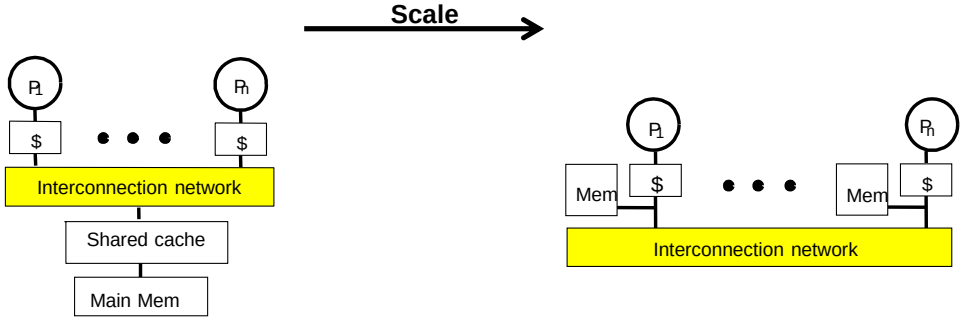
Outline

- Shared-centralized vs. Distributed Memory
- Challenges of Parallel Processing
- Centralized shared-memory architecture
- Snoop based Coherence
- Directory based Coherence

Back to Basics

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

Centralized vs. Distributed Memory MultiProcessors



1. Centralized Shared-memory Multiprocessor

2. Distributed-memory Multiprocessor

1. Centralized Memory MultiProcessor

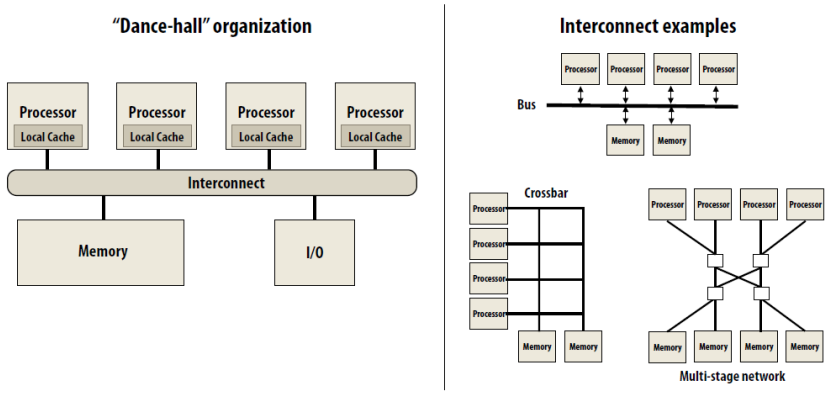
- Also called Symmetric MultiProcessors (SMPs) because single main memory has a symmetric relationship to all processors
- Large caches ⇒ single memory can satisfy memory demands of small number of processors
- Can scale to a few dozen processors by using a switch and by using many memory banks
- Although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

5

5

Centralized Memory Multiprocessor

Any processor can directly reference any memory location

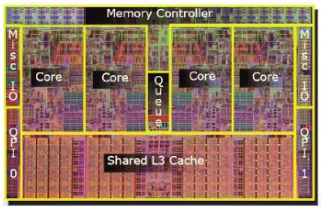


- **Symmetric (shared-memory) multi-processor (SMP):**
  - **Uniform memory access time: cost of accessing an uncached* memory address is the same for all processors**

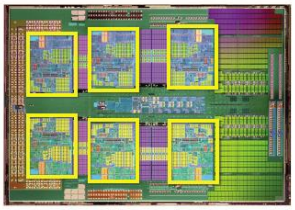
6

6

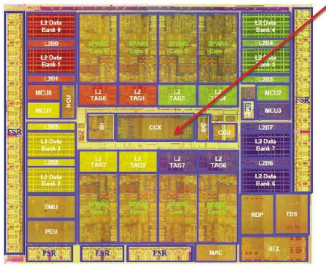
SMP Examples: Commodity processors



Intel Core i7 (quad core)
(network is a ring)



AMD Phenom II (six core)



Eight cores

Note size of crossbar: about die area of one core

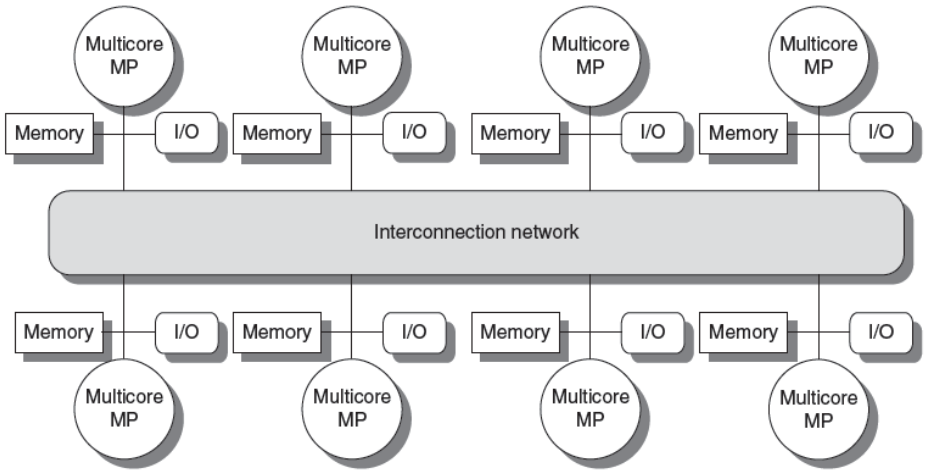
7

2. Distributed Shared Memory (DSM) MultiProcessor

- Also called Non Uniform Memory Access (NUMA) since the access time depends on the location of a data word in memory
- Pros:
  - Cost-effective way to scale memory bandwidth
    - If most accesses are to local memory
  - Reduces latency of local memory accesses
- Cons:
  - Communicating data between processors more complex
  - Must change software to take advantage of increased memory BW

8

Distributed Shared Memory Multiprocessor



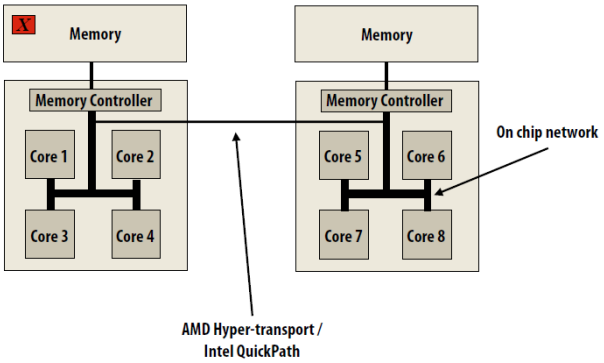
9

9

NUMA Example

Example: latency to access location X is higher from cores 5-8 than cores 1-4

Example: modern dual-socket configuration



10

10

Two Models for Communication and Memory Architecture

1. Communication in DSM and SMP 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
2. Communication occurs by explicitly passing messages among the processors: message-passing multiprocessors
   - Mostly clusters and warehouse scale systems

11

11

Outline

- Shared-centralized vs. Distributed Memory
- Challenges of Parallel Processing
- Centralized shared-memory architecture
- Snoop based Coherence
- Directory based Coherence

12

12

Challenges of Parallel Processing

- First challenge: percentage of program that is inherently sequential
- Suppose we want to achieve 80X speedup from 100 processors. What fraction of original program can be sequential?
  a. 10%
  b. 5%
  c. 1%
  d. <1%
  e. Impossible

13

Amdahl's Law in Action

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

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

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

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

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

14

Challenges of Parallel Processing

- Second challenge: long latency to remote memory
- Suppose 32 CPU MP, 2GHz, 200ns to handle remote memory accesses; all local accesses hit memory hierarchy and Base CPI is 0.5. (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

15

15

CPI Equation

- CPI = Base CPI +
  Remote request rate x Remote request cost
- CPI = 0.5 + 0.2% x 400 = 0.5 + 0.8 = 1.3
- So, multiprocessor with all local references is 1.3/0.5 = 2.6 times faster, than if 0.2% instructions involved remote access!

16

Challenges of Parallel Processing

Possible solutions to the two challenges:

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 freq. of remote accesses by:
  - Caching shared data (HW), or
  - Restructuring the data layout to make more accesses local (SW)

17

17

Outline

- Shared-centralized vs. Distributed Memory
- Challenges of Parallel Processing
- Centralized shared-memory architecture
- Snoop based Coherence
- Directory based Coherence

18

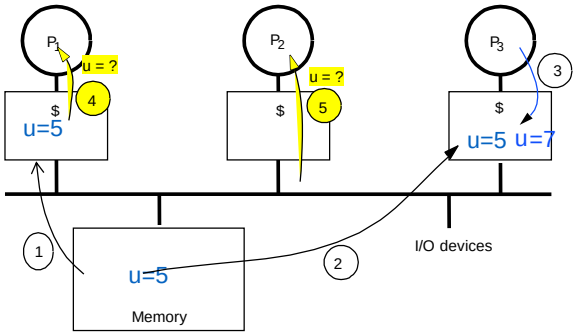
18

Symmetric Shared-Memory Architecture

- Caches both:
  - Private data - used by a single processor
  - Shared data - used by multiple processors
- Caching shared data
  ⇒ reduces latency to shared data, memory bandwidth for shared data, and interconnect bandwidth
- But caching creates a cache coherence problem

19

Example: Cache Coherence Problem



- Processors see different values for u after event 3
- Unacceptable for programming, and it is frequent!
- Reading an address should return the last value written to that address

20

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: 2 writes to same location by any 2 processors are seen in the same order by all processors
   - For example, if 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

21

21

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

$\Rightarrow$ 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

22

22

Migration and Replication

- In a coherent multiprocessor, caches provide both migration and replication
- Migration and replication: key to performance of shared data
  1. 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
  2. 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

23

23

Enforcing Coherence
Two Classes of Cache Coherence Protocols

SMPs use a HW protocol to maintain coherent caches

1. 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
2. Directory based - Sharing status of a block of physical memory is kept in just one location, the directory

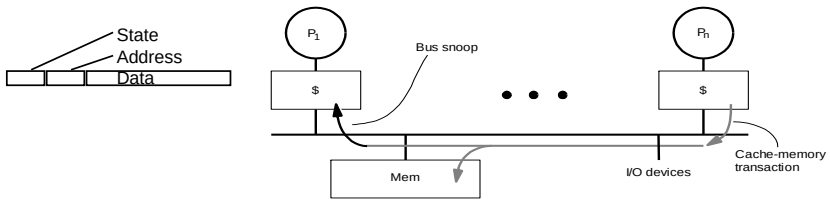
24

24

Outline

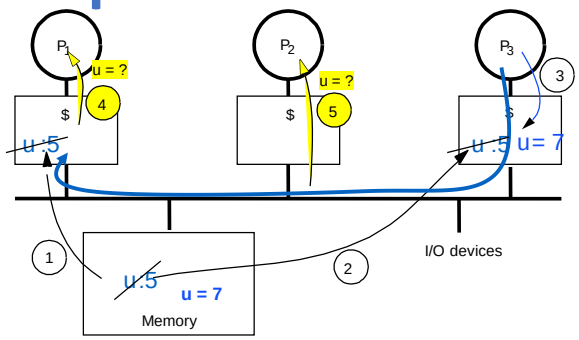
- Shared-centralized vs. Distributed Memory
- Challenges of Parallel Processing
- Centralized shared-memory architecture
- Snoop based Coherence
- Directory based Coherence

Snoopy Cache-Coherence Protocols



- Cache Controller "snoops" all transactions on the shared medium (bus or switch)
  - Relevant transaction if transaction if for a block the cache contains
  - Take action to ensure coherence
    - invalidate, update, or supply value
  - Depends on state of the block and the protocol

Example: Write Invalidate



- Must invalidate before Step 3
- All recent MPUs use write invalidate

27

27

Architectural Building Blocks

- Cache block state transition diagram
  - FSM specifying how disposition of block changes
    - invalid, valid, dirty
- Broadcast Medium (e.g., bus)
  - Fundamental system design abstraction
  - Logically single set of wires connect several devices
  - Protocol: arbitration, command/address, 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 mechanism to find up-to-date copy of cache block
  - Easy for write-thru; more challenging for write-back

28

28

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 (WB) 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 finds that it 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

29

29

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

30

30

Cache Resources for WB Snooping

- To track whether a cache block is shared, add extra state bit ("shared") 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
  - 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)
- *Recall:*
  - *Valid bit: says whether or not this cache entry contains a valid address*
  - *Dirty bit: status bit that indicates whether the block in cache is dirty (modified while in the cache) or clean (not modified). If clean, the block is not written back on a miss, since identical information to the cache is found in lower levels.*

Example Snooping Coherence 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

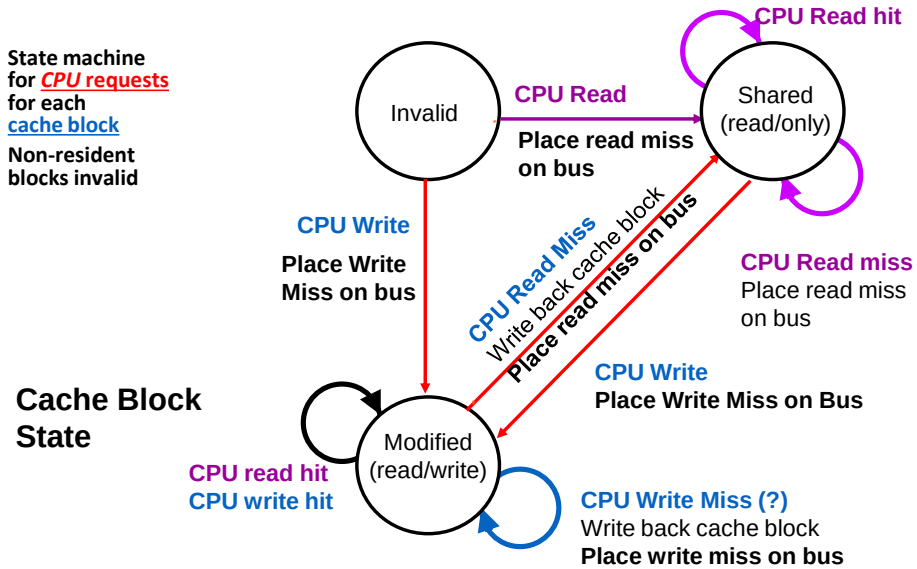
Write-Back Snoopy Protocol

- Invalidation protocol, **write-back** cache
  - Snoops every address on bus
- Each memory block is in one state:
  - Clean in all caches and up-to-date in main memory (Shared)
  - OR Dirty in exactly one cache (Modified)
  - OR Not in any caches
- Each cache block is in one state:
  - Shared: block can be read
  - OR Modified: cache has only copy, it's writeable, and dirty
  - OR Invalid: block contains no data

33

Write-Back State Machine – Requests from CPU

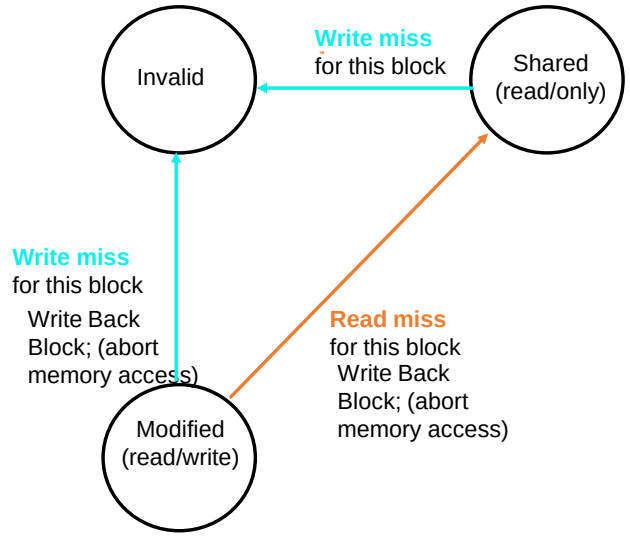
- State machine for *CPU* requests for each cache block
- Non-resident blocks invalid



34

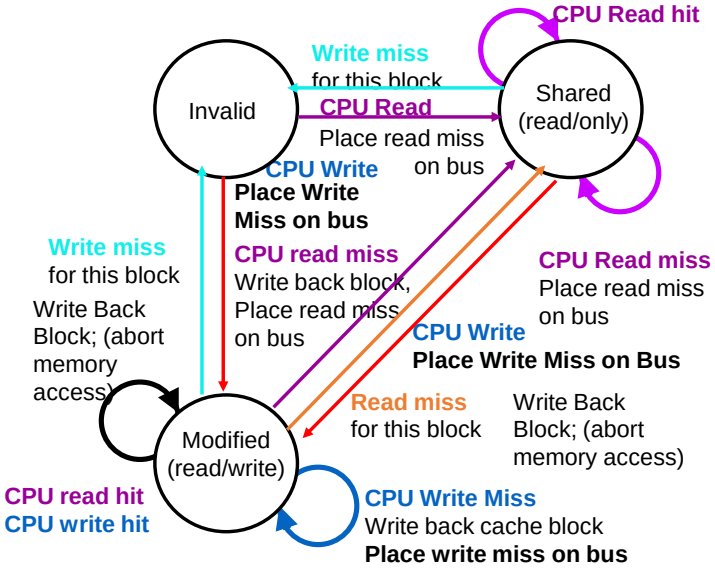
Write-Back State Machine – Requests from Bus

- State machine for *bus requests* for each cache block
- Labels = commands received from bus



Write-Back State Machine - Complete

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



Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Add	Valu
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

37

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Add	Valu
P1 Write 10 to A1	Mod.	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

38

Example

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

Assumes A1 and A2 map to same cache block

Example

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

Assumes A1 and A2 map to same cache block

Example

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

Assumes A1 and A2 map to same cache block

41

41

Example

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

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

42

42

Extensions to the Basic Snoopy Coherence Protocol

- Simple three-state protocol is often referred to by the first letter of the states, making it a MSI (Modified, Shared, Invalid) protocol
- Extensions to MSI are created by adding additional states and transactions, which optimize certain behaviors, resulting in improved performance
- Three popular extensions:

1. MESI

- Adds the state Exclusive to the basic MSI protocol to indicate when a cache block is resident only in a single cache but is clean
- If a block is in the E state, it can be written without generating any invalidates => optimizes the case where a block is read by a single cache before being written by that same cache

43

43

Extensions to the Basic Snoopy Coherence Protocol

2. MESIF

- Intel i7 uses a variant of a MESI protocol, called MESIF, which adds a state (Forward) to designate which sharing processor should respond to a request
  - In MESI, a cache line request that is received by multiple caches may either be satisfied from (slow) main memory, or *all* the sharing caches could respond, bombarding the requestor with redundant responses

3. MOESI

- Adds the state Owned to the MESI protocol to indicate that the associated block is owned by that cache and out-of-date in memory
- In MSI and MESI protocols, when there is an attempt to share a block in the M state, the state is changed to S (in both the original and newly sharing cache), and the block must be written back to memory
- In MOESI, the block can be changed from the Modified to Owned state in the original cache without writing it to memory
- Other caches, which are newly sharing the block, keep the block in the S state; the O state, which only the original cache holds, indicates that main memory copy is out of date and that the designated cache is the owner.
- owner of the block must supply it on a miss
- AMD Opteron uses the MOESI protocol

44

44

Limitations in Symmetric Shared-Memory Multiprocessors and Snooping Protocols

- Single memory accommodates 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)

45

45

Outline

- Shared-centralized vs. Distributed Memory
- Challenges of Parallel Processing
- Centralized shared-memory architecture
- Snoop based Coherence
- Directory based Coherence

46

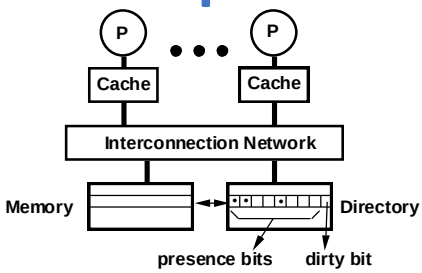
46

Scalable Approach: Directories

- Each 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
- Example: Intel core i7
  - keep a bit vector of size equal to number of cores for each L3 block
  - bit vector indicates which private caches may have copies of a block that is in L3
  - invalidations are only sent to those caches
- Many alternatives for organizing directory information

47

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 processor (change 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; ... }
  - ...

48

Directory Protocol

- Similar to Snoopy Protocol: Three states
  - Shared: ≥ 1 processors have data, memory up-to-date
  - Uncached (no processor has it; not valid in any cache)
  - Modified: 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 for every block, 1 if processor has copy)
- No bus and don't want to broadcast:
  - Interconnect no longer single arbitration point!
- Terms: typically 3 processors involved
  - Local node where a request originates
  - Home node where the memory location and directory of an address resides
  - Remote node has a copy of a cache block, whether modified or shared

49

Directory Protocol Messages

Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Node P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Node P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	Local cache	Home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	Home directory	Remote cache	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	D	Return a data value from the home memory.
Data write-back	Remote cache	Home directory	A, D	Write-back a data value for address A.

P = processor number, A = address, D = Data contents

50

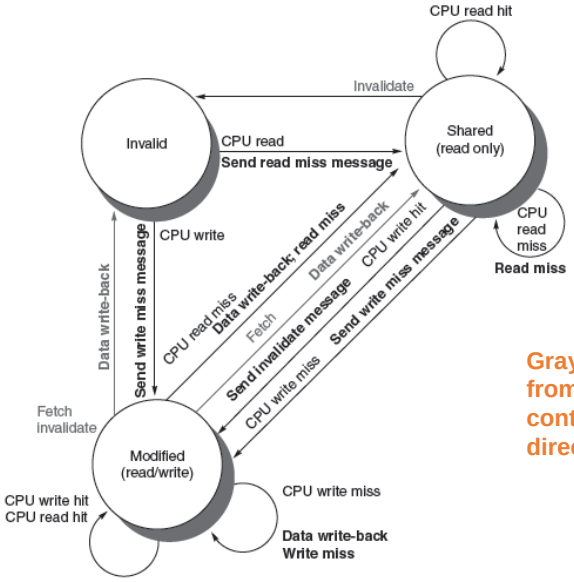
State Transition Diagram for One Cache Block in Directory Based System

- States identical to snoopy case; transactions very similar
- Transitions caused by read misses, write misses, invalidates, data fetch requests
- Generates read miss & write miss messages to home directory
  - Write misses that were broadcast on the bus for snooping => explicit invalidate & data fetch requests

51

CPU - Cache State Machine

- State machine for *CPU* requests for each block
- Write miss operation, which was broadcast on the bus (or other network) in the snooping scheme, is replaced by the data *fetch and invalidate* operations selectively sent by the directory controller



Gray lines originate from directory controller directives

52

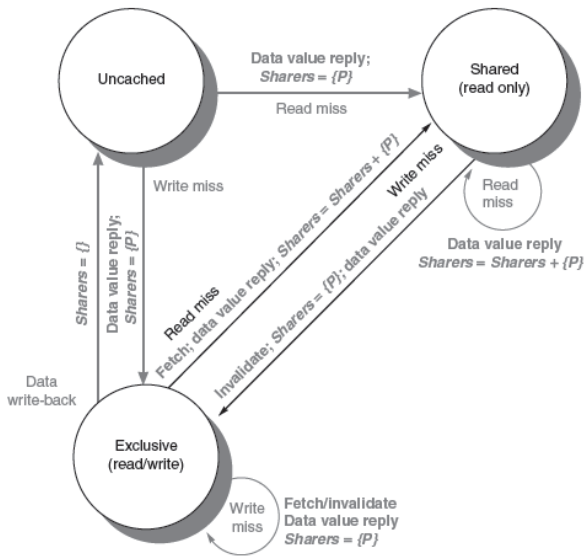
State Transition Diagram for Directory

- 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 updates the sharing set, Sharers, when necessary, and sends back the *value* or *invalidate* messages

53

Directory - State Machine

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



Data write back here refers to writing to DRAM

54

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, 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.

Example

Processor 1 **Processor 2** **Interconnect** **Directory** **Memory**

	P1			P2			Bus				Directory			Memor
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	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

Example

Processor 1 **Processor 2** **Interconnect** **Directory** **Memory**

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

Example

Processor 1 Processor 2 Interconnect Directory Memory

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

A1 and A2 map to the same cache block

Example

Processor 1 Processor 2 Interconnect Directory Memory

	P1			P2			Bus				Directory			Memor
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	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	A1			10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	P1,P2}	10
P2: Write 20 to A1														
P2: Write 40 to A2														

Write Back

A1 and A2 map to the same cache block

Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			P2			Bus				Directory			Memor
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	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	A1			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														

A1 and A2 map to the same cache block

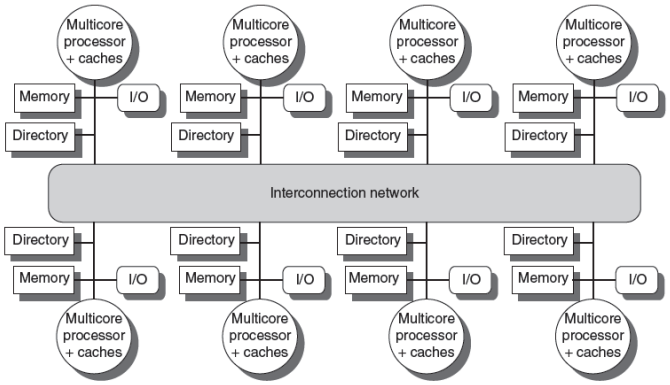
Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			P2			Bus				Directory			Memor
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	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	A1			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

Distributed Directories

- Single directory is not scalable
- Distributed directory:
  - Each directory is responsible for tracking the caches that share the memory addresses of the portion of memory in the node



63

63

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

64

64

Characteristics of high-end processors

Feature	AMD Opteron 8439	IBM Power 7	Intel Xenon 7560	Sun T2
Transistors	904 M	1200 M	2300 M	500 M
Power (nominal)	137 W	140 W	130 W	95 W
Max. cores/chip	6	8	8	8
Multithreading	No	SMT	SMT	Fine-grained
Threads/core	1	4	2	8
Instruction issue/clock	3 from one thread	6 from one thread	4 from one thread	2 from 2 threads
Clock rate	2.8 GHz	4.1 GHz	2.7 GHz	1.6 GHz
Outermost cache	L3; 6 MB; shared	L3; 32 MB (using embedded DRAM); shared or private/core	L3; 24 MB; shared	L2; 4 MB; shared
Inclusion	No, although L2 is superset of L1	Yes, L3 superset	Yes, L3 superset	Yes
Multicore coherence protocol	MOESI	Extended MESI with behavioral and locality hints (13-state protocol)	MESIF	MOESI
Multicore coherence implementation	Snooping	Directory at L3	Directory at L3	Directory at L2
Extended coherence support	Up to 8 processor chips can be connected with HyperTransport in a ring, using directory or snooping. System is NUMA.	Up to 32 processor chips can be connected with the SMP links. Dynamic distributed directory structure. Memory access is symmetric outside of an 8-core chip.	Up to 8 processor cores can be implemented via Quickpath Interconnect. Support for directories with external logic.	Implemented via four coherence links per processor that can be used to snoop. Up to two chips directly connect, and up to four connect using external ASICs.

65

65

Conclusion

- 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
- MPs are highly effective for multiprogramed workloads
- MPs proved effective for intensive commercial workloads

66

66