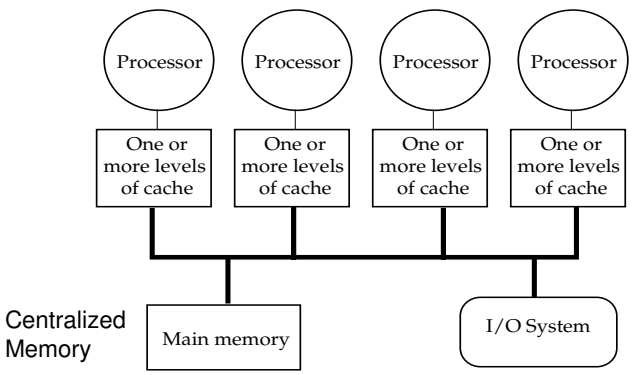
Lecture 31: Multiprocessors—Directory Schemes

Professor Randy H. Katz
Computer Science 252
Spring 1996

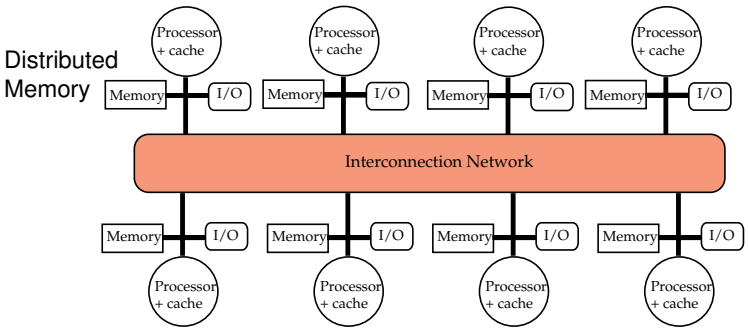
Review: Small-Scale MIMD Designs

- **Memory: centralized with uniform access time and bus interconnect**
- **Examples: SPARCCenter, Challenge, SystemPro**



Review: Large-Scale MIMD Designs

- **Memory: distributed with nonuniform access time and scalable interconnect (distributed memory)**
- **Examples: T3D, Exemplar, Paragon, CM-5**



RHK.S96 3

Review: Communication Models

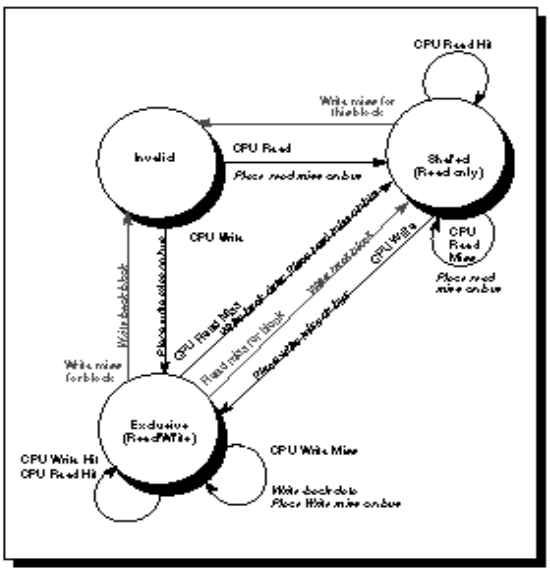
- **Shared Memory**
  - **Processors communicate with shared address space**
  - **Easy on small-scale machines**
  - **Advantages:**
    - » **Model of choice for uniprocessors, small-scale MPs**
    - » **Ease of programming**
    - » **Lower latency**
    - » **Easier to use hardware controlled caching**
- **Message passing**
  - **Processors have private memories communicate with messages**
  - **Advantages:**
    - » **less hardware, easier to design**
    - » **focuses attention on costly operations**
- **Can support either model on either HW base**

RHK.S96 4

Review: Basic Snoopy Protocols

- **Write Invalidate Protocol:**
  - Write to shared data: an invalidate is sent to all caches which snoop and *invalidate* any copies
  - Read miss:
    - » Write-through: memory is always up-to-date
    - » Write-back: snoop in caches to find most recent copy
- **Write Broadcast Protocol:**
  - Write to shared data: broadcast on bus, processors snoop, and *update* copies
  - Read miss: memory is always up-to-date
- **Write Serialization: bus serializes requests**

Snoop Cache: State Machine



Extensions:

- Fourth State
- Clean-> dirty, need invalidate only (upgrade request) Berkeley Protocol
- Clean exclusive state (no miss for private data on write) Illinois Protocol

Snoop Cache Variations

Berkeley Protocol	Basic Protocol	Illinois Protocol
Owned Exclusive	Exclusive	Private Dirty
Owned Shared	Shared	Private Clean
Shared	Invalid	Shared
Invalid		Invalid

Owner can update via bus invalidate operation
Owner must write back when replaced in cache

If read sourced from memory, then Private Clean
if read sourced from other cache, then Shared
Can write in cache if held private clean or dirty

Implementing Snooping Caches

- **Multiple processors must be on bus, access to both addresses and data**
- **Add a few new commands to perform coherency, in addition to read and write**
- **Processors continuously snoop on address bus**
  - **If address matches tag, either invalidate or update**

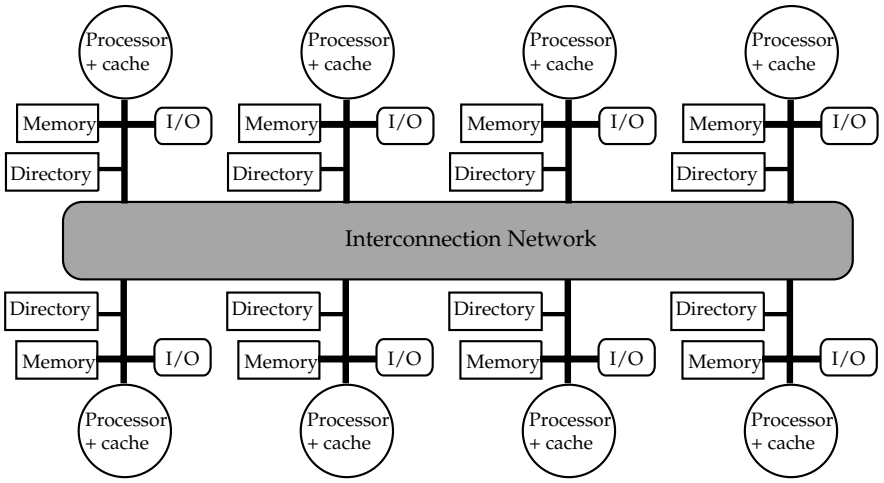
Implementing Snooping Caches

- Bus serializes writes, getting bus ensures no one else can perform operation
- On a miss in a write back cache, may have the desired copy and its dirty, so must reply
- Add extra state bit to cache to determine shared or not
- Since every bus transaction checks cache tags, could interfere with CPU just to check: solution is a duplicate set of tags just to allow checks in parallel with CPU or second level cache that obeys inclusion

Larger MPs

- Separate Memory per Processor
- Local or Remote access via memory controller
- Cache Coherency solution: non-cached pages
- Alternative: directory per cache that tracks state of every block in every cache
  - Which caches have a copies of block, dirty vs. clean, ...
- Info per memory block vs. per cache block?
  - PLUS: In memory => simpler protocol (centralized/one location)
  - MINUS: In memory => directory is f(memory size) vs. f(cache size)
- Prevent directory as bottleneck: distribute directory entries with memory, each keeping track of which Procs have copies of their blocks

Distributed Directory MPs



RHK.S96 11

Directory Protocol

- **Similar to Snoopy Protocol: Three states**
  - **Shared: ≥ 1 processors have data, memory up-to-date**
  - **Uncached**
  - **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**
- **Terms:**
  - **Local node is the node where a request originates**
  - **Home node is the node where the memory location of an address resides**
  - **Remote node is the node that has a copy of a cache block, whether exclusive or shared.**

RHK.S96 12

Directory Protocol Messages

Message type	*Source*	*Destination*	*Msg*
Read miss	Local processor	Home directory	P, A
– *Processor P reads data at address A; send data and make P a read sharer*			
Write miss	Local processor	Home directory	P, A
– *Processor P writes data at address A; send data and make P the exclusive owner*			
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*			
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*			

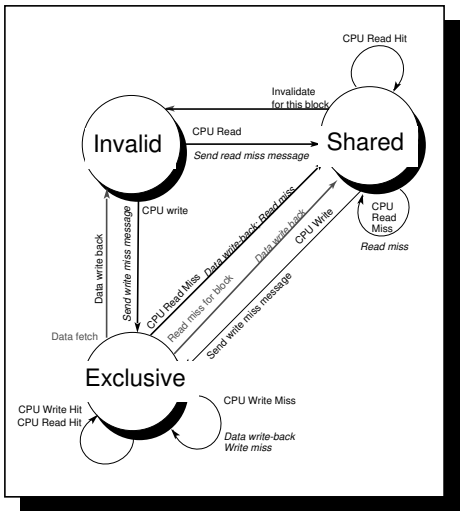
Example Directory Protocol

- **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 is sent back the data from memory and the requestor is the only sharing node. The state of the block is made Shared.**
  - **Write miss: requesting processor is sent the value and 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

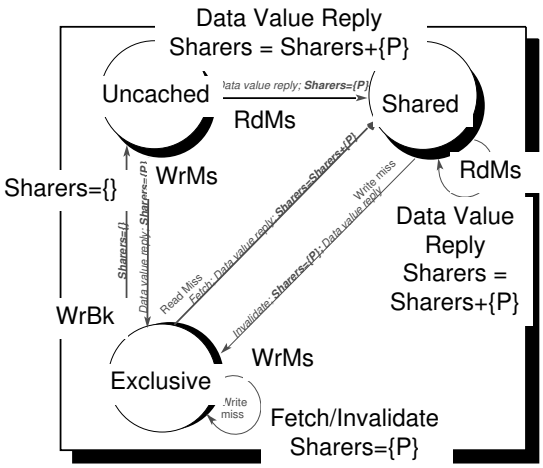
- **Block is Exclusive: current value of the block is held in the cache of the processor identified by the set Sharers (the owner) & three possible directory requests:**
  - **Read miss: owner processor is sent a data fetch message, which causes state of block in owner's cache to transition to Shared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

State Transition Diagram for an Individual Cache Block in a Directory Based System



- **The states are identical to those in the snoopy case, and the transactions are very similar with explicit invalidate and write-back requests replacing the write misses that were formerly broadcast on the bus.**

State Transition Diagram for the Directory



- **The same states and structure as the transition diagram for an individual cache**
  - **All actions are in color since they all are externally caused. Italics indicates the action taken the directory in response to the request. Bold italics indicate an action that updates the sharing set, Sharers, as opposed to sending a message.**

Example

	P1		P2		Bus			Directory		Memor
step	State	Addr	Value	State	Addr	Value	Actio	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

	P1			P2			Bus				Directory			Memor
step	State	Addr	Value	State	Addr	Value	Actio	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

	P1			P2			Bus				Directory			Memor
step	State	Addr	Value	State	Addr	Value	Actio	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

	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				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

A1 and A2 map to the same cache block

Example

	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				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

	P1			P2			Bus				Directory			Memor
step	State	Addr	Value	State	Addr	Value	Actio	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				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