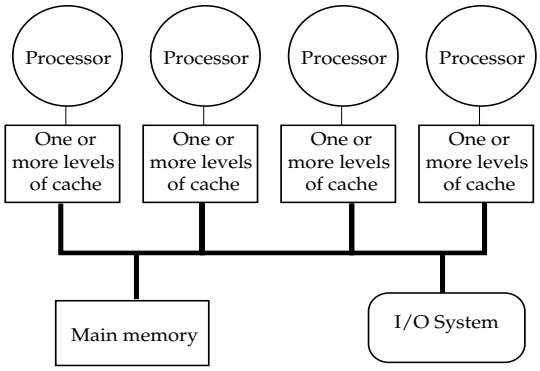
Review: Small-Scale MIMD Designs

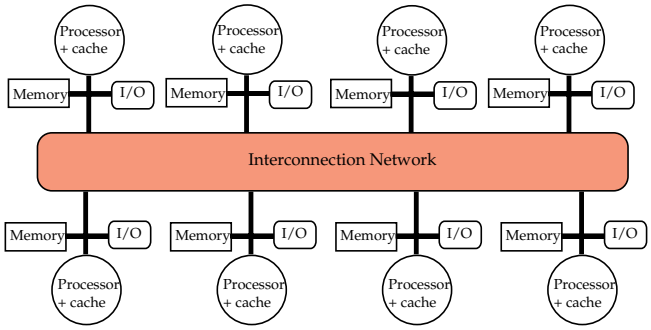
- **Memory: centralized with uniform access time and bus interconnect**
- **Examples: SPARCCenter, Challenge, SystemPro**



© Patterson 1994

Review: Large-Scale MIMD Designs

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



© Patterson 1994

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

© Patterson 1994

Review: Important Communication Properties

- Bandwidth
  - need high bandwidth in communication
  - cannot scale, but stay close
  - make limits in network, memory, and processor
- Processor Overhead + Interconnect Latency
  - overhead to communicate is a problem in many machines
  - affects performance, since processor may have to wait
  - affects ease of programming, since requires more thought to overlap communication and computation
- Latency hiding
  - how can a mechanism help hide latency
  - Examples: overlap message send with computation, prefetch

© Patterson 1994

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**

© Patterson 1994

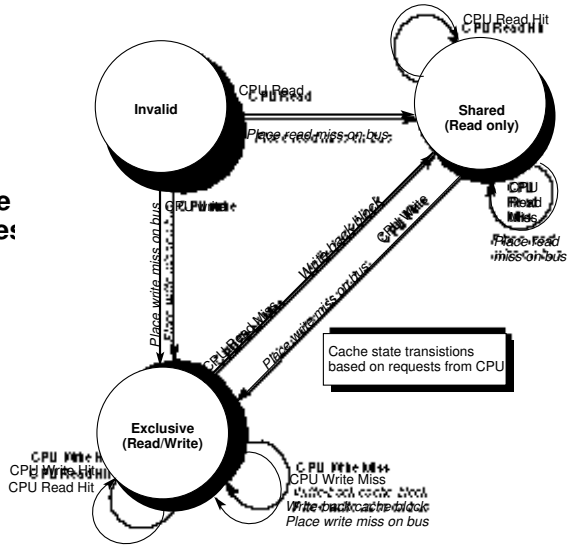
An Example Snoopy Protocol

- **Invalidation protocol, write-back cache**
- **Each block of memory is in one state:**
  - **clean in all caches and up-to-date in memory**
  - **dirty in exactly one cache**
  - **not in any caches**
- **Each cache block is in one state:**
  - **Invalid: block contains no data**
  - **Shared: block can be read**
  - **Exclusive: cache has only copy, its writeable, and dirty**
- **Read misses: cause all caches to snoop**
- **Writes to clean line are treated as misses**

© Patterson 1994

Snoopy-Cache State Machine-I

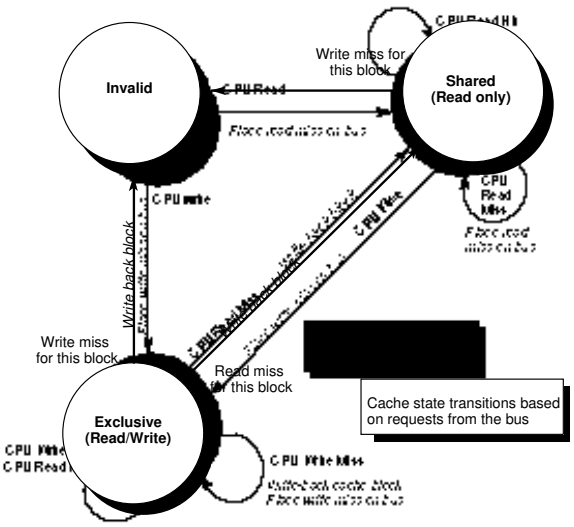
- **State machine for *CPU* reques**



© Patterson 1994

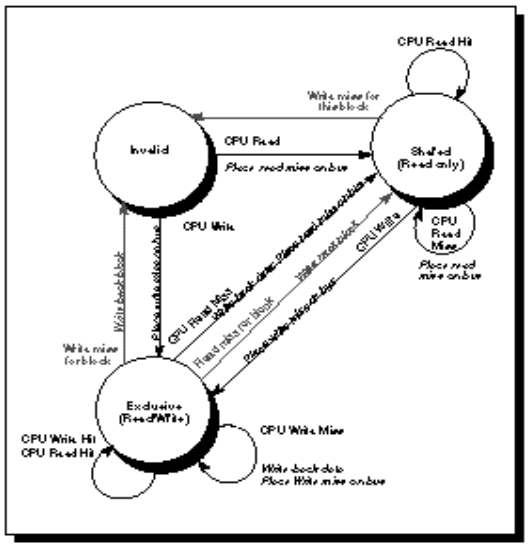
Snoopy-Cache State Machine-II

- **State machine for *bus* reques**



© Patterson 1994

Snoop Cache: State Machine



Extensions:

- **clean-> dirty, need invalidate only (upgrade request)**
- **clean exclusive state (no miss for private data on write)**

© Patterson 1994

Implementation Complications

- **Write races:**
  - **cannot update cache until bus is obtained**
    - » **otherwise, another processor may get bus first, and write the same cache line**
  - **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=> multiple outstanding transactions for a line**
    - » **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**

© Patterson 1994

Example: Assume x1, x2 map to same cache block

Step	P1	P1 Cache State		P2	P2 Cache State x2		Bus
		x1	x2		x1	x2	
0.		Inv	Inv		Inv	Inv	
1.	Write x1						
2.				Read x1			
3.	Write x1						
4.				Write x2			
5.	Read x2						

© Patterson 1994

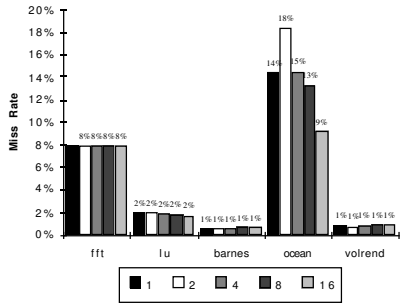
Implementing Snooping Caches

- 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 contiuously snoop on address bus
  - If address matches tag, either invalidate or update
- Bus serializes writes, getting bus ensures no one else can performa 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

© Patterson 1994

Miss Rates for Snooping Protocol

- 4th C: Coherency Misses
- More processors: increase Coherency, decrease Capacity



Fast Fourier Transform (FFT): Matrix transposes + comp.

LU factorization of dense 2D matrix (linear algebra)

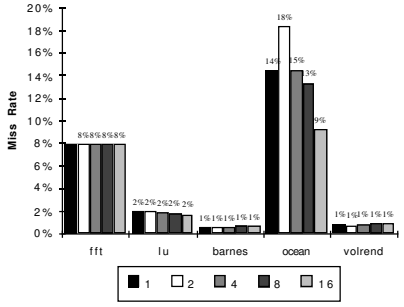
Barnes-Hut n-body algorithm solving galaxy evolution probem

Ocean simluates influence of eddy & boundary currents on large-scale flow in ocean: dynamic arrays per grid

VolRend is parallel volume rendering: scientific visualization

© Patterson 1994

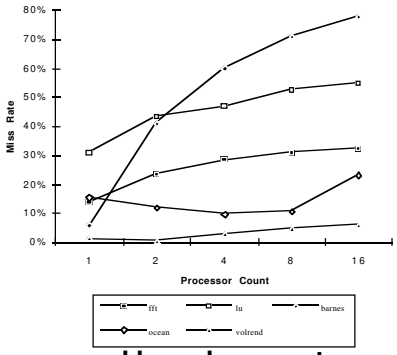
Miss Rates for Snooping Protocol



- Cache size is 64KB, 2-way set associative, with 32B blocks.
- With the exception of Volrend, the misses in these applications are generated by accesses to data that is potentially shared.
- Except for Ocean, data is heavily shared; in Ocean only the boundaries of the subgrids are shared, though the entire grid is treated as a shared data object. Since the boundaries change as we increase the processor count (for a fixed size problem), different amounts of the grid become shared. The analmous increase in miss rate for Ocean in moving from 1 to 2 processors arises because of conflict misses in accessing the subgrids.

© Patterson 1994

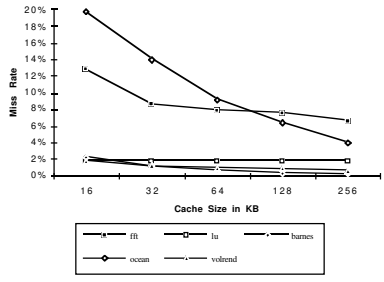
% Misses caused by cohernecy traffic vs. Processors



- % cache misses caused by coherency transactions typically rises when a fixed size problem is run on more processors.
- The absolute number of coherency misses is increasing in all these benchmarks, including Ocean. In Ocean, however, it is difficult to separate out these misses from others, since the amount of sharing of the grid varies with processor count.
- Invalidations increases significantly; FFT the miss rate arising from coherency misses increases from nothing to almost 7%.

© Patterson 1994

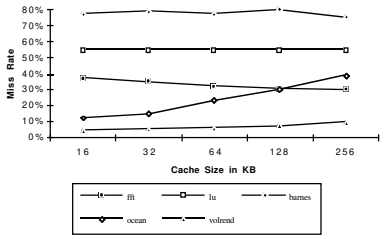
Miss rates as increase cache size/processor



- Miss rate drops as the cache size is increased, unless the miss rate is dominated by coherency misses.
- The block size is 32B and the cache is 2-way set-associative. The processor count is fixed at 16 processors.

© Patterson 1994

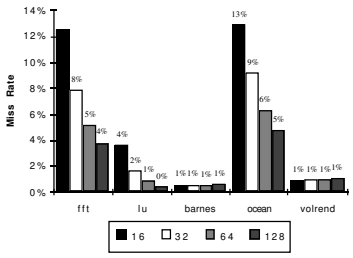
% Misses caused by cohernecy traffic vs. Cache size



© Patterson 1994

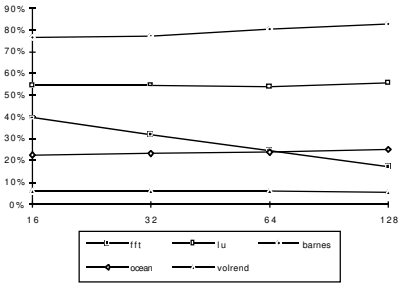
Block Size

- Since cache block hold multiple words, may get coherency traffic for unrelated variables in same block
- False sharing arises from the use of an invalidation-based coherency algorithm. False sharing occurs when a block is invalidated (and a subsequent reference causes a miss) because some word in the block, other than the one being read, is written into.



© Patterson 1994

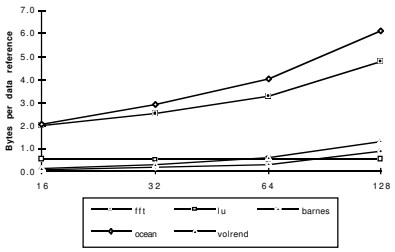
% Misses caused by cohernecy traffic vs. Block size



- FFT communicates data in large blocks & communication adapts to the block size (it is a parameter to the code); makes effective use of large blocks.
- Ocean competing effects that favor different block size
  - accesses to the boundary of each subgrid, in one direction the accesses match the array layout, taking advantage of large blocks, while in the other dimension, they do not match. These two effects largely cancel each other out leading to an overall decrease in the coherency misses as well as the capacity misses.

© Patterson 1994

Bus Traffic as increase Block Size



- Bus traffic climbs steadily as the block size is increased.
- Volrend the increase is more than a factor of 10, although the low miss rate keeps the absolute traffic small.
- The factor of 3 increase in traffic for Ocean is the best argument against larger block sizes.
- Remember that our protocol treats ownership misses the same as other misses, slightly increasing the penalty for large cache blocks: in both Ocean and FFT this effect accounts for less than 10% of the traffic.

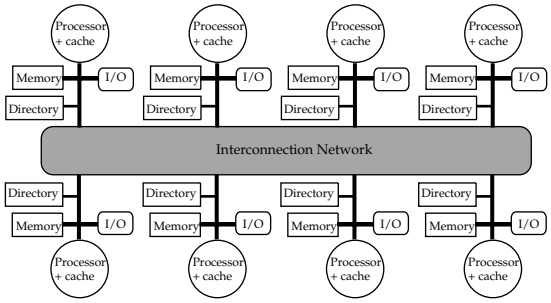
© Patterson 1994

Larger MPs

- Separate Memory per Processor
- Local or Remote access via memory controller
- Cache Coherency solution: non cached pages
- Alternative: directory/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?
  - in memory => simpler protocol (one location)
  - 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

© Patterson 1994

Distributed Directory MPs



© Patterson 1994

Directory protocol

- Similar to snoopy protocol: 3 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 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.

© Patterson 1994

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 either from the home memory.*			
Data write-back	Remote cache	Home directory	A, Data
– *Write-back a data value for address A.*			

© Patterson 1994

Example Diretory Protocol

- Message sent to directory causes 2 acitons:
  - update the directory
  - more messages to satifty 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 set is to identity of requesting processor. The state of the block is made Exclusive.

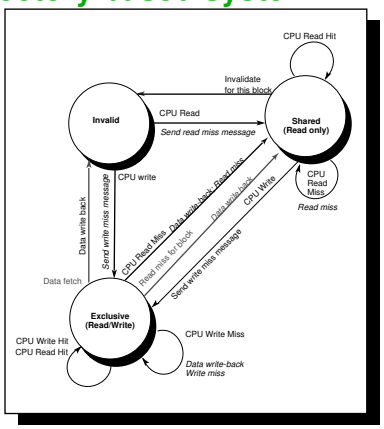
© Patterson 1994

Example Diretory Protocol

- block is Exclusive: current value of the block is held in the cache of the processor identified by the set Sharers (the owner), & 3 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 send to the requesting processor, which becomes the new owner. Sharers is set to identity of new owner, and state of block is made Exclusive.

© Patterson 1994

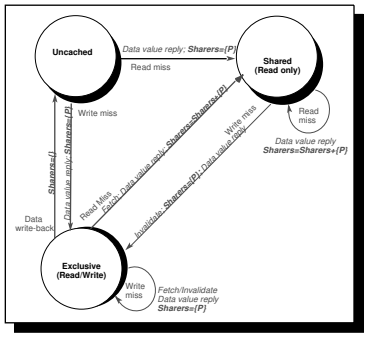
state transition diagram for an individual cache block in a directory based system



- **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.**

© Patterson 1994

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.**

© Patterson 1994

Example: Assume x1, x2 map to same cache block

Step	P1	P1 Cache State		P2	P2 Cache State x2		Bus
		x1	x2		x1	x2	
0.		Inv	Inv		Inv	Inv	
1.	Write x1						
2.				Read x1			
3.	Write x1						
4.				Write x2			
5.	Read x2						

© Patterson 1994