Multiprocessors

- **Potential for both performance and reliability**
- **Shared, uniform memory access vs. Shared non-uniform memory access vs. Message Passing**
- **Cache coherency protocols: Snooping vs. directory**
- **Successful today for file servers, time sharing, databases**
- **Will parallel programming become popular for production programs?**

Why is programming hard?

- **Communication: analogy % communication in committee as number of people goes up**
- **Everything speedup to achieve great speedup**
- **Real questions:**
  - **(1) Do problems exist with sufficient parallelism?**
  - **(2) Can people/compilers be trained to exploit parallelism?**

Multiprocessor taxonomy:

1. **Single instruction stream, single data stream (SISD, the uniprocessor)**
2. **Single instruction stream, multiple data streams (SIMD)**
3. **Multiple instruction streams, single data stream (MISD)**
4. **Multiple instruction streams, multiple data streams (MIMD)**

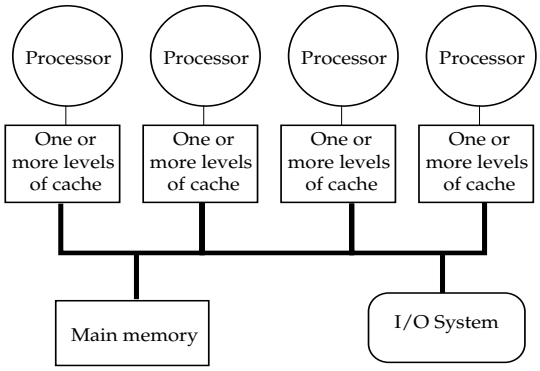
- **Multiple compared to what? SISD=Pentium**

SIMD vs. Vector?

- **ISA same**
- **N parallel streams & adders vs pipeline**
- **(Array processor)**
- **Single Program Counter (a "uniprocessor")**
- **Best case: data parallelism, if not lots of data, nothing to do**
- **Worst case: case statement, only singe data item**

Small-Scale MIMD Designs

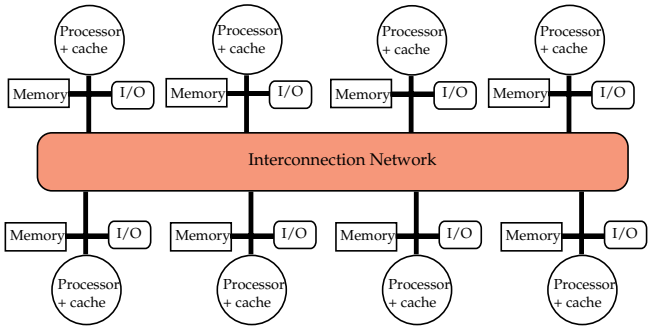
- **Memory: centralized with uniform access time and bus interconnect**
- **Examples: SPARCCenter, Challenge, SystemPro**



MPs.intro 5

Large-Scale MIMD Designs

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



MPs.intro 6

The Bet (made in 1989)

- **Danny Hillis believes that while supercomputers with a small number of data streams may be best sellers, the biggest machines will be machines with many data steams, and these will perform the bulk of the computations. Gordon Bell bet Hillis that in the last quarter of calendar year 1995 more sustained MFLOPS will be shipped in machines using few data streams (<100) rather than many data streams (>1000). This bet concerns only supercomputers, defined as machines costing more than $1,000,000 and used for scientific applications. Sustained MFLOPS is defined for this bet as the number of floating-point operations per month, so availability of machines affects their rating. The loser must write and publish an article explaining why his prediction failed.**

MPs.intro 7

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**

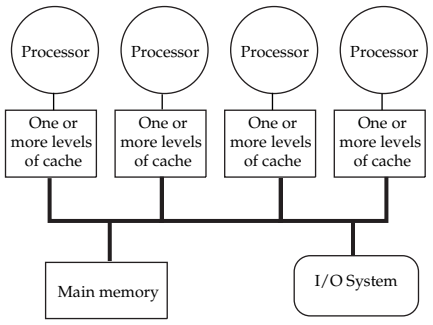
MPs.intro 8

Important Communication Properties

- **Bandwidth**
  - need high bandwidth in communication
  - cannot scale, but stay close
  - make limits in network, memory, and processor
  - overhead to communicate is a problem in many machines
- **Latency**
  - 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

Small-Scale, Shared Memory

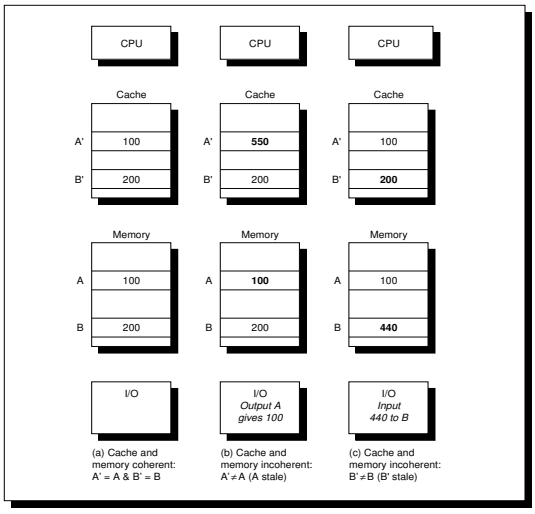
- **Caches serve both to**
  - increase bandwidth versus bus/memory
  - reduce latency of access
  - valuable for both private data and shared data



Key Issues for MPs

- **Measuring Performance**
  - **not just time on one size, but how performance scales with P**
  - **for fixed size problem (same memory per processor) and scaled up problem (fixed execution time)**
  - **care to compare to best uniprocessor algorithm, not just parallel program on 1 processor (unless its best)**
- **Multilevel caches, cohernecy, and inclusion**
  - **get duplication by using coherency at level of second cache provided data of all higher levels are included**
  - **block size of L1 & L2?**
- **Nonblock caches and Prefetching**
  - **more latency to hide so nonblocking more important**
  - **predicated on free memory bandwidth, may be in conflict with other processors**
- **Virtual Memory to get Shared Memory MP: Distributed VIrtual Memory (DVM); pages are units of coherency**

The Problem of Cache Coherency



What does coherency mean?

- **Informally:**
  - **any read must return the most recent write (too strict)**
- **Better:**
  - **any write must eventually be seen by a read**
  - **all writes are seen in order**
- **Two rules to ensure this:**
  - **If P writes x and P1 reads it, P's write will be seen if the read and write are sufficiently far apart.**
  - **Writes to a single location are serialized: seen in one order**
    - » **latest write will be seen**
    - » **could see writes in illogical order (could see older value after a newer value)**

Potential Solutions

- **Snooping solution(snoopy bus):**
  - **Send all requests for data to all processors.**
  - **Processors snoop to see if they have a copy and respond accordingly**
  - **requires broadcast, since caching information is at processors**
  - **works well with bus**
- **Directory based schemes**
  - **keep track of what is being shared in one place**
  - **distributed memory => distributed directory**
  - **send point-to-point requests to processors**

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**

Basic Snoopy Protocols

- **Write invalidate versus broadcast:**
  - **invalidate requires one transaction per write-run**
  - **invalidate uses spatial locality: one transaction per block**
  - **broadcast has lower latency between write and read**
  - **BW vs. latency tradeoff**

Name	Protocol Type	Memory-write policy	Machines using
Write Once	Write invalidate	Write back after first write	First snoopy protocol.
Synapse N+1	Write invalidate	Write back	1st cache-coherent MPs
Berkeley	Write invalidate	Write back	Berkeley SPUR .
Illinois	Write invalidate	Write back	SGI Power and Challenge
"Firefly"	Write broadcast	Write back private, Write through shared	SPARCCenter 2000

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

Snoopy Example: Assume cache block = 1 word

Step	P1 Cache Adr/State/Val	P2 Cache Adr/State/Val	Bus P/S/A/V	Memory Adr/Val

1. P1: Write A1
2. P1: Read A1
3. P2: Read A1
4. P2: Write A1

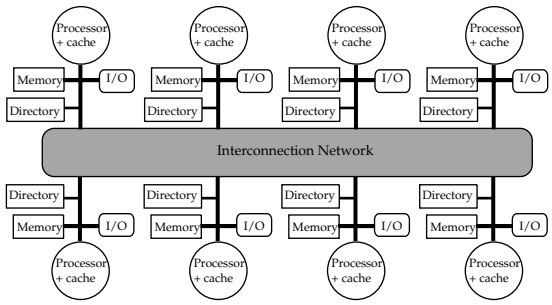
Implementing Snooping Caches

- 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 perform an 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/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

Distributed Directory MPs

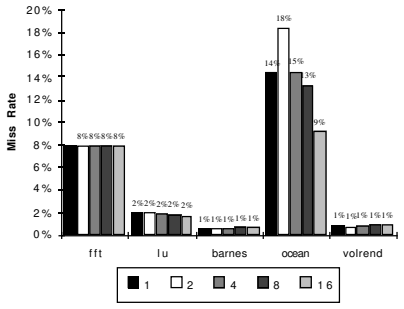


Directory protocol

- **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.**

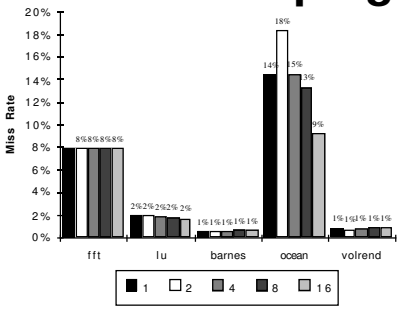
Miss Rates for Snooping Protocol

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



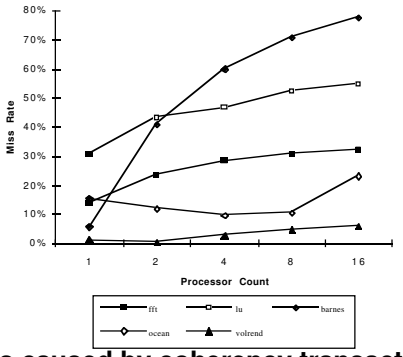
  - **Fast Fourier Transform (FFT): Matrix transposes + comp.**
  - **LU factorization of dense 2D matrix (linear algebra)**
  - **Barnes-Hut n-body algorithm solving galaxy evolution probem**
  - **Ocean simluates influence of eddy & boundary currents on large-scale flow in ocean: dynamic arrays per grid**

Miss Rates for Snooping Protocol



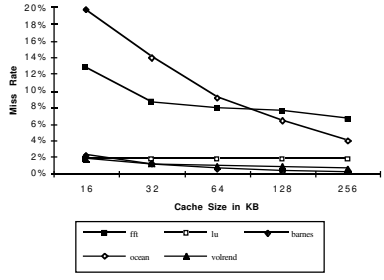
  - **Cache size is 64KB, 2-way set associative, with 32B blocks.**
  - **With the exception of Volrend, the misses in these applications are generated by accesses to data that is potentially shared.**
  - **Except for Ocean, data is heavily shared; in Ocean only the boundaries of the subgrids are shared, 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.**

% 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%.**

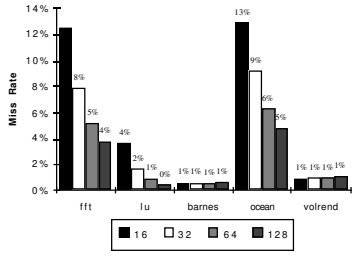
Miss rates as increase cache size/processor



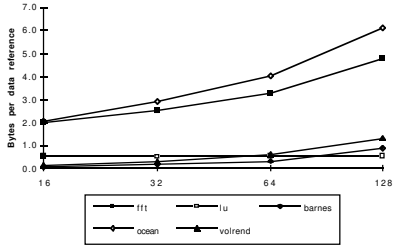
- **Miss rate drops as the cache size is increased, unless the miss rate is dominated by coherency misses.**
- **The block size is 32B and the cache is 2-way set-associative. The processor count is fixed at 16 processors.**

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.**

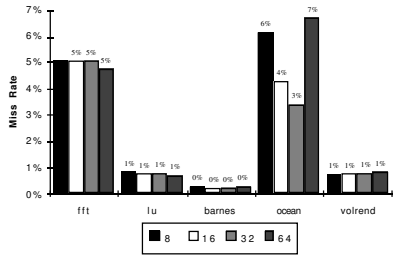


Bus Traffic as increase Block Size



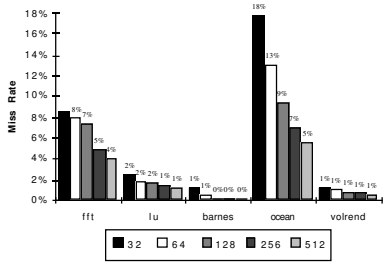
- **Bus traffic climbs steadily as the block size is increased.**
- **Volrend the increase is more than a factor of 10, although the low miss rate keeps the absolute traffic small.**
- **The factor of 3 increase in traffic for Ocean is the best argument against larger block sizes.**

Miss Rates for Directory



- **Cache size is 128 KB, 2-way set associative, with 64B blocks.**
- **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 increase in miss rate for Ocean in moving from 32 to 64 processors arises because of conflict misses in accessing small subgrids & for cohernecy misses for 64 processors.**

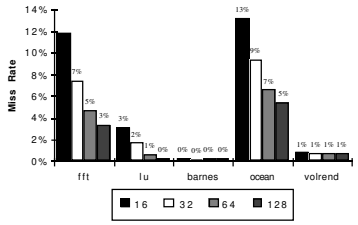
Miss rates as increase cache size/processor for Directory



- **Miss rate drops as the cache size is increased, unless the miss rate is dominated by coherency misses.**
- **The block size is 64B and the cache is 2-way set-associative. The processor count is fixed at 16 processors.**

Block Size for Directory

- **Assumes 128 KB cache & 64 processors**



Synchronization

- **Why Synchronize? know when its safe for different processes to use shared data**
- **3 Issues for Syncronization**

1) Uninterruptable instruction to fetch and update memory (atomic operation)

2) User level synchronization operation using this primitive

3) For large scale MPs, synchronization can be a bottleneck; techniques to reduce contention and latency of synchronization

1) Uninterruptable instruction to fetch and update memory

- Atomic exchange: interchange a value in a register for a value in memory
  - 0 => synchronization variable is free
  - 1 => synchronization variable is locked and unavailable
  - Set register to 1 & swap
  - New value in register determines success in getting lock
    0 if you are first
    1 if some other processor had already claimed access
  - key is that exchange is indivisible
- Test-and-set: tests a value and sets it if the value passes the test
- Fetch-and-increment: it returns the value of a memory location and atomically increments it
  - 0 => synchronization variable is free

MPs.intro 33

2) User level synchronization operation using this primitive

- spin locks: processor continuously tries to acquire, spinning around a loop trying to get the lock

```
            li    R2,#1
lockit:     exch  R2,0(R1)     ;atomic exchange
            bnez  R2,lockit    ;already locked?
```

- What about MP with cache coherency?
  - Want to spin on cache copy
  - Likely to get cache hits for such variables
- Problem: exchange includes a write, which invalidates all other copies => lots of bus traffic
- Solution: first repeated just read variable; when it changes, then try exchange:

```
try:        li    R2,#1
lockit:     lw    R3,0(R1)     ;load var
            bnez  R3,lockit    ;not free=>spin
            exch  R2,0(R1)     ;atomic exchange
            bnez  R2,try       ;already locked?
```

MPs.intro 34

3) For large scale MPs, synchronization can be a bottleneck

- Example: 20 processors try to lock, 50 cycles for bus

Read miss all waiting processors to fetch lock	1000
Write miss by releasing processor and invalidates	50
Read miss by all waiting processors	1000
Write miss by all waiting processors , one successful lock, & invalidate all copies	1000
Total time for 1 proc. to acquire & release lock	3050

- Each time one gets a lock, drops out of competition= 1525
- 20 x 1525 = 30,000 cycles for 20 processors to pass through the lock
- Problem is contention for lock and serialization of lock access: once lock is fre all compete to see who gets it
- Alternative: create a list of waiting processors, go through list: called a "queuing lock"
  - special HW to recognize 1st lock access & lock release
- Another laternative: fetch-and-increment; can be used to create barrier; wait until everyone reaches same point

Another MP Issue: Memory Consistency Models

- How consistent? When must a processor see the new value? e.g., seems that

```
P1:   A = 0;                  P2:   B = 0;
      .....                         .....
      A = 1;                        B = 1;
L1:   if (B == 0) ...         L2:   if (A == 0) ...
```

- Impossible for both if statements L1 & L2 to be true?
  - What if write invalidate is delayed & processor continues?
- Memory consitency models: what are the rules for such cases
- Sequential consistency: result of any execution be the same as if the accesses of each processor were kept in order and the accesses among different processors were interleaved => assignments before ifs above
- SC => delay all memory accesses until all invalidates done

Memory Consistency Model

- **Schemes faster execution to sequential consistency**
- **Not really an issue for most programs; they are synchronized**
  - **A program is synchronized if all access to shared data are ordered by synchronization operations**

    write (x)
    ...
    release (s) *{unlock}*
    ...
    acquire (s) *{lock}*
    ...
    read(x)
- **Only those programs willing to be nondeterminisitc are not sychronized**
- **Several Relaxed Models for Memory Consistency since most programs are synchronized: charcerized by their attitude towards: RAR, WAR, RAW, WAW to different addresses**