COSC4201
Multiprocessors

Prof. Mokhtar Aboelaze

Parts of these slides are taken from Notes by
Prof. David Patterson (UCB)

Multiprocessing

° **We are dedicating all of our future product development to multicore designs. We believe this is a key inflection for the industry**

Intel President Paul Otellini 2005

Introduction

- Microprocessors are getting more and more complex. Can we sustain the 2X/1.5Y anymore?
- The major obstacle for parallel processing is the software. Servers exhibits parallelism that could be exploited using multiprocessors.
- ILP is limited by the application (even if we have unlimited issues.
- Vector processors is very good for scientific applications, but not very useful in desktops or servers.

Processor	Micro architecture	Fetch / Issue / Execute	FU	Clock Rate (GHz)	Transis-tors Die size	Power
Intel Pentium 4 Extreme	Speculative dynamically scheduled; deeply pipelined; SMT	3/3/4	7 int. 1 FP	3.8	125 M 122 mm^2	115 W
AMD Athlon 64 FX-57	Speculative dynamically scheduled	3/3/4	6 int. 3 FP	2.8	114 M 115 mm^2	104 W
IBM Power5 (1 CPU only)	Speculative dynamically scheduled; SMT; 2 CPU cores/chip	8/4/8	6 int. 2 FP	1.9	200 M 300 mm^2 (est.)	80W (est.)
Intel Itanium 2	Statically scheduled VLIW-style	6/5/11	9 int. 2 FP	1.6	592 M 423 mm^2	130 W

Definitions

° **Flynn's classification**

- ***Single instruction stream, single data stream*** SISD
  - Uniprocessor
- ***Single instruction stream, multiple data streams*** SIMD
  - Same instruction is sent to every processor, Illiac IV
- ***Multiple instruction stream, single data stream*** MISD
  - No commercial multiprocessor ever built Systolic arrays?
- ***Multiple instruction stream, multiple data stream*** MIMD
  - Each processor has its own instruction and data, IBM SP2, NCUBE, SUN T1 (8 processors, 32 threads)

Definition

° **In MIMD, each processor is executing its own instruction stream.**

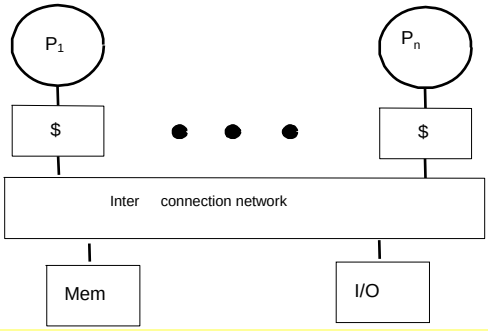
° **Multiple processors executing a single program sharing code and data (*threads*).**

° **The term *threads* sometimes is loosely defined to include processes not sharing address space.**

° **The thread could be a function or a subroutine written by the programmer, or one iteration of a loop extracted by the compiler.**

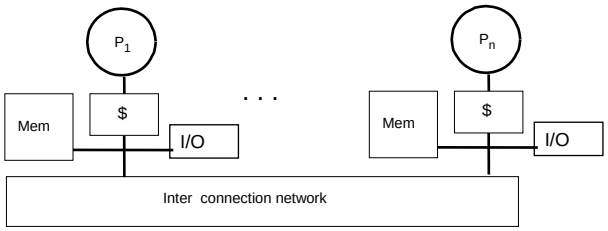
Centralized Shared Memory

Relatively small number of processors
Processors share a single centralized (maybe interleaved or banked) memory
An interconnection network (maybe a bus) is used to access the memory
Also known as shared memory multiprocessors SMP and the style is known as UMA (uniform memory access)



Distributed Memory Multiprocessors

- **Can support a larger number of processors than SMP.**
- **Usually a better-than-bus interconnection network is used.**
- **Every processor has its own memory and I/O**
- **Local memory is accessed faster than remote memory (message passing or shared memory?)**



Models for Communication and Memory

- Communication occurs through a shared address space (memories could be physically separate, but logically form a single memory). DSM, or NUMA.
- The same physical address means the same thing at each processor.
- Message Passing Multiprocessors: each processor has its own memory, communication is performed by passing messages between processors.
- Communication can be performed from the reader or writer point of view.

Performance Metrics

- Communication bandwidth:
  - Ideally it should be determined by the processor, memory, and interconnection network bandwidth.
  - The architecture of the node, and the communication mechanism may affect the bandwidth.
  - Buffer management and resource occupancy can affect bandwidth

Performance Metrics

° Communication latency:

- The latency is sender overhead, + receiver overhead + time of flight + transmission time
- Time of flight, and transmission time are fixed.
- The overhead at both sender and receiver (H/W and S/W) determine the latency.
- Overhead includes naming (remote addresses), protection handling, and communication mechanism.
- Does it include O/S or not ?

Performance Metrics

° Communication latency Hiding:

- How will we can hide latency?
- Can we overlap communication with computation or another communication?
- Application dependent

Compariosn

° Shared Memory

- **Compatibility with the well-understood model**
- **Ease of programming**
- **Lower overhead for small amount of data transfer**
- **The ability to use hardware controlled caching**

° Message Passing

- **Simpler hardware especially compared to supporting coherent caching for remote data.**
- **Communication is explicit, simpler to understand**
- **Explicit communication focuses programmer attention on it.**
- **Easier to do sender initiated communication**

Array sum: Message Passing

```
#define ASIZE 1024
#define NUMPROC 4
double myArray[ASIZE/NUMPROC];
double mySum=0;
for( i=0;i<ASIZE/NUMPROC;i++)
  mySum+=myArray[i];
```

```
if(myPID=0){
  for(int
 p=1;p<NUMPROC;p++){
    int pSum;
    recv(p,pSum);
    mySum+=pSum;
  }
  printf("Sum:
 %lf\n",mySum);
}else
  send(0,mySum);
```

Array Sum: Shared Memory

```
#define ASIZE 1024
#define NUMPROC 4
shared double array[ASIZE];
shared double allSum=0;
shared mutex sumLock;
double mySum=0;
for(int
  i=myPID*ASIZE/NUMPROC;i<(
  myPID+1)*ASIZE/NUMPROC;i+
  +)
  mySum+=array[i];
lock(sumLock);
allSum+=mySum;
unlock(sumLock);
if(myPID=0)
  printf("Sum:
  %lf\n",allSum);
```


Implementation

- The desired communication model can be created on top of a hardware model that supports either.
- Supporting message passing using shared memory is easing, message passing is simply copying data between 2 locations (How to deal with a memory system that is oriented towards transfer of cache lines? Performance degradation).
- Supporting shared memory on top of message passing is not easy. All shared memory reference must involve the operating system for translation and memory protection.

Workload

- Commercial Workload
- Online Transaction-Processing (OLTP)
  - Using Oracle 7.3.2
  - Clients generating requests and servers responding to them
- Decision Support System (DSS)
  - Using also Oracle 7.3.2
  - Parallelism within and across queries
- Web index search (AltaVista)
  - Search on a mapped version of the AltaVista database

Workload

- Scientific
- FFT Kernel
- LU Kernel
- Barnes Application
  - *n*-body algorithm
- Ocean application
  - Simulates eddy and boundary currents on large-scale flow in the ocean
  - Solve elliptical partial differential equations using Gauss Seidel multigrid technique

Computation/Communication

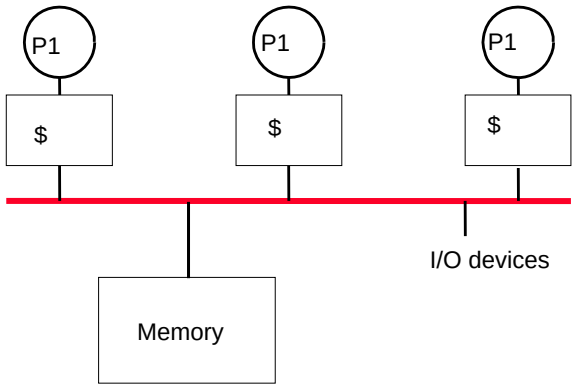
- **Computation to communication ratio determine the speedup in multiprocessors.**
- **Also the scaling of computation and communication vs. the number of processors.**
- **Assume *p* is the number of processors and *n* is the problem size**

Application	Scaling of computation	Scaling of communication	Scaliong of Com/Com
FFT	***(nlogn)/p***	***n/p***	***log n***
LU	***n/p***	$\sqrt{n}/\sqrt{p}$	$\sqrt{n}/\sqrt{p}$
Barnes	***(nlogn)/p***	$\sqrt{n}\log n/\sqrt{p}$	$\approx \sqrt{n}/\sqrt{p}$
Ocean	***n/p***	$\sqrt{n}/\sqrt{p}$	$\sqrt{n}/\sqrt{p}$

Symmetric Shared Memory

- For symmetric shared memory multiprocessors, the memory is equally accessible by all the processors.
- Memory bandwidth is usually a problem with SMP
- To alleviate the bandwidth problem, caches are used.
- Private and Shared data are cached
- Cache coherence problem

SMP



Cache Coherence

° A memory system is coherent if

1. A read by P to location X that follows a write by P to location X with no writes to X in between (by any processor) returns the value written by P.
2. A read by processor p1 to X that follows a write by P2 to X returns the value written by P2 if the read and write are sufficiently separated in time, and no other writes to X occurred between the two accesses.
3. Writes to the same location are *serialized* Two writes by two processors to the same location are seen in the same order by all processors

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

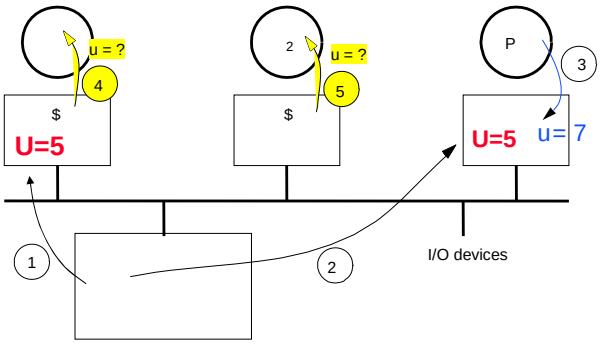
⇒ if a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

° These restrictions allow the processor to reorder reads, but forces the processor to finish writes in program order

Cache Coherence

- In coherent multiprocessors, the cache provide both migration and replication of shared data items
- Migration: Data can move from the main memory to the cache for faster access (and less memory B.W.)
- Replication: An item may reside in more than one cache where it will be read by many processors without consuming memory bandwidth.

Example



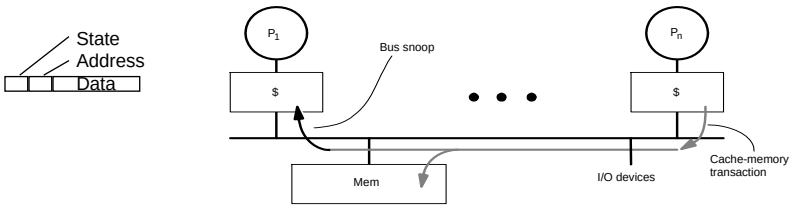
Assuming a write back cache

Different processors see different values for the same variable

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**

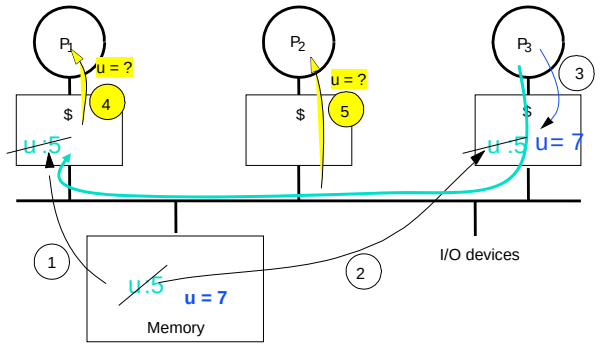
Snoopy Cache Coherence Protocol



° **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**

Snoopy Cache Coherence Protocols



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

Comparison

- Multiple writes to the same word with no intervening reads require multiple write broadcast for an update protocol, and one invalidate for invalidate protocols.
- With multiword cache blocks, write to multiple words (bytes) in the same line require multiple broadcast, while only one invalidate (assuming no intervening reads).
- The delay between writing a word in a processor, and reading it by another processor is less in write update

Implementation Techniques

- A shared bus
- All processors continuously snoop on the bus checking if the address on the bus is in their cache or not.
- A shared bus enforces serialization
- We need to locate the requested data.
  - In a write through, the memory
  - Ina write back, may be another processor
- For writes, we need to know if there are other copies shared of the block or not.
  - If yes, we have to invalidate on a write
  - If not, no invalidation is necessary

Resources

- 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

Resources

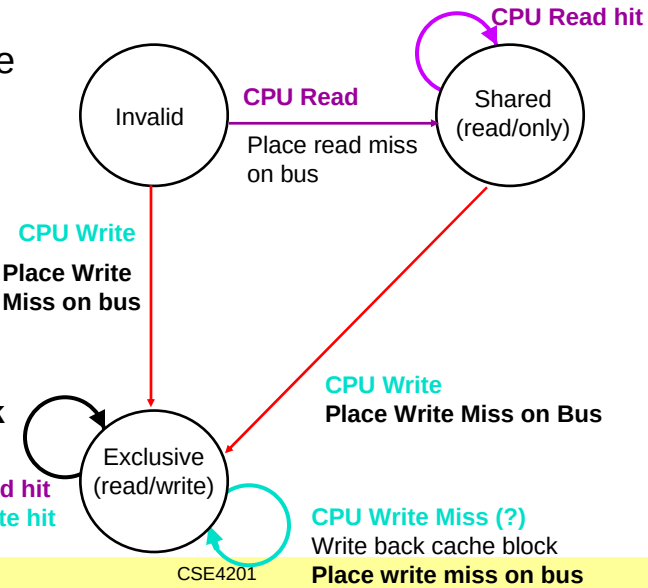
- To track whether a cache block is shared, add extra state bit associated with each cache block, like valid bit and dirty bit
  - Write to Shared block ⇒ Need to place invalidate on bus and mark cache block as private (if an option)
  - No further invalidations will be sent for that block
  - This processor called owner of cache block
  - Owner then changes state from shared to unshared (or exclusive)

Example 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

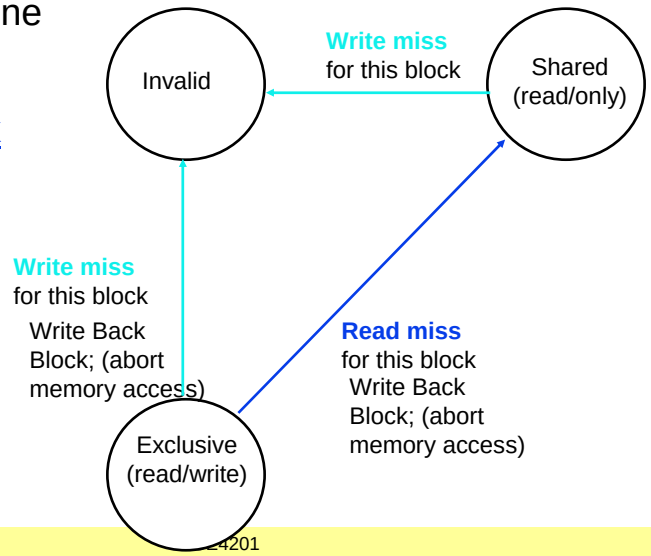
Write Back: CPU requests

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



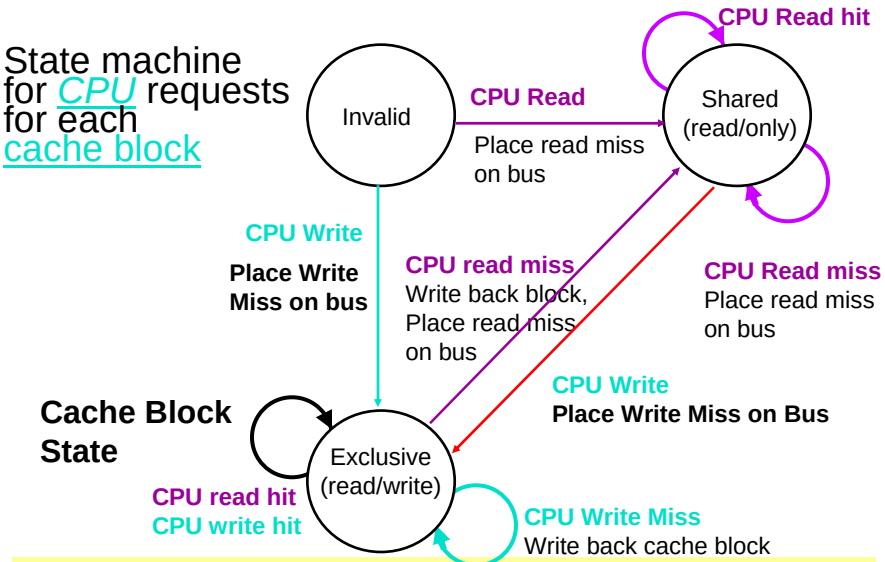
Write Back: Bus requests

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



Write Back: Block Replacement

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



Write Back: Write Back Cache

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