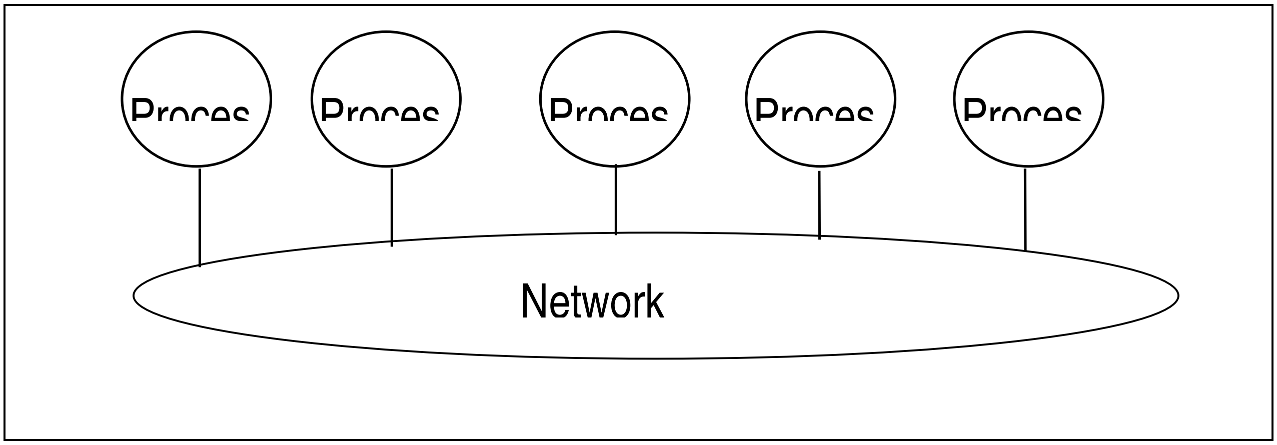
Lecture #34

****

Review -- 1 min

****

Convergence of MP designs



Design questions

- Interface to network (want low overhead, high BW)
- Network design (scalable, low latency, high BW)

Key challenge

programming MPs

****

Outline - 1 min

****

Programming MP's: shared memory v. message passing

NUMA v. UMA

Network: Bus v. Switched

Consistency v. Coherency

Cache consistency

CC-numa v. non-cc-numa

Snooping v. Directories

Snooping Protocol

****
Preview - 1 min
****
Directory protocol
Consistency models
NOW, SGI Origin

****
Lecture - 20 min
****
fundamental issues

naming/programming model
synchronization
latency, bandwidth
consistency, coherency

Programming model – shared memory v. msg passing

Shared Memory	Message Passing
easier to program	interface makes costly operations explicit
lower latency	less hardware, easier to design
easier to do HW controlled caching	

Message passing v. shared memory

Message Passing
<slide>

- Computers (processor + memory) communicate via network
- Explicit network commands to access other computers
- Send: specifies local buffer + receiving process on remote computer
- Receive: specifieds sending process on remote computer + local buffer to place data
- pairwise syncrhonization – match send and receive;
    - other MP models relax sync for better performance (e.g. AM)

History of Message Passing

- ♦ network topology important b/c could only send to immediate neighbor
- ♦ typically synchronous, blocking sends and receives
- ♦ later: DMA w. non-blocking sends; DMA for recv.; DMA to buffer until processor does receive, then copy to correct destination in processor local memory
- ♦ later: SW libraries to allow arbitrary communication

Message Passing example: IBM SP-2 = RS6000 workstations in rack

- ♦ Network interface card has Intel 960
- ♦ 8x8 crossbar building block of network
- ♦ 40 MB/s link

Shared memory

- Communicate via Load and Store
- Usual model: share code, private stack, some shared heap, some private heap <slide>

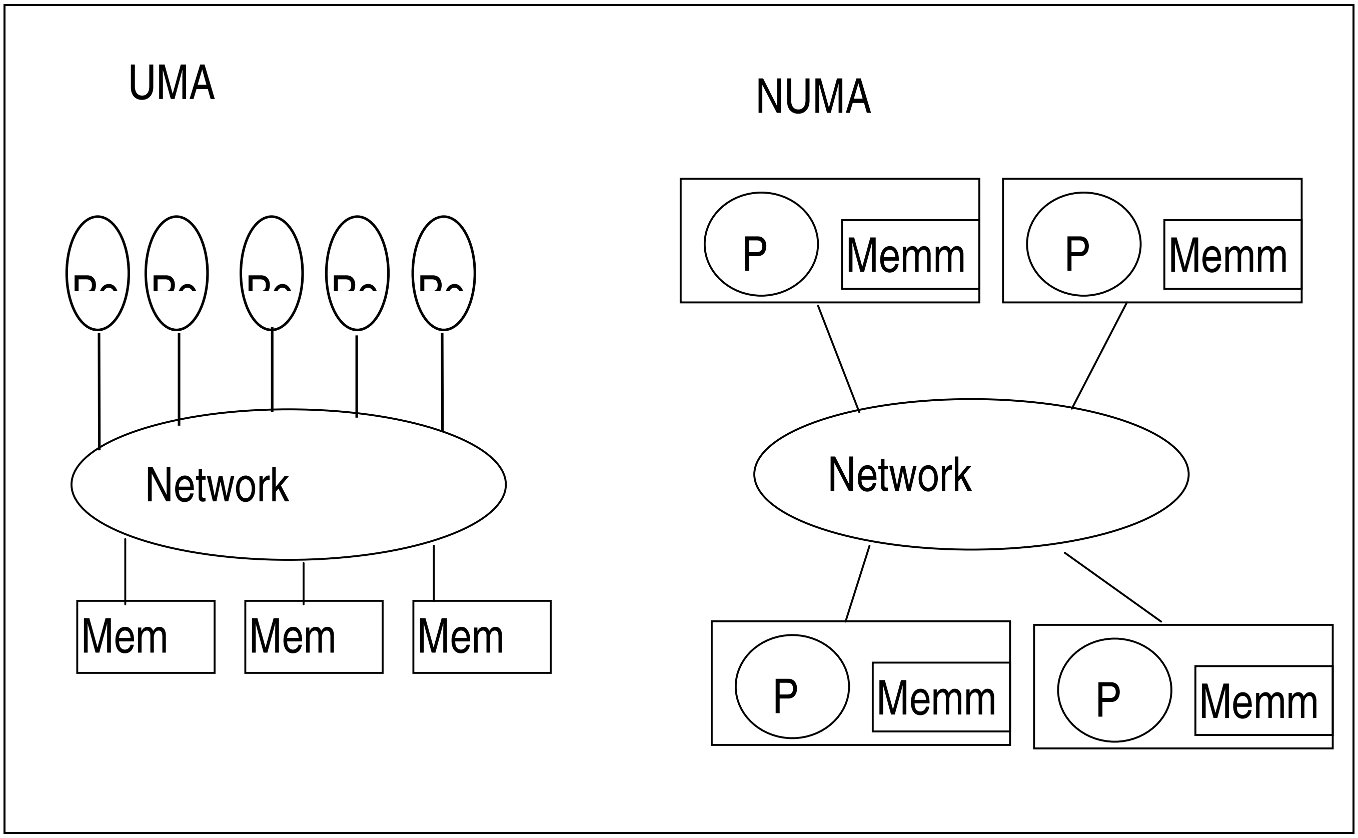
  - ♦ each processor can name every physical location in macine
  - ♦ data xfer via load,store
  - ♦ data size: byte, word, … cache block
  - ♦ Use virtual memory to map virtual to local or remote physical
  - ♦ memory hierarchy model applies: communication moves data to local cache (as load moves data from memory to cache)

Comparison: message passing v. shared memory

- Shared memory
  - proc communicate with shared address space
  - easy to implement 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 via messages
    - advantages
        - ♦ less hardware, easier to design
        - ♦ focuses attention on costly, non-local operations

UMA v. NUMA



UMA – Uniform memory access

aka SMP – symmetric multi-processing
network – usually a bus
good for small-scale systems

QUESTION: why hard for large systems?

examples – SGI challenge, Intel Systempro

<slide>

NUMA - nonuniform

each node: a processor (or several), caches, local memory

e.g. CRAY T3D

<slide>

UMA v. NUMA

QUESTION: advantage/DA of each

UMA – easier to program

NUMA – more scalable

Network for Shared Memory -- bus v. switched

QUESTION: advantage/DA of each

Bus – fast, but expensive to scale (adds pins)

<slide: bus BW v. year>

Multistage network: less expensive to expand than bus/crossbar

→ more bandwidth

<slide -- bus, crossbar, multistage>
<slide – cray t3e>

Synchronization

To cooperate, processes must coordinate

message passing – implicit coordination when data arrives/transmitted

shared adddress → addtional operations needed to expliclity coordinate e.g. set a flag, awaken a thread, interrupt a processor

Latency, bandwidth

Bandwidth

need high BW in communication

cannot scale, but stay close

make limits in network, memory, processor (not software)

communication overhead is 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

efforts to reduce latency increase hardware complexity (e.g. caching)

Latency hiding

Examples: prefetch, overlap send with computation

consistency, coherency

****

Admin - 3 min

****

Sermon: stay broad

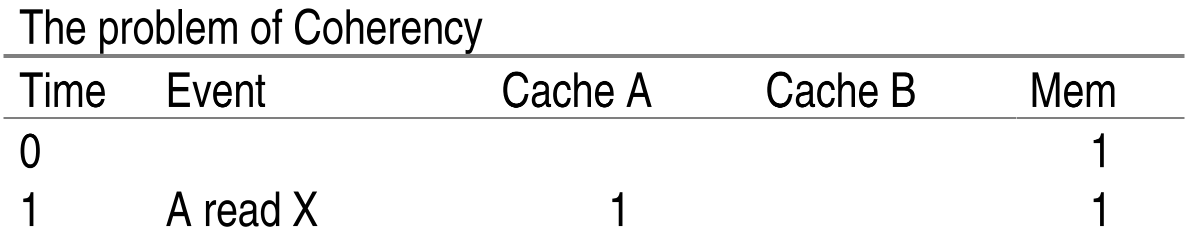
HW5 due today

****

Lecture - 24 min

****

Cache coherency and consistency



The problem of Coherency

Time	Event	Cache A	Cache B	Mem
0				1
1	A read X	1		1

2	B read X	1	1	1
3	A write 0 to X	0	1	0

What does coherency mean?
Informally:

- Any read must return most recent write
- Too strict – dificult to implement

Better:

- Any write must eventually be seen by a read
- All writes seen in order

Key idea – can a processor detect a bug in the memory system by looking at writes other processors make?
e.g. – another processor count to 10 by writing numbers into a memory location.

Are the following legal or illegal?

0 1 2 3 … 10 -- legal

0 4 10 – legal (I might have been slow)

0 2 1 3 4 5 6 7 8 9 10 – illegal – something bad happened to memory system

0 1 2 3 4 5 6 7 8 9 9 9 9 9 9 9 … -- after a couple days, I can suspect that it is illegal

Consistency v. Coherency
Note: there is another way to detect that the memory system is playing tricks on me

Suppose other processor counts to 10 by first writing to addresss A and then writing to B and I see

A: 0, B: 0, A: 1, B: 1, A: 1, B: 2, A: 1, B: 3 …

This is a consistency bug in the memory system

coherency – a single address looks right

consistency – multiple addresses are consistent

People (including me) don't always make distinction as clear as could be.

Approaches to coherency

Key idea: when a write occurs, there can be only one copy of the data (that way other copies won't be out of date)

Strategies:

1) No caching remote data
   example: cray T3D/T3E
2) Snooping
   - Send all requests for data to all processors
   - processors snoop and update cache state appropriately
   - requires a broadcast b/c cache state distributed
     - works well on a bus
     - Bus also provides point of serialization
   - dominates small scale machines (most of market)

3) Directory
   - Directory tracks sharing
   - distributed memory→distributed directory (avoid bottleneck)
   - send point-to-point requests to processors
     - scales better than snoop
   - Note: directory existed BEFORE snoop

Snooping Protocol

Simplified 3-state protocol

Slide: figure 8-11 H&P

2 notes on figure

1) left side (processor side) : figure is per cache block not per memory location – this is why we can have a read miss while we're in the shared state – read miss is to a different block that maps to this state
2) right side (bus side): notice the "abort memory access" action. This means that suppose I'm trying to do a write just as someone else is. They are in invalid and I am in exclusive, but they get the bus first – I have to go to invalid and restart my write request (otherwise – deadlock)

BUS SERIALIZES REQUESTS!

Snoopy details

1) Intermediate states
   <figure E.1>

   any set of actions that includes getting the bus is not atomic
   - ♦ other actions could happen while I'm waiting for bus

   <example – "Pending read" state>

2) Write back
   <figure E.1, again>
   key idea – writes are a 2-step process
   1) detect miss and request bus
   2) get bus, place miss on bus, get data, complete write

3) Split transaction bus
    more pending states

4) squash memory access
    when data in exclusive state, it must come from cache when anothe cache reads,
    but memory doesn't know cache state, so it will try to respond with wrong value

    solution – extra wire on bus that allows caches to signal memory when it has dirty data

    note: snooping takes variable time so caches signal with wired-or when it is OK to proceed (they release the line when they know they do NOT have the data)

5) 4-state protocol
    split exclusive to exclusive-clean and exclusive-dirty
    "MESI"

Summary:

1) Bus makes broadcast possible -→ allows snooping
2) Bus serializes requests → simplifies protocol
3) protocol more complicated than simple diagram

****

Summary - 1 min

****