Lecture #33

****

Review -- 1 min

****

Beyond the NI:

Media
Topologies
Routing
Connections and flow control
Topology

Finish up flow control: and topology

Switch topologies

Factors

degree – number of links from a node
diameter – max # links crossed between nodes
avg distance – number of hops to random destination
bisection – minimum number of links that separate the network into two halves

These factors relate to higher level properties

latency – diameter, distance
bandwidth – bisection
cost – degree (larger degree increases cost per switch and reduces number of switches)

Warnings against beautiful topologies

1) 3-d or N-d drawings must be mapped onto chip and boards

- elegant when sketched on blackboard may be awkward to build from chips, cables, boards, and boxes

2) subtlety – routing

up*down* routing leads to symmetries → all packets try to go through same link

e.g. 2-d mesh (see slide)

3) Simple, fast v. beautiful, slow

4) Behavior “in the limit” not terribly relevant

- Biggest machine = 2048 processors
- Most machines < 32 processors

Switch topology: Reliability
----
another consideration – how many nodes become disconnected when a switch fails? How many switches must fail to partition the network?

Solution – redundant connections, careful topologies

****

Outline - 1 min

****

Into to parallel architecture
- motivation
- applications
- arch categories
  - flynn categories
  - what level parallelism
  - shared address v. message passing
  - numa v. uma
  - cc-numa v. shared memory
- convergence of architectures
- fundamental issues
  - naming
  - synchronization
  - latency, bandwidth
  - consistency, coherency

****

Preview - 1 min

****

caching, consistency, and coherency
snooping v. directory-based coherency
synchronization
consistency models
case study: SGI Origin
case study: NOW

****

Lecture - 20 min

****
Motivation

Almasi and Gottlieb Highly Parallel Computing 1989
"A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast."

Key questions
- how large a collection?
- how powerful are processing elements?
- How do they cooperate and communicate?
- What type of interconnect?
- What are HW and SW primitives for programmer?
- Does it translate into performance (is it fast)?
  - Wouldn't voluntarily parallelize program if no more performance

Parallel processing: the dream of architects for 30 years – replicate processors to add performance v. design a faster processor
- rather than design a new processor from scratch (hard), just get people to buy more current processors
- Rather than come up with more complex ways to get ILP out of processors, get people to buy more current processors
- dream v. reality
  - dream: better performance, easier to program, more reliable
  - reality: ??? performance, HARD to program, less reliable

The hype:
speed of light limits → must go parallel (since at least 1972)

The reality:
1) processors faster today than 1972
at least 5 years of future improvements understood

beyond that?
I wouldn't bet against improved performance

2) MPP bankruptcy
KSR, TMC, Cray…

Jim gray: "Chapter 11"

Opportunities and Applications
----

1) Scientific computing

Grand challenge problems: climate simulation, 3-d airflow, nuclear weapons testing, human genome…

problem w scientific – much tied to govt funding
collapse of TMC, etc. due to end of cold war
Total NSF budget $3B – even if every penny went to buying parallel computers, that is still a small market v. PCs

Success in real industries
- petroleum – reservoir modeling
- automotive: crash simulation, drag analysis, engines
- aeronautics: airflow analysis, engine design, structural mechanics
- pharmaceuticals – molecular modeling

2) Commercial computing

Transaction processing & TPC-C benchmark

- all vendors serious about TPC are doing MPP
- range – small to medium scale

Others: CAD, entertainment (“Toy story”)…

Application challenge: Programming
<mortar shot picture Amber molecular dynamics 4 months of effort to go from 2x speedup on 8 processors to 5x speedup on 8>

- load balancing,
- optimize communication

“Attach of the killer micros”
----

1) uniprocessor performance closing in on supercomputers
<slide>
2) MPP performance surpassing vector supercomputers
<slide>
3) bus-based machines getting bigger, more common
<slide>
4) Top-500 – micros dominate; bus based micros increasing
<slide>

Categories of Parallel Architectures
----
basic questions
what level parallelism
how communicate
how cache?
…

What level parallelism?
----
1) Processor evolution: Bit-level v. instruction-level v. thread level?

<slide>

2) larger-scale parallelism (multiple processors)

Flynn categories
----

1. SISD – single instruction single data
   -- uniprocessors

2. MISD – multiple instruction single data
   -- ???

3. SIMD – single instruction multiple data
   --- examples – Illiac IV, CM-2

   <slide>

   simple programming model
      simple compiler model!
   low overhead
   flexibility
   all custom integrated circuits

   might also classify: vector machines, MMX as SIMD

4. MIMD – multiple instruction multiple data
   - examples : sun Enterprise 5000, Cray T3D, SGI origin
   - flexible
   - use off-the-shelf micros

****
Admin - 3 min
****
sign up for project presentations

quiz next week

****
Lecture - 24 min
****

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: specifies sending process on remote computer + local buffer to place data
- pairwise synchronization – 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 machine
    - ♦ 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

Shared memory: numa v. uma

UMA – uniform memory access
aka SMP – symmetric MP

design – interconnect (usu bus) connects all processors, memories, IO devices

Uniform access time to memory from all processors

good for small-scale systems
often built with shared bus, single memory
key to design – aggressive caching to reduce traffic to centralized resources

examples: SGI challenge, Intel systemPro

<slide>

NUMA – nonuniform memory access
each node = a processor (or several), caches, and local memory
example: CRAY T3D
<slide>

UMA v. NUMA
NUMA more scalable (most memory can be "far away")
UMA easier to program

<slide: Sun E5000 – NUMA or UMA?>
→ uma for simplicity

Shared memory: bus v. switched network
Have alluded to fact that NUMA more scalable than UMA b/c UMA usually built with a bus, but NUMA built with more complex network

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>

Shared memory: cc-numa v. non-cc numa

CC = cache coherence

basic problem – caches vital to performance (see above)

but, what happens when data change

→ cached data stale

fundamental rule – more copies of data makes reads faster,
but makes writes slower/more complex

cache consistency/coherency – how we keep multiple copies in caches in sync

CC adds complexity

T3E avoids by not keeping caches coherent

- only cache reads to local data
- all remote reads go to remote node
- "cache" remote data by copying to local memory
  - SW responsible for consistency

convergence of architectures

- Complete computers connected to scalable network
- different programming models place different requirements on communications layer
  - shared address space: tightintegration with memory to capture memory events that interact with others and to accept requests from other nodes

- message passing: send messages quickly and respond to incoming messages: tag match, allocate buffer, transfer data, wait for recv posting.

fundamental issues

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

****

Summary - 1 min

****