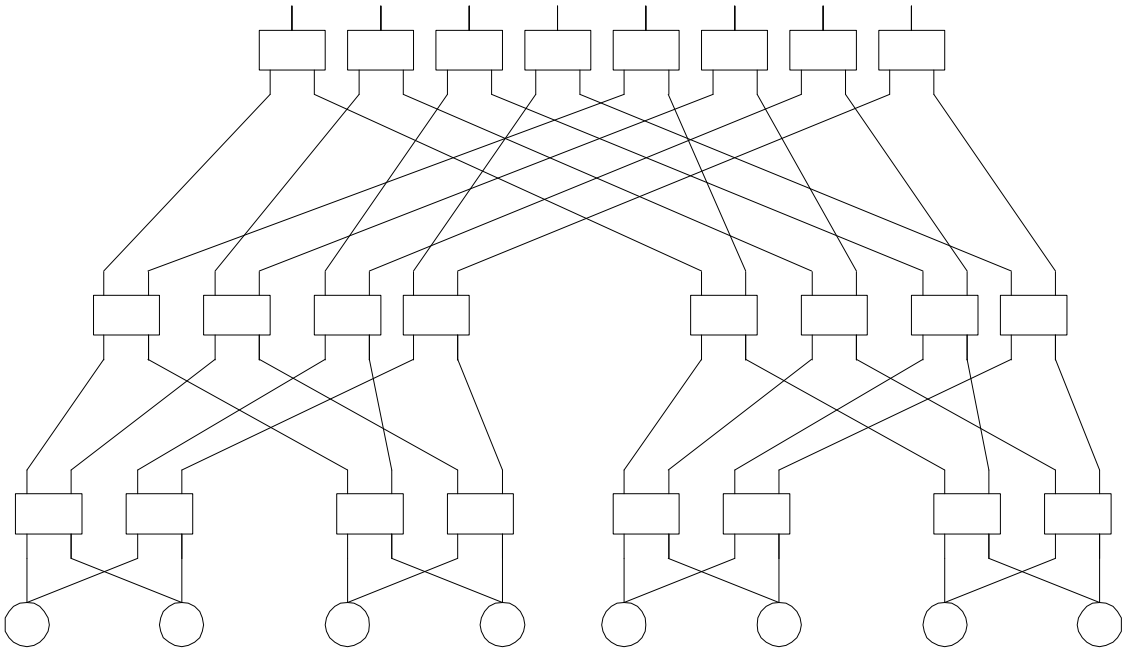
Introduction

- Reading
  - for today Chapter 10 (skip dataflow)
  - for Tuesday finish Chapter 10
  - for next Thursday Active Messages & T3E paper
    - new link for T3E paper on web site
  - delete Chapter 9 & 11 from reading
- Please email me the class you would like to lead discussion
  - only have about three volunteers now

Fat Trees

- adds extra bandwidth near the root
  - improves bi-section bandwidth
  - can use more hardware or faster clock rates
  - TMC CM-5 and Meiko CS-2 are examples of this design
  - can choose not to fully scale any any level (for cost savings)



Combining

- Try to reduce hot spots in a communication network
  - combine requests for the same location
    - really cache line
- Use topology of the network to match comm ops
  - broadcast
    - flood network - everyone sends to everyone else
    - form a broadcast tree
  - reduction
    - put arithmetic operations into switches
    - CM-5 fast hardware for and,or,min,max

MIMD

- Replicate Processor
  - can use commodity parts
- Shared vs. Private Memory
  - shared memory permits any programming model
    - can share data as desired
    - can build message passing using shared buffers
  - private memory is cheaper
    - no expensive interconnect to build
    - interconnect is only to communicate shared info
- Grain Size
  - larger then SIMD machines
  - macro dataflow
    - parts of large datasets flow: $(A^{-1} \times B \times A^{T})^{-1}$
  - macro pipeline
    - cpp | compile | as | ld

Message Passing Machines

- Network of Workstations (NOW)
  - use normal LANs (TCP/IP or custom protocols)
  - message passing libraries (PVM, Express, or MPI)
  - cheap to build
  - communication bandwidth and latency are poor
- Hypercubes
  - Examples: Cosmic Cube and Intel IPSC and IPSC/860
  - original didn't have hardware message forwarding
  - recent systems have included hardware routers
  - how much OS is required
    - just enough to send messages?
    - what about resource allocation?
      - do you need/get a sub-cube?
    - how about tools: debugging performance measurement

Message Passing Machines (cont.)

- Intel Paragon
  - mesh connected machine (2-d mesh)
  - i860 processors
    - 75 Mflops
    - can issue one multiple and one add per cycle
  - communication
    - 200 MB/sec bandwidth (per link)
    - second i860 used for communication (shares memory)
  - memory
    - 16-128 MB/node
    - OS takes over 1MB/node
      - for 1,000 nodes that a giga-kernel

CM-5

- interconnection network
  - fat tree
  - combining network for data reduction (fast barrier)
  - 20MB/sec local 5MB/sec arbitrary
  - 30usec latency for messages
- processors
  - SPARC processors
  - 4 Vector units per node (128Mflops/node)
  - systems up to 1024 nodes have been delivered

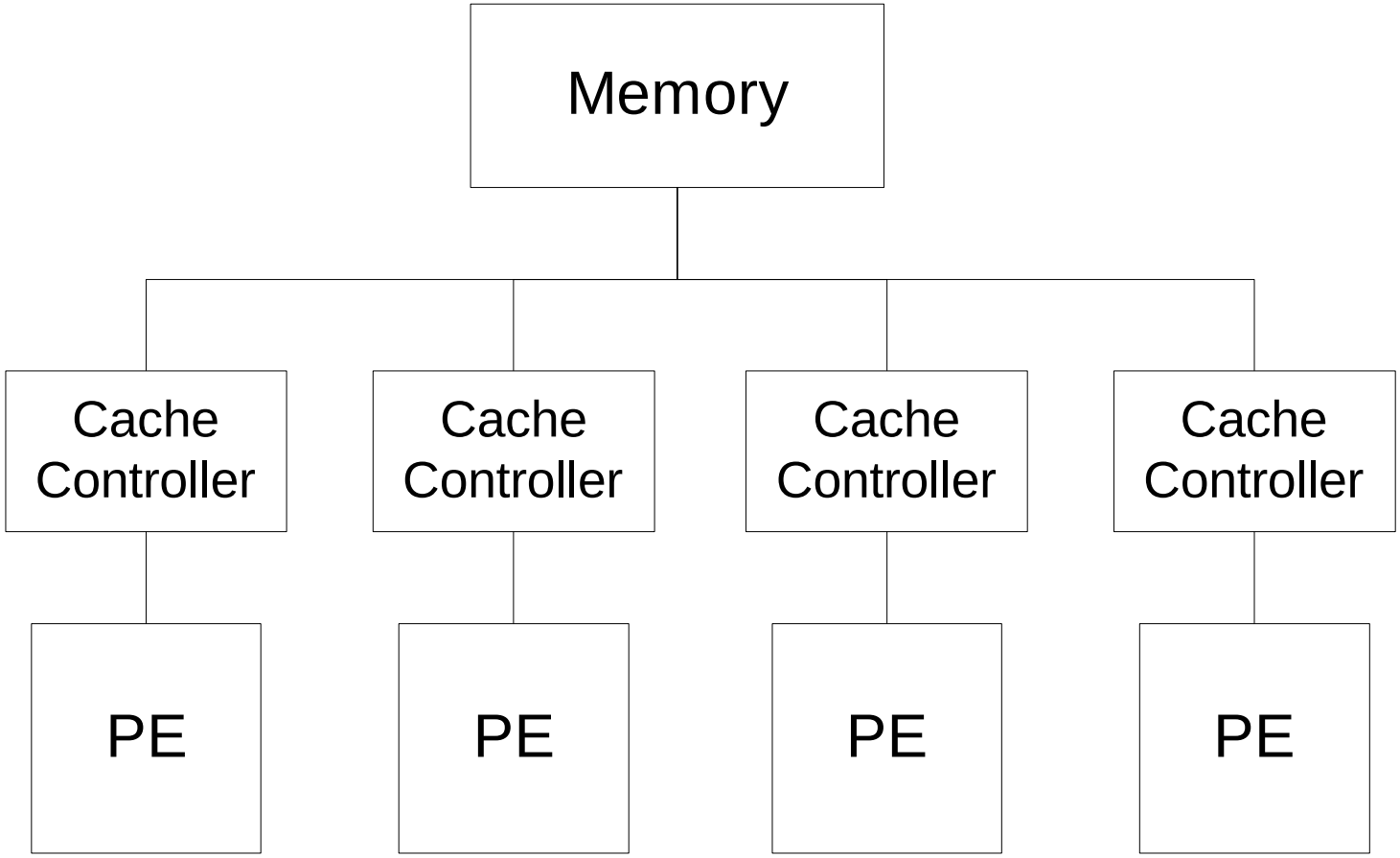
Message Passing Machines (cont.)

- SP-2
  - processors
    - IBM power II
    - 65Mhz (330 Mhz Now)
    - 125 Mflops per processor
  - Entire UNIX workstation is used
    - each runs a full UNIX operating system
    - POE Environment sits on top to provide parallel access
      - individual nodes can be allocated
  - Network
    - MCA (now PCI) plug in card
    - omega network of 8x8 cross bar switches
    - 40 MB/second (now 120MB/sec)

Bus based Multiprocessors

- biggest commercial success of parallel computing
- limited number of processors can share a single bus
- use caches to keep the bus traffic lower
  - the cache is useful even if it is not that much faster than main memory
  - caches act as private memory to reduce bus requests
- Cache coherency
  - need to ensure that each processor get latest version of data
  - how soon does a processors sees changes by other processors is a design parameter
    - by the next instruction “sequential consistency”
    - by the next synchronization operation “released consistency”

Basic Structure of Bus-based SMP



Cache Coherency (simple)

- Read only values are cached
- Writeable values
  - not cached, all reads and writes go to main memory
  - good performance for frequently updated values
    - if many processors update the same location
  - poor performance for
    - many updates by the same processor
    - infrequent updates and frequent writes
- Who marks regions for caching?
  - static: compiler marks shared writeable areas
  - dynamic: runtime support to change cachability of lines
    - compiler emits code to change status
    - user makes explicit calls