Introduction

- Class is an introduction to parallel computing
  - topics include: hardware, applications, compilers, system software, and tools
- Counts for Masters/PhD Comp Credit
- Work required
  - small programming assignments (two) - MPI/OpenMP
  - midterm
  - classroom participation
    - Everyone will have to present (1 or 2) papers for 1 class
    - For each class with readings, everyone has to send me one question to discuss in class
  - group project (2-3 students per group)
- Will be asked to participate in a study of parallel programming

What is Parallel Computing?

- Does it include:
  - super-scalar processing (more than one instruction at once)?
  - client/server computing?
    - what if RPC calls are non-blocking?
  - vector processing (same instruction to several values)?
  - collection of PC’s not connected to a (fast) network?
- For this class, parallel computing requires:
  - more than one processing element
  - nodes connected to a communication network
  - nodes working together to solve a single problem

Why Parallelism

- Speed
  - need to get results faster than possible with sequential
    - a weather forecast that is late is useless
  - could come from
    - more processing elements (P.E.’s)
    - more memory (or cache)
    - more disks
- Cost: cheaper to buy many smaller machines
  - this is only relatively recently true due to
    - VLSI
    - commodity parts

What Does a Parallel Computer Look Like?

- Hardware
  - processors
  - communication
  - memory
  - coordination

- Software
  - programming model
  - communication libraries
  - operating system

Processing Elements (PE)

- Key Processor Choices
  - How many?
  - How powerful?
  - Custom or off-the-shelf?
- Major Styles of Parallel Computing
  - SIMD - Single Instruction Multiple Data
    - one master program counter (PC)
  - MIMD - Multiple Instruction Multiple Data
    - separate code for each processor
  - SPMD - Single Program Multiple Data
    - same code on each processor, separate PC’s on each
  - Dataflow – instruction (or code block) waits for operands
    - “automatically” finds parallelism
SIMD

- Program Counter
- Mask Flag
- Processors
- Program

MIMD

- Processors
- Program Counter
- Program #1
- Program #2
- Program #3

SPMD

- Processors
- Program Counter
- Program
- Program
- Program

Dataflow

- Instruction
- I1
- I2
- I3
- I4

Communication Networks

- **Connect**
  - PE’s, memory, I/O
- **Key Performance Issues**
  - Latency: time for first byte
  - Throughput: average bytes/second
- **Possible Topologies**
  - Bus - simple, but doesn’t scale
  - Ring - orders delivery of messages

Topologies (cont)

- **Tree** - needs to increase bandwidth near the top
- **Mesh** - two or three dimensions
- **Hypercube** - needs a power of number of nodes
Memory Systems

- **Key Performance Issues**
  - latency: time for first byte
  - throughput: average bytes/second

- **Design Issues**
  - Where is the memory
    - divided among each node
    - centrally located (on communication network)
  - Access by processors
    - can all processors get to all memory?
    - is the access time uniform?
      - UMA vs. NUMA