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 prepare questions for the readings for several classes (4 students per class with readings), and help explain the papers
    - Group project (3-4 students per group)

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
PARALLEL ARCHITECTURE

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

### 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

- **PE**
- **MEM**

Topologies (cont)

- tree - need to increase bandwidth near the top

- mesh - two or three dimensions

- hypercube - needs a power of (2) number of nodes

Current state of the art is dragonfly network – local groups with mesh + global links between groups

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

- 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