CMSC 714
High Performance Computing
Lecture 1 - Introduction
http://www.cs.umd.edu/class/fall2022/cmsc714
Alan Sussman
Introduction

• Class is an introduction to parallel computing
  • Seminar style, on history and recent advances
  • topics include: programming models, hardware, applications, compilers, system software, and tools

• Qualifying course for MS/PhD: Computer Systems

• Work required
  • small programming assignments (two) – MPI and OpenMP
  • Midterm exam
  • classroom participation
    • Everyone will have to prepare questions for the readings for several classes (3 students per class with readings), and help explain the papers
  • group project (3 students per group)
Course Topics

• Introduction to parallel computing – 1 week
• Programming Models – 3 weeks
• Parallel Architectures and Networks – 2 weeks
• Debugging and Instrumentation – 1 week
• Performance Tools – 2 weeks
• OS, Runtime Systems, and Parallel I/O – 2 weeks
• Commercial and Scientific Applications – 2 weeks
Additional class info

• Syllabus, lecture slides, project descriptions on course web site:
  • [https://www.cs.umd.edu/class/fall2022/cmsc714/](https://www.cs.umd.edu/class/fall2022/cmsc714/)
• Project submissions via email to me
• In-class midterm – date TBD soon
• Cluster accounts on university resource (zaratan) will be coming soon
  • We will email you a login ID and initial password
  • Further instructions with first project
Introductions

• Name
• MS or PhD, and department
• Area of research
• Why this course?
• Something interesting /unique about yourself
What is Parallel Computing?

• Does it include:
  • super-scalar processing (more than one instruction at once)?
  • vector processing (same instruction to several values)?
  • collection of PC’s **not** connected to a (fast) network?
  • cloud computing?
  • Accelerators (GPUs, FPGAs)?

● **For this class, parallel computing requires:**
  – more than one processing element/core
  – nodes (with one or more cores) connected to a communication network
  – nodes working together to solve a single problem
    – Sometimes a single node is enough
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/secondary storage
  - example is speeding up scientific simulations
  - another reason is to get results in (near) realtime

- **Cost: cheaper to buy many smaller machines**
  - this has been true for the last 15-20 years due to
    - VLSI
    - commodity parts
HPC is needed for real applications

Weather forecasting

https://www.ncl.ucar.edu/Applications/wrf.shtml

Cosmology studies

https://www.nas.nasa.gov/SC14/demos/demo27.html
Parallel Architecture
What Does a Parallel Computer Look Like?

• **Hardware**
  - processors
  - communication
  - memory
  - coordination

• **Software**
  - programming model
  - communication libraries
  - operating system
Parallel architecture – the current answer

- A set of nodes or processing elements connected by a network.

https://computing.llnl.gov/tutorials/parallel_comp
Processing Elements (PE)

• **Key Processor/Core 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

0 1 1

0 1 1 1

Processors

Program
MIMD

Processors

Program Counter

Program #1

Program Counter

Program #2

Program Counter

Program #3
SPMD

Processors

Program Counter

Program

Program Counter

Program

Program Counter

Program

Program
Dataflow
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 - need to increase bandwidth near the top (fat-tree)

  PE  PE  PE  PE

  PE  PE  PE  PE

  PE  PE  PE  PE

  PE  PE  PE  PE

  PE  PE  PE  PE

  PE  PE  PE  PE

  PE  PE  PE  PE

–Mesh/torus - 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