Course Details

- Time and Location: TuTh, 9:10-10:30 AM, WEB L112
- Course Website
  - http://www.eng.utah.edu/~cs4961/
- Instructor: Mary Hall, mhall@cs.utah.edu
  - Office Hours: Tu 10:45-11:15 AM; Wed 11:00-11:30 AM
- TA: Sriram Aananthakrishnan, sriram@cs.utah.edu
  - Office Hours: TBD
- SYMPA mailing list
  - cs4961@list.eng.utah.edu
- Textbook
  - “Principles of Parallel Programming,” Calvin Lin and Lawrence Snyder.
  - Also, readings and notes provided for MPI, CUDA, Locality and Parallel Algs.

Today's Lecture

- Overview of course (done)
- Important problems require powerful computers ...
  - ... and powerful computers must be parallel
  - Increasing importance of educating parallel programmers (you)
- What sorts of architectures in this class
  - Multimedia extensions, multi-cores, GPUs, networked clusters
- Developing high-performance parallel applications
  - An optimization perspective

Outline

- Logistics
- Introduction
- Technology Drivers for Multi-Core Paradigm Shift
- Origins of Parallel Programming:
  - Large-scale scientific simulations
- The fastest computer in the world today
- Why writing fast parallel programs is hard
- Algorithm Activity

Some material for this lecture drawn from:
Kathy Yelick and Jim Demmel, UC Berkeley
Quentin Stout, University of Michigan,
(see http://www.eecs.umich.edu/~qstout/parallel.html)
Top 500 list (http://www.top500.org)
Course Objectives

• Learn how to program parallel processors and systems
  - Learn how to think in parallel and write correct parallel programs
  - Achieve performance and scalability through understanding of architecture and software mapping
• Significant hands-on programming experience
  - Develop real applications on real hardware
• Discuss the current parallel computing context
  - What are the drivers that make this course timely
  - Contemporary programming models and architectures, and where is the field going

Why is this Course Important?

• Multi-core and many-core era is here to stay
  - Why? Technology Trends
• Many programmers will be developing parallel software
  - But still not everyone is trained in parallel programming
  - Learn how to put all these vast machine resources to the best use!
• Useful for
  - Joining the work force
  - Graduate school
• Our focus
  - Teach core concepts
  - Use common programming models
  - Discuss broader spectrum of parallel computing

Parallel and Distributed Computing

• Parallel computing (processing):
  - the use of two or more processors (computers), *usually within a single system*, working simultaneously to solve a single problem.
• Distributed computing (processing):
  - any computing that involves *multiple computers remote from each other* that each have a role in a computation problem or information processing.
• Parallel programming:
  - the human process of developing programs that express what computations should be executed in parallel.

Detour: Technology as Driver for “Multi-Core” Paradigm Shift

• Do you know why most computers sold today are parallel computers?
• Let’s talk about the technology trends
**Technology Trends: Microprocessor Capacity**

Moore's Law:
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

**Clock speed flattening sharply**

**Transistor count still rising**

*Slide source: Maurice Herlihy*

---

**Technology Trends: Power Density Limits**

**Serial Performance**

**Moore's Law Extrapolation:**
Power Density for Leading Edge Microprocessors

---

**The Multi-Core Paradigm Shift**

**What to do with all these transistors?**

- **Key ideas:**
  - Movement away from increasingly complex processor design and faster clocks
  - Replicated functionality (i.e., parallel) is simpler to design
  - Resources more efficiently utilized
  - Huge power management advantages

**All Computers are Parallel Computers.**

---

**Proof of Significance: Popular Press**

- August 2009 issue of *Newsweek*
- Article on 25 things "smart people" should know
- See [http://www.newsweek.com/id/212142](http://www.newsweek.com/id/212142)
Scientific Simulation: The Third Pillar of Science

• Traditional scientific and engineering paradigm:
  1) Do theory or paper design.
  2) Perform experiments or build system.

• Limitations:
  - Too difficult -- build large wind tunnels.
  - Too expensive -- build a throw-away passenger jet.
  - Too slow -- wait for climate or galactic evolution.
  - Too dangerous -- weapons, drug design, climate experimentation.

• Computational science paradigm:
  3) Use high performance computer systems to simulate the phenomenon
     - Base on known physical laws and efficient numerical methods.

The quest for increasingly more powerful machines

• Scientific simulation will continue to push on system requirements:
  - To increase the precision of the result
  - To get to an answer sooner (e.g., climate modeling, disaster modeling)

• The U.S. will continue to acquire systems of increasing scale
  - For the above reasons
  - And to maintain competitiveness

A Similar Phenomenon in Commodity Systems

• More capabilities in software
• Integration across software
• Faster response
• More realistic graphics
• ...

The fastest computer in the world today

• What is its name? Jaguar (Cray XT5)

• Where is it located? Oak Ridge National Laboratory

• How many processors does it have? ~37,000 processor chips (224,162 cores)

• What kind of processors? AMD 6-core Opterons

• How fast is it? 1.759 Petaflop/second One quadrillion operations/s 1 x 10^16

See http://www.top500.org
The **SECOND** fastest computer in the world today

- What is its name?  RoadRunner
- Where is it located?  Los Alamos National Laboratory
- How many processors does it have?  ~19,000 processor chips (~129,600 "processors")
- What kind of processors?  AMD Opterons and IBM Cell/BE (in Playstations)
- How fast is it?  1.105 Petaflop/second

One quadrillion operations/s  \(1 \times 10^{15}\)

See http://www.top500.org

---

**Example: Global Climate Modeling Problem**

- Problem is to compute:
  \[ f(\text{latitude}, \text{longitude}, \text{elevation}, \text{time}) \rightarrow \text{temperature, pressure, humidity, wind velocity} \]
- Approach:
  - Discretize the domain, e.g., a measurement point every 10 km
  - Devise an algorithm to predict weather at time \(t+\delta t\) given \(t\)
- Uses:
  - Predict major events, e.g., El Nino
  - Use in setting air emissions standards

---

**Some Characteristics of Scientific Simulation**

- Discretize physical or conceptual space into a grid
  - Simpler if regular, may be more representative if adaptive
- Perform local computations on grid
  - Given yesterday’s temperature and weather pattern, what is today’s expected temperature?
- Communicate partial results between grids
  - Contribute local weather result to understand global weather pattern.
- Repeat for a set of time steps
- Possibly perform other calculations with results
  - Given weather model, what area should evacuate for a hurricane?
Example of Discretizing a Domain

Parallel Programming Complexity

An Analogy to Preparing Thanksgiving Dinner

- Enough parallelism? (Amdahl’s Law)
  - Suppose you want to just serve turkey
- Granularity
  - How frequently must each assistant report to the chef?
    - After each stroke of a knife? Each step of a recipe? Each dish completed?
- Locality
  - Grab the spices one at a time? Or collect ones that are needed prior to starting a dish?
- Load balance
  - Each assistant gets a dish? Preparing stuffing vs. cooking green beans?
- Coordination and Synchronization
  - Person chopping onions for stuffing can also supply green beans
  - Start pie after turkey is out of the oven

All of these things makes parallel programming even harder than sequential programming.

Finding Enough Parallelism

- Suppose only part of an application seems parallel
- Amdahl’s law
  - let s be the fraction of work done sequentially, so (1-s) is fraction parallelizable
  - P = number of processors
    - Speedup(P) = Time(1)/Time(P)
    - <= 1/(s + (1-s)/P)
    - <= 1/s
- Even if the parallel part speeds up perfectly, performance is limited by the sequential part

Overhead of Parallelism

- Given enough parallel work, this is the biggest barrier to getting desired speedup
- Parallelism overheads include:
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - extra (redundant) computation
- Each of these can be in the range of milliseconds (milliseconds of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work
**Locality and Parallelism**

- Large memories are slow, fast memories are small
- Program should do most work on local data

**Load Imbalance**

- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase)
  - unequal size tasks
- Examples of the latter
  - adapting to "interesting parts of a domain"
  - tree-structured computations
  - fundamentally unstructured problems
- Algorithm needs to balance load

**Summary of Lecture**

- Solving the "Parallel Programming Problem"
  - Key technical challenge facing today's computing industry, government agencies and scientists
- Scientific simulation discretizes some space into a grid
  - Perform local computations on grid
  - Communicate partial results between grids
  - Repeat for a set of time steps
  - Possibly perform other calculations with results
- Commodity parallel programming can draw from this history and move forward in a new direction
- Writing fast parallel programs is difficult
  - Amdahl's Law ➔ Must parallelize most of computation
  - Data Locality
  - Communication and Synchronization
  - Load Imbalance

**Next Time**

- An exploration of parallel algorithms and their features
- First written homework assignment