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
  - http://www.cs.utah.edu/~mhall/
  - Office Hours: Tu 10:45-11:15 AM; Wed 11:00-11:30 AM
- TA: Nikhil Mishrikoti, Nikhil Mishrikoti, nikmys@cs.utah.edu
  - Office Hours: TBD
- SYMPA mailing list
  - cs4961@list.eng.utah.edu
  - https://sympa.eng.utah.edu/sympa/info/cs4961
- Textbook
  - Also, readings and notes provided for other topics as needed

Administrative

- Prerequisites:
  - C programming
  - Knowledge of computer architecture
  - CS4400 (concurrent may be ok)
- Please do not bring laptops to class!
- Do not copy solutions to assignments from the internet (e.g., wikipedia)
- Read Chapter 1 of textbook by next lecture
- We will discuss the first written homework assignment, due Aug. 30 before class (two problems and one question)
  - Homework is posted on the website and will be shown at the end of lecture

Today's Lecture

- Overview of course
- 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

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

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

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

Clock speed flattening sharply
Transistor count still rising

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.

The Multi-Core or Many-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
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
- ...

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

Source: [http://www.epa.gov/chammp/chammp.html](http://www.epa.gov/chammp/chammp.html)
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

Processors in adjacent blocks in the grid communicate their result.

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?
- All of these things makes parallel programming even harder than sequential programming.
  - 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
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 (=millions 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
Homework 1: Parallel Programming Basics

Turn in electronically on the CADE machines using the handin program: "handin cs4961 hw1 <ProblemFile>

- Problem 1: #1.3 in textbook
- Problem 2: I recently had to tabulate results from a written survey that had four categories of respondents: (I) students; (II) academic professionals; (III) industry professionals; and, (IV) other. The respondents selected to which category they belonged and then answered 32 questions with five possible responses: (i) strongly agree; (ii) agree; (iii) neutral; (iv) disagree; and, (v) strongly disagree. My family members and I tabulated the results in parallel (assume there were four of us).
  - (a) Identify how data parallelism can be used to tabulate the results of the survey. Keep in mind that each individual survey is on a separate sheet of paper that only one "processor" can examine at a time. Identify scenarios that might lead to load imbalance with a purely data parallel scheme.
  - (b) Identify how task parallelism and combined task and data parallelism can be used to tabulate the results of the survey to improve upon the load imbalance you have identified.

Homework 1, cont.

- Problem 3: What are your goals after this year and how do you anticipate this class is going to help you with that? Some possible answers, but please feel free to add to them. Also, please write at least one sentence of explanation.
  - A job in the computing industry
  - A job in some other industry that uses computing
  - As preparation for graduate studies
  - To satisfy intellectual curiosity about the future of the computing field
  - Other

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 \(\Rightarrow\) 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