Lecture 1:

Why Parallelism?

Why Efficiency?
Hi!

Plus . . .

An evolving collection of teaching assistants

Randy Bryant

Nathan Beckmann
Getting into the Class

- **Status (Mon Jan. 13, 09:30)**
  - 157 students enrolled
  - 103 on wait list
  - 175 max. enrollment
  - ~20 slots

- **If you are registered**
  - Do Assignment 1
    - Due Jan. 29
  - If find too challenging, then please drop by Jan. 27

- **Clearing Wait List**
  - Complete Assignment 1 by Jan. 22, 23:00
  - No Autolab account required
  - We will enroll top-performing students
  - *It’s that simple!*
  - You will know by Jan. 27
What will you be doing in this course?
Assignments

- Four programming assignments
  - First assignment is done individually, the rest will be done in pairs
  - Each uses a different parallel programming environment
  - Each also involves measurement, analysis, and tuning

Assignment 1: SIMD and multi-core parallelism

Assignment 2: CUDA programming on NVIDIA GPUs

Assignment 3: Parallel Programming via a Shared-Address Space Model

Assignment 4: Parallel Programming via a Message Passing Model
Final project

- 6-week self-selected final project
- Performed in groups (by default, 2 people per group)
- Keep thinking about your project ideas starting TODAY!
- Poster session at end of term

Check out previous projects:

http://15418.courses.cs.cmu.edu/spring2016/competition

http://15418.courses.cs.cmu.edu/fall2017/article/10

http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15418-s18/www/15418-s18-projects.pdf

http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15418-s19/www/15418-s19-projects.pdf
Exercises

- Five homework exercises
  - Scheduled throughout term
  - Designed to prepare you for the exams
  - We will grade your work only in terms of participation
    - Did you make a serious attempt?
    - Only a participation grade will go into the gradebook
Grades

40%  Programming assignments (4)
30%  Exams (2)
25%  Final project
5%   Exercises

Each student gets up to five late days on programming assignments (see syllabus for details)
Getting started

- Visit course home page
  - http://www.cs.cmu.edu/~418/
- Sign up for the course on Piazza
  - http://piazza.com/cmu/spring2020/1541815618
- Textbook
  - There is no course textbook, but please see web site for suggested references
- Find a Partner
  - Assignments 2–4, final project
Regarding the class meeting times

- **Class MWF 3:00–4:20**
  - Lectures (mostly)
  - Some designated “Recitations”
    - Targeted toward things you need to know for an upcoming assignment
- **No classes last part of the term**
  - Let you focus on projects
Collaboration (Acceptable & Unacceptable)

- **Do**
  - Become familiar with course policy
    - [http://www.cs.cmu.edu/~418/academicintegrity.html](http://www.cs.cmu.edu/~418/academicintegrity.html)
  - Talk with instructors, TAs, partner
  - Brainstorm with others
  - Use general information on WWW

- **Don’t**
  - Copy or provide code to anyone
  - Use information specific to 15-418/618 on WWW
  - Leave your code in accessible place
    - Now or in the future
A Brief History of Parallel Computing

- Initial Focus (starting in 1970s): "Supercomputers" for Scientific Computing

C.mmp at CMU (1971)
16 PDP-11 processors

Cray XMP (circa 1984)
4 vector processors

Thinking Machines CM-2 (circa 1987)
65,536 1-bit processors +
2048 floating-point co-processors

800+ compute nodes
Heterogenous Structure

Bridges at the Pittsburgh Supercomputer Center
A Brief History of Parallel Computing

- Initial Focus (starting in 1970s): “Supercomputers” for Scientific Computing
- Another Driving Application (starting in early ‘90s): Databases
  - Especially, handling millions of transactions per second for web services

- Sun Enterprise 10000 (circa 1997)
  - 16 UltraSPARC-II processors

- Oracle Supercluster M7 (today)
  - 4 X 32-core SPARC M2 processors

- RIP 2019. Killed by cloud computing
A Brief History of Parallel Computing

- Cloud computing (2000–present)
  - Build out massive centers with many, simple processors
    - Connected via LAN technology
  - Program using distributed-system models

- Not really the subject of this course (take 15-440)
Setting Some Context

Before we continue our multiprocessor story, let’s pause to consider:

- **Q**: what had been happening with single-processor performance?
- **A**: since forever, they had been getting **exponentially faster**
- **Why?**
A Brief History of Processor Performance

- **Wider data paths**
  - 4 bit → 8 bit → 16 bit → 32 bit → 64 bit

- **More efficient pipelining**
  - e.g., 3.5 Cycles Per Instruction (CPI) → 1.1 CPI

- **Exploiting instruction-level parallelism (ILP)**
  - “Superscalar” processing: e.g., issue up to 4 instructions/cycle
  - “Out-of-order” processing: extract parallelism from instruction stream

- **Faster clock rates**
  - e.g., 10 MHz → 200 MHz → 3 GHz

- **During the 80s and 90s:** large exponential performance gains
  - and then…
A Brief History of Parallel Computing

- Initial Focus (starting in 1970s): “Supercomputers” for Scientific Computing
- Another Driving Application (starting in early ‘90s): Databases
- Inflection point in 2004: Intel hits the Power Density Wall

Pat Gelsinger, ISSCC 2001
From the New York Times

Intel's Big Shift After Hitting Technical Wall

The warning came first from a group of hobbyists that tests the speeds of computer chips. This year, the group discovered that the Intel Corporation's newest microprocessor was running slower and hotter than its predecessor.

What they had stumbled upon was a major threat to Intel's longstanding approach to dominating the semiconductor industry - relentlessly raising the clock speed of its chips.

Then two weeks ago, Intel, the world's largest chip maker, publicly acknowledged that it had hit a "thermal wall" on its microprocessor line. As a result, the company is changing its product strategy and disbanding one of its most advanced design groups. Intel also said that it would abandon two advanced chip development projects, code-named Tejas and Jayhawk.

Now, Intel is embarked on a course already adopted by some of its major rivals: obtaining more computing power by stamping multiple processors on a single chip rather than straining to increase the speed of a single processor.

...
ILP tapped out + end of frequency scaling

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)

- Transistor density
- Clock frequency
- Power
- Instruction-level parallelism (ILP)

Processor clock rate stops increasing
No further benefit from ILP

Image credit: “The free Lunch is Over” by Herb Sutter, Dr. Dobbs 2005
Programmer’s Perspective on Performance

**Question:** How do you make your program run faster?

**Answer before 2004:**
- Just wait 6 months, and buy a new machine!
- (Or if you’re really obsessed, you can learn about parallelism.)

**Answer after 2004:**
- You need to write parallel software.
Parallel Machines Today

Examples from Apple’s product line:

- **Mac Pro**
  - 28 Intel Xeon W cores

- **iMac Pro**
  - 18 Intel Xeon W cores

- **MacBook Pro Retina 15”**
  - 8 Intel Core i9 cores

- **iPhone XS**
  - 6 CPU cores (2 fast + 4 low power)
  - 6 GPU cores

(images from apple.com)
Intel Coffee Lake Core i9 (2019)

6-core CPU + multi-core GPU integrated on one chip
NVIDIA GeForce GTX 1660 Ti GPU (2019)

24 major processing blocks
(but much, much more parallelism available... details coming soon)
Mobile parallel processing

Power constraints heavily influence design of mobile systems

Apple A12: (in iPhone XR)
- 4 CPU cores
- 4 GPU cores
- Neural net engine
- + much more

NVIDIA Tegra K1:
- Quad-core ARM A57 CPU + 4 ARM A53 CPUs +
- NVIDIA GPU + image processor...
Supercomputing

- Today: clusters of multi-core CPUs + GPUs
- Oak Ridge National Laboratory: Summit (#1 supercomputer in world)
  - 4,608 nodes
  - Each with two 22-core CPUs + 6 GPUs
Supercomputers vs. Cloud Systems

Supercomputers
- Few, big tasks
- Customized
- Optimized for reliability
- Low latency interconnect
- Minimal
- Static scheduling

Data Center Clusters
- Many small tasks
- Consumer grade
- Optimized for low cost
- Throughput-optimized interconnect
- Provides reliability
- Dynamic allocation

Target Applications

Hardware

Run-Time System

Application Programming
- Low-level, processor-centric model
- Programmer manages resources
- High level, data-centric model
- Let run-time system manage resources
Supercomputer / Data Center Overlap

- **Supercomputer features in data centers**
  - Data center computers sometimes used to solve problem
    - E.g., learn neural network for language translation
  - Data center computers sometimes equipped with GPUs

- **Data center features in supercomputers**
  - Also used to process many small–medium jobs
What is a parallel computer?
One common definition

A parallel computer is a collection of processing elements that cooperate to solve problems quickly.

We care about performance *
We care about efficiency

We’re going to use multiple processors to get it

* Note: different motivation from “concurrent programming” using pthreads in 15-213
DEMO 1

(This semester’s first parallel program)
Speedup

One major motivation of using parallel processing: achieve a speedup

For a given problem:

\[
\text{speedup( using P processors )} = \frac{\text{execution time (using 1 processor)}}{\text{execution time (using P processors)}}
\]
Class observations from demo 1

- Communication limited the maximum speedup achieved
  - In the demo, the communication was telling each other the partial sums

- Minimizing the cost of communication improves speedup
  - Moving students ("processors") closer together (or let them shout)
DEMO 2

(scaling up to four “processors”)

Class observations from demo 2

- Imbalance in work assignment limited speedup
  - Some students (“processors”) ran out work to do (went idle), while others were still working on their assigned task

- Improving the distribution of work improved speedup
DEMO 3

(massively parallel execution)
Class observations from demo 3

- The problem I just gave you has a significant amount of communication compared to computation.
- Communication costs can dominate a parallel computation, severely limiting speedup.
Course theme 1: Designing and writing parallel programs ... that scale!

- Parallel thinking
  1. Decomposing work into pieces that can safely be performed in parallel
  2. Assigning work to processors
  3. Managing communication/synchronization between the processors so that it does not limit speedup

- Abstractions/mechanisms for performing the above tasks
  - Writing code in popular parallel programming languages
Course theme 2: Parallel computer hardware implementation: how parallel computers work

- Mechanisms used to implement abstractions efficiently
  - Performance characteristics of implementations
  - Design trade-offs: performance vs. convenience vs. cost

- Why do I need to know about hardware?
  - Because the characteristics of the machine really matter
    (recall speed of communication issues in earlier demos)
  - Because you care about efficiency and performance
    (you are writing parallel programs after all!)
Course theme 3: Thinking about efficiency

- **FAST != EFFICIENT**

- Just because your program runs faster on a parallel computer, it does not mean it is using the hardware efficiently

  - Is 2x speedup on computer with 10 processors a good result?

- **Programmer’s perspective:** make use of provided machine capabilities

- **HW designer’s perspective:** choosing the right capabilities to put in system (performance/cost, cost = silicon area?, power?, etc.)
Fundamental Shift in CPU Design Philosophy

Before 2004:
- within the chip area budget, maximize performance
  - increasingly aggressive speculative execution for ILP

After 2004:
- area *within* the chip matters (limits # of cores/chip):
  - maximize performance per area
- power consumption is critical (battery life, data centers)
  - maximize performance per Watt
- upshot: major focus on *efficiency* of cores
Summary

- Today, single-thread performance is improving very slowly
  - To run programs significantly faster, programs must utilize multiple processing elements
  - Which means you need to know how to write parallel code

- Writing parallel programs can be challenging
  - Requires problem partitioning, communication, synchronization
  - Knowledge of machine characteristics is important

- I suspect you will find that modern computers have tremendously more processing power than you might realize, if you just use it!

- Welcome to 15-418!