# Lecture 01—Introduction
## ECE 459: Programming for Performance

Patrick Lam

University of Waterloo

January 7, 2014

[Thanks to Jon Eyolfson for slides!]

# Course Website

**http://patricklam.ca/p4p/**

I also added everyone enrolled as of Sunday to Piazza.

# Staff

### Instructor
Patrick Lam    p.lam@ece.uwaterloo.ca    DC 2597D/DC2534

### Teaching Assistants
| | | |
|---|---|---|
| [redacted] | [redacted]@uwaterloo.ca | DC 2553 |
| Morteza Nabavi | mnabavi@uwaterloo.ca | E5-4131 |
| Saeed Nejati | snejati@uwaterloo.ca | E5-4132 |

# Schedule

|  |  |
|---|---|
| **Lectures:** | January 7—April 4 |
| | Tuesday 2:30 PM, RCH 302 |
| | Thursday 2:30PM QNC 2502 |
| **Tutorials:** | not used |
| **Midterm:** | Wednesday February 26 |
| | 19:00–20:20 DC 1530 |

# Office Hours

Let's have them. When?

[Academic, and other, advice also available!]

# Recommended Textbook
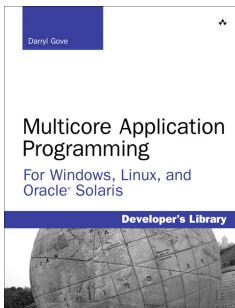


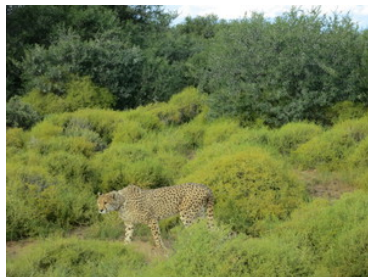Multicore Application Programming For Windows, Linux, and Oracle Solaris. Darryl Gove. Addison-Wesley, 2010.

# Goal

Make programs run faster!

# Making Programs Faster

Two main ways:





---

[1]credit: Chensiyuan, Wikimedia Commons, CC-BY-SA
[2]credit: me

# Making Programs Faster

- Increase bandwidth (tasks per unit time); or
- Decrease latency (time per task).

**Examples of bandwidth/latency:**
Network (connection speed/ping), traffic (lanes/speed)

## Our Focus

Primarily on increasing bandwidth (more tasks/unit time).

- Do tasks in parallel

Decreasing time/task usually harder, with fewer gains.

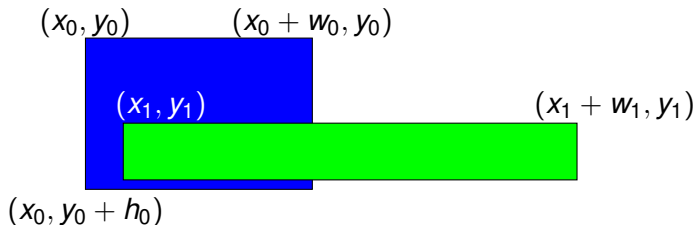CPUs have been going towards more cores rather than raw speed.

# A Bit on Improving Latency

We won't return to these topics, but we'll touch on them now.

- Profile the code;
- Do less work;
- Be smarter; or
- Improve the hardware.

# Intermission

While working on Assignment 1, I ran into this puzzle:



$(x_0, y_0)$ $(x_0 + w_0, y_0)$

$(x_1, y_1)$ $(x_1 + w_1, y_1)$

$(x_0, y_0 + h_0)$

When do these rectangles intersect?

# Increasing Bandwidth: Parallelism

Some tasks are easy to run in parallel.

**Examples:** web server requests, computer graphics, brute-force searches, genetic algorithms

Others are more difficult.

**Example:** linked list traversal (why?)

# Hardware

- Use pipelining (all modern CPU do this):
  - Implement this in software by spliting a task into subtasks and running the subtasks in parallel

- Increase the number of cores/CPUs.

- Use multiple connected machines.

- Use specialized hardware, such as a GPU which contains hundreds of simple cores.

# Barriers to parallelization

- Independent tasks ("embarrassingly parallel problems") are trivial to parallelize, but dependencies cause problems.

- Unable to start task until previous task finishes.

- May require synchronization and combination of results.

- More difficult to reason about, since execution may happen in any order.

# Limitations

- Sequential tasks in the problem will always dominate maximum performance

- Some sequential problems may be parallelizable by reformulating the implementation

- However, no matter how many processors you have, you won't be able to speed up the program as a whole (known as **Amdahl's Law**)

# Data Race

- Two processors accessing the same data.

- For example, consider the following code:
  ```
  x = 1
  print x
  ```
  **You run it and see it prints 5**

- **Why?** Before the print, another thread wrote a new value for x. This is an example of a data race.

# Deadlock

Two processors trying to access a shared resource.

- Consider two processors trying to get two resources:

  | **Processor 1** | **Processor 2** |
  | --- | --- |
  | Get Resource 1 | Get Resource 2 |
  | Get Resource 2 | Get Resource 1 |
  | Release Resource 2 | Release Resource 1 |
  | Release Resource 1 | Release Resource 2 |

- Processor 1 gets Resource 1, then Processor 2 gets Resource 2, now they both wait for each other **(deadlock)**.

# Objectives

- Implement parallel programs which use 1) synchronization primitives and 2) asynchronous I/O

- Describe and use parallel computing frameworks

- Be able to investigate software and improve its performance

- Use and understand specialized GPU programming/programming languages

# Assignments

1. Manual parallelization using Pthreads/async I/O

2. Automatic parallelization and OpenMP

3. Application profiling and improvement

4. GPU programming

# Breakdown

- 40% Assignments (10% each)

- 10% Midterm

- 50% Final

# Grace Days

- 4 grace days to use over the semester for late assignments

- **No mark penalty** for using grace days

- Try not to use them just because they're there

# Homework for Thursday

We'll be doing exercises based on this presentation:

```
http://www.infoq.com/presentations/
click-crash-course-modern-hardware
```

I'll post the exercises on Wednesday.

# Suggestions?

- Just let me know