Lecture 01 - Introduction

ECE 459: Programming for Performance

Jon Eyolfson

University of Waterloo

January 4, 2012

Course Website

`http://ece459.eyolfson.com/`

If the website is down, it may be due to moving the domain

2011 Website: `http://www.patricklam.ca/p4p/`

Staff

Instructor

Jon Eyolfson	jeyolfso@uwaterloo.ca	DC 2553

Teaching Assistants

Shay Berkovich	sberkovi@uwaterloo.ca	E5 4124
Mohammad Rostami	m2rostam@uwaterloo.ca	EIT 3137

Schedule

Lectures: January 4 - April 2, MWF, 10:30 AM, DWE 3522

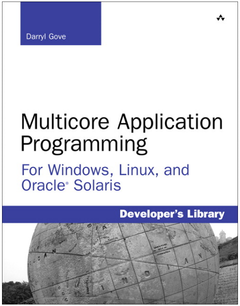
Tutorials: January 6 - March 30, F, 1:30 PM, DWE 3522

Midterm: March 2, F, 6:30 PM, RCH 105/110 (tentatively)

Office Hours

- Open vote
- Likely sometime after each lecture?

Textbook



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

Goal

- To make programs run faster!

"T-T-T-TODAY, JUNIOR!"

Making Programs Faster

- There are two main ways
- Increase bandwidth (tasks per unit time)
- Decrease latency (time per task)
- **Examples of bandwidth/latency:**
  Network (connection speed/ping), Traffic (lanes/speed)

Our Focus

- Primarily on increasing bandwidth (more tasks per unit time)
- Do tasks in parallel
- Decreasing the amount of time per task usually more difficult with lower gains
- Trends for CPUs have been going towards more cores rather than raw speed

Parallelism

- Some tasks are easy to run in parallel
- **Examples:** computer graphics, brute-force searches, and genetic algorithms
- Others are more difficult
- **Examples:** simple linked list traversal, why?

Hardware

- Use pipelining (all moderns CPU do this)
  - Implement this in software by spliting a task into subtasks and running the subtasks in parallel
- Obviously, we can increase the number of cores/CPUs
- Run problem on multiple connected machines
- Use specialized hardware, such as a GPU which contains hundreds of simple cores

Difficulties

- Independent tasks are trivial to parallelize, dependencies cause problems
- Unable to start task until previous task runs to completion
- 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 eachother (deadlock)

Objectives

- Implementation of parallel programming involving synchronization
- Understanding of parallel computing frameworks
- Ability to investigate software and improve its performance
- Specialized GPU programming/programming languages

Assignments

1. Manual parallelization using Pthreads
2. Automatic parallelization and OpenMP
3. Application profiling and improvement (groups of 2)
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

Suggestions?

- Just let me know