CSCI 322
Principles of Concurrent Programming

Filip Jagodzinski
Labs will be held *most* Fridays, at the same time as lecture is scheduled
- CF 164
- I’ll announce each Wednesday if on Friday there is lab or lecture
- You do not need to register explicitly for a lab section
Preliminaries: Course Website

http://facultyweb.cs.wwu.edu/~jagodzf/teaching/csci322/

- Two exams
  - Score higher on the second exam and it will replace your midterm score
- Sample midterm and final exams will be posted to the course website prior to each exam
- Homework assignments: submitted via Canvas, a mix of book and programming questions
- Project: individual or in-group (max 2 people) during which you’ll use a HLL to implement a program that employs concurrent programming constructs, AND you’ll assess its performance
- Writing assignment: read and review a research article (during 2nd half of course)
Introduction

Who has read the course description?
Introduction

Who has read the course description?

What is this class about?
  Why do we care?
  What will you learn?
  What will you not learn?
Introduction

From the course catalog:


Task: Define each of the words in the course catalog description of this course (on the board exercise)
Introduction

From the course catalog: Algorithms for mutual exclusion. Synchronization and communication techniques: semaphores; monitors; rendezvous; conditional critical regions. Multi-process and multi-threaded programming. Concurrent programming facilities in HLL's.

An important component of learning is knowing WHY we do things the way we do them.

Why do we need synchronization?
What are semaphores?
What is the difference between parallel and concurrent programming?

If you aren’t asking “why” then you are just memorizing
Q: What are the main features of a computer, and how do they work together to transfer information (data) back and forth among different hardware components?

(there’s a reason that architecture is a prerequisite for this course)
Q: What are the main features of a computer, and how do they work together to transfer information (data) back and forth among different hardware components?

Define and explain the Following

- Register
- ALU
- Control Unit
- Opcode
- Cache L1
- Cache L2
- Memory
- Hard Drive
- CPU
Motivation: Architecture 101 review

Q: What are the main features of a computer, and how do they work together to transfer information (data) back and forth among different hardware components?

Task: draw a computer component hierarchy (hard drive, memory, L2 cache, L1 cache, registers, and ALU) which includes an axis for speed of data transfer and data capacity (on the board exercise)
Motivation: Architecture 101 review

Q: What does all of this have to do with concurrent programming?
Motivation: Architecture 101 review

Q: What does all of this have to do with concurrent programming?

Q: How many of you run a SINGLE program each time that you use a computer?

Demo: the `top` command
Motivation: Architecture 101 review

Q: What does all of this have to do with concurrent programming?

Q: How many of you run a SINGLE program each time that you use a computer?

Q: What are possible issues that might arise, and what are a few possible solutions?

(on the board exercise)
Q: What does all of this have to do with concurrent programming?

Q: How many of you run a SINGLE program each time that you use a computer?

Q: What are possible issues that might arise, and what are a few possible solutions?

You might have said: just build faster and faster computers, which means placing more transistors on a chip ... why is this not feasible?
Q: Assume that the number of transistors on a die keeps on going up ... is memory (access) speed keeping up?

Source: Computer Architecture: A Quantitative Approach, Hennessy, Patterson, Arpcaci-Dusseau
Motivation: Architecture 101 review

Q: Can the number of transistors on a die continue to increase? Why or why not?
Motivation: Architecture 101 review

Q: Can the number of transistors on a die continue to increase? Why or why not?
Hence we are “stuck” ... but we want to run more applications at the same time ... Going back to our architecture hierarchy ...

Q: If we want to run more and more programs (applications) on a single-chip computer, but the number of registers and ALUs asymptotes, what else can we do?

(brainstorm, on-the-board exercise)
Hence we are “stuck” ... but we want to run more applications at the same time ... Going back to our architecture hierarchy ...

Q: If we want to run more and more programs (applications) on a single-chip computer, but the number of registers and ALUs asymptotes, what else can we do?

Structure of this course

Labs 1-7: some are “simple,” others are programming assignments in disguise
Structure of this course

Homework Assignments: a mixture of book questions and programming exercises

Question 3: (25 pts) Assume the following three threads with two instructions each:

<table>
<thead>
<tr>
<th>Thread A</th>
<th>Thread B</th>
<th>Thread C</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA1: x = x + 1</td>
<td>IB1: x = x - 2</td>
<td>IC1: x = x + 2</td>
</tr>
<tr>
<td>IA2: print(x)</td>
<td>IB2: print(x)</td>
<td>IC2: print(x)</td>
</tr>
</tbody>
</table>

Further assume that all three threads are sharing the variable x. As a first step, rewrite the instructions for the threads into a register/ALU view (as was done in lecture), however assume that the “print” instruction of each thread cannot be decomposed further.

Under these conditions, which of the following outputs are possible when all three threads execute to completion? If the output is NOT possible, then explain why. If the output IS possible, then provide the order of execution of the register/ALU instructions.

A. 777  
B. 787  
C. 676  
D. 767  
E. 799

Many of the lab tasks and homework assignments are open-ended.

My expectation of you: ask questions, be thorough, start early
Structure of this course

Midterm and final exams

III. Semaphores. 14 points. Partial credit given.

8. The pseudocode below contains three functions, callA() and callB(), each with two instructions, and main(). All instructions inside callA and callB are atomic. Assume Semaphore class is available, which has a constructor Semaphore(int initSharedValue), and functions increment() and decrement(). Variables x and y are global variables (i.e., shared among all functions) saved in shared memory. Declare semaphores wherever needed, so that when the main method runs to completion, the values of x and y are 7 and 12.

ONLY declare and use semaphores. You code other than semaphores.

```c
int x = 44, y = 7;
```

Your answer:

V. Short Answer: Provide a concise answer to each question. Partial credit. 6 points each.

11. Are either of the arms in the below concurrent program at-most-once? Why or why not. Explain.

```c
int x = 44, y = 7;
```

Your answer:

12. Assume a 5 step pipeline with the following stage latencies

- Stage_1 = 5ms
- Stage_2 = 5ms
- Stage_3 = 4ms
- Stage_4 = 3ms
- Stage_5 = 4ms

Under these conditions, what will be the pipeline’s throughput when the 12,435th of 20,000 instructions is executing, and what will be the pipeline’s latency, also for instruction 12,435th? Do not assume any stalls other than those that might be imposed by the stage latencies listed above.

Pipeline throughput:

Pipeline latency:

Your Answer:

13. What cache reuse ratio is needed so that a program that uses cache is 2 times better (performance gain) than a program that does not use cache? Assume cache access time of 1ns and memory access time of 34ns.

Your Answer:
Final Project – your choice

CSCI 322: Project Report

December 5, 2015

Executive Summary: In Strassen’s algorithm for matrix multiplication, a number of intermediate matrices are calculated that are not dependent on each other. Because of this lack of dependency, many of these calculations could be run concurrently without affecting the accuracy of the output. Even with matrices (1000x1000) on two thread machines, we see a speedup faster than serial implementation of Strassen or a naive implementation with no overhead. With larger matrices and more threads, we would expect a greater difference in run-time.

Statement of Need & Motivation: Since matrix multiplication has a number of applications (cryptography, image, and applied math), yet has a relatively high order of growth of $O(n^2)$, we want to optimize this task as much as possible. In 1969, Volker Strassen introduced his optimized algorithm for matrix multiplication, improving the upper bound of growth from $O(n^2)$ to $O(n^{log_27}) = O(n^{2.8074})$. Strassen’s algorithm has many intermediate calculations that are not dependent on each other, which we can prove run-time even more by concurrently executing these calculations.

Program Description: For $AB=C$, Strassen breaks up these matrices into block matrices:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Calculates these intermediate matrices:

$$M_1 = (A_{11} + A_{22})B_{11}$$
$$M_2 = (A_{21} + A_{22})B_{12}$$
$$M_3 = A_{11}(B_{11} - B_{21})$$
$$M_4 = A_{22}(B_{22} - B_{12})$$
$$M_5 = (A_{11} + A_{12})B_{22}$$
$$M_6 = (A_{11} - A_{12})(B_{21} - B_{11})$$
$$M_7 = (A_{21} - A_{22})(B_{11} + B_{22})$$

Concurrent Merge Sort

Exclusion Time (s)

Elements in Array

1 Thread
2 Threads
4 Threads
8 Threads
Why should you take this course ...

A. Because it’s a required course
B. Because I needed to take an extra class to get above 12 credits to keep my financial aid
C. Because it has real world applications
Why should you take this course ...

A. Because it’s a required course
B. Because I needed to take an extra class to get above 12 credits to keep my financial aid
C. Because it has real world applications

CDN
Review of Architecture