Concurrency
Welcome to cs378

Chris Rossbach
CS378 Spring 2018
1/16/18
Outline for Today

• Questions?
• Administrivia
• Course Overview
• Course Details and Logistics
• Parallelism Basics

• Acknowledgments: some materials in this lecture borrowed from
  • Emmett Witchel
    • Who borrowed them from: Kathryn McKinley, Ron Rockhold, Tom Anderson, John Carter, Mike Dahlin, Jim Kurose, Hank Levy, Harrick Vin, Thomas Narten, and Emery Berger
  • Mark Silberstein
    • Who borrowed them from: Blaise Barney, Kunle Olukoton, Gupta
# Course Details

<table>
<thead>
<tr>
<th>Course Name:</th>
<th>CS378 -- Concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Number:</td>
<td>51665</td>
</tr>
<tr>
<td>Lectures:</td>
<td>TTH 9:30-11AM GDC 5.302</td>
</tr>
<tr>
<td>Instructor:</td>
<td>Chris Rossbach</td>
</tr>
<tr>
<td>TA:</td>
<td>Joshua Landgraf</td>
</tr>
</tbody>
</table>

*Please read the syllabus!*
The “Takeaway”

• Concurrency is super-cool, and super-important
• You’ll learn important concepts and background
• ...but have *fun* programming cool systems
  • GPUs! FGPAs!
  • “BigData” engines like MapReduce (or better yet, its successors)
  • Interesting synchronization primitives (not just boring old locks)
  • Programming tools people use to program *super-computers* (*ooh...*)
The “Takeaway”

• Concurrency is super-cool, and super-important
• You’ll learn important concepts and background
• ...but have fun programming cool systems
  • GPUs! FGPAs!
  • “BigData” engines like MapReduce (or better yet, its successors)
  • Interesting synchronization primitives (not just boring old locks)
  • Programming tools people use to program super-computers (ooh...)

Outline:
• The “just eat your kale and quinoa” argument
• The “it’s going to be fun” argument
My first computer
My first computer
My first computer
My first computer
My first computer

CPU

Storage

Tape drive!
(also good for playing heavy metal music)
My first computer

CPU

Storage

screen

Tape drive!
(also good for playing heavy metal music)
My first computer

- CPU
- Wires + gobble-dy-gook (sp?)
- Screen
- Storage
- Tape drive! (also good for playing heavy metal music)
My current computer
My current computer

Too boring...
Another of my current computers
Another of my current computers
Another of my current computers
Another of my current computers
Another of my current computers

A lot has changed but... the common theme is...??

- CPU
- GPU
- Image DSP
- Crypto
...
Abstractions are Evolving Fast
Abstractions are Evolving Fast
Abstractions are Evolving Fast
Abstractions are Evolving Fast
Abstractions are Evolving Fast
Abstractions are Evolving Fast
Abstractions are Evolving Fast
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are Everywhere
Concurrency and Parallelism are everywhere
Concurrency and Parallelism are everywhere

How much parallel and concurrent programming have you learned so far?
Concurrency and Parallelism are everywhere

How much parallel and concurrent programming have you learned so far?
• Concurrency/parallelism can’t be avoided anymore (want a job?)
• A program or two playing with locks and threads isn’t enough
• I’ve worked in industry a lot—I know

Course goal is to expose you to lots of ways of programming systems like these

...So “you should take this course because it’s good for you” (eat your #5(* & kale!)
Outline: Different Approaches to Concurrency

<table>
<thead>
<tr>
<th>Abstract</th>
<th>Concrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locks and Shared Memory Synchronization</td>
<td>Basic Locking</td>
</tr>
<tr>
<td>Language Support</td>
<td>K-Means – pthreads</td>
</tr>
<tr>
<td>Parallel Architectures</td>
<td>Go lab with condition variables, channels, go</td>
</tr>
<tr>
<td></td>
<td>routines</td>
</tr>
<tr>
<td>HPC</td>
<td>GPU Programming lab</td>
</tr>
<tr>
<td>Big Data</td>
<td>MPI lab</td>
</tr>
<tr>
<td>Modern/Advanced Topics</td>
<td>Map-Reduce / Spark Lab</td>
</tr>
<tr>
<td>Whatever Interests YOU</td>
<td>• Specialized Runtimes / Programming Models</td>
</tr>
<tr>
<td></td>
<td>• Auto-parallelization</td>
</tr>
<tr>
<td></td>
<td>• Race Detection</td>
</tr>
<tr>
<td></td>
<td>• Project</td>
</tr>
</tbody>
</table>
Serial vs. parallel program

One instruction at a time (apparently)

Multiple instructions in parallel
Free lunch...

35 YEARS OF MICROPROCESSOR TREND DATA

Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore
Free lunch – is over 😞

35 YEARS OF MICROPROCESSOR TREND DATA

- Transistor number grows (Moore’s law)
- Sequential performance no longer improves
- Cores number grows

Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore
Flynn’s Taxonomy

- **SISD**
  - Single Instruction stream
  - Single Data stream
- **SIMD**
  - Single Instruction stream
  - Multiple Data stream
- **MISD**
  - Multiple Instruction stream
  - Single Data stream
- **MIMD**
  - Multiple Instruction stream
  - Multiple Data stream
• Example: vector operations (e.g., Intel SSE/AVX, GPU)
MIMD

• Example: multi-core CPU
Problem Partitioning

- Domain Decomposition
  - SPMD
  - Input domain
  - Output Domain
  - Both
- Functional Decomposition
  - MPMD
  - Independent Tasks
  - Pipelining
Game of Life

- Given a 2D Grid
- \( v_t(i, j) = F(v_{t-1}(\text{of all its neighbors})) \)
How to Partition Game of Life

• What model fits “best”?
Try domain decomposition

• Each CPU gets part of the input

Issues:
• Can we access v(i+1, j) from CPU 0 as in serial program?
  • Shared memory? Distributed?
  • Time to access v(i+1,j) = Time to access v(i-1,j) ?
  • Scalability vs Latency
• Control
  • Can we assign one v per CPU?
  • Can we assign one v per process/logical task?
  • Task Management Overhead
• Load Balance
• Is it Correct?
  • order of reads and writes is non-deterministic
  • synchronization is required to enforce the order
  • locks, semaphores, barriers, conditionals....
Load Balancing

• Slowest task determines performance
Granularity

\[ G = \frac{\text{Computation}}{\text{Communication}} \]

- Fine-grain parallelism
  - G is small
  - Good load balancing
  - Potentially high overhead
  - Hard to get correct

- Coarse-grain parallelism
  - G is large
  - Load balancing is tough
  - Low overhead
  - Easier to get correct
Performance: Amdahl’s law

- Speedup is bound by serial component
- Split program serial time ($T_{\text{serial}} = 1$) into
  - Ideally parallelizable portion: $A$
    - assuming perfect load balancing, identical speed, no overheads
  - Cannot be parallelized (serial) portion: $1 - A$
  - Parallel time:
    $$T_{\text{parallel}} = \frac{A}{\#\text{CPUs}} + (1 - A)$$

$$\text{Speedup}(\#\text{CPUs}) = \frac{T_{\text{serial}}}{T_{\text{parallel}}} = \frac{1}{\frac{A}{\#\text{CPUs}} + (1 - A)}$$
Performance: Amdahl’s law

\[ \text{Speedup} = \frac{\text{serial run time}}{\text{parallel run time}} \]

\[ \text{Speedup}(\#\text{CPUs}) = \frac{T_{\text{serial}}}{T_{\text{parallel}}} = \frac{1}{\frac{A}{\#\text{CPUs}} + (1 - A)} \]
Amdahl’s law in action

Amdahl's Law

(sources: Wikipedia)

(Thanks Wikipedia)
Strong Scaling vs Weak Scaling

- Amdahl vs. Gustafson

- \( N = \#\text{CPUs}, \ S = \text{serial portion} = 1 - A \)

  - Amdahl's law: \( \text{Speedup}(N) = \frac{1}{\frac{A}{N} + S} \)
    - **Strong scaling**: \( \text{Speedup}(N) \) calculated given total amount of work is fixed
    - Solve same problems faster when problem size is fixed and #CPU grows
    - Assuming parallel portion is fixed, speedup soon seizes to increase

  - Gustafson's law: \( \text{Speedup}(N) = N + (N-1) \cdot S \)
    - **Weak scaling**: \( \text{Speedup}(N) \) calculated given amount of work per CPU is fixed
    - Keep the amount of work per CPU when adding more CPUs to keep the granularity fixed
    - Problem size grows: solve larger problems
    - **Consequence**: speedup upper bound much higher
Super-linear speedup
Super-linear speedup

• Possible due to cache
• But usually just poor methodology
• Baseline: *best* serial algorithm
Super-linear speedup

• Possible due to cache
• But usually just poor methodology
• Baseline: *best* serial algorithm
• Example:
Super-linear speedup

• Possible due to cache
• But usually just poor methodology
• Baseline: *best* serial algorithm
• Example:
  • Efficient **bubble sort** takes:
    • Parallel 40s
    • Serial 150s
    • $Speedup = \frac{150}{40} = 3.75$ ?
  
  • NO!
    • Serial quicksort runs in 30s
    • $\Rightarrow Speedup = 0.75$
Concurrent and Correctness

If two threads execute this program concurrently, how many different final values of X are there?

Initially, $X == 0$.

Thread 1

```java
void increment() {
    int temp = X;
    temp = temp + 1;
    X = temp;
}
```

Thread 2

```java
void increment() {
    int temp = X;
    temp = temp + 1;
    X = temp;
}
```

Answer:
A. 0
B. 1
C. 2
D. More than 2
Schedules/Interleavings

Model of concurrent execution

- Interleave statements from each thread into a single thread
- If any interleaving yields incorrect results, synchronization is needed

Thread 1

```
tmp1 = X;
tmp1 = tmp1 + 1;
X = tmp1;
```

Thread 2

```
tmp2 = X;
tmp2 = tmp2 + 1;
X = tmp2;
```
Schedules/Interleavings

Model of concurrent execution

- Interleave statements from each thread into a single thread
- If any interleaving yields incorrect results, synchronization is needed

If $X=0$ initially, $X = 1$ at the end. WRONG result!
Locks fix this with Mutual Exclusion

void increment() {
    lock.acquire();
    int temp = X;
    temp = temp + 1;
    X = temp;
    lock.release();
}

• Mutual exclusion ensures only safe interleavings
  • *But it limits concurrency, and hence scalability/performance*
Lab #1

• Basic synchronization

• [http://www.cs.utexas.edu/~rossbach/cs378/lab/lab0.html](http://www.cs.utexas.edu/~rossbach/cs378/lab/lab0.html)

• *Start early!!!*
Questions?