Parallel Systems
Welcome to cs380p

Chris Rossbach + Calvin Lin
CS380p
Outline for Today

• Course Overview
• Course Details and Logistics
• Concurrency & Parallelism Basics
  • Motivation
  • Problem Decomposition

Acknowledgments: some materials in this lecture borrowed from or built on materials 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 Olukotan, Gupta
<table>
<thead>
<tr>
<th><strong>Course Details</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Course Name:</strong></td>
</tr>
<tr>
<td><strong>Lectures:</strong></td>
</tr>
<tr>
<td><strong>Class Web Page:</strong></td>
</tr>
<tr>
<td><strong>Instructors:</strong></td>
</tr>
<tr>
<td><strong>Text:</strong></td>
</tr>
</tbody>
</table>

*Please read the syllabus!*
Why you should take this course
Why you should take this course

• Parallelism is super-cool and super-important
Why you should take this course

• Parallelism is super-cool and super-important
• You’ll learn important concepts and background
Why you should take this course

• Parallelism is super-cool and super-important
• You’ll learn important concepts and background
• Have fun programming cool systems
  • GPUs! Multi-core!
  • Modern infrastructure and programming languages
  • Interesting synchronization primitives (not just about locks!)
Why you should take this course

• Parallelism is super-cool and super-important
• You’ll learn important concepts and background
• Have *fun* programming cool systems
  • GPUs! Multi-core!
  • Modern infrastructure and programming languages
  • Interesting synchronization primitives (not just about locks!)

Two perspectives:
• 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

CPU
My first computer
My first computer

CPU

Storage

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

CPU

screen

Storage

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

- CPU
- Wires + gobbledygook
- Storage
- Screen
- 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
CPU
GPU
Image DSP
Crypto

...
Modern Technology Stack
Modern Technology Stack
Modern Technology Stack
Modern Technology Stack
Modern Technology Stack
Modern Technology Stack

- FPGA
- GPU
- ASIC
- NVM
- DSP
- CRYPT
Modern Technology Stack

Introduction
Concurrency and Parallelism are Everywhere

Applications

- device APIs
- Runtime

- ioctl
- mmap

- driver

- Vendor-specific

- CPU
- GPU
- DISK
- ASIC

- NVM
- FPGA
- DSP
- CRYPT
Concurrency and Parallelism are Everywhere

Hypervisor

Applications

 Device APIs
 Runtime

 IOCTL

 Mmap

 Driver

 vCPU

 vGPU

 vDisk

 vASIC

 vNVM

 vFPGA

 vDSP

 vCRYPT

 HYPervisor

 CPU

 GPU

 DISK

 ASIC

 NVM

 FPGA

 DSP

 CRYPT

 Applications

 Vendor-specific driver

 Runtime device APIs

 Applications

 Devices

 Vendor-specific driver

 Runtime device APIs

 Runtime device APIs

 Runtime device APIs

 Runtime device APIs

 Runtime device APIs

 Runtime device APIs

 Introduction
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

- CPU(s)
- GPU
- Image DSP
- Crypto
- ...
Concurrency and Parallelism are everywhere
Concurrency and Parallelism are everywhere

Key concerns:

- CPU(s)
- GPU
- Image DSP
- Crypto
- ...
Concurrency and Parallelism are everywhere

Key concerns:
• Concurrency/parallelism can’t be avoided anymore (want a job?)
• A program or two playing with locks and threads isn’t enough

Course goal is to expose you to lots of ways of programming systems like these
**Goal**: Make Parallelism Your Close Friend  
**Method**: Use Many Different Approaches

<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></td>
<td>Prefix sum – pthreads</td>
</tr>
<tr>
<td>Language Support</td>
<td>Go lab: condition variables, channels, go routines</td>
</tr>
<tr>
<td>Parallel Architectures</td>
<td>GPU Programming Lab</td>
</tr>
<tr>
<td>HPC</td>
<td>MPI: Barnes-Hut lab</td>
</tr>
<tr>
<td>Modern/Advanced Topics</td>
<td>• Specialized Runtimes / Programming Models</td>
</tr>
<tr>
<td></td>
<td>• Auto-parallelization</td>
</tr>
<tr>
<td></td>
<td>• Race Detection</td>
</tr>
</tbody>
</table>
Serial vs. Parallel Program
Serial vs. Parallel Program

One instruction at a time (apparently)
Serial vs. Parallel Program

One instruction at a time (apparently)

Multiple instructions in parallel
Serial vs. Parallel Program

Key concerns:

Multiple instructions in parallel
Serial vs. Parallel Program

Key concerns:
- Programming model

Multiple instructions in parallel
Serial vs. Parallel Program

Key concerns:
- Programming model
- Execution Model

Multiple instructions in parallel
Serial vs. Parallel Program

Key concerns:
• Programming model
• Execution Model
• Performance/Efficiency

Multiple instructions in parallel
Serial vs. Parallel Program

Key concerns:
• Programming model
• Execution Model
• Performance/Efficiency
• Exposing parallelism

Multiple instructions in parallel
Technology Trends

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
Execution Models: Flynn’s Taxonomy
Execution Models: Flynn’s Taxonomy

<table>
<thead>
<tr>
<th></th>
<th>SISD</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>Single Instruction stream</td>
<td>Single Instruction stream</td>
</tr>
<tr>
<td>Data</td>
<td>Single Data stream</td>
<td>Multiple Data stream</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>MISD</th>
<th>MIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>Multiple Instruction stream</td>
<td>Multiple Instruction stream</td>
</tr>
<tr>
<td>Data</td>
<td>Single Data stream</td>
<td>Multiple Data stream</td>
</tr>
</tbody>
</table>
Execution Models: Flynn’s Taxonomy

- Normal Serial program

<table>
<thead>
<tr>
<th></th>
<th>Single Instruction stream</th>
<th>Multiple Instruction stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD</td>
<td>Multiple Data stream</td>
<td>Single Data stream</td>
</tr>
<tr>
<td>MISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIMD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Execution Models: Flynn’s Taxonomy

<table>
<thead>
<tr>
<th>SISD</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instruction stream</td>
<td>Single Instruction stream</td>
</tr>
<tr>
<td>Single Data stream</td>
<td>Multiple Data stream</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Instruction stream</td>
</tr>
<tr>
<td>Multiple Data stream</td>
</tr>
</tbody>
</table>

Uncommon architecture: Fault – tolerance  
Pipeline parallelism
Execution Models: Flynn’s Taxonomy

Our main focus
SIMD
SIMD

P1
- prev instrct
- load A(1)
- load B(1)
- C(1)=A(1)*B(1)
- store C(1)
- next instrct

P2
- prev instrct
- load A(2)
- load B(2)
- C(2)=A(2)*B(2)
- store C(2)
- next instrct

Pn
- prev instrct
- load A(n)
- load B(n)
- C(n)=A(n)*B(n)
- store C(n)
- next instrct

time
SIMD

- Example: vector operations (e.g., Intel SSE/AVX, GPU)
SIMD

- Example: vector operations (e.g., Intel SSE/AVX, GPU)
MIMD
MIMD

• Example: multi-core CPU
MIMD

• Example: multi-core CPU
Problem Partitioning
Problem Partitioning

• Decomposition: Domain v. Functional
Problem Partitioning

• Decomposition: Domain v. Functional
• Domain Decomposition (Data Parallel)
  • SPMD
  • Input domain
  • Output Domain
  • Both
Problem Partitioning

• Decomposition: Domain v. Functional
• Domain Decomposition (Data Parallel)
  • SPMD
  • Input domain
  • Output Domain
  • Both
Problem Partitioning

• Decomposition: Domain v. Functional
• Domain Decomposition (Data Parallel)
  • SPMD
  • Input domain
  • Output Domain
  • Both
• Functional Decomposition (Task Parallel)
  • MPMD
  • Independent Tasks
  • Pipelining
Problem Partitioning

• Decomposition: Domain v. Functional
• Domain Decomposition (Data Parallel)
  • SPMD
  • Input domain
  • Output Domain
  • Both
• Functional Decomposition (Task Parallel)
  • MPMD
  • Independent Tasks
  • Pipelining
Game of Life
Game of Life

• Given a 2D Grid:

\[ v_t(i, j) = F(v_{t-1}(\text{of all its neighbors})) \]
Game of Life

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

What decomposition fits “best”?
- Domain (data parallel)
- Functional (task parallel)
Domain decomposition
Domain decomposition

Each CPU gets part of the input
Domain decomposition

Each CPU gets part of the input

CPU 0

CPU 1
Domain decomposition

Each CPU gets part of the input

For next time:
Domain decomposition

Each CPU gets part of the input

For next time:
• What issues/challenges might arise with this solution?
Domain decomposition

Each CPU gets part of the input

For next time:
• What issues/challenges might arise with this solution?
• How could we do a functional decomposition?