

# Introducing Parallel Computing in a Second CS Course

Tia Newhall, Kevin C. Webb,  
Vasanta Chaganti, Andrew Danner

Computer Science Dept. Swarthmore College  
Swarthmore, PA USA



# Expose Students to PDC Concepts Early

P&D computing increasingly important

- P&D world (and students know it)
- Prepare work force, gradschool, undergraduate research

All CS majors see PDC concepts early and again

- Teach “Parallel Thinking” early
- Natural to think about P&D questions in different ways later
- Our curriculum ensures that they see it later
- Prepares students for upper-level courses with P&D and systems

# Our Course: Introduction to Computer Systems (CS31)

## CS in the Liberal Arts: Fewer Courses & Shallow Prereq Hierarchy

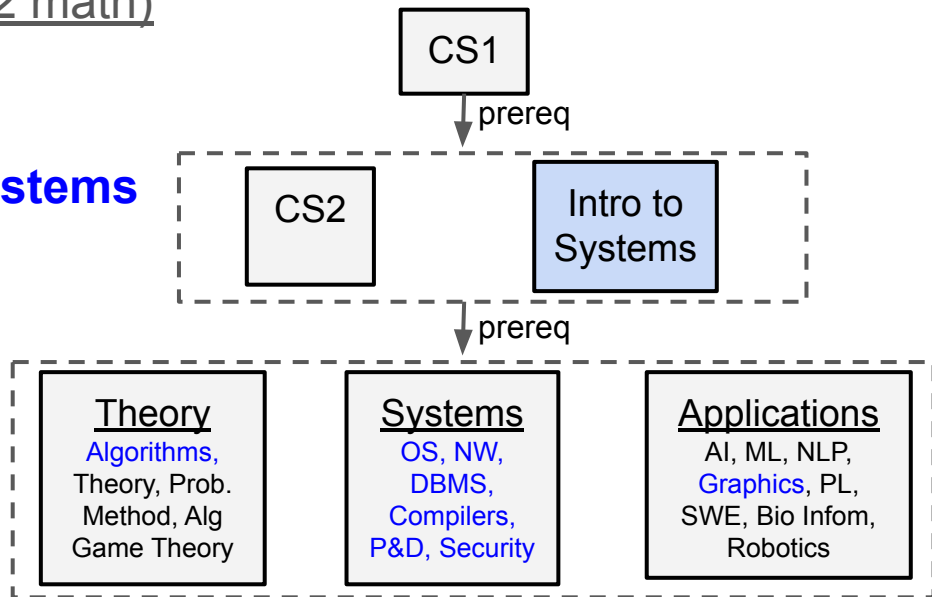
Swarthmore: 8 CS courses in major (+ 2 math)

1 Introductory-level: CS1

2 Intermediate-level: CS2 & **Intro to Systems**

5 Upper-level:

- Grouped in 3 buckets
- At least 1 in each for breadth
- **P&D in all Systems, some other**



# Why in a Second Course?

- Early Curricular Exposure to PDC
  - Provides common introduction to systems and PDC topics to every student in any upper-level course
  - Increased Depth vs. in CS1: Can focus more on PDC and less on basic algorithmic problem solving & intro programming
  - Increased Breadth at intro level: introduce many concepts will see again in different contexts
- Easier Part of our Curriculum to Change to incorporate PDC topics
  - Replaced more traditional Computer Organization course
  - More constraints/goals/opinions about CS1 and CS2 (Data structures & Algs)

# CS31 Introduces Many TCPP Topics

TCPP Category	Intro to Systems Topics
Pervasive	concurrency, asynchrony, locality, performance
Architecture	multicore, caching, latency, bandwidth, atomicity, consistency, coherency, pipelining, instruction execution, memory hierarchy, multithreading, buses process ID, interrupts
Programming	shared memory parallelization, pthreads, critical sections, producer-consumer, synchronization, deadlock, race cond, memory data layout, locality, signals
Algorithms	dependencies, space/memory, speedup, Amdahl's law, synchronization, efficiency

(Table 1)

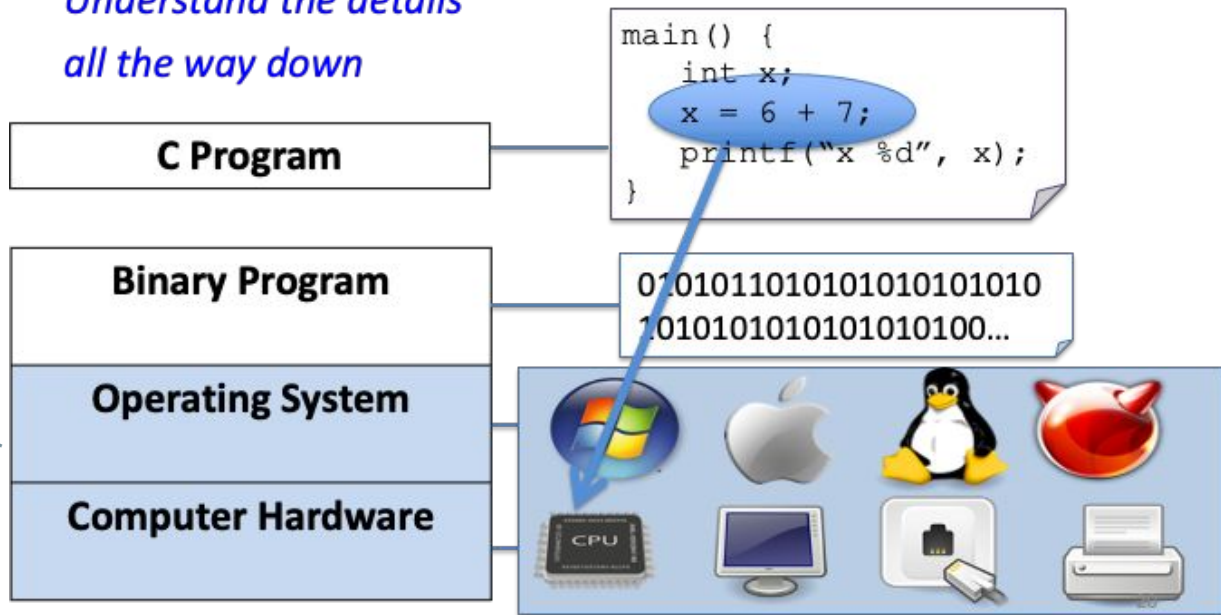
# Course Goal 1: How a Computer Runs a Program

*A vertical slice  
through the  
computer*

*Understand the details  
all the way down*

from HLL to binary  
instructions executed  
by HW circuitry

Computer  
System

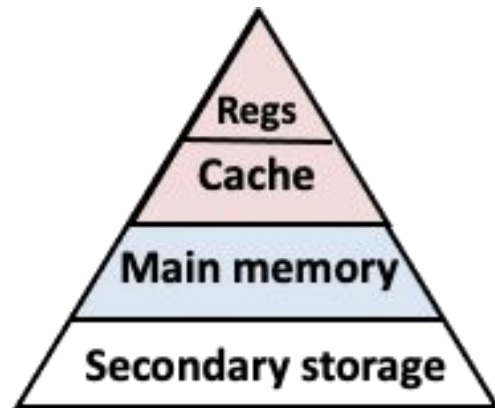


# Course Goal 2: Systems Costs and Efficiency

*It is not all big-O*

Recognize & Evaluate systems costs associated with running a program

- Focus on Memory Hierarchy & caching
- Also in context of OS and parallel

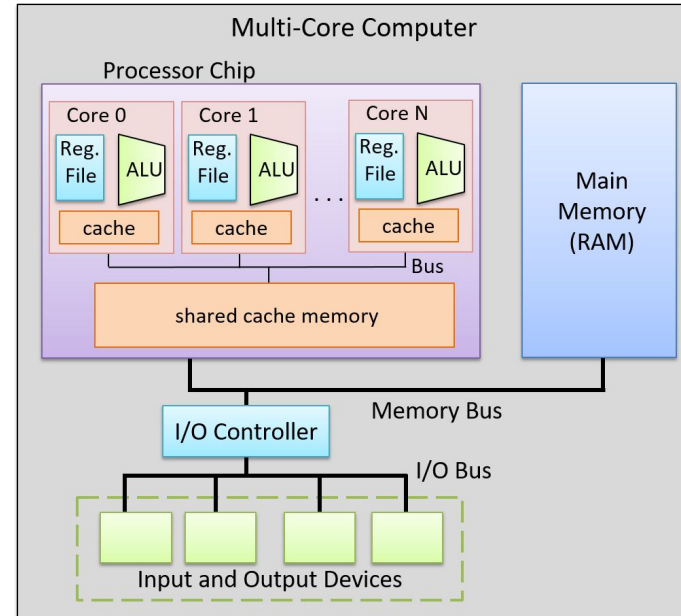
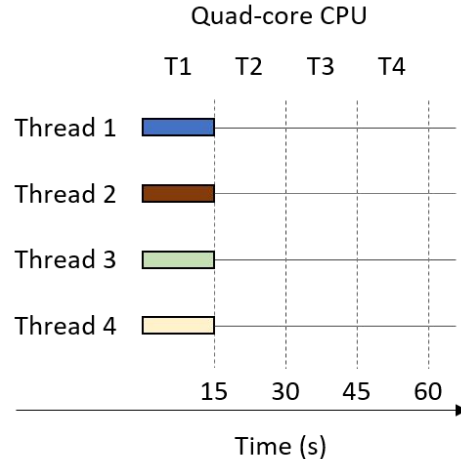


# Course Goal 3: Intro to Parallel Computing

*Taking advantage of the power of parallel computing*

In the context of goals 1 and 2: how programs run & efficiency

- Focus on Shared Memory parallelism
- Multicore and pthreads
- Synchronization



# Class Structure

Lecture: typically 60 students (large for LACS)

Lecture with active learning: clicker questions, group discussion, reading quiz

Weekly Lab Session (90 minutes per week)

Learn tools for doing lab work: C, gdb, valgrind, logisim, pthreads

Lab Assignments

C, assembly, and pthreads programming, logisim, gdb assembly tracing

Weekly Written Homeworks

Short, reinforce lecture topics, have used assigned HW-groups

Course-Specific Student Mentors (CS Ninjas, SIGCSE'14)

Help in Lab sessions, in-class activities, Run evening help sessions for lab work

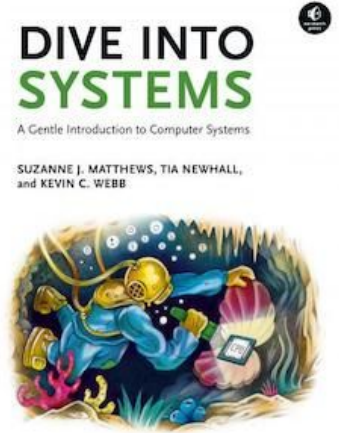
# Issue: finding the right textbook

## Problem:

- Needed introductory-level coverage of a broad set of organization, systems, parallel topics

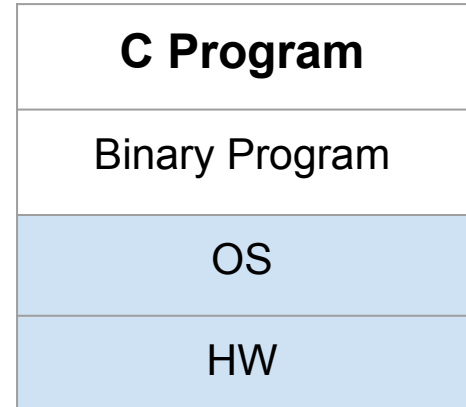
## Solution: write one (Matthews, Newhall, Webb)

- Free, online textbook [diveintosystems.org](https://diveintosystems.org)
- Broad set of topics at Intro level
- Useful to a range of courses/uses (30+ institutions)
- SIGCSE'21 early adopter evaluation paper
- New NSF funded effort to add interactive exercises
- Print version out by fall (will always remain free online)



# C Programming: used throughout the course


- 1 week intro to C
  - Students have a CS1 background in Python (or Java)
- New features introduced in context of other topics
  - Pointers
  - Dynamic memory allocation
  - 2D arrays, structs



*Part of a Vertical Slice  
through a Computer  
(Goal 1)*

# C Programming: Labs and HWs

- Learning C Labs
  - Intro: 1D static arrays, searching, sorting
  - 1D dynamic arrays, searching, sorting
  - 2D dynamic arrays, GOL w/ASCII & ParaVis visualization (EduPar'19)

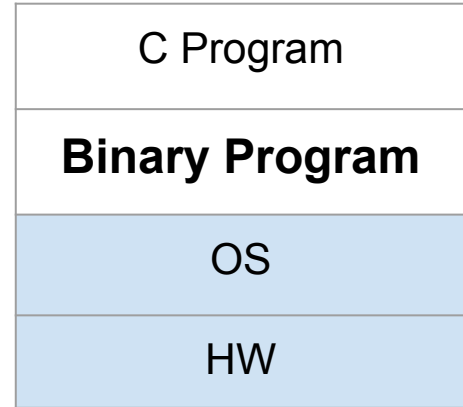


```
File Edit View Terminal Tabs Help
Round: 100
. . . . .
. @ @ @ . . . . .
. . . . . @ @ @
. . . . . @
. . . . .
. . . . .
. @ @ @
. . . @
. . . . . @ @
. . . . . @ @
. . . . .
Live cells: 17
Total time: 10.867 seconds
After 100 rounds on 20x20, number of live cells is 17
```

- HW: stack drawing, code tracing, expressions & type

# Binary Representation

- Usually 1st topic, before intro C
- Topics
  - C types in binary: unsigned and signed (2's complement)
  - Binary arithmetic (+, -) signed unsigned, overflow
  - Other binary operators



*Part of how program  
encoded to run on computer  
And how arithmetic on  
binary values works  
(Goal 1)*

# Binary Representation Labs and HW

- Simple C program to evaluate answers to questions about binary representation and arithmetic
  - First C program: introduces compilation, basic syntax, simple functions
- Written part: convert decimal/binary/hex, binary arithmetic

```
  11111111 (-1)
+  00000001 (1)
-----
1 00000000 (0)
```

carry out

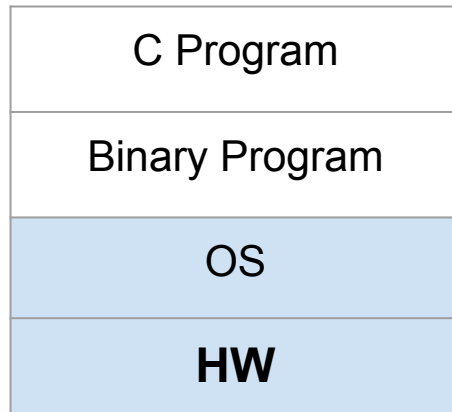
```
  10101100
& 11010111
-----
 10000100
```

```
0xff → 0b11111111 -> 255
0xff → 0b11111111 -> -1
```

...

# Architecture: focus on CPU architecture

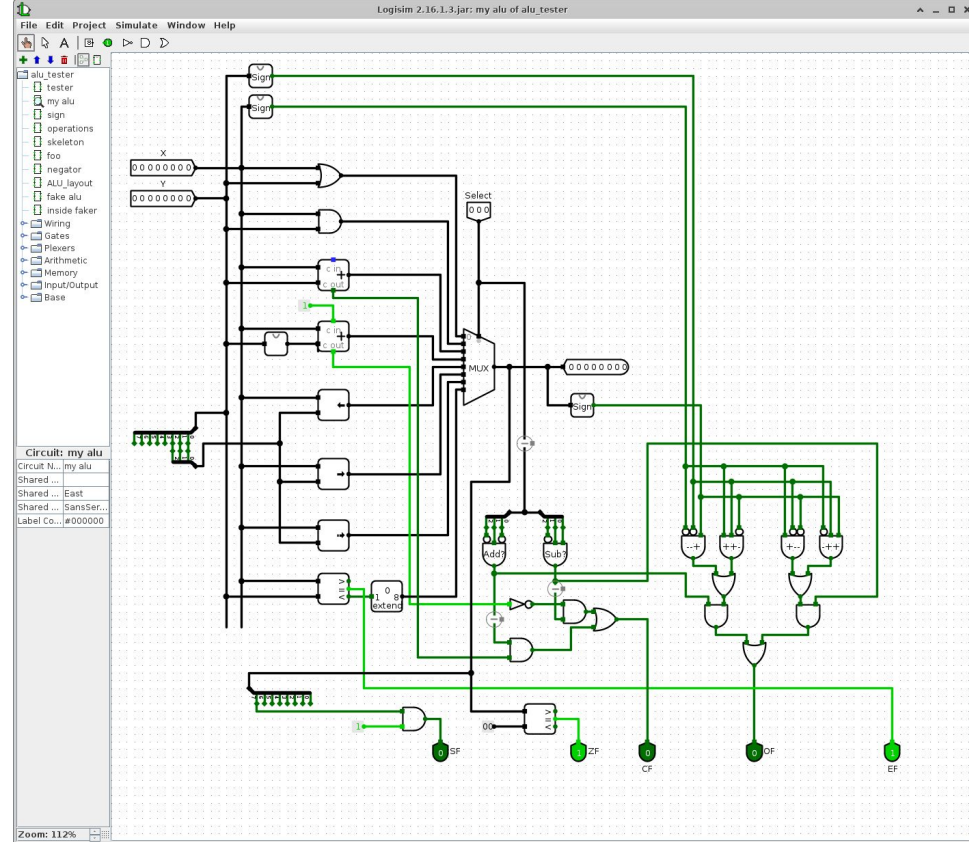
- Von Neumann Architecture
- ISA
- Simple circuits from basic gates (AND, OR, NOT)
- Arithmetic-logic, Storage, Control
- ALU, Register File, Clock
- Instruction Execution, Pipelining
- Some Modern Architectures (**multicore**, multithreaded, ILP)
- Theme of Abstraction as add complexity (from gates to CPU)



*How HW is designed to execute a stream of binary instructions on binary data (Goal 1)*

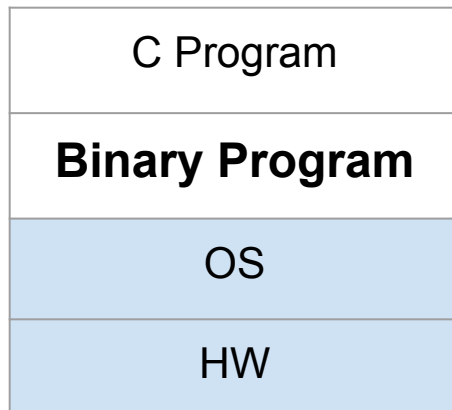
# Architecture Labs and HWs

- Labs using Logisim
  - 4-bit adder from AND, OR, NOT gates
  - Build CPU for 8 instruction ISA (ALU, Reg File, IR, PC, clock) w/condition codes
- Truth table to/from circuit
- Circuit Tracing



# Assembly Programming

- Translating C to/from Assembly
- X86 (ARM-64 in Fall'22)
  - ISA: instruction set, registers
  - Arithmetic/logic
  - Control, Loops
  - Functions, the Stack
  - Pointers
  - Array & Struct layout & access



*Part of how program  
encoded to run on computer  
(Goal 1)*

*Tie to C code and to what  
know about how CPU  
designed to run program  
instructions*

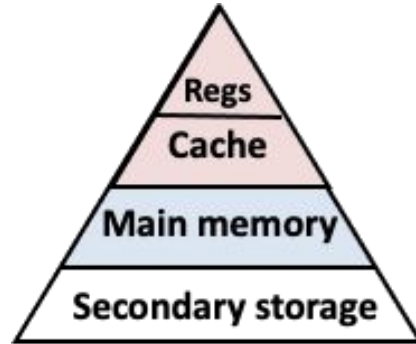
# Assembly Programming Labs and HWs

- Translating between C & Assembly
  - simple instructions
  - loops, if-else
  - functions
  - arrays and pointers
- Binary Maze Program
  - gdb to find way out
  - based on Bryant & O'Hallaron's Binary Bomb
- Code tracing showing stack & register contents

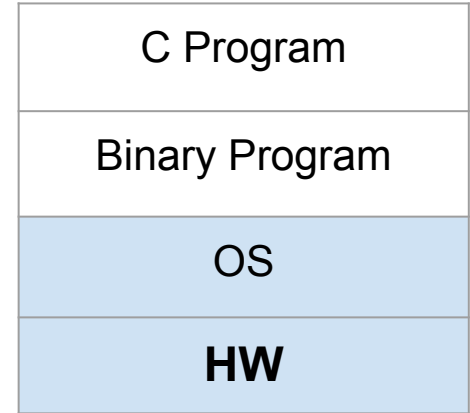
```
0x00001a38 <+73>: mov    (%ebx),%eax
0x00001a3a <+75>: add   %eax,%eax
0x00001a3c <+77>: cmp   %eax,0x4(%ebx)
0x00001a3f <+80>: je    0x1a31 <phase_2+66>
0x00001a41 <+82>: sub   $0xc,%esp
0x00001a44 <+85>: push $0x2
0x00001a46 <+87>: call  0x237b <tripped_up>
0x00001a4b <+92>: add   $0x10,%esp
0x00001a4e <+95>: jmp   0x1a31 <phase_2+66>
0x00001a50 <+97>: mov   0x1c(%esp),%eax
0x00001a54 <+101>: xor   %gs:0x14,%eax
0x00001a5b <+108>: jne   0x1a63 <phase_2+116>
0x00001a5d <+110>: add   $0x24,%esp
0x00001a60 <+113>: pop   %ebx
0x00001a61 <+114>: pop   %esi
0x00001a62 <+115>: ret
0x00001a63 <+116>: call  0x1a64 <phase_2+117>
End of assembler dump.
(gdb) break *0x00001a38
Breakpoint 1 at 0x1a38
(gdb) ni
```

# Memory Hierarchy and Caching

- Memory Hierarchy
  - buses, devices
  - W/R mechanics



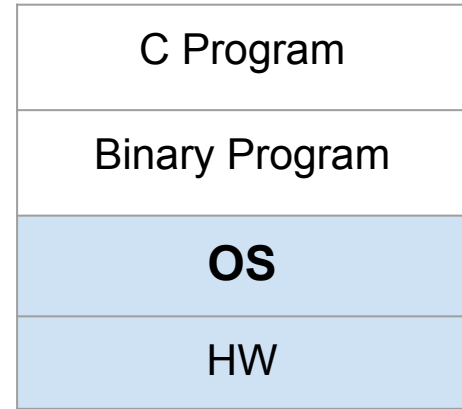
- CPU Caching
  - Direct Mapped, Set Associative
  - Locality
  - Data layout, access patterns
- Written HW
  - Address Translation
  - DM & SA Cache effects on sequence of W/R



*How Computer System  
Designed to Efficiently Run  
programs and Systems  
Costs with program  
execution (Goal 2)*

# Introduction to Operating Systems

- Role of OS & how works: boot, interrupts, kernel/user mode
- Two main Abstractions Related to How Computer Runs Program
  1. **Processes**: state, context switch, multiprogramming/timesharing, process hierarchy, fork, exec, wait, exit, signals, asynchrony, concurrency
  2. **VM**: VAS, VA/PA, paging, replacement, TLB



*Part of how program is run on a computer (Goal 1)*

*Part of how to efficiently run program on computer and systems costs assoc with running program (Goal 2)*

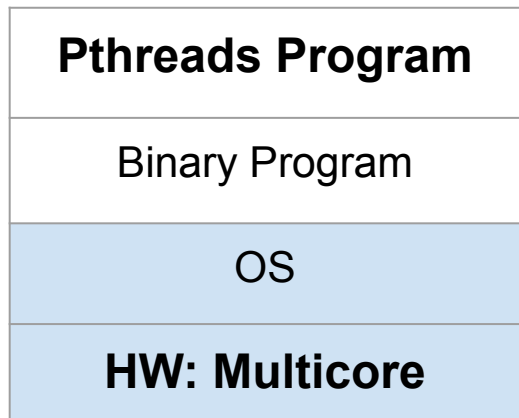
# Operating System Labs & HWs

- Unix shell & parse command library:
  - foreground, background (asynch, signals), history
  - starting point for lab in UL OS
- Process HW: tracing code with fork/exec/wait/, concurrency
- VM: page table translations, replacement

```
tiascs31shell> sleeper &
tiascs31shell> ps
      PID TTY          TIME CMD
1248792 pts/0    00:00:00 tcsh
1249679 pts/0    00:00:00 cs31shell
1249703 pts/0    00:00:00 sleeper
1249705 pts/0    00:00:00 ps
tiascs31shell> history
  0  ls
  1  sleeper &
  2  ps
  3  history
tiascs31shell> !1
tiascs31shell> history
  0  ls
  1  sleeper &
  2  ps
  3  history
  4  sleeper &
  5  history
tiascs31shell> ps
      PID TTY          TIME CMD
1248792 pts/0    00:00:00 tcsh
1249679 pts/0    00:00:00 cs31shell
1249736 pts/0    00:00:00 sleeper
1249749 pts/0    00:00:00 ps
tiascs31shell>
```

# Parallel Computing

- Focus on Shared Memory
  - Already seen Multicore
  - Parallelism vs. Concurrency
  - Speedup
- OS's Thread Abstraction:
  - Follows from Process and VM
  - Thread Scheduling
  - VAS Sharing

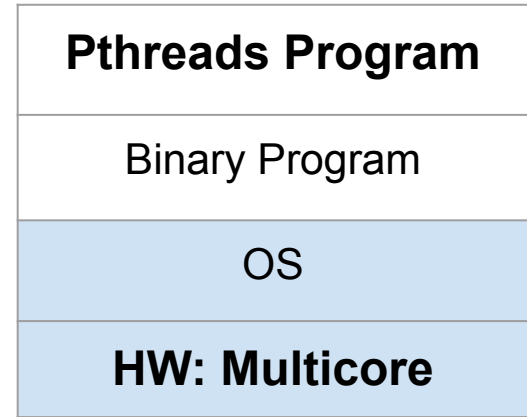


*Part of how a parallel program is run on a computer*

*Part of how to efficiently write program to run on multicore computer*

# Parallel Computing (cont.)

- Parallel Pthreads Programming
  - Thread create-join
  - Synchronization, mutual exclusion, critical sections
  - mutex, barrier, condition vars
  - Race condition, deadlock
- Speed-up, Amdahl's law
  - Embarrassingly Parallel
  - Parallel Costs/Overheads

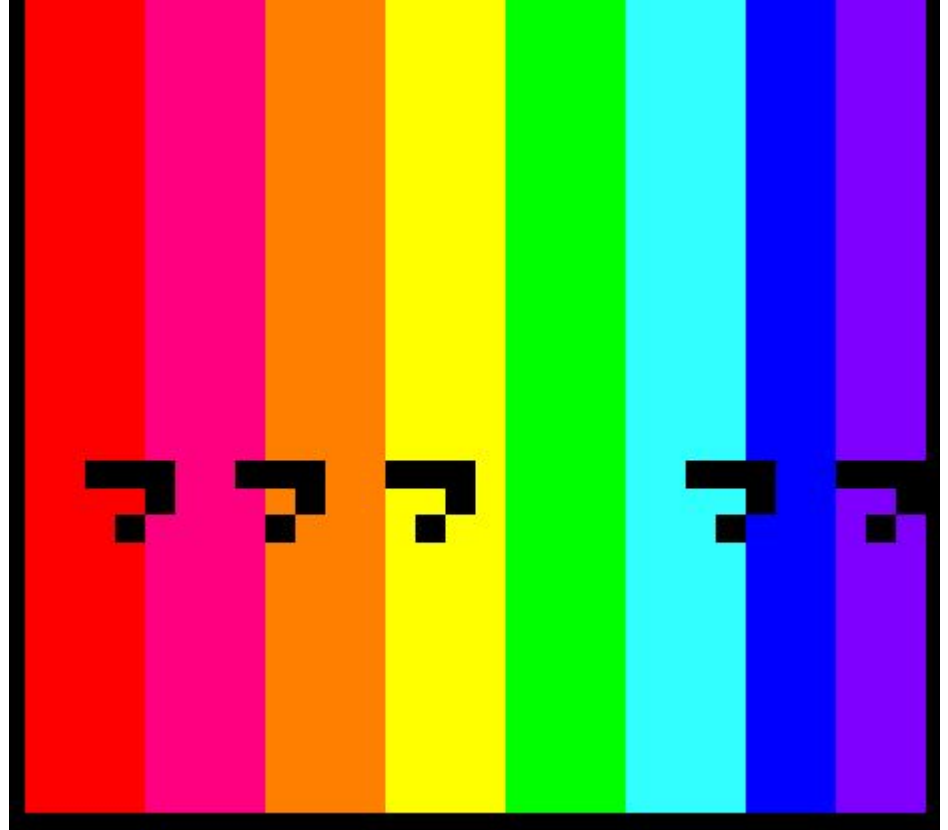


*Part of how a parallel program is run on a computer*

*Part of how to efficiently write program to run on multicore computer*

# Parallel Computing Labs and HWs

- Written synchronization, parallel execution
- Producer/Consumer bounded buffer
- Pthreads GOL w/ASCII & ParaVis (EduPar'19)
  - Vary partitioning & number threads
  - ParaVis helps debug column/row partitioning
  - Starting point for lab in UL P&D Computing



# CS31 Student Survey Results (~300 students over 5 offerings)

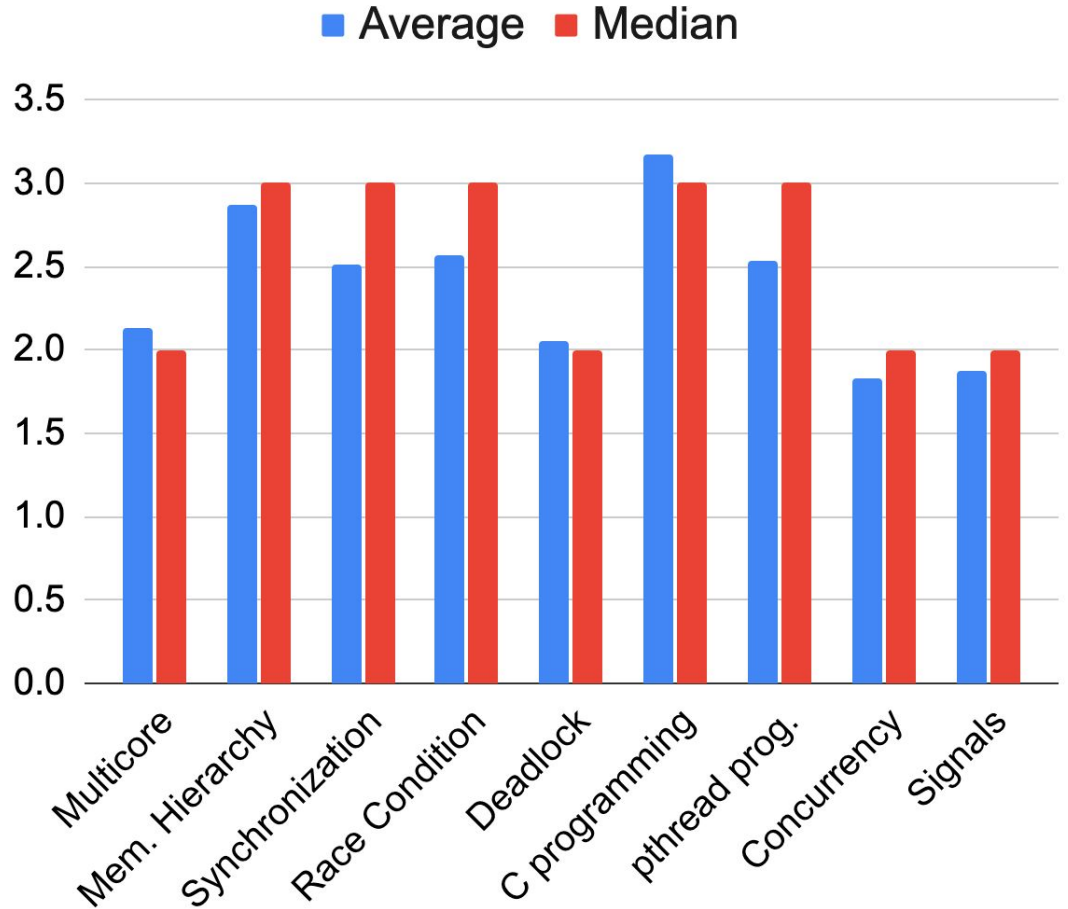
- Appreciated Exposure to PDC
  - *“Parallelism is great! Also pipelining. Those two concepts are super applicable to life broadly”*
  - *“exposed to a great deal of new concepts, especially...new ways for managing programs like threading and forking”*
- Gained new Systems Perspective, evaluating trade-offs
  - “I like how we unpacked a lot of what goes on behind the scenes”
  - “really interesting to think about...questions of efficiency in different things we talked about, and see how those really apply to computer systems that we use every day”
  - “should keep in mind both BigO and what happens in hardware”
- Built Confidence in thinking about systems & parallelism
  - “I can’t wait to take more systems courses!”
  - “learning how to parallelize programs were really interesting, and now I want to take parallel and distributed computing”

# Upper-level Student's Understanding of PDC Concepts from CS31

## Bloom's Taxonomy Rantings

- 0: don't know
- 1: recognize
- 2: can define
- 3: can explain
- 4: can apply

Don't expect level 4 understanding on all topics after CS31 (intro level, 1st introduction)



# Conclusions

- Early PDC instruction in a second course works well in our curriculum, and we believe more generally too (10+ years of CS31 at Swarthmore)
  - Students with CS1 background  $\Rightarrow$  can focus more on PDC
  - CS31 Pre-requisite to upper-level courses ensures all students see PDC early
- Focus on Shared Memory Parallelism
  - Fits naturally with larger course goals, follows naturally from previous course topics
  - Allows for more depth of coverage
- Faculty teaching Upper-level courses with PDC content note w/CS31:
  - Students naturally think in parallel and distributed ways from day one
  - Students start with programming and thinking skills and PDC background that allows us to spend more time on advanced PDC in upper-level courses (can cover more PDC and cover more in-depth than before)

Thank you.

Questions?

CS31 Webpages with resources: [www.cs.swarthmore.edu/~newhall/cs31](http://www.cs.swarthmore.edu/~newhall/cs31)

Dive into Systems Textbook: [diveintosystems.org/](http://diveintosystems.org/)