CS 220:
Introduction to Parallel Computing

Welcome to CS 220!

- Glad to have you all in class!

- Lecture Information:

  Time: MWF 3:30 – 4:35pm

  Room: HR 148

  Course website:

  http://www.cs.usfca.edu/~mmalensek/cs220

Staff

- Instructor: **Matthew Malensek**
  - Office Hours: MF 10-11am, Th 1-2pm (HR 416)
  - Research: distributed systems, big data

- TA: **Ivy An**
  - Office Hours: MW 1-2pm

Enrollment Status

- We currently have 25 students enrolled with 19 more on the waitlist
- If you're enrolled, great!
- If not:
  - Come put your name on the sign-up sheet after class
  - Email me if you're a special case (need the course to make progress, can't get in to any CS courses, etc…)
    - You may also want to let our chair, Prof. Wolber, know

Quick Note: Prerequisites

- CS 110 with a B or better and instructor permission

  - or -

- CS 112 with a C or better

Today's Agenda

- Syllabus
- Why C?
- Modern CPU Performance
- Parallel Computing Background

Today's Agenda

- **Syllabus**
- Why C?
- Modern CPU Performance
- Parallel Computing Background

Staying up to Date

- Check the course website before class for:
  - Syllabus
  - Recent announcements
  - Assignments
  - Printable lecture notes
- Grades will be posted on Canvas
- Project submissions: GitHub

Course Roadmap

1. C programming
2. Parallel computing with MPI
3. Programming with threads and parallelism primitives
4. Wrapping up: GPU programming

- → A tentative schedule is available online

Books

- *Required*:

  Brian Kernighan and Dennis Ritchie, *The C Programming Language*, 2nd edition, Prentice-Hall, 1988.

- *Optional*:

  Peter Pacheco, *An Introduction to Parallel Programming*, Morgan-Kaufmann, 2011.

Course Structure

- Class sessions will introduce C and parallel programming theory
- We will work through programming examples together as a class
  - Remember to ask questions!
- You will start on homework assignments (labs) to apply what we've learned
- We'll also work on 4-5 larger projects

Evaluation

- Besides homework and projects, you'll be evaluated in two ways:
  - Midterms
  - Final exam
- We'll have ~2 midterms
- Cumulative final

Grade Distribution

- Homework: 15%
- Projects: 50%
- Midterms: 20%
- Final: 15%

Grading

Score	Grade
100 – 93	A
92 – 90	A-
89 – 87	B+
86 – 83	B
82 – 80	B-
79 – 77	C+
76 – 73	C
72 – 70	C-
69 – 67	D+
66 – 63	D
62 – 60	D-
59 – 0	F

Policies

- Assignments are due at **11:59 pm** on the due date
- Late homework is not accepted
- Late projects are penalized 10% per day, for a maximum of three days (no credit thereafter)

Today's Agenda

- Syllabus
- **Why C?**
- Modern CPU Performance
- Parallel Computing Background

Oh Say Can you C?

- The C programming language was invented around 1970
  - It's old.
  - Legend has it that Dennis Ritchie invented it while he was riding around in his horse-drawn carriage
- Jokes aside, C can be a tough language to learn
- The good news? C is a very simple language!
- The bad news? C is a very simple language!

So, why learn this old thing?

- Nearly all operating systems are written in C
  - Linux: almost all C
  - macOS: most of the low-level functionality is C
  - Windows: C and C++
  - Two of the most popular mobile operating systems are based on these…
- Embedded systems: elevators, refrigerators, routers, TVs are all often written in C
- High-performance software is often written in C

TIOBE Language Rankings

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215%	-3.06%
2	2		C	11.037%	+1.69%
3	3		C++	5.603%	-0.70%
4	5	^	Python	4.678%	+1.21%
5	4	v	C#	3.754%	-0.29%

https://www.tiobe.com/tiobe-index/

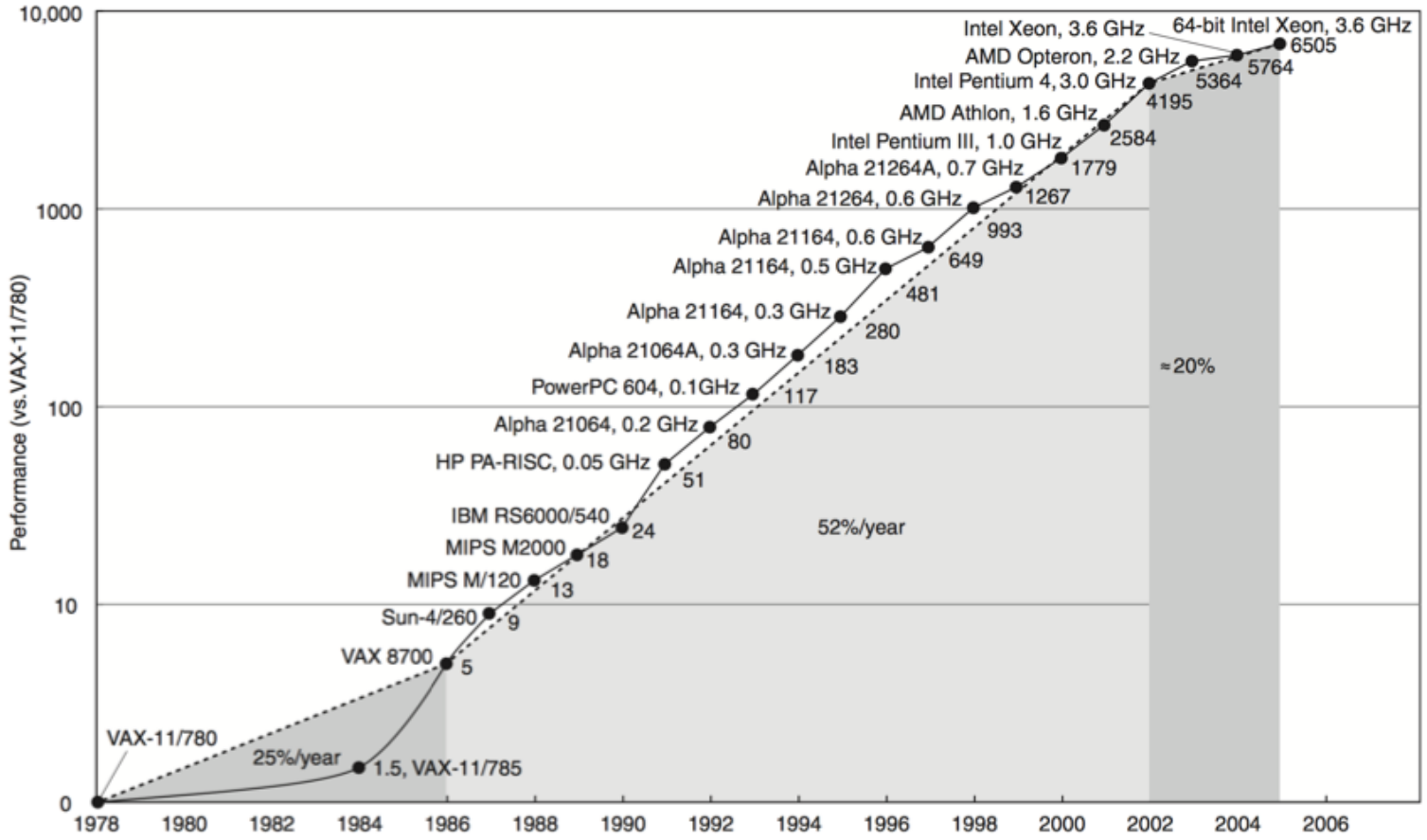
C's Popularity isn't just Historical

- C is very fast and efficient
  - It's a thin layer above the actual hardware
  - Languages like Python or Java operate on higher levels of abstraction
- C is always in the back of hardware designers' minds
- C is easy to interoperate with
  - Slow Python code? Re-implement the function in C and call it easily from your Python app

Today's Agenda

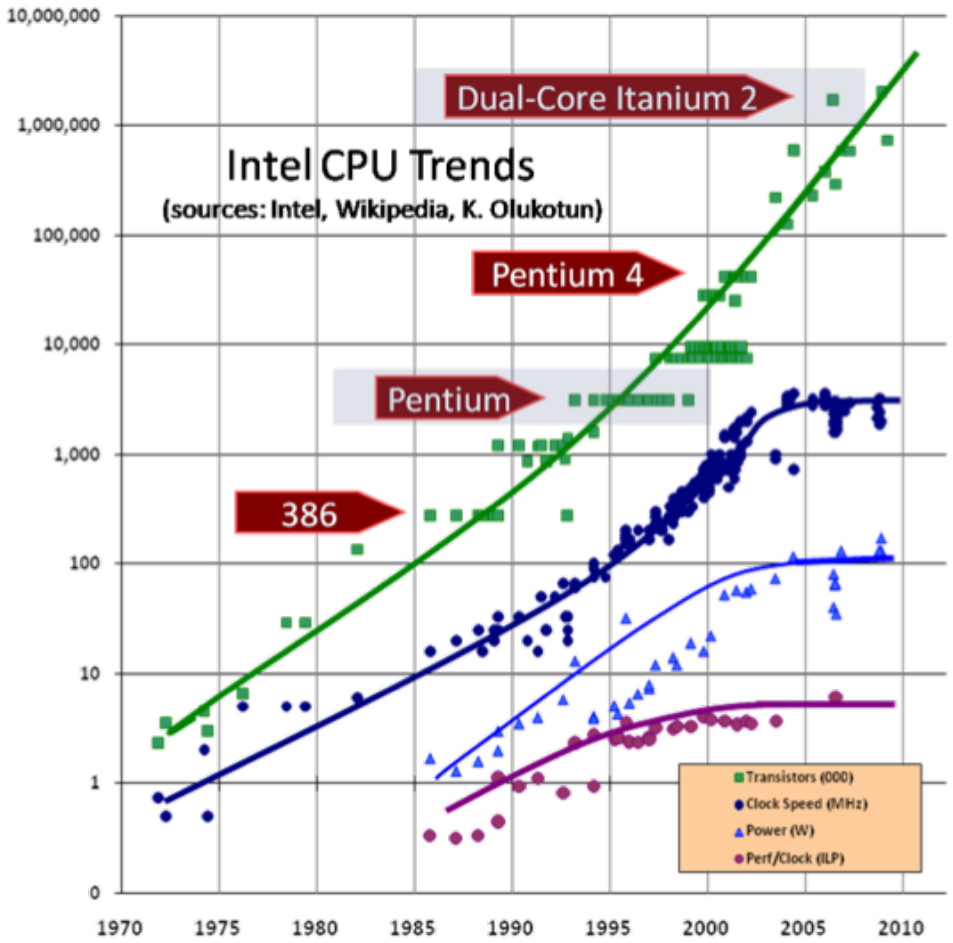
- Syllabus
- Why C?
- **Modern CPU Performance**
- Parallel Computing Background

Single-Threaded Performance



Source: Stephen A. Edwards. *History of Processor Performance.*

Another Look



Clock Speed

Power

Source: Herb Sutter. *The Free Lunch Is Over.* Dr. Dobb's Journal.

The “Free Lunch” is over?

- In the past, we could just wait and CPUs would get faster and faster (in terms of clock speed)
  - Your CPU’s clock speed: 3.2 GHz (for example)
- Performance improvements have slowed over time
  - Too much power consumption, too much heat
- So as we all know, “There ain't no such thing as a free lunch”
  - TANSTAAFL
  - *Unless you work at a tech company (?)

Today's Reality

- These days, we get better performance through **horizontal scaling**
  - Rather than making one really fast processor (**vertical scaling**), we'll make a processor running at a reasonable speed with multiple **cores**
- And in some cases, we'll have multiple processors, or even multiple machines
  - Clusters

An Example: Google

- Google doesn't buy the world's craziest, most expensive supercomputers
- They buy **commodity hardware** in huge quantities
- A single Google search may query tens or even hundreds of servers
- MPI is one option for cluster computing that we'll be learning in this class

Another Example: AMD

- AMD's recent Ryzen/EPYC CPUs push the multicore concept further
- Let's say we want to build a 128-core CPU
  - The chance of a manufacturing defect is fairly high
- AMD's latest approach is taking four 32-core CPUs and fusing them together
  - **MCM**: multi chip module

Performance Challenges

- In some cases, this slowdown is okay:
  - Most applications run “fast enough”
  - I don’t have to buy a new laptop every year!
- But some use cases still need more power:
  - Climate models, large-scale, more realistic simulations
  - Machine Learning (deep learning)
  - Bioinformatics
  - Games! VR requires massive computational capabilities

Solution: IPC

- We can increase the IPC (instructions per clock cycle) of the CPU
  - This is not easy!
  - Modern CPUs even use machine learning to help optimize instruction throughput
- We can keep decreasing the size of transistors

Solution: Die Shrink

- We can shrink the size of the transistors in our CPUs
  - With increases in density, we can have more CPU cores
- This lowers costs and allows us to pack even more transistors in a small area
  - The latest from Intel is around 10 nm
- 5 nm is currently considered the limit for die shrinks
  - Physical limits; quantum tunneling

Solution: Parallel Programming

- The current approach is parallel programming, and in some cases distributed programming
- So really, my point is:
  If you care about performance, you're going to have to parallelize your workloads
- So, what exactly is parallel programming?

Today's Agenda

- Syllabus
- Why C?
- Modern CPU Performance
- **Parallel Computing Background**

Parallel Programming [1/2]

- Adding more cores is only helpful if we can make use of them!
  - The bad news is *parallelizing* applications can be difficult
- The basic idea behind parallel computing is:

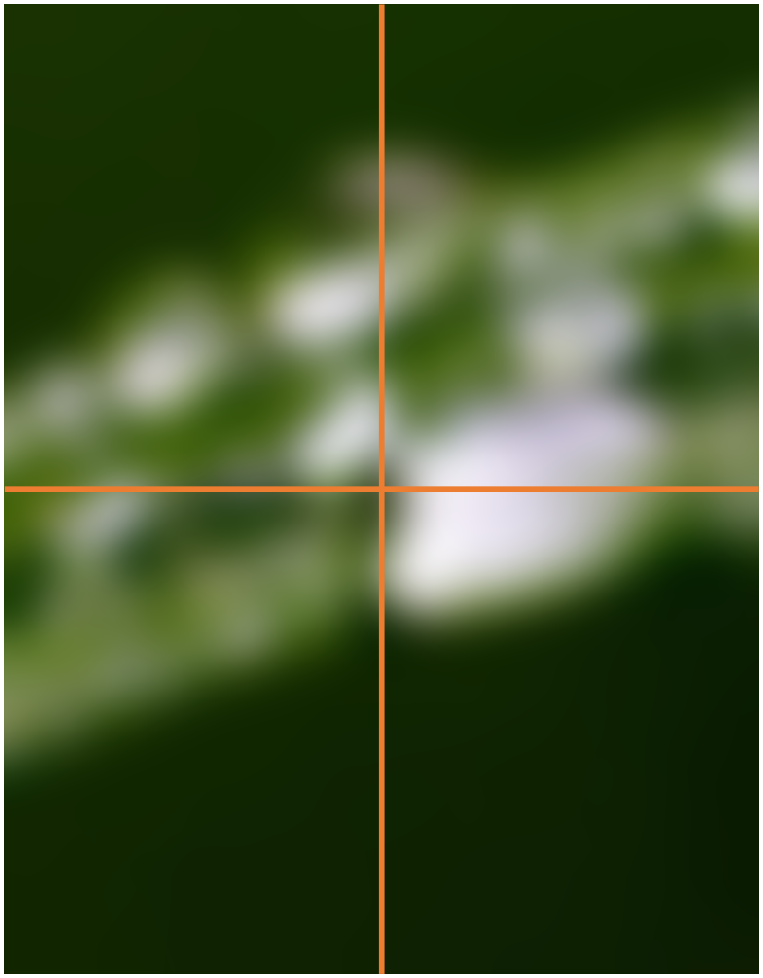
  Divide and Conquer
- If we can split a problem up into many smaller problems, then each core (or each machine) can take care of part of the work

Parallel Programming [2/2]

- Some problems are **embarrassingly parallel**
  - If I told everyone to raise their hands, no coordination is necessary
  - Another example: running simulations
- Unfortunately, not all problems are as easy to parallelize
  - Communication is required between CPUs and machines

Blurring an Image

- Let's blur an image in parallel: first, we split it up across our CPUs
- Next, each CPU loops through each pixel and inspects its neighbors to blend them together
- The only problem? We need to stitch the edges together



Other Issues

- Some algorithms are extremely difficult (or even impossible) to parallelize
  - **Global Shared State**

- In some situations, you may select a less efficient algorithm simply because it is parallelizable

Types of Parallel Architectures

- Distributed memory systems (clusters)
- Shared memory systems
- Heterogeneous systems
  - Specialization for a particular task
- Graphics Processing Units (GPUs)
  - Single instruction, multiple data (SIMD) systems

Wrapping Up

- Welcome to class (again!)
  - Ask questions, come to instructor/TA office hours, we're here to make sure you succeed
- Next class:
  - Parallel architectures
  - Getting started with C