

Two vertical lines, one blue and one orange, are positioned on the left side of the slide.

ECE408/CS483/CSE408 Fall 2022

Applied Parallel Programming

Lecture 1: Introduction

Before We Get Started

- Welcome to the course!
- The course is taught in-person
 - Lectures are in-class; they are also recorded and posted on Mediaspace
 - Labs (aka MPs) & Projects are on your own out of class activities
 - Midterm exams are (*most likely*) in-person
- Lecture slides will be posted on-line prior to the lecture on the course's Canvas page
 - <https://canvas.illinois.edu/courses/38813>

People

Instructors: **Prof. Sanjay Patel** <sjp@illinois.edu>

Prof. Volodymyr Kindratenko <kindrtnk@illinois.edu>

TAs: **Shuangliang Chen** <sc60@illinois.edu>

Howie Liu <hanwenl4@illinois.edu>

Yifei Song <yifeis7@illinois.edu>

Heyi Tao <heyitao2@illinois.edu>

Huili Tao <huilit2@illinois.edu>

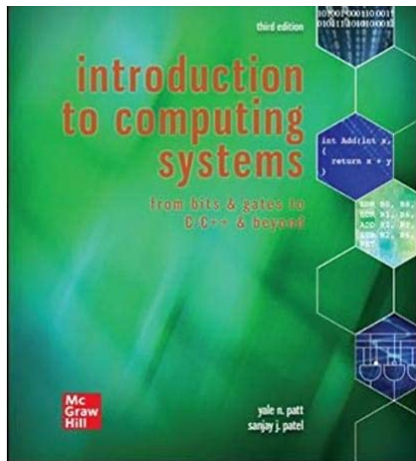
Zixuan Tian <zixuant5@illinois.edu>

Ze Yang <zeyang2@illinois.edu>


RAI administrator: **Andrew Schuh** (aschuh@illinois.edu)

About Prof. Sanjay Patel

- Ph.D: University of Michigan, Ann Arbor, 1999
- Faculty: University of Illinois since 1999
- Research: Computing Systems, Architecture, Computer Vision, ML
- Author: Introduction to Computing Systems (now in 3rd edition!)
- Entrepreneur: Founded/Built Several Successful Companies (AGEIA, Personify, Digital Gemini)
- Director of Alchemy Foundry



Creating Technology that has Impact



2004-2008
Acquired by Nvidia

Notable Alum:
Chris Lamb




2011-2018
Acquired by Foxconn

Founding Team:
Quang Nguyen
Dennis Lin
Simon Venshtain



2019-2022
Acquired by Uhnder

Founding Team:
Henry Haase
Yuan Zhang
Yumai Sun
Ke Sun



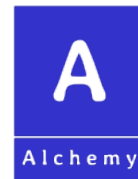
2020-

Founding Team:
John Paul
Stephen Hurwitt
Siqi Zhang
Ryan Oberlander



2020-

Founding Team:
Yucheng Liang
Chak Ho Chan
Shuchen Zhang



About Prof. V. Kindratenko

- Ph.D. from University of Antwerp, Belgium, 1997
- At NCSA since 1997
 - Past: Director of Innovative Systems Lab
 - Current: Director of the Center for AI Innovation (CAII)
- Research: Computing Systems, HPC, Computational Accelerators (FPGAs, GPUs), ML systems & applications



AC – first GPU
HPC cluster built in
2008 (used to teach
this course too)

- 32 S1070 GPUs



HAL – first AI-
oriented cluster
built in 2018

- 64 V100 GPUs

Course Goals

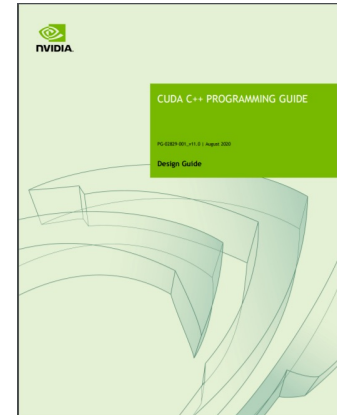
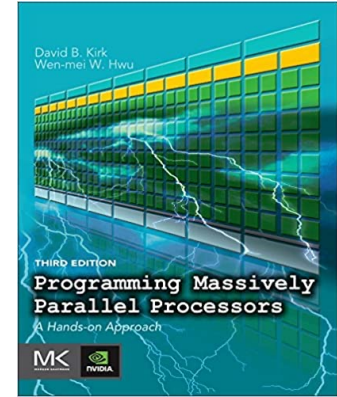
- Learn to program massively parallel processors
 - High performance
 - Functionality and maintainability
 - Scalability across future generations
- Technical subjects
 - Parallel programming basics
 - Principles and patterns of parallel algorithms
 - Programming API, tools and techniques
 - Processor architecture features and constraints
 - Computational concepts in AI, High-Performance Computing

Grading

- Exams: 40%
 - Midterm 1: 20% -- ~ first 10+ lectures
 - Midterm 2: 20% -- ~ the remaining lectures
- Labs (Machine Problems): 35%
 - Passing Datasets 90%
 - Correct answers to questions
 - Lowest graded lab will be dropped
- Project: 25%
 - Demo/Functionality/Coding Style: ~50%
 - Performance with full functionality: ~50%
 - Detailed Rubric will be posted

Text/Notes

1. D. Kirk and W. Hwu, “Programming Massively Parallel Processors – A Hands-on Approach,” Morgan Kaufman Publisher, 3rd edition, 2016, ISBN 978-0123814722
2. NVIDIA, *NVidia CUDA C Programming Guide*, version 7.5 or later (reference book)
<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>



Tentative Schedule

- Week 1:

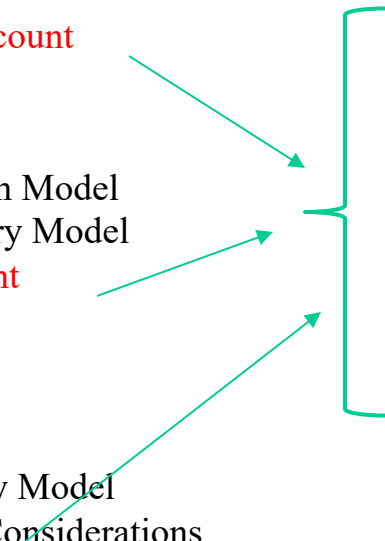
- Tuesday: Lecture 1: Introduction
- Thursday: Lecture 2: CUDA Intro
- Release: Lab 0, Installation, Test account

- Week 2:

- Tuesday: Lecture 3: Data Parallelism Model
- Thursday: Lecture 4: CUDA Memory Model
- Due: Lab 0, Installation, Test account
- Release: Lab 1, Vector Addition

- Week 3:

- Tuesday: Lecture 5: CUDA Memory Model
- Thursday: Lecture 6: Performance Considerations
- Due: Lab 1, Vector Addition
- Release: Lab 2, Simple Matrix Multiply



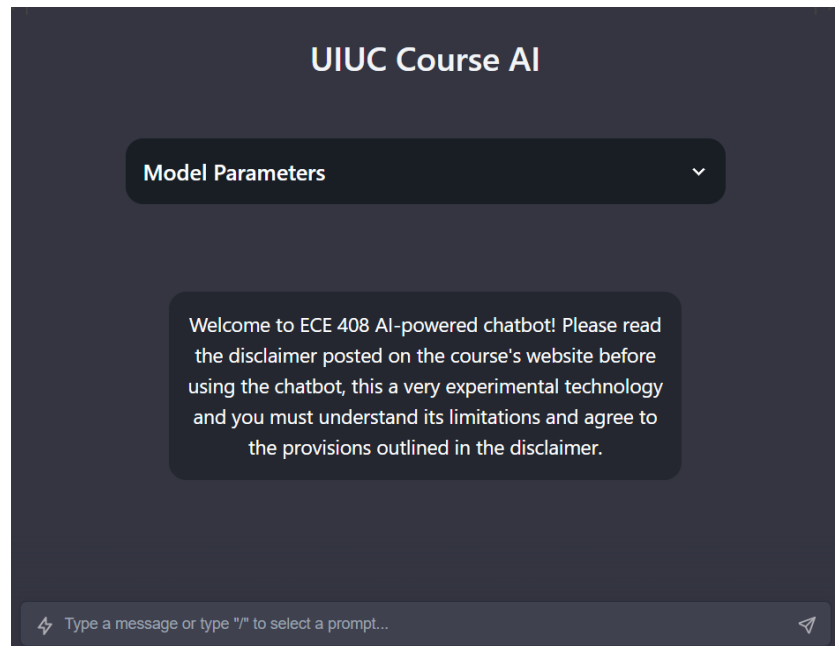
**all Labs (MPs)
and Project
Milestones (PMs) are
due on Fridays at
8:00pm US Central Time**

RAI

- Framework for submitting labs (MPs) and projects and grading them
 - You will receive an email with your RAI account instructions
 - Lab 0 will include all the details about deploying and using RAI client
- All Labs (MPs) and Project Milestones (PMs) are due on
 - Fridays at 8:00pm US Central Time
- Lab workflow:
 - #1: Get base code from GitHub → write code & compile & run (using RAI) → submit final code for grading (via RAI)
 - #2: Answer questions on Canvas AFTER #1 is completed
- **Lab 0 will have all the instructions to get started with RAI**

Course ChatBot

- <https://uiuc.chat/ECE408FA23>
- Under development by CAII team at NCSA
- GPT-4 based (OpenAI API)
- Context Stuffing and Mega-Prompting (with course materials)
- Please read the disclaimer posted on the course's website before using the chatbot



Academic Honesty

- You are allowed and encouraged to discuss assignments with other students in the class. Getting verbal advice/help from people who've already taken the course is also fine.
- Any reference to assignments from previous terms or web postings is unacceptable.
- Any copying of non-trivial code is unacceptable
 - Non-trivial = more than a line or so
 - Copying includes reading someone else's code, including code produced by course chatbot, and then going off to write your own.
 - Those who have allowed copying will also be penalized

Academic Honesty (cont'd)

- Giving/receiving help on an exam is unacceptable.
- Deliberately sidestepping the lab requirements is unacceptable.
- Penalties for academic dishonesty:
 - Zero on the assignment/exam for the first occasion
 - Automatic failure of the course for repeat offenses

A decorative element on the left side of the slide consisting of two vertical lines, one blue and one orange.

Old Paradigm

**We were able to understand, design, and
manufacture what we can measure**

- Physical instruments allowed us to see farther, capture more, communicate better, understand natural processes, control artificial processes...

A decorative element on the left side of the slide consisting of two vertical lines, one blue and one orange.

New Paradigm

We are able to understand, design, and create what we can compute

- Computational models are allowing us to see even farther, going back and forth in time, learn better, test hypothesis that cannot be verified any other way, create safe artificial processes...

Examples of Paradigm Shift

- Conventional semiconductor Lithography → Computational correction
- Electron Microscopy → Computational Microscopy
- Film Photography → Deep-learning driven, computational imaging
- X-Rays → CT & MRI Scans with reconstruction
- Land-line phones → Zoom video conferencing
- Cars → Self-driving electric taxis
- Print and Broadcast Ads → AI-assisted Digital Ad Placement
- ??? → Multimodal Generative AI that can Plan

Why is this happening now?

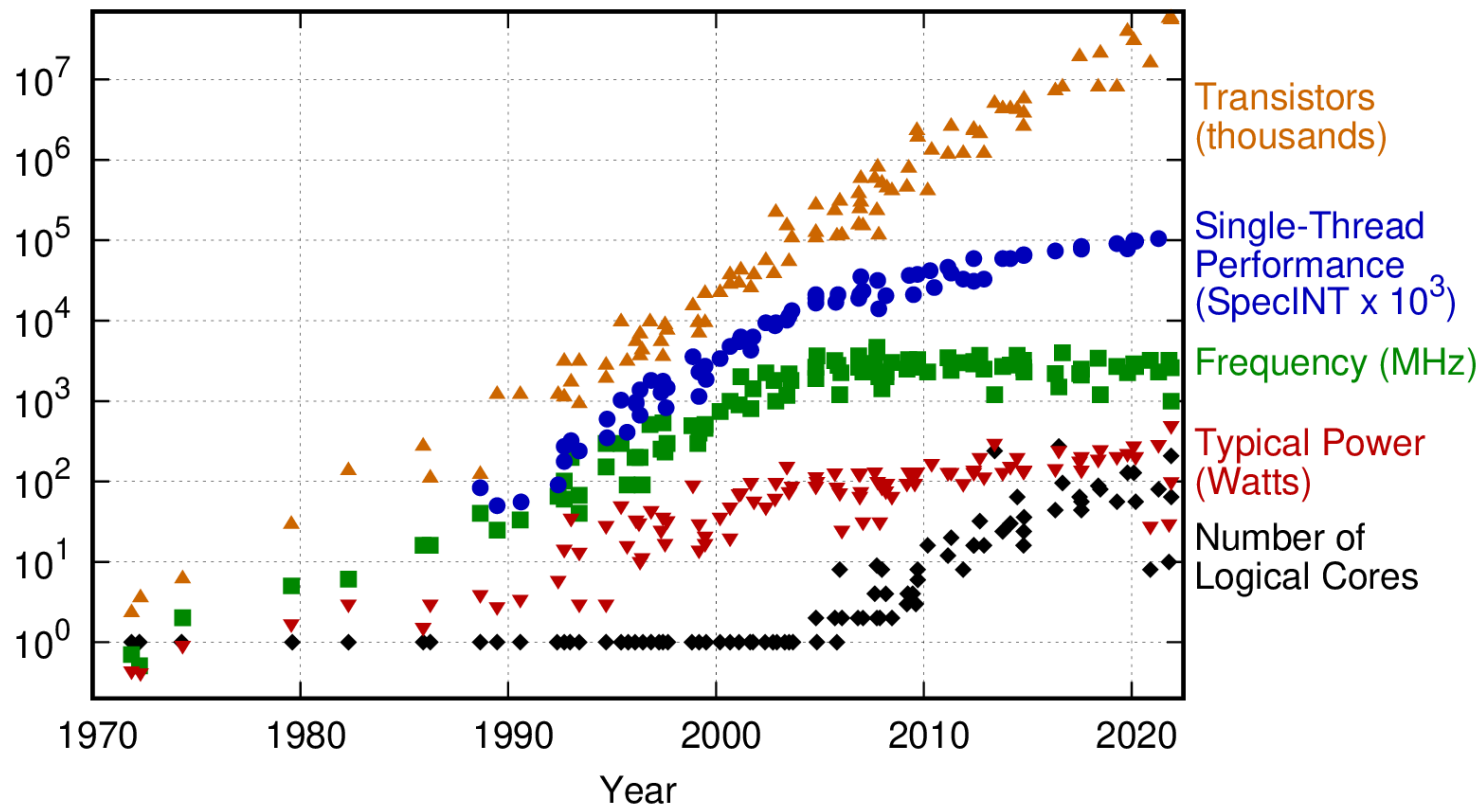
“Old” Paradigm



New Paradigm

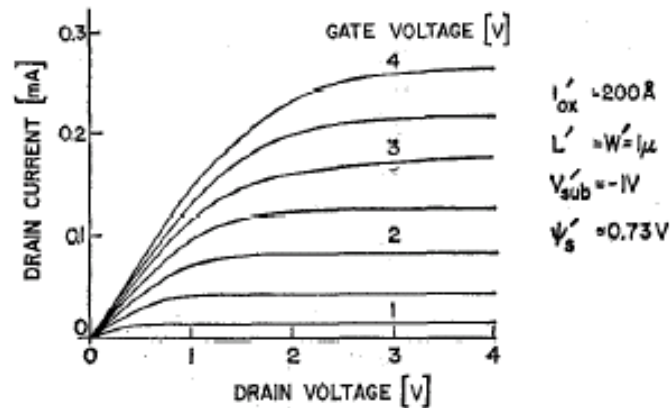
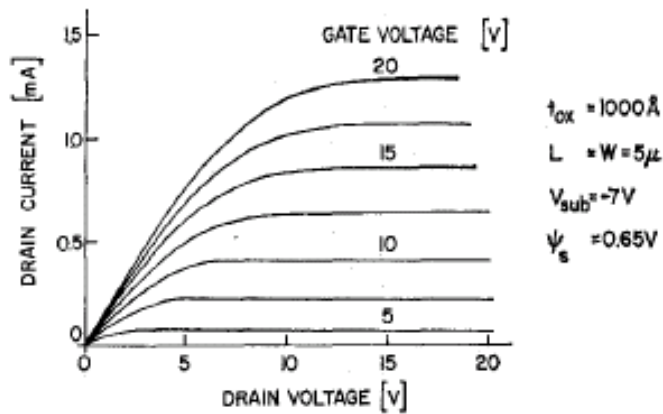


50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

Dennard Scaling of MOS Devices



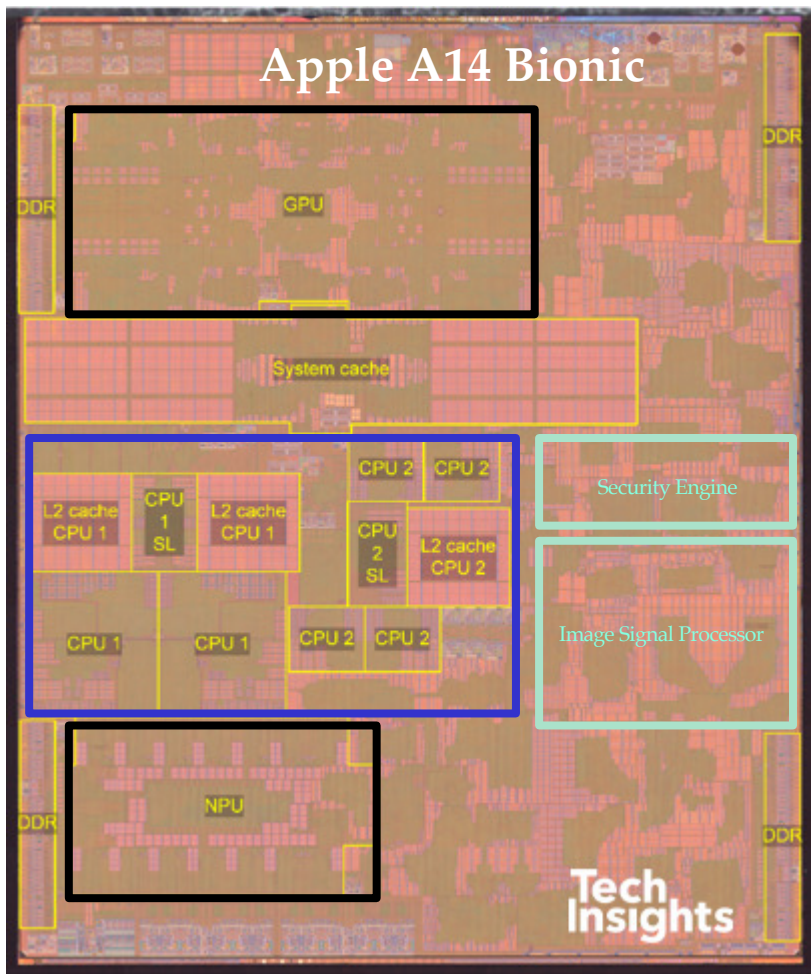
JSSC Oct 1974, page 256

- In this ideal scaling, as $\lambda \rightarrow \alpha^* \lambda$
 - $V_{DD} \rightarrow \alpha^* V_{DD}$, $C \rightarrow \alpha^* C$, $I \rightarrow \alpha^* I$
 - Delay = CV_{DD}/I scales by α , so $f \rightarrow 1/\alpha$
 - Power for each transistor is $CV^2 f$ and scales by α^2
- Total power constant for same chip area

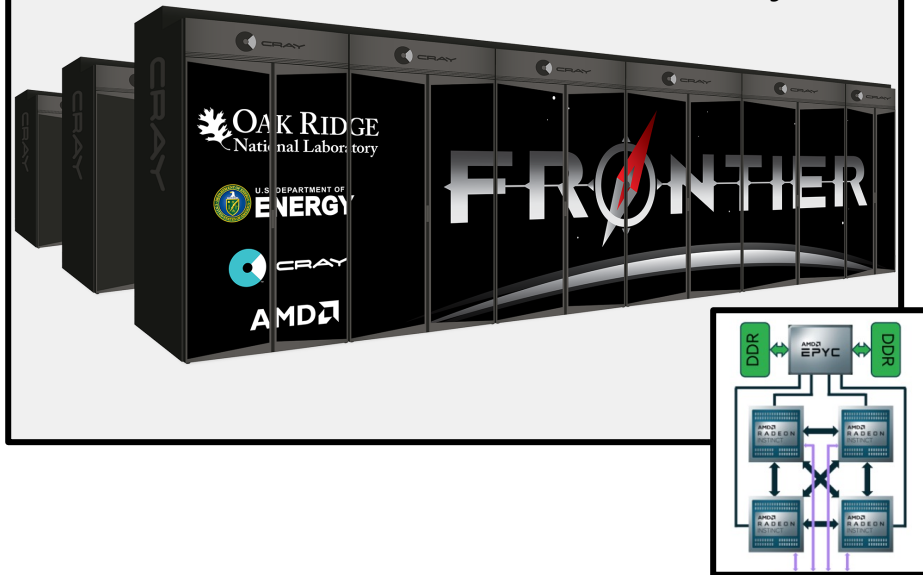
Post-Dennard Computer Architecture

- Multiple cores with more moderate clock frequencies, heavy use of threading, vector execution
- Systems-on-a-chip with various cores: latency-optimized cores, throughput-optimized cores, and special-purpose cores
- Optimized memory architectures – 3D integration, higher bandwidth interfaces, deeper hierarchies

Apple A14 Bionic



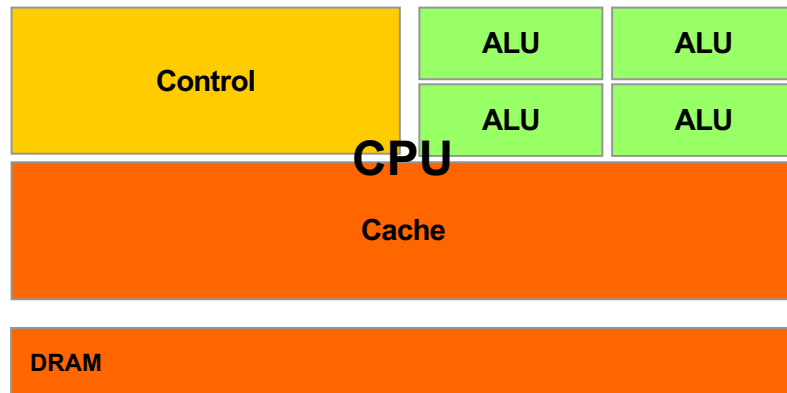
ORNL Frontier Exascale System



From the smallest to the largest, computing today involves accelerators architectures.

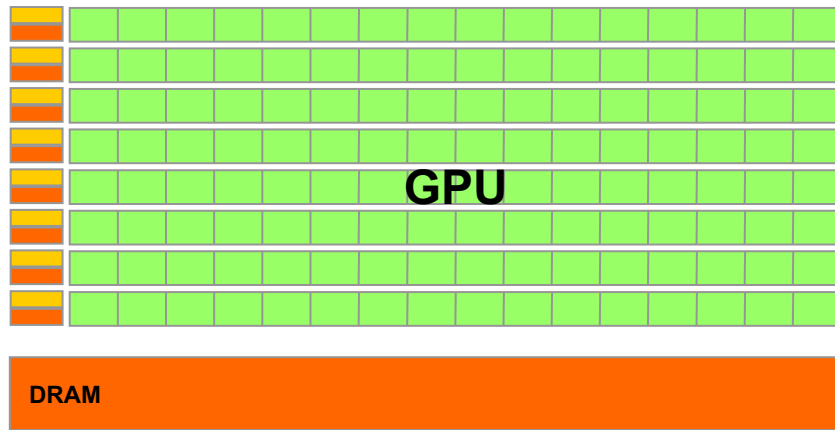
CPU: Latency Oriented Design

- High clock frequency
- Large caches
 - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
 - Branch prediction for reduced branch latency
 - Data forwarding for reduced data latency
- Powerful ALU
 - Reduced operation latency



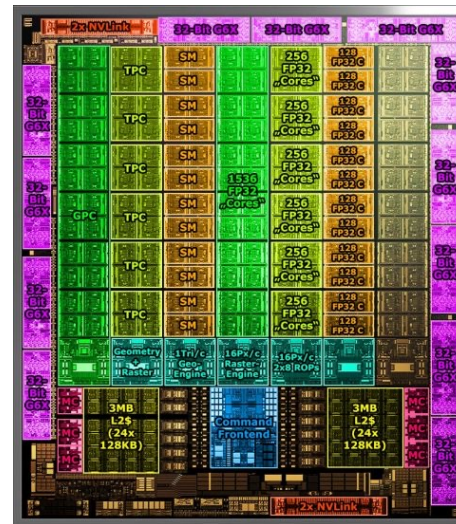
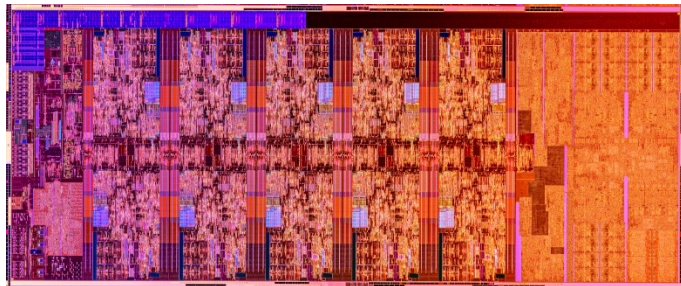
GPUs: Throughput Oriented Design

- Moderate clock frequency
 - To boost memory throughput
- Small caches
 - No branch prediction
 - No data forwarding
- Energy efficient ALUs
 - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies



CPU vs GPU

- 10th Gen Intel Core
 - 10 cores in silicon
 - 14 nm process
- NVIDIA A100
 - 3456 CUDA cores
 - 7 nm process



Canonical View of Accelerator Model



Two vertical lines, one blue and one orange, are positioned on the left side of the slide.

Today's approach for compute-intensive applications

- CPUs for sequential, latency sensitive parts
 - CPUs can be $>10\times$ faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
 - GPUs can be $>10\times$ faster than CPUs for data parallel code

Heterogeneous Parallel Computing Applications

**Financial
Analysis**

**Scientific
Simulation**

**Engineering
Simulation**

**Data
Intensive
Analytics**

**Medical
Imaging**

**Digital Audio
Processing**

**Digital Video
Processing**

**Computer
Vision**

**Machine
Learning**

**Electronic
Design
Automation**

**Biomedical
Informatics**

**Statistical
Modeling**

**Ray Tracing
Rendering**

**Interactive
Physics**

**Numerical
Methods**

LLMs

Digital Twins

**Digital
Agents**

Robotics

**Autonomous
Vehicles**

Parallel Programming Work Flow

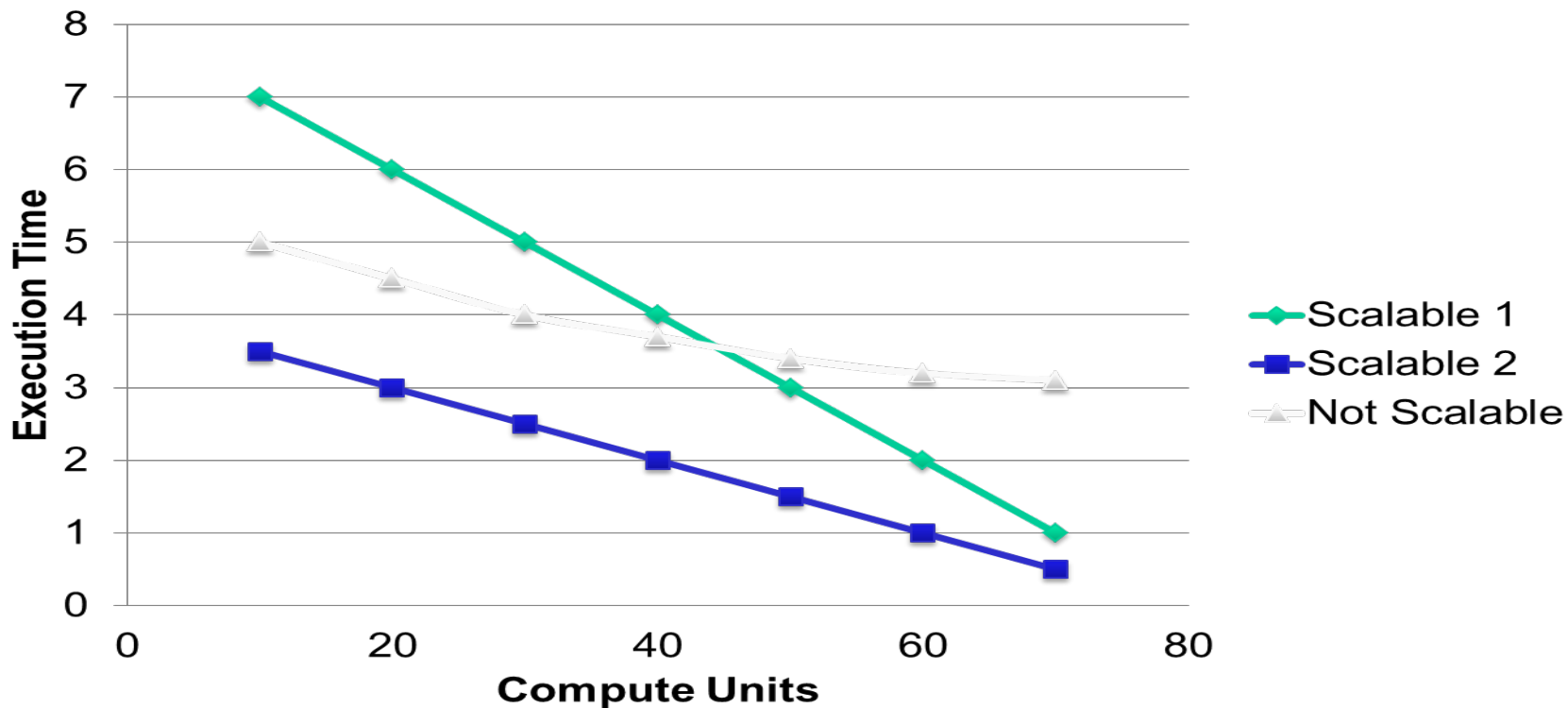
- Identify compute intensive parts of an application
- Adopt/create scalable algorithms
- Optimize data arrangements to maximize locality
- Performance Tuning
- Pay attention to code **portability**, **scalability**, and **maintainability**

Simple Parallelism

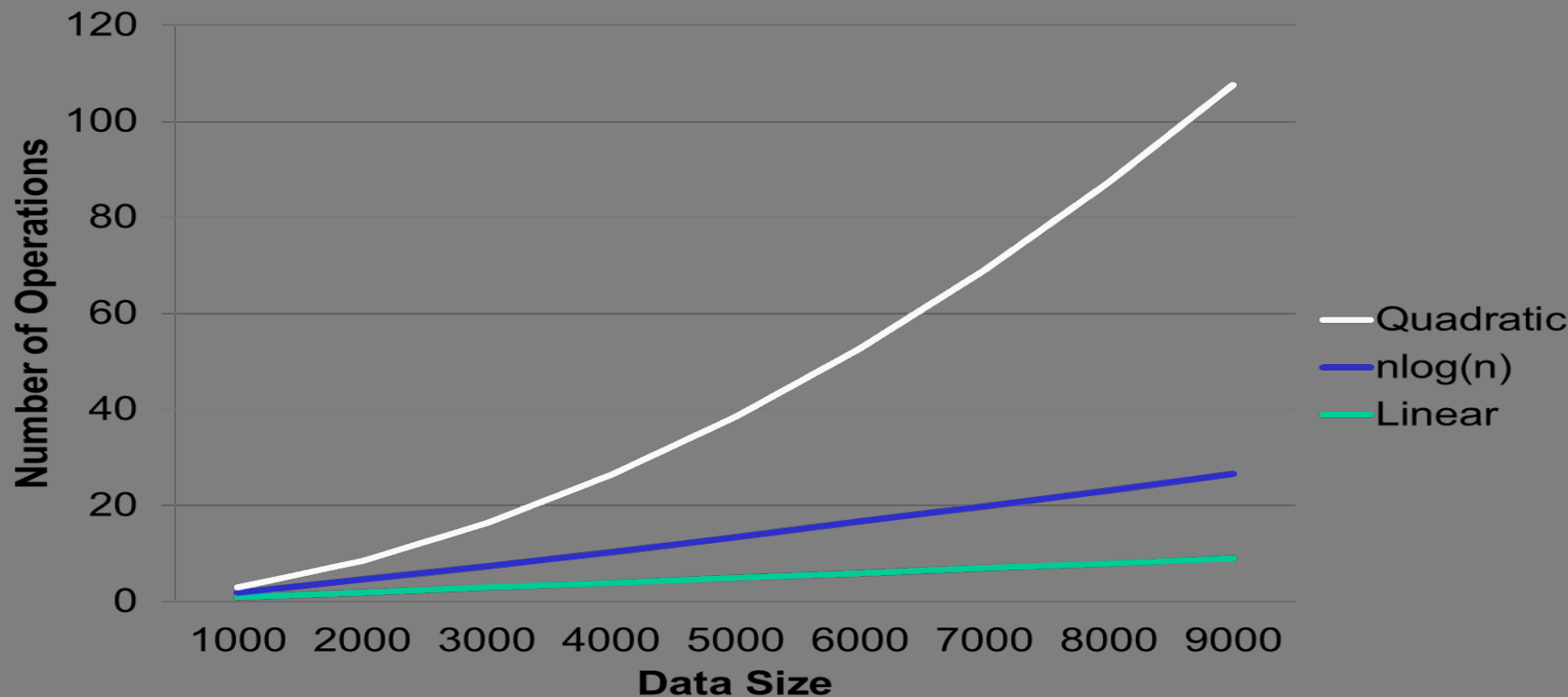
```
for (i = 0, i < n, i++) {  
    C[i] = A[i] + B[i];  
}
```

- Loop iterations are independent of each other (caveat: what about i ?)
- We want to express this loop in a form where the parallelism can be converted into independent operations.
- In this course, we will use CUDA to accomplish this

Parallelism Scalability



Algorithm Complexity and Data Scalability



Global Memory Bandwidth

Ideal

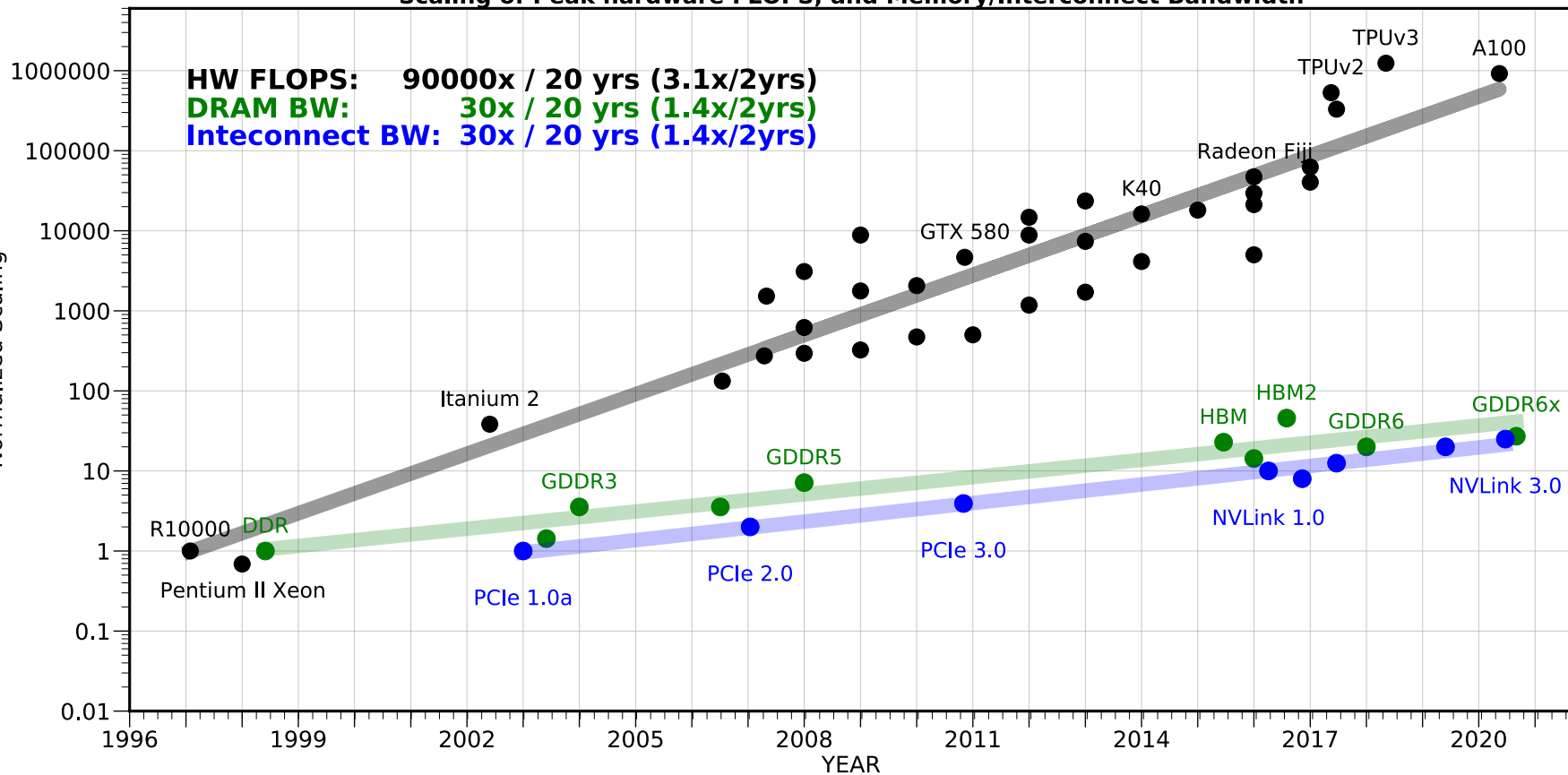


Reality



Scaling of Peak hardware FLOPS, and Memory/Interconnect Bandwidth

Normalized Scaling



Conflicting Data Accesses Cause Serialization and Delays

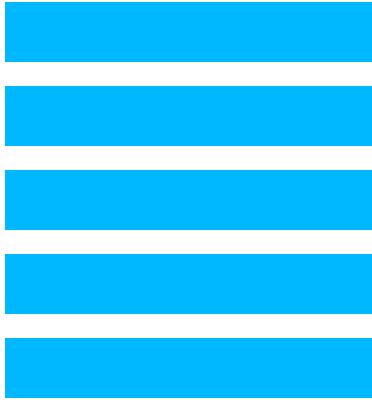
- Massively parallel execution cannot afford serialization



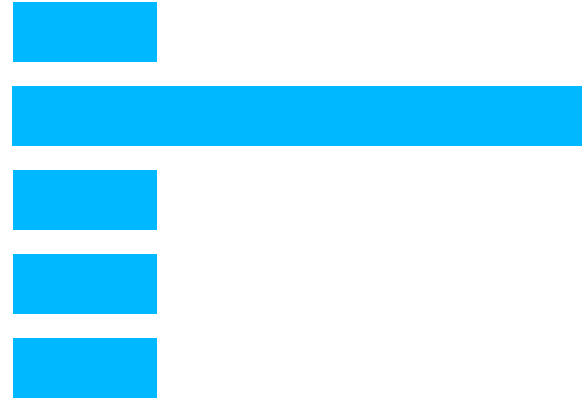
- Contentions in accessing critical data causes serialization

Load Balance

- The total amount of time to complete a parallel job is limited by the thread that takes the longest to finish



good



bad!

Two vertical lines, one blue and one orange, are positioned on the left side of the slide.

ANY MORE QUESTIONS?