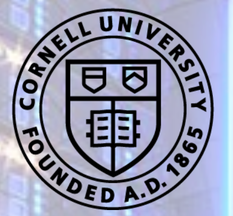


Cornell Bowers CIS
Computer Science

Applied High-Performance and Parallel Computing

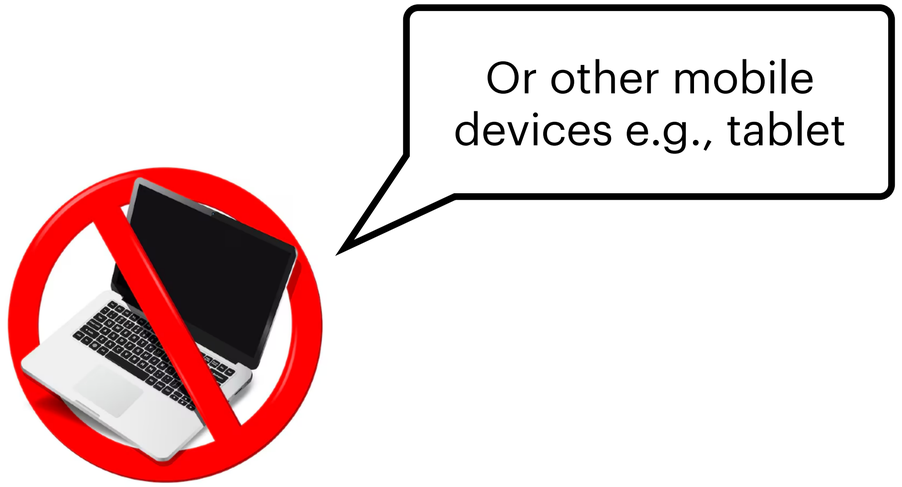
Introduction and Performance Basics

Professor Giulia Guidi

01/20/2026

Course policy on mobile devices

no smartphones



no laptop

Why? Research shows that they slow us down in a learning environment.

I recommend **paper and pen** for taking notes

Credit to: https://users.cs.utah.edu/~pandey/courses/cs6530/fall23/index.html

Ask questions!

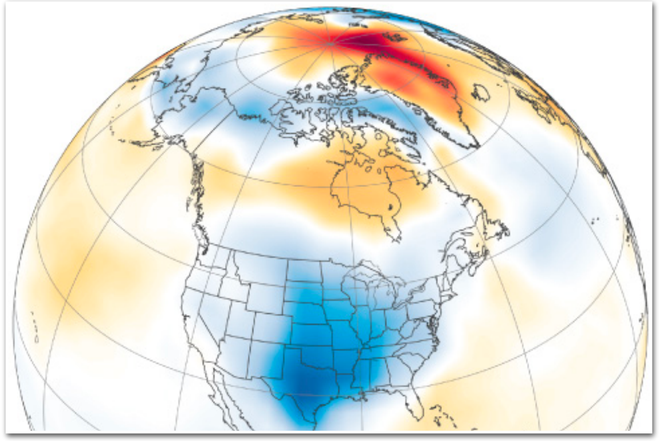
...and **answers** my questions.

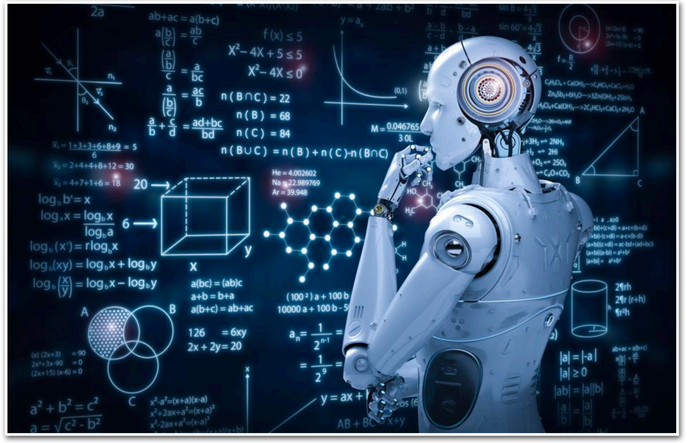
Please **speak up** if anything is unclear / with any questions / comments!

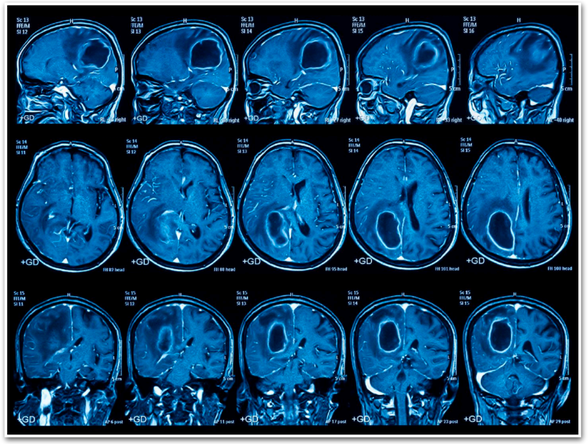
Credit to: https://users.cs.utah.edu/~pandey/courses/cs6530/fall23/index.html

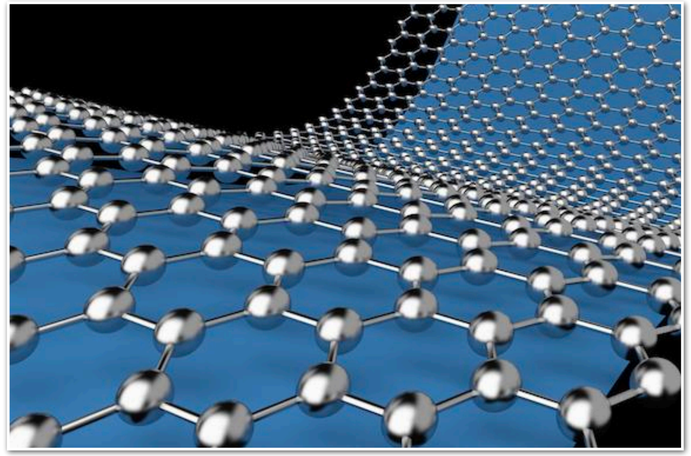
Why Parallel Computing?

Applications everywhere!









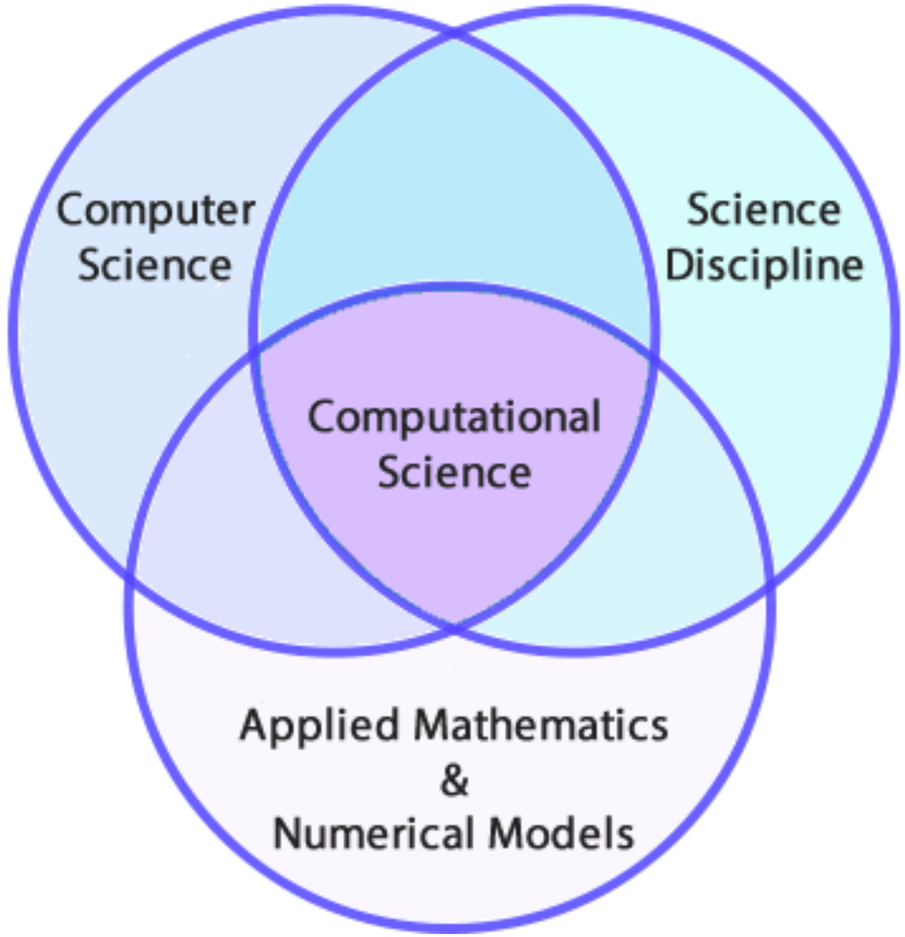
Why Parallel Computing?

Applications everywhere!

Scientific computing figured that out a long time ago!

In scientific computing:

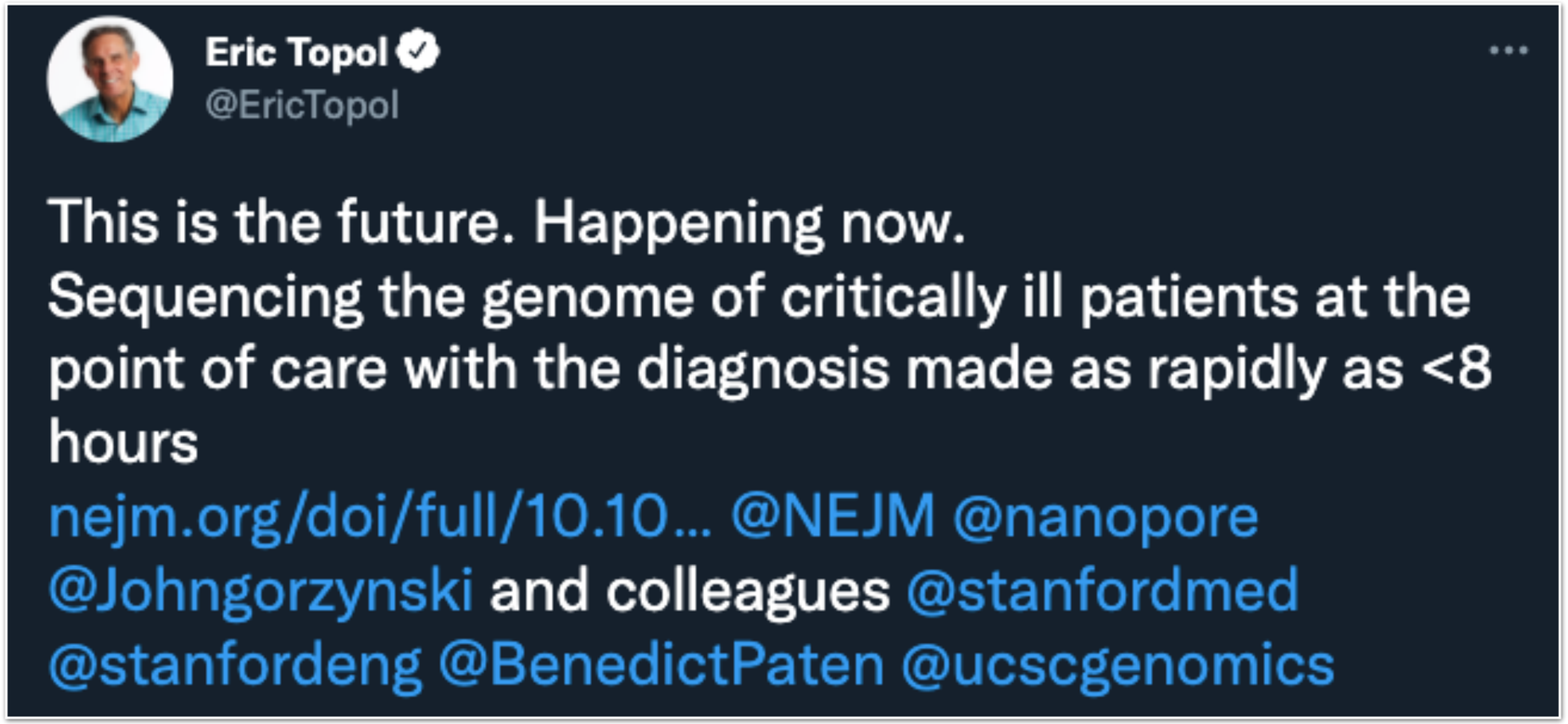
- A right enough answer, fast enough!
  - Either of those might imply a lot of work!
- Love to ask for more as machines get bigger!
- Lot of data, too!



Why Parallel Computing?

Because it matters!

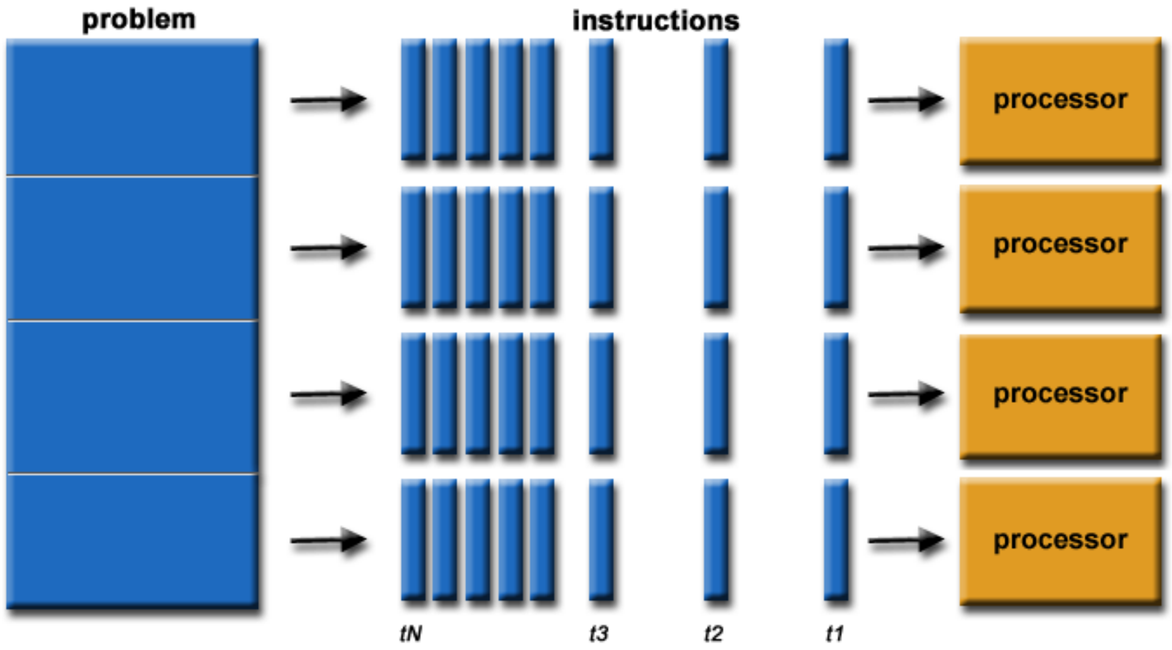
Parallel computing enabled fast **critical** medical diagnostic in early 2022:



Why Parallel Computing?

Today, hard to get a non-parallel computer!

Ah, almost **all modern computers are parallel**! The algorithms and code we develop need to run efficiently on these machines.



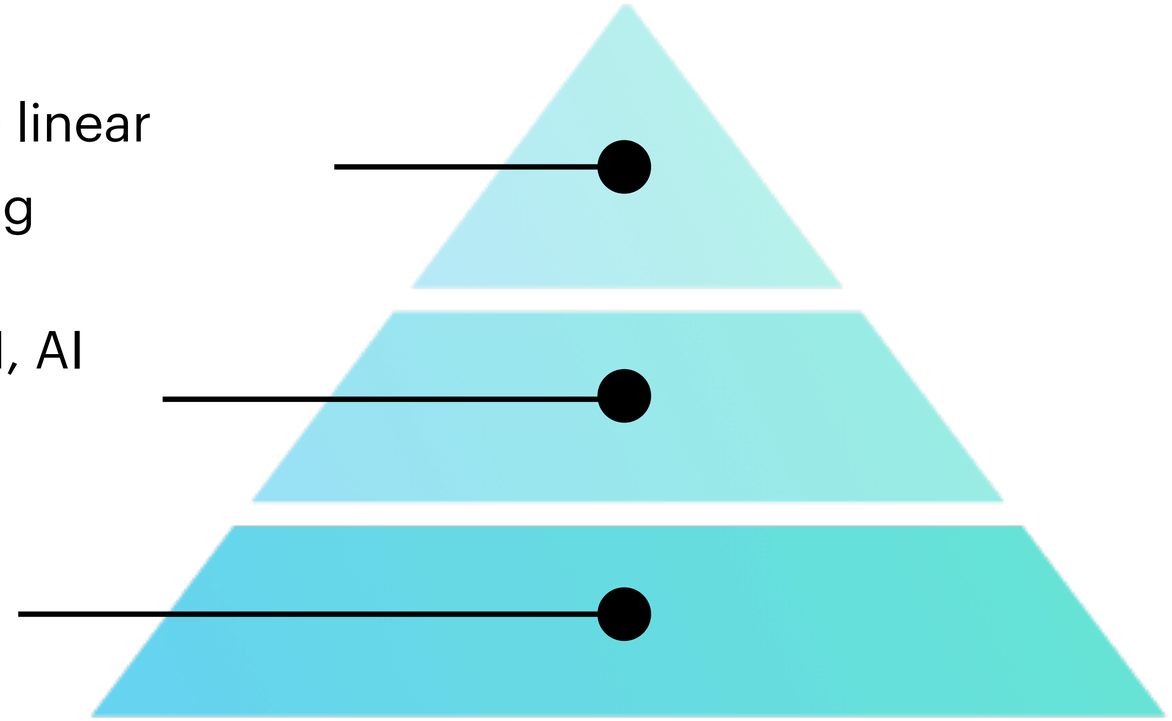
Parallel Computing Pyramid

How can we leverage these parallel machines?

Patterns and Applications: dense and sparse linear algebra, graph partitioning and load balancing

Technology: OpenMP, MPI, CUDA, NVSHMEM, AI accelerators

Basics: architecture, parallel concepts, locality and parallelism

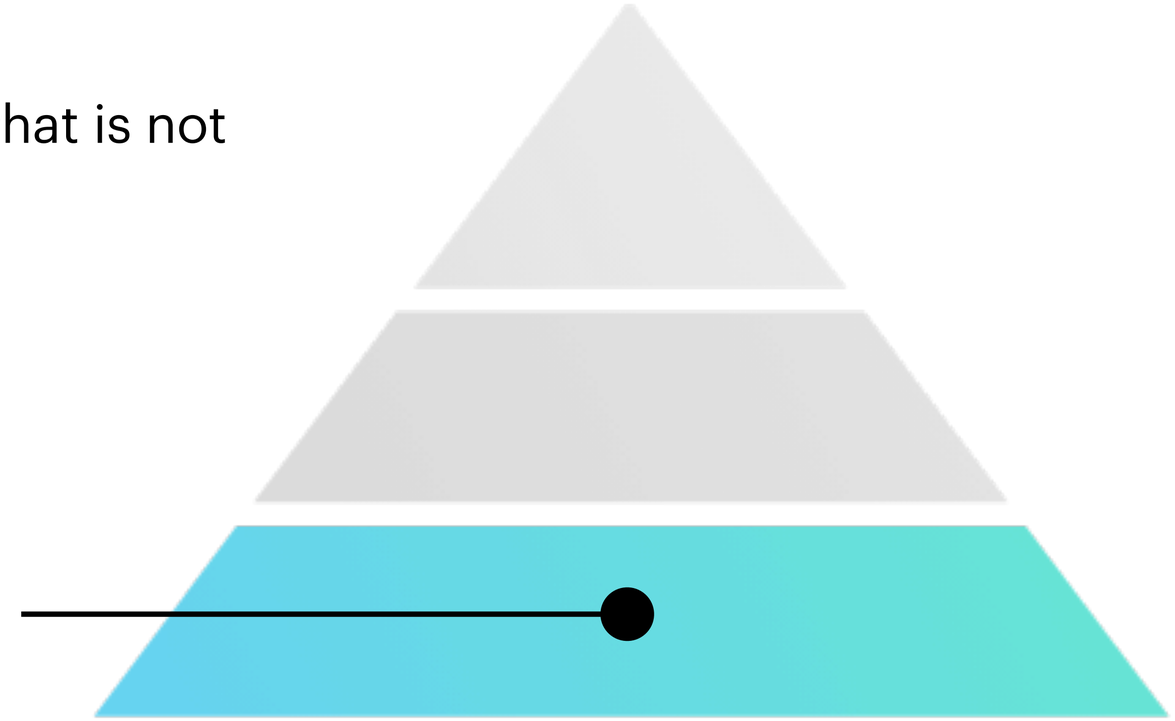


Parallel Computing Pyramid

Plan for today

- Compare what is parallel computing and what is not
- Define performance and its historical trend
- Course logistics and HW0
- Define and use basic performance metrics

Basics: architecture, parallel concepts, locality and parallelism



CS 5220 Instru

Dr. Giulia Guidi — Professor Guid

<u>Favorite Course to Teach:</u>

CS 5220, of course!

<u>Research Interest:</u>

High-Performance Computing ar

Computational Sciences, Sparse

Combinatorics

<u>Fun Fact:</u>

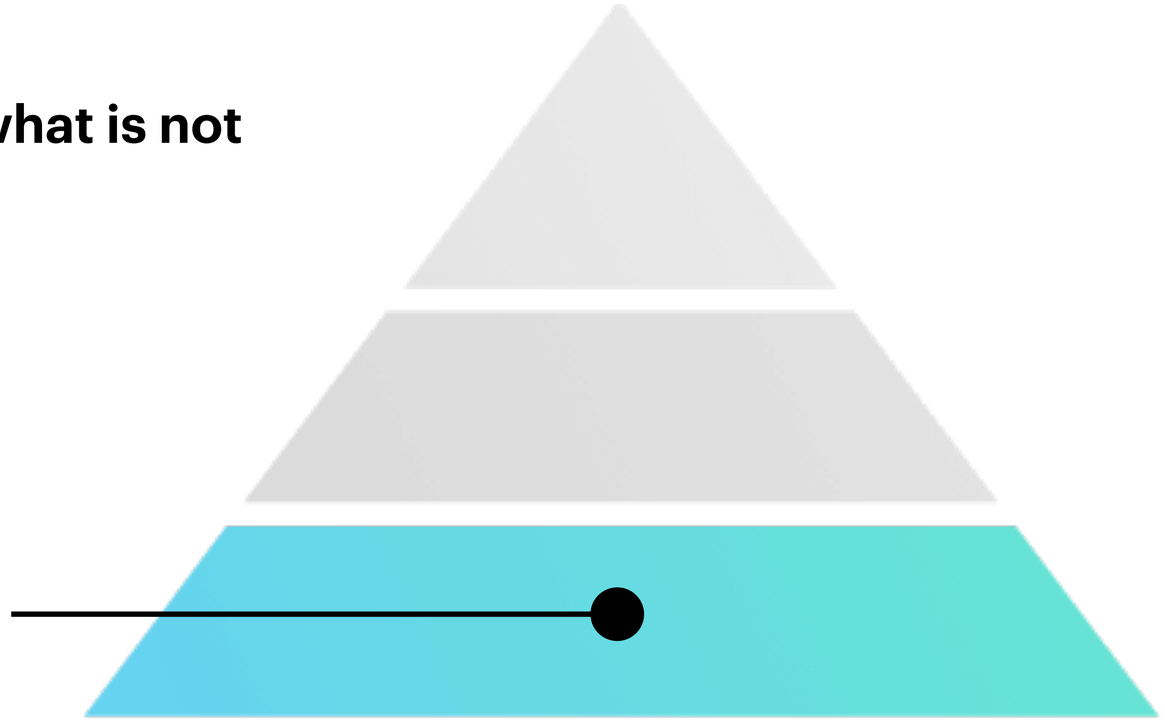
I have two dogs and enjoying pottery!

Parallel Computing Pyramid

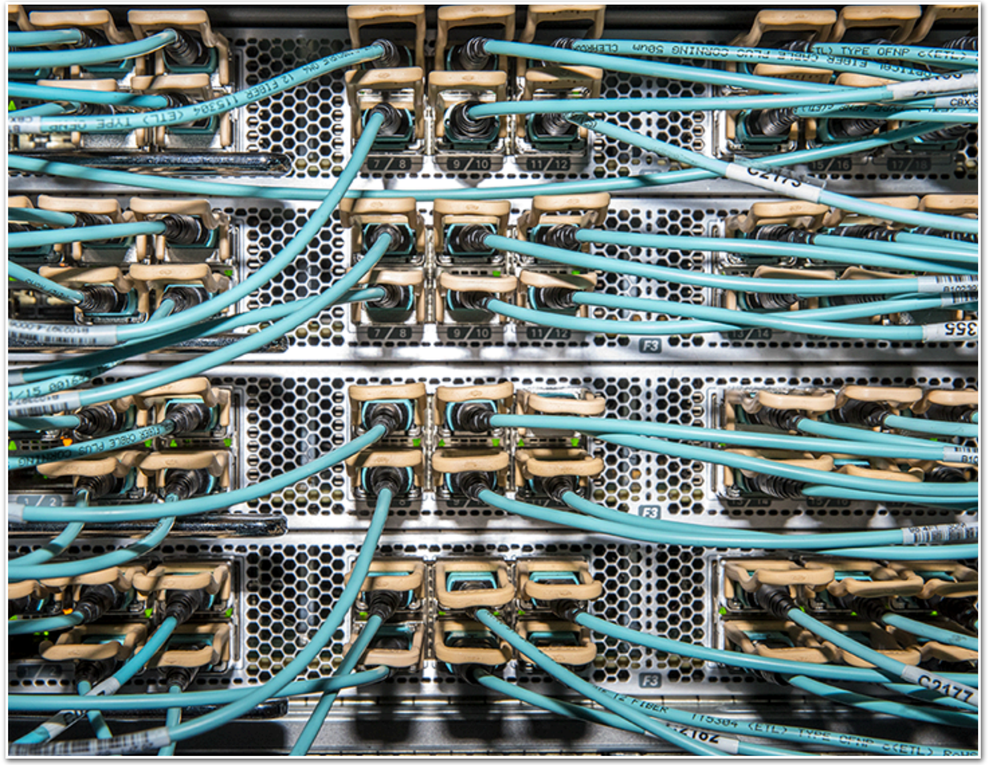
Plan for today

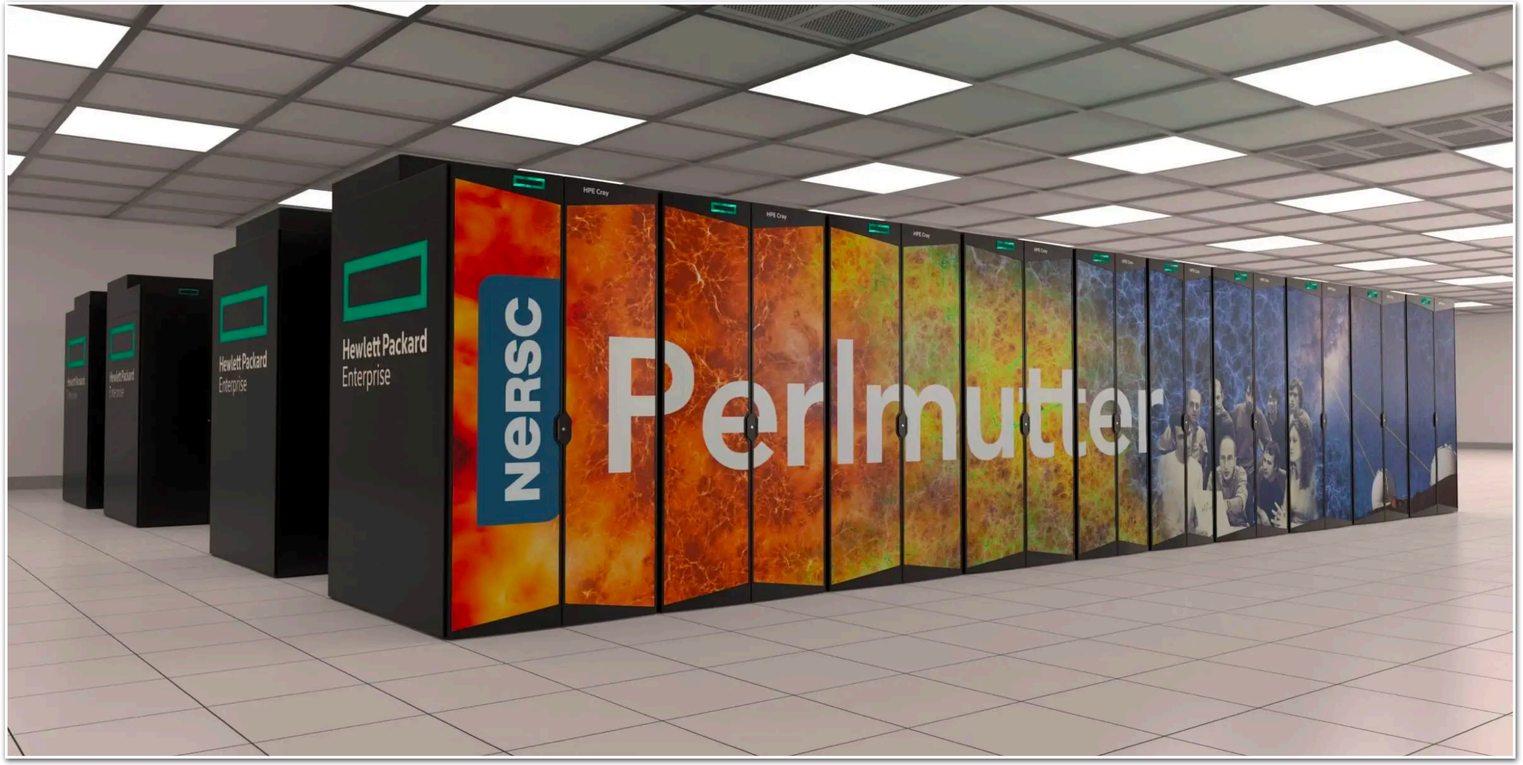
- **Compare what is parallel computing and what is not**
- Define performance and its historical trend
- Course logistics and HW0
- Define and use basic performance metrics

Basics: architecture, parallel concepts, locality and parallelism



What's a Parallel Computer?

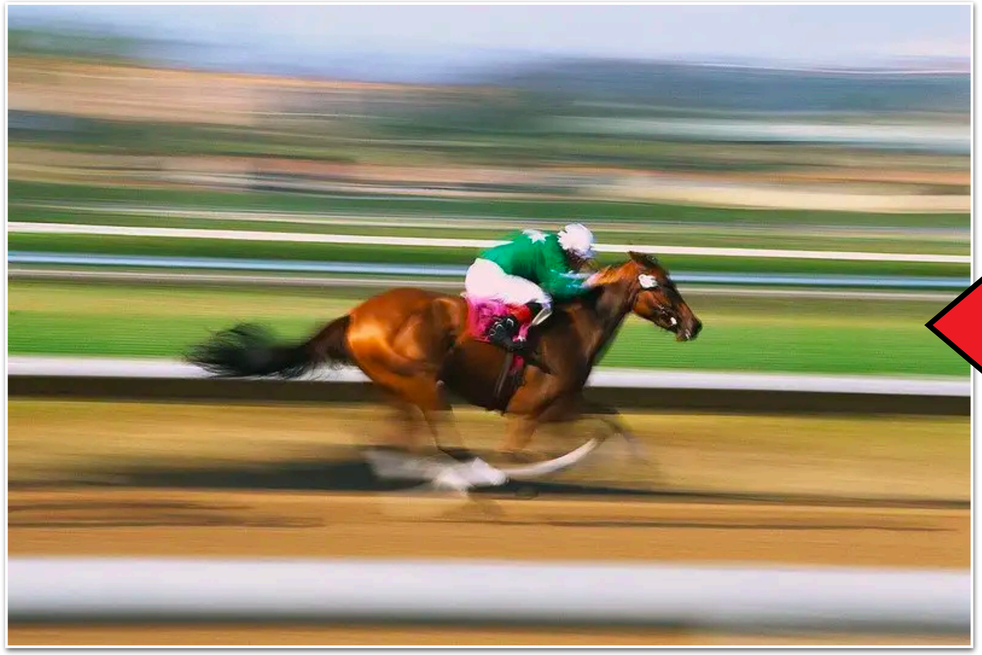




A number of slides were taken from or inspired by slides by Aydin Buluç, Kathy Yelick, and David Bindel

Parallel Computing

It is all about the **need for speed** — but sometimes is also about the **need for memory.**



Parallel Computing: Faster Solutions

The core idea is to **use multiple processors in parallel** to solve a problem faster than a single processor — or to use less memory per processor if it is limited

Compute the prime factors of 1 billion numbers:

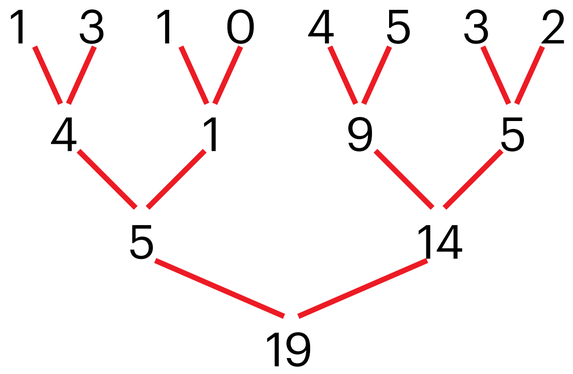
If we had 1 million processes ...

An **embarrassingly parallel problem** is one in which little or no effort is required to divide the problem into a set of parallel tasks, e.g., when there is little or no dependence between these parallel tasks

Parallel Computing: Faster Solutions

The core idea is to **use multiple processors in parallel** to solve a problem faster than a single processor — or to use less memory per processor if it is limited

For example: How can we speedup the sum computation on an array with *n* values if we have multiple *n* processors?
The **sum (reduction) in parallel**:



- The **serial algorithm** to compute the sum is O(*n*)
- The **parallel sum-reduction algorithm** is O(*n*·log(*n*)) using a single processor and O(log(*n*)) using n processors
- The parallel algorithm takes advantage of associativity in **+**

Class Activity

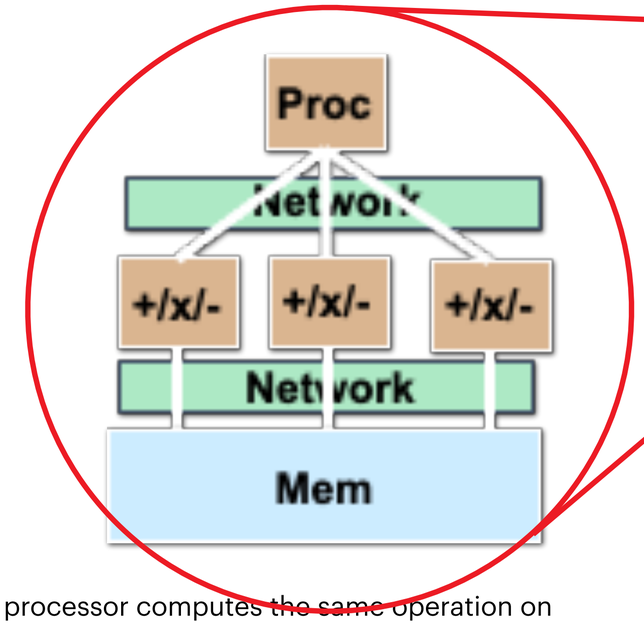
Can you think of something parallel or of parallelism in the world that is not a computer?

Please take 1 minute to think about it and then 2 minutes to discuss it with your neighbor

Examples of Parallel Computers

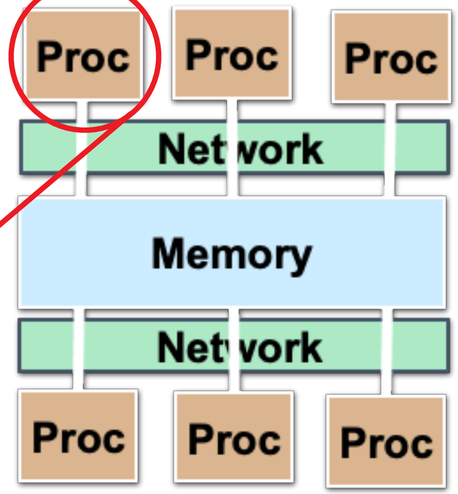
To solve a problem faster we need multiple processors — but **where do we find these processors** and how they communicate with each other when needed?

A **Single Instruction Multiple Data** machine:



- A processor computes the same operation on multiple data in parallel
- A processor often have SIMD with 2-8 way parallelism
- **GPUs use this as well**

A **shared memory** or **multicore** machine:

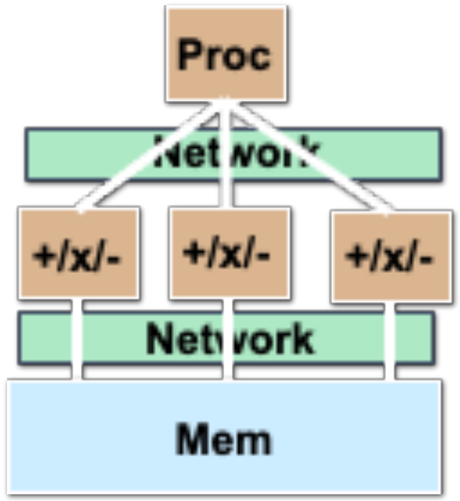


- A single memory system is shared among multiple processors
- A multicore processor has multiple cores (**proc** in the figure) on a single chip
- A multicore processor can be called a **node**

Examples of Parallel Computers

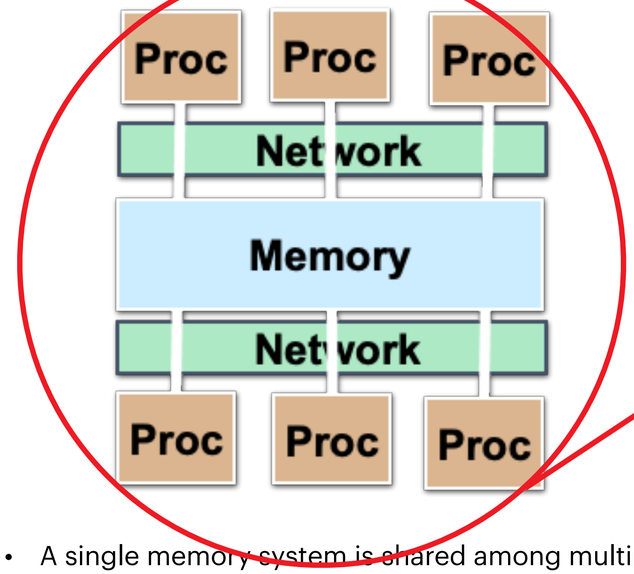
To solve a problem faster we need multiple processors — but **where do we find these processors** and how they communicate with each other when needed?

A **Single Instruction Multiple Data** machine:



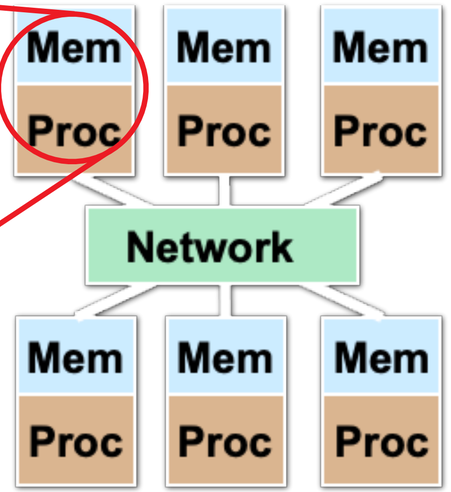
- A processor computes the same operation on multiple data in parallel
- A processor often have SIMD with 2-8 way parallelism
- **GPUs use this as well**

A **shared memory** or **multicore** machine:



- A single memory system is shared among multiple processors
- A multicore processor has multiple cores (**proc** in the figure) on a single chip
- A multicore processor can be called a **node**

A **distributed memory** or **HPC** machine:

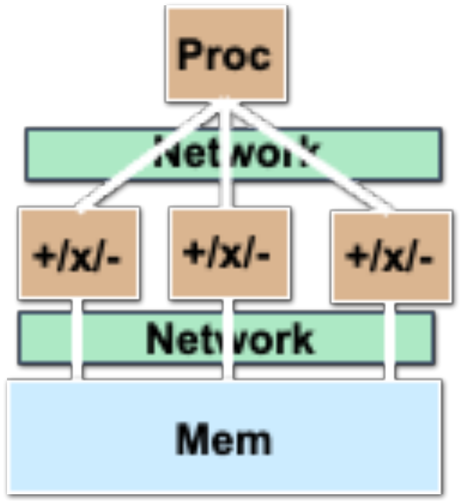


- A distributed memory multiprocessor has multiple nodes connected by a high speed **network**
- A High Performance Computing (HPC) system or **cluster** has 100s or 1000s nodes

Examples of Parallel Computers

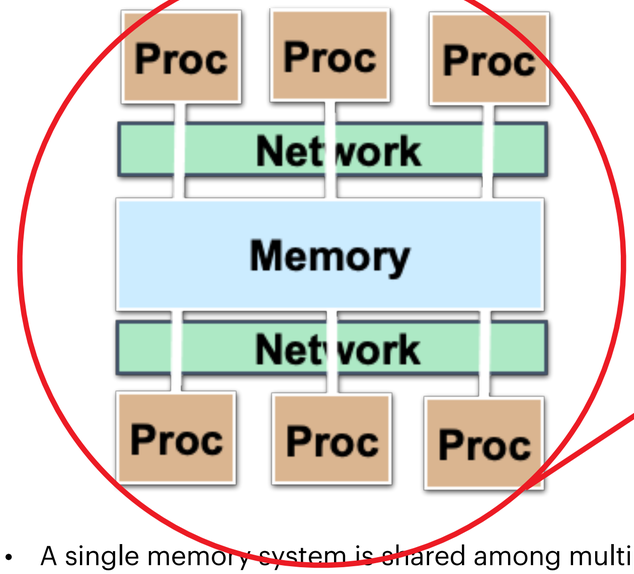
Based on the problem's demand for speed and memory, one can exploit **hierarchical parallelism**

A **Single Instruction Multiple Data** machine:



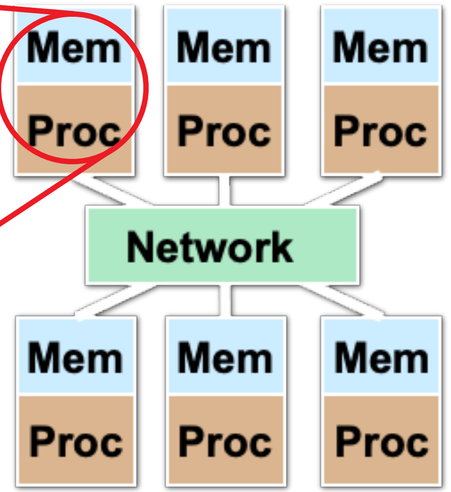
- A processor computes the same operation on multiple data in parallel
- A processor often have SIMD with 2-8 way parallelism
- **GPUs use this as well**

A **shared memory** or **multicore** machine:



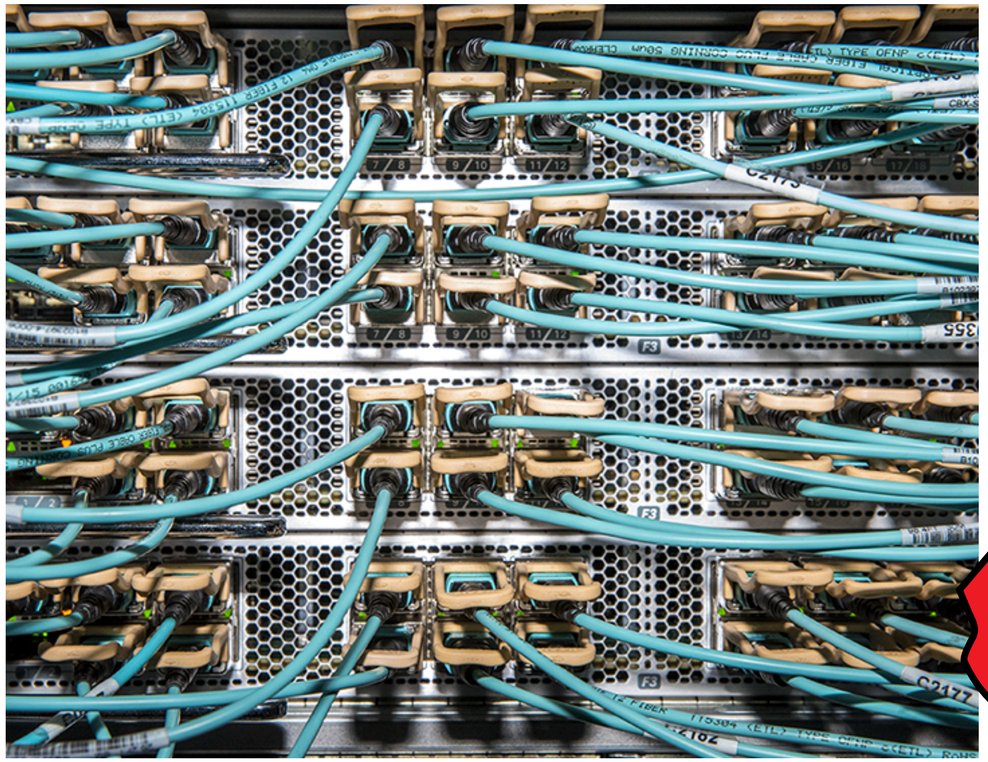
- A single memory system is shared among multiple processors
- A multicore processor has multiple cores (**proc** in the figure) on a single chip
- A multicore processor can be called a **node**

A **distributed memory** or **HPC** machine:



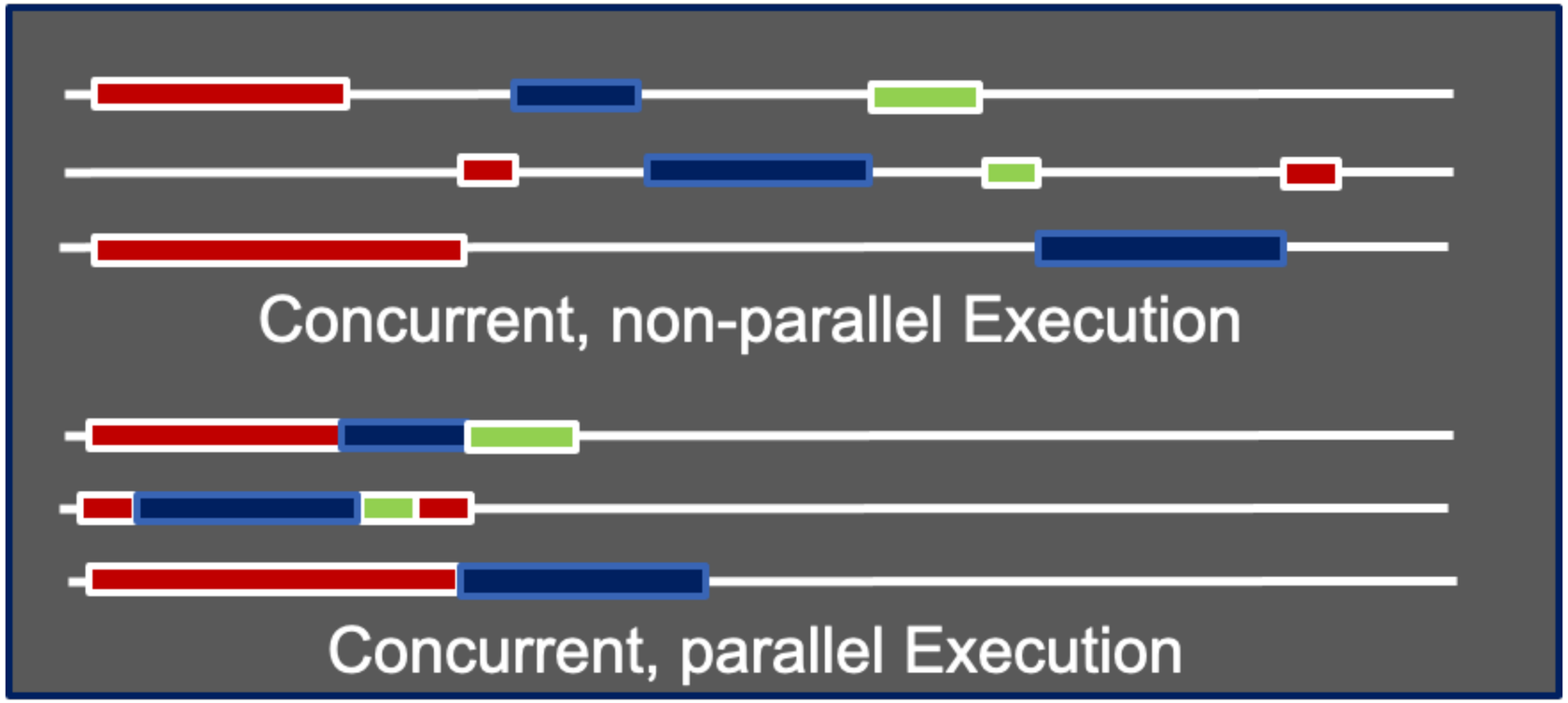
- A distributed memory multiprocessor has multiple nodes connected by a high speed **network**
- A High Performance Computing (HPC) system or **cluster** has 100s or 1000s nodes

What's NOT a Parallel Computer?



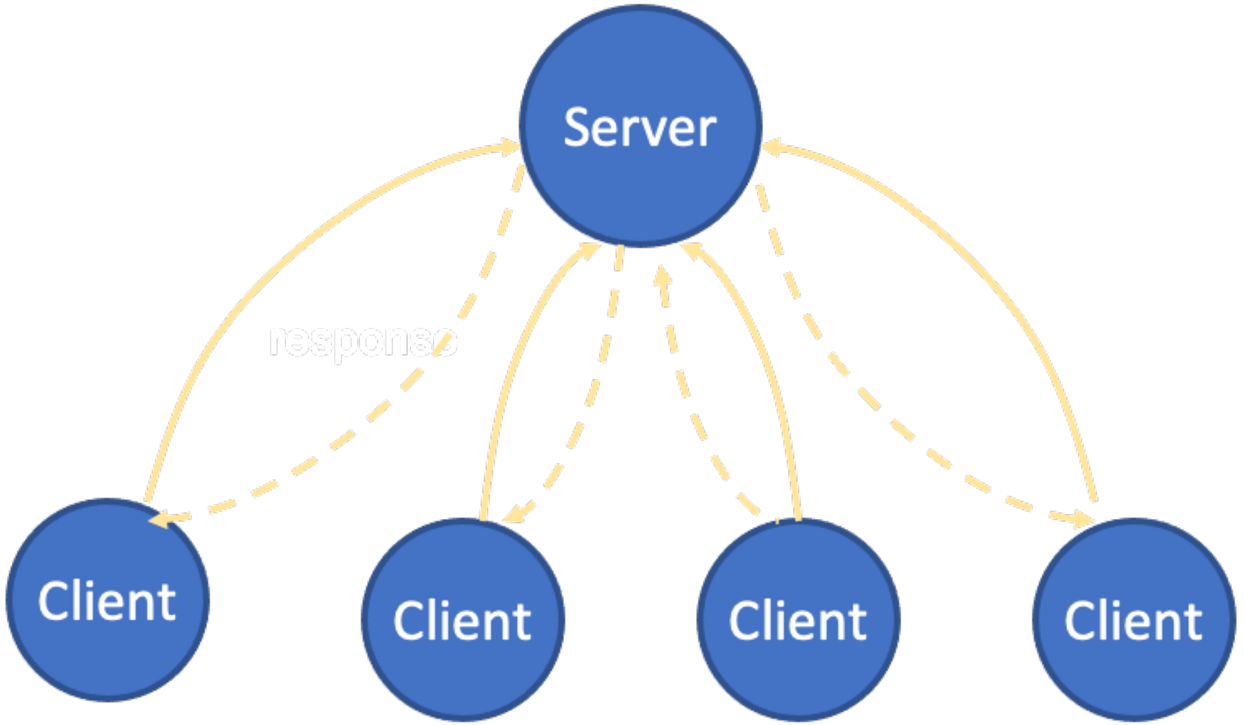
Concurrency versus Parallelism

- **Concurrency:** multiple tasks are ***logically*** active at one time
- **Parallelism:** multiple tasks are ***actually*** active at one time



Parallel Computer versus Distributed System

- A distributed system is ***inherently*** distributed, i.e., serving clients at different locations
- A parallel computer may use **distributed memory** (multiple processors with their own memory) for more performance

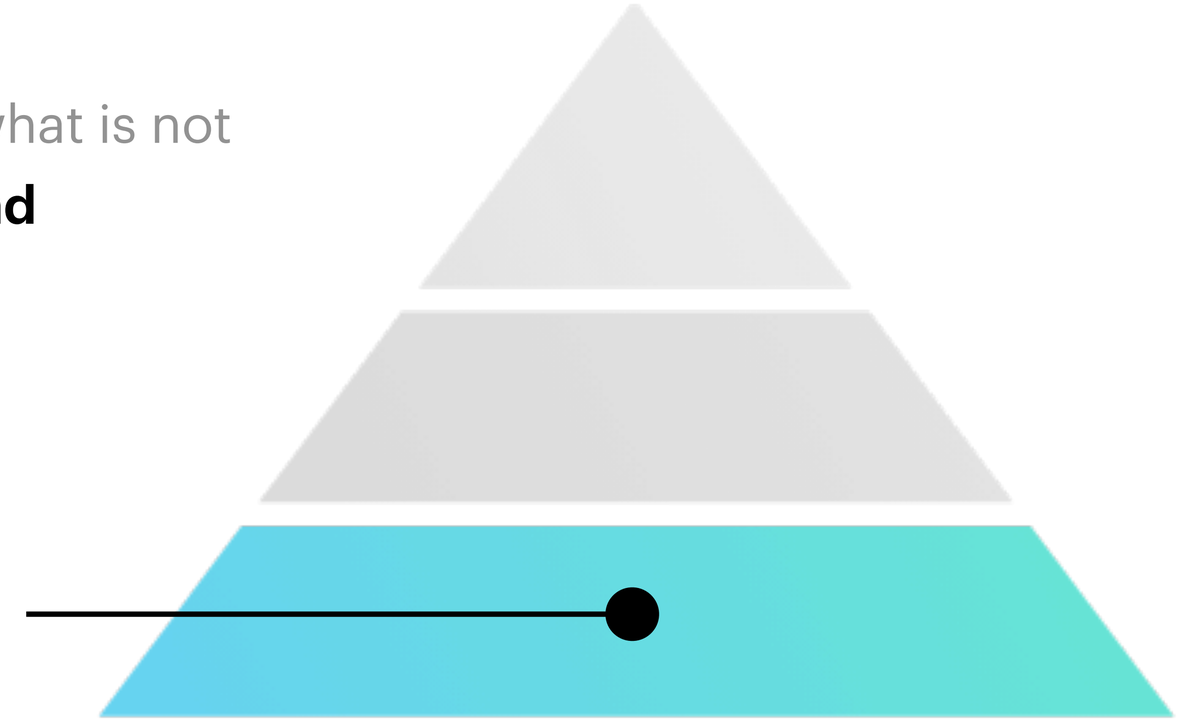


Parallel Computing Pyramid

Plan for today

- Compare what is parallel computing and what is not
- **Define performance and its historical trend**
- Course logistics and HW0
- Define and use basic performance metrics

Basics: architecture, parallel concepts, locality and parallelism



How Fast Can We Go?

Rankings and **speed records** for the fastest supercomputers in the world:

Speed is measured in **FLOPS** (floating point operations per second) instead of **MIPS** (millions of instructions per second):

- MIPS talk about instructions, i.e., any type of instruction
- **FLOPS** talk about a **specific type of instruction related to decimal numbers**

FLOPS are more accurate for scientific computing

How Fast Can We

Different machines achieve different FLOPs rates:

- Giga (10^9) — a single core
- Tera (10^{12}) — a big machine
- Peta (10^{15}) — most of current top 10 machines in the
- Exa (10^{18}) — the current top 2 machines in the top50

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,340,000	1,809.00	2,821.10	29,685
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	**JUPITER Booster** - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, EVIDEN EuroHPC/FZJ Germany	4,801,344	1,000.00	1,226.28	15,794
5	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
6	**HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461
7	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
8	**Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cray OS, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	2,121,600	434.90	574.84	7,124

How Fast Can We Go?

Different machines achieve different FLOPs rates:

- Giga (10^9) — a single core
- Tera (10^{12}) — a big machine
- Peta (10^{15}) — most of current top 10 machines in the top500.org
- Exa (10^{18}) — the current top 2 machines in the top500.org

You'll carry out your homework on the **world's 30th fastest supercomputer**: Perlmutter (NERSC)

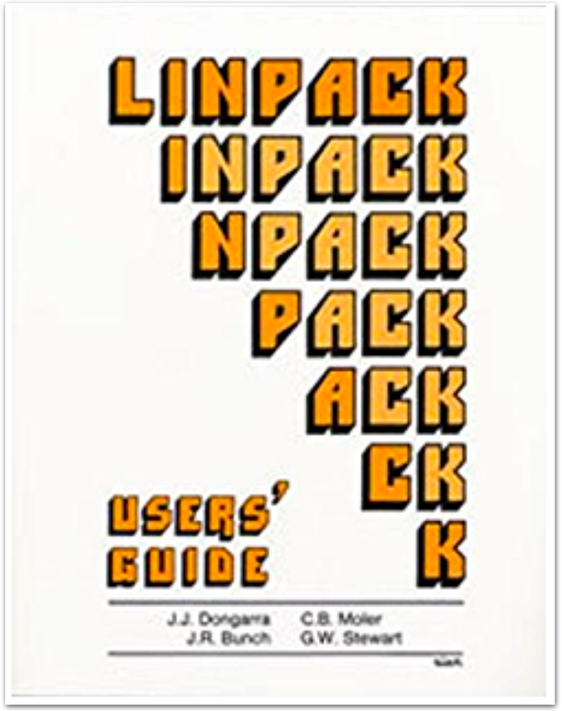
How is Top500 Created?

The Top500 is based on a portable implementation of the **high-performance LINPACK (HPL)** benchmark written in Fortran for distributed-memory machines

LINPACK is a collection of subroutines that analyze and solve linear equations and linear least-squares problems

Originally created by Jack Dongarra 2021 Turing Award

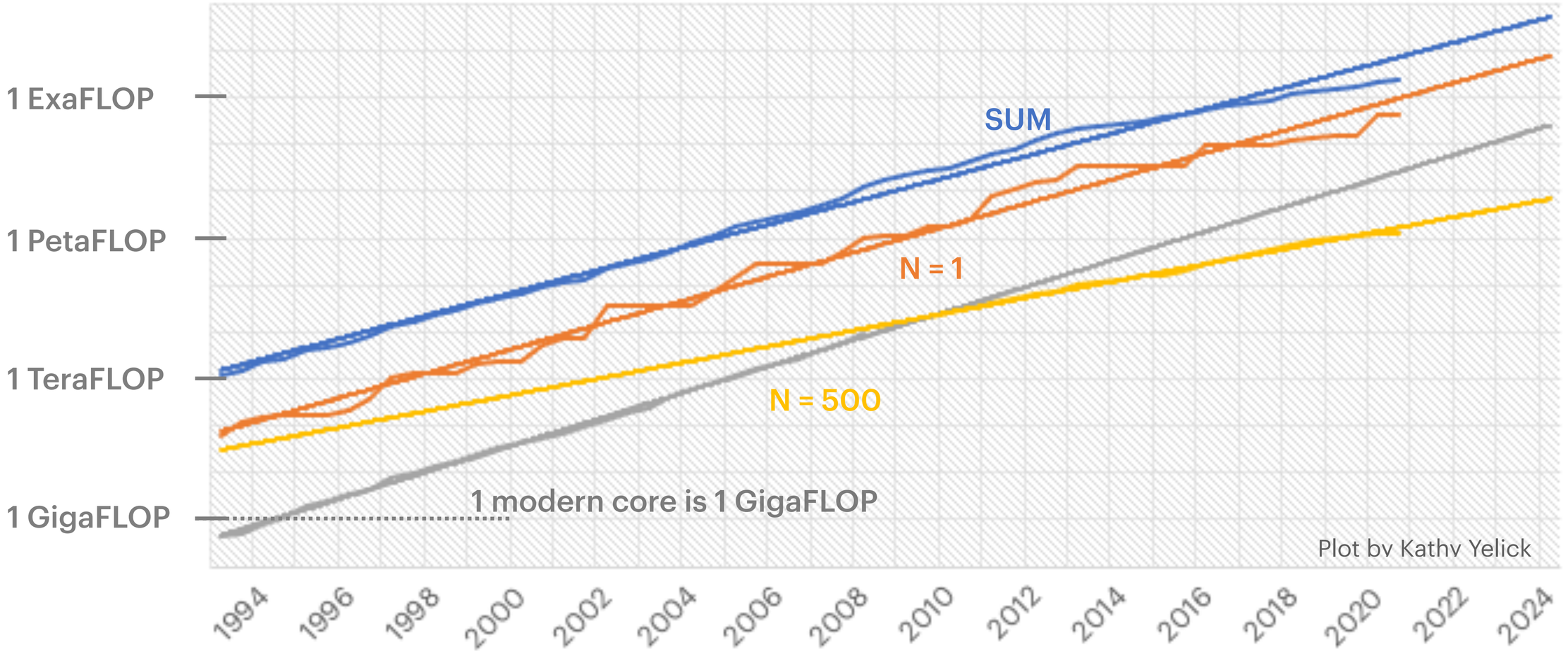
Click for Jack's talk recording (CS Colloquium Fall 2024)



LINPACK is very computationally intensive because it is dominated by dense matrix-matrix multiply

It's the **best case scenario for a supercomputer!**

Top500 Performance History



How Fast Can I Make Computation on Perlmutter?

Theoretical peak performance (Rpeak): 93.75 PFlop/s

LINPACK performance (Rmax): 70.87 PFlop/s (~76% of Rpeak)

Rpeak of **one single CPU core** of Perlmutter:

3.5 GHz * 4 vector width * 2 vector pipelines * 2 flops for FMA = 56 GFlop/s

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
30	**Perlmutter** - HPE Cray EX 235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-11, HPE DOE/SC/LBNL/NERSC United States	888,832	79.23	113.00	2,945

How Fast Can I Make Computation on Perlmutter?

Theoretical peak performance (Rpeak): 93.75 PFlop/s

LINPACK performance (Rmax): 70.87 PFlop/s (~76% of Rpeak)

Rpeak of **one single CPU core** of Perlmutter:

3.5 GHz * 4 vector width * 2 vector pipelines * 2 flops for FMA = 56 GFlop/s

56 GFlop/s * 888,832 = 47.8 PFlop/s

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
30	**Perlmutter** - HPE Cray EX 235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-11, HPE DOE/SC/LBNL/NERSC United States	888,832	79.23	113.00	2,945

How Fast Can I Make Computation on Perlmutter?

Theoretical peak performance (Rpeak): 93.75 PFlop/s

LINPACK performance (Rmax): 70.87 PFlop/s (~76% of Rpeak)

Rpeak of **one single CPU core** of Perlmutter:

3.5 GHz * 4 vector width * 2 vector pipelines * 2 flops for FMA = 56 GFlop/s

Perlmutter has GPUs too!

56 GFlop/s * 888,832 = 47.8 PFlop/s

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
30	**Perlmutter** - HPE Cray EX 235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-11, HPE DOE/SC/LBNL/NERSC United States	888,832	79.23	113.00	2,945

How Fast Can I Make Computation on Perlmutter?

Theoretical peak performance (Rpeak): 93.75 PFlop/s

LINPACK performance (Rmax): 70.87 PFlop/s (~76% of Rpeak)

In practice, the **achievable performance** is:

- << Rpeak
- It's application-dependent

Rpeak > Rmax (LINPACK) > **Gordon Bell** > Typical

Gordon Bell: Science at Scale

Established in 1987 with a cash award of $10,000 (since 2011) and funded by Gordon Bell

ACM CAREERS

2022 ACM Gordon Bell Prize Finalists Announced

Extreme-Scale Many-against-Many Protein Similarity Search

Session: Gordon Bell Prize Finalist Session 1

Description: Similarity search is one of the most fundamental computations that are regularly performed on ever-increasing protein datasets. Scalability is of paramount importance for uncovering novel phenomena that occur at very large scales. We unleash the power of over 20,000 GPUs on the Summit system to perform all-vs-all protein similarity search on one of the largest publicly available datasets with 405 million proteins, in less than 3.5 hours, cutting the time-to-solution for many use cases from weeks. The variability of protein sequence lengths, as well as the sparsity of the space of pairwise comparisons, make this a challenging problem in distributed memory. Due to the need to construct and maintain a data structure holding indices to all other sequences, this application has a huge memory footprint that makes it hard to scale the problem sizes. We overcome this memory limitation by innovative matrix-based blocking techniques, without introducing additional load imbalance.

Event Type: ACM Gordon Bell Finalist, Awards Presentation

Time: Tuesday, 15 November 2022, 10:30am - 11am CST

Location: C144-145

Tags:
Awards

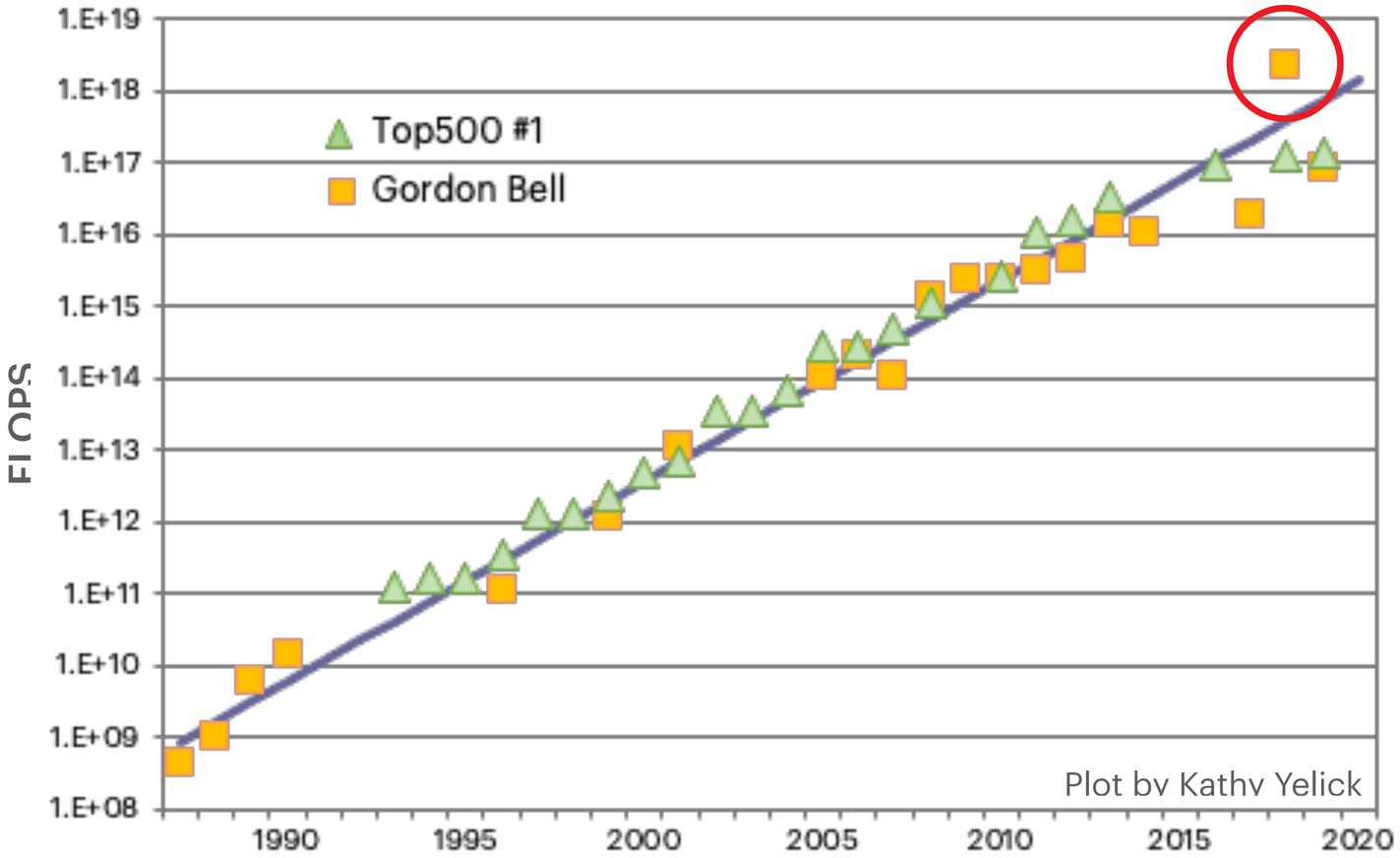
Session Formats:
Recorded

Registration Categories:
TP

The **Gordon Bell Prize** is awarded every year to reward innovation in the application of high-performance computing to large-scale science, engineering, and data analytics

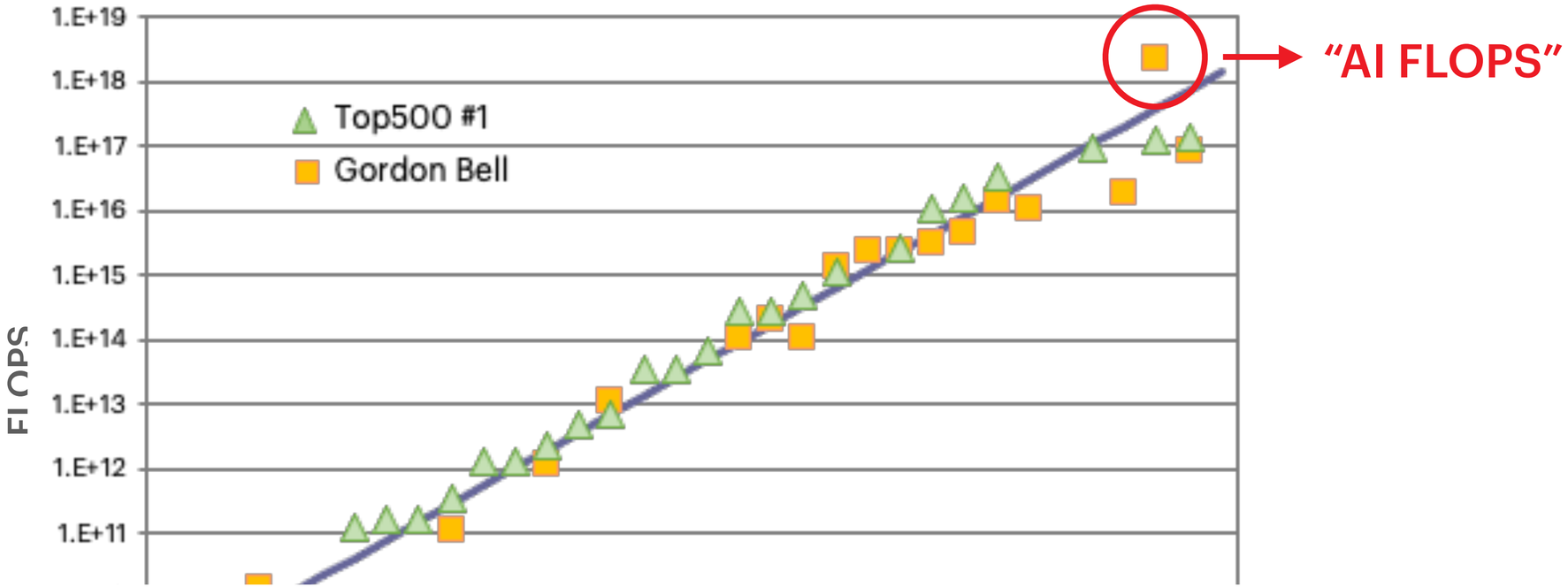
Gordon Bell versus Top500

How can the FLOPS of a real application beat Top500 #1?



Gordon Bell versus Top500

How can the FLOPS of a real application beat Top500 #1?



- The application doesn't need to use double precision ("AI FLOPS", **reduced precision**)
- LINPACK, while having high arithmetic intensity, isn't embarrassingly parallel

High Performance Conjugate Gradients (HPCG)

LINPACK is the best case scenario, **HPCG** is a newer, **more realistic benchmark** for Top500

Proposed by Michael Heroux, Jack Dongarra, and Piotr Luszczek

HPCG was developed to:

- model the data access patterns of real-world applications (e.g., sparse matrix multiply)
- test the *impact of the memory subsystem and network on computational performance*

HPCG is I/O-bound:

- Only **achieves a small fraction** of the theoretical peak performance that the computer

Key Takeaway

HPC machines are over-provisioned for FLOPS but **suffer (a lot) from data movement**

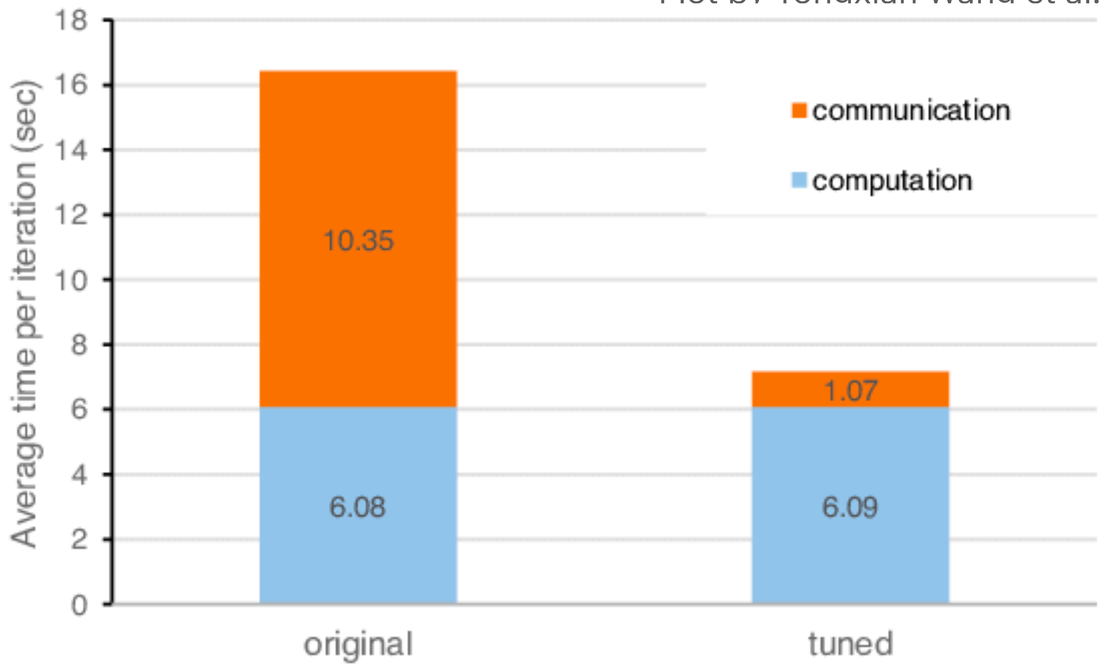
Parallel programming **objectives**:

- ⬆ parallelism (aka parallel computation)
- ⬇ data movement (aka communication)

In one sentence:

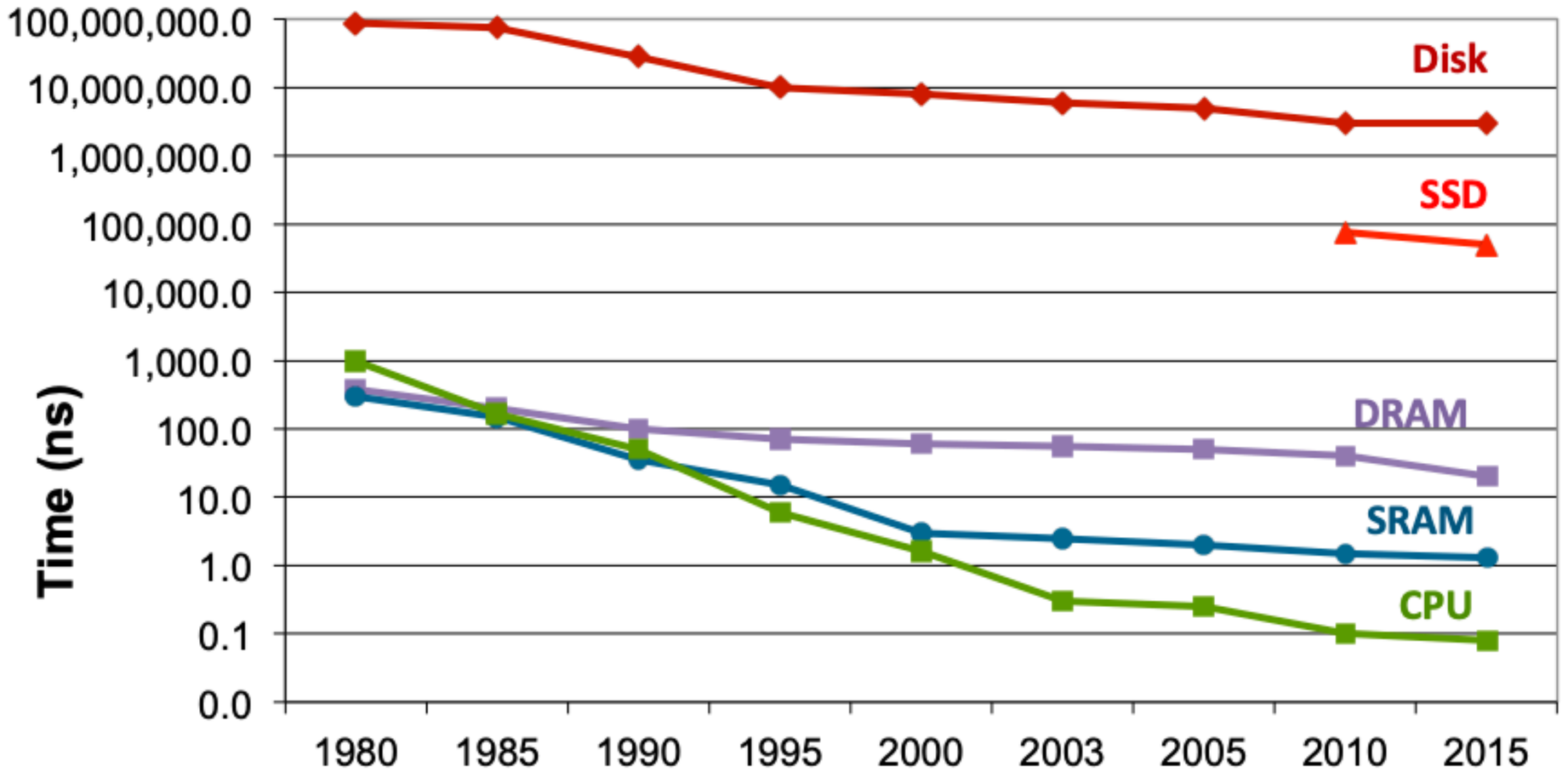
- ⬆ computation-to-communication ratio

Plot by Yonaxian Wang et al.



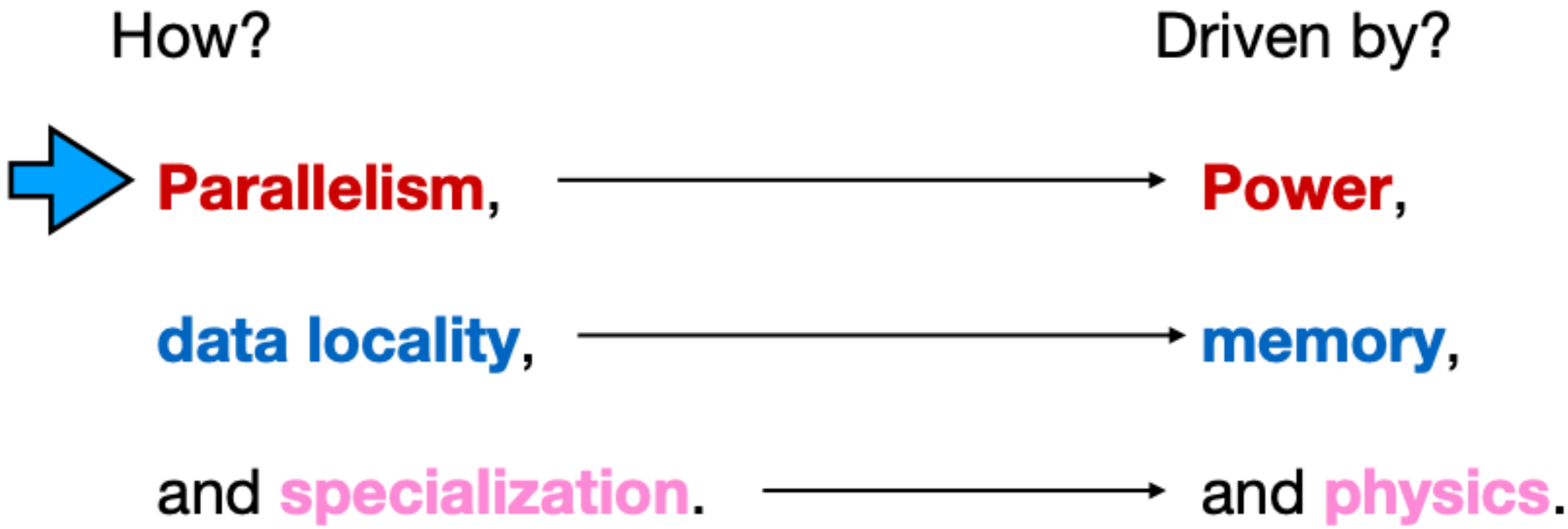
Data Movement is Expensive

Comparing CPU Cycle Time with Memory Access Time



Plot by hfp://csapp.cs.cmu.edu/2e/figures.html

Motivating Trends

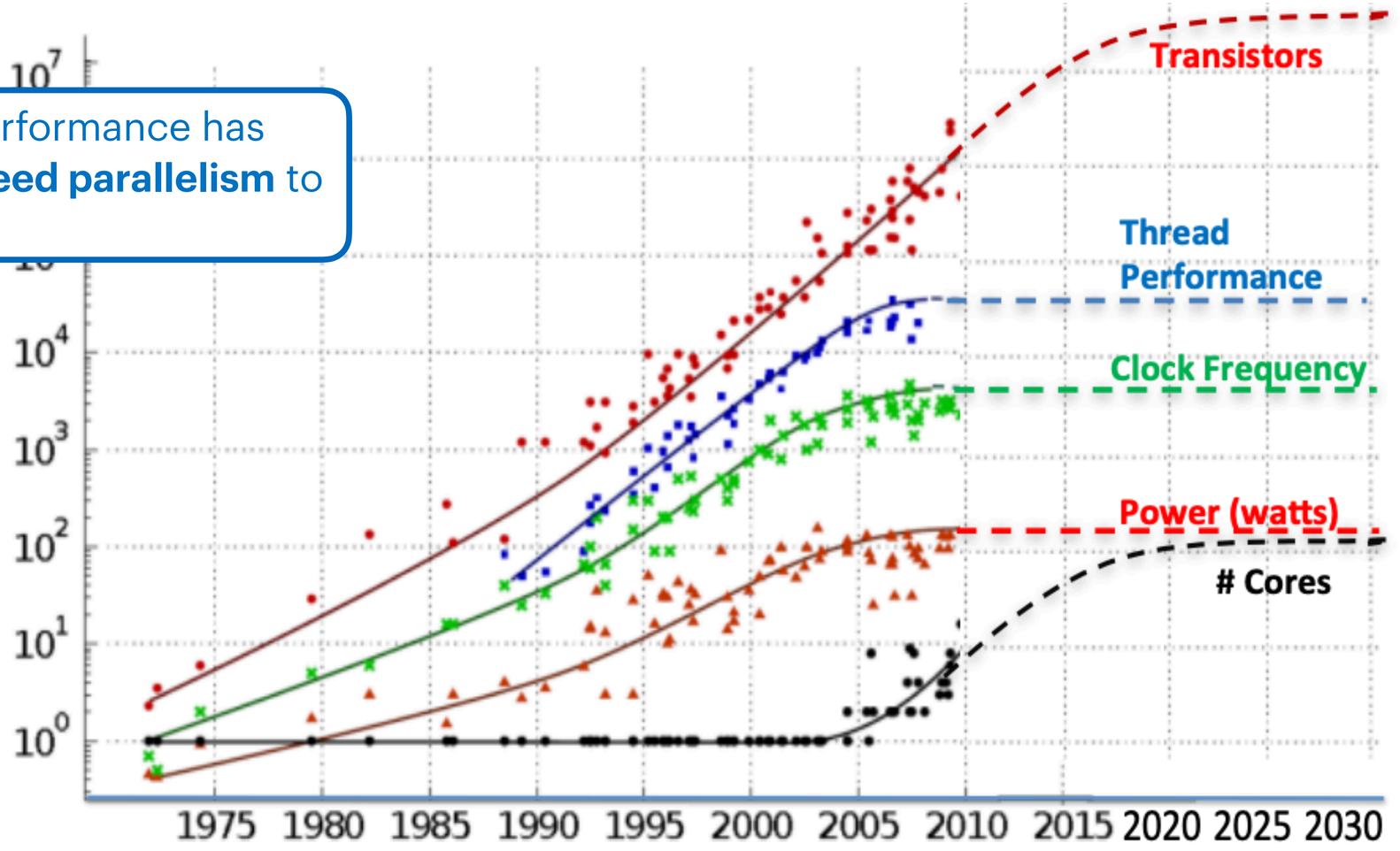


This slide was taken or inspired from Rich Vuduc (Georgia Tech)

Historical Context

Dennard Scaling and Moore's Law

Single-thread performance has stagnated, we **need parallelism** to hit peak FLOPS

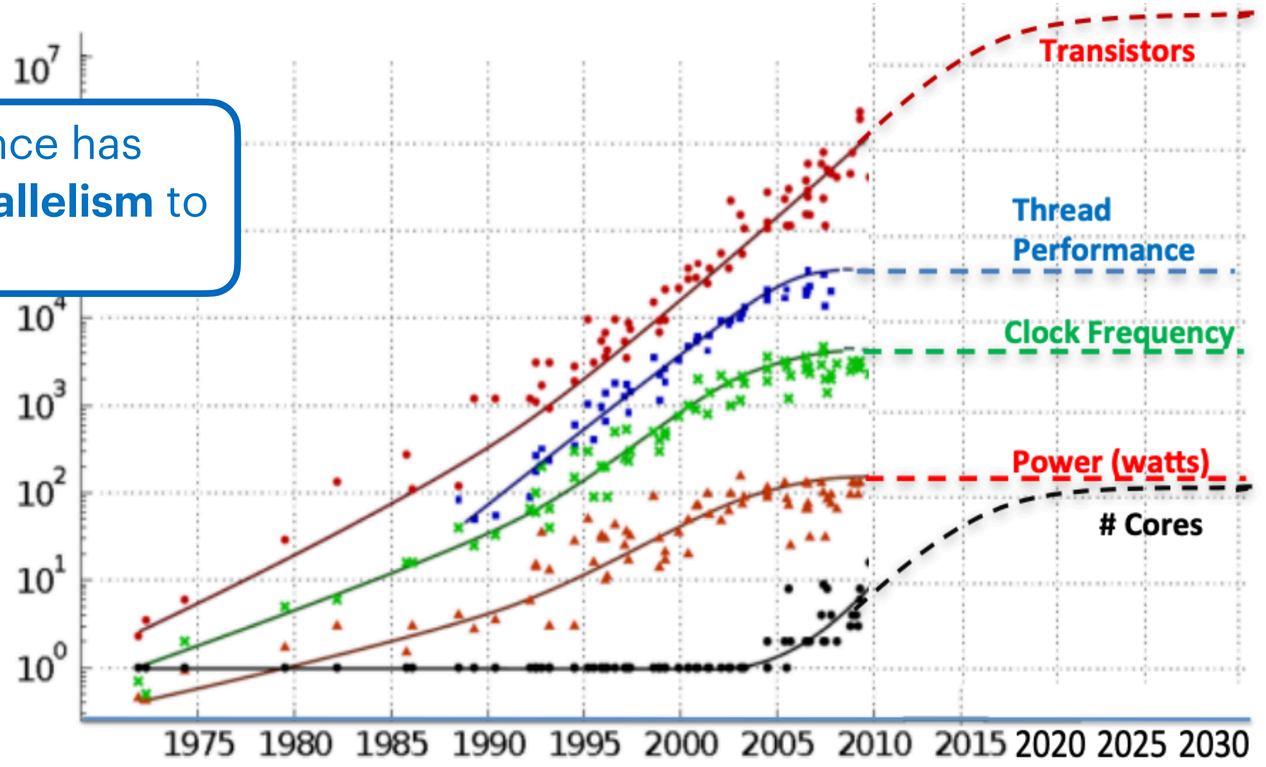


Plot by Kunle Olukotun, Lance Hammond, Herb SuJer, and Burton Smith

Historical Context

Dennard Scaling and Moore's Law

Single-thread performance has stagnated, we **need parallelism** to hit peak FLOPS



To scale performance, processor manufacturers put **many processing cores** on the microprocessor chip. Each generation of **Moore's Law** potentially doubles the number of cores.

Plot by Kunle Olukotun, Lance Hammond, Herb SuJer, and Burton Smith

Beyond Parallelism

This class is about more than just parallelism. It’s about getting the most out of your hardware, e.g., through **data locality**.

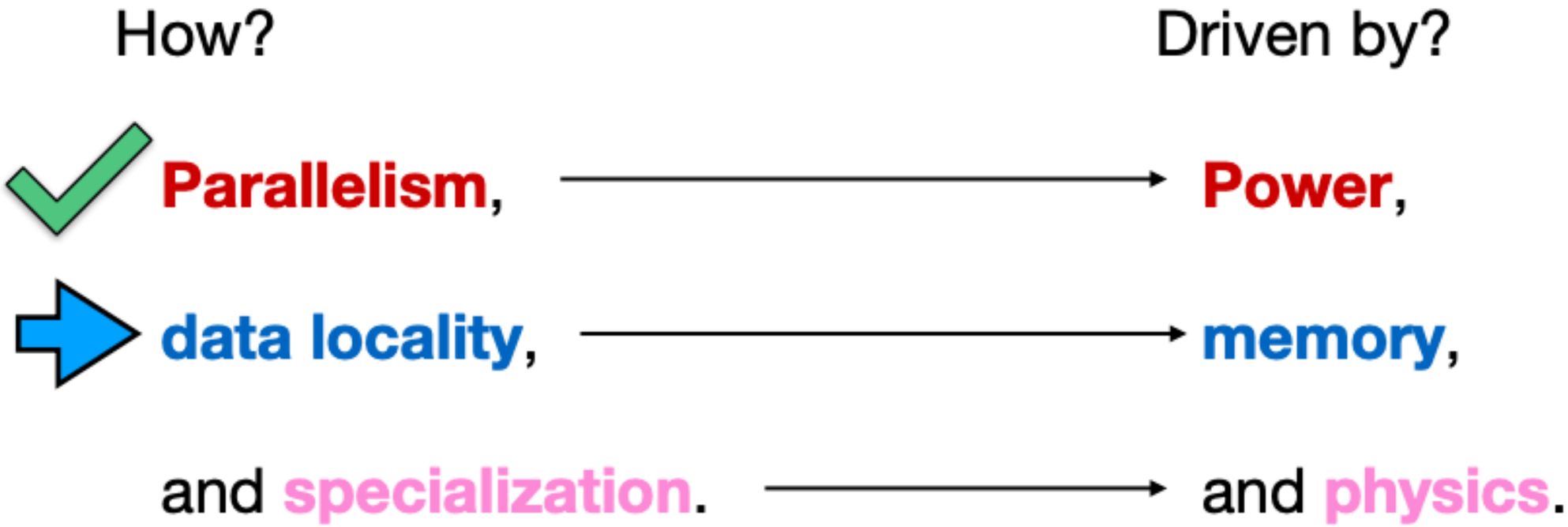
Why? The faster you run, the sooner you can stop, saving time, energy, cost, etc.

“Race-to-halt” — finish quickly then shutdown to save energy

Use resources more efficiently, rather than using more resources

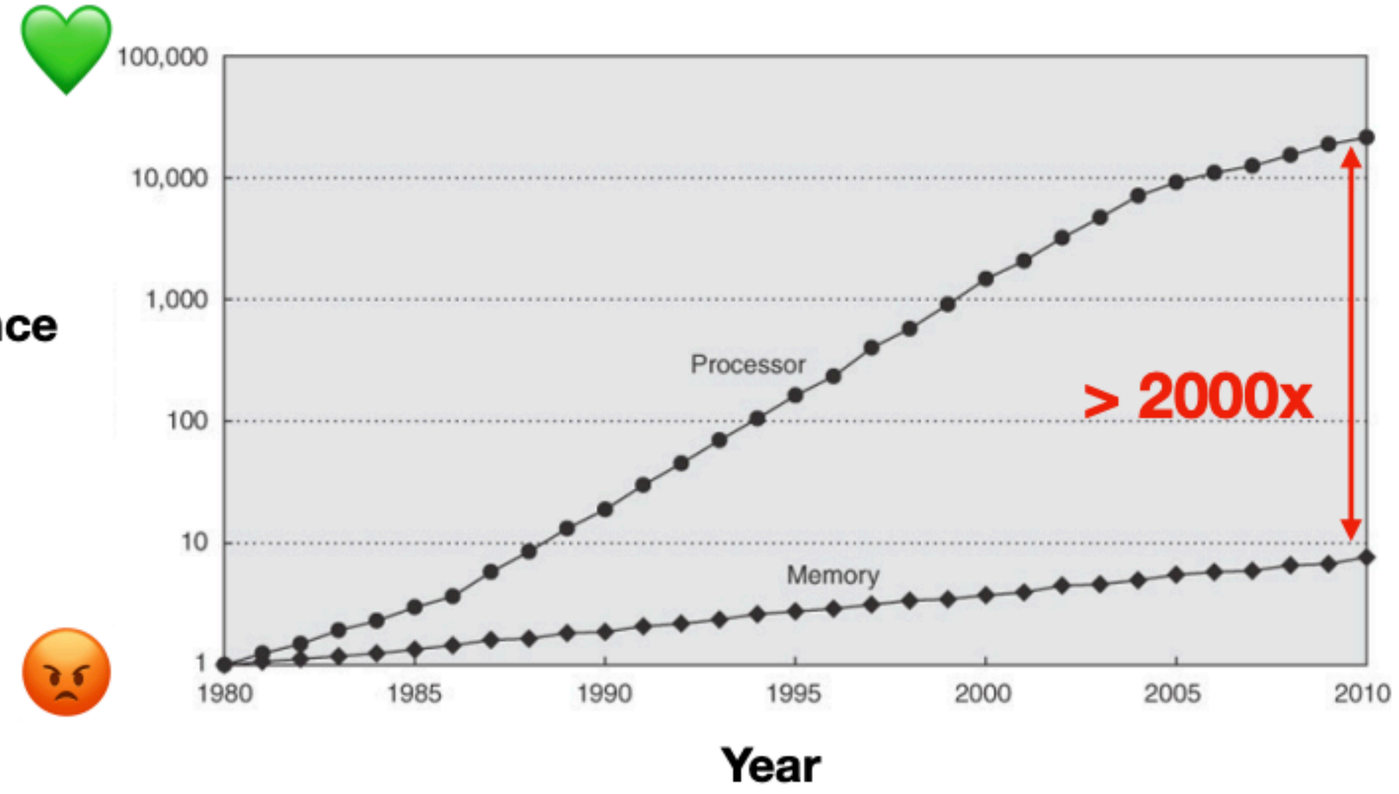
This slide was taken or inspired from Rich Vuduc (Georgia Tech)

Motivating Trends



This slide was taken or inspired from Rich Vuduc (Georgia Tech)

Data Locality and the Memory Wall

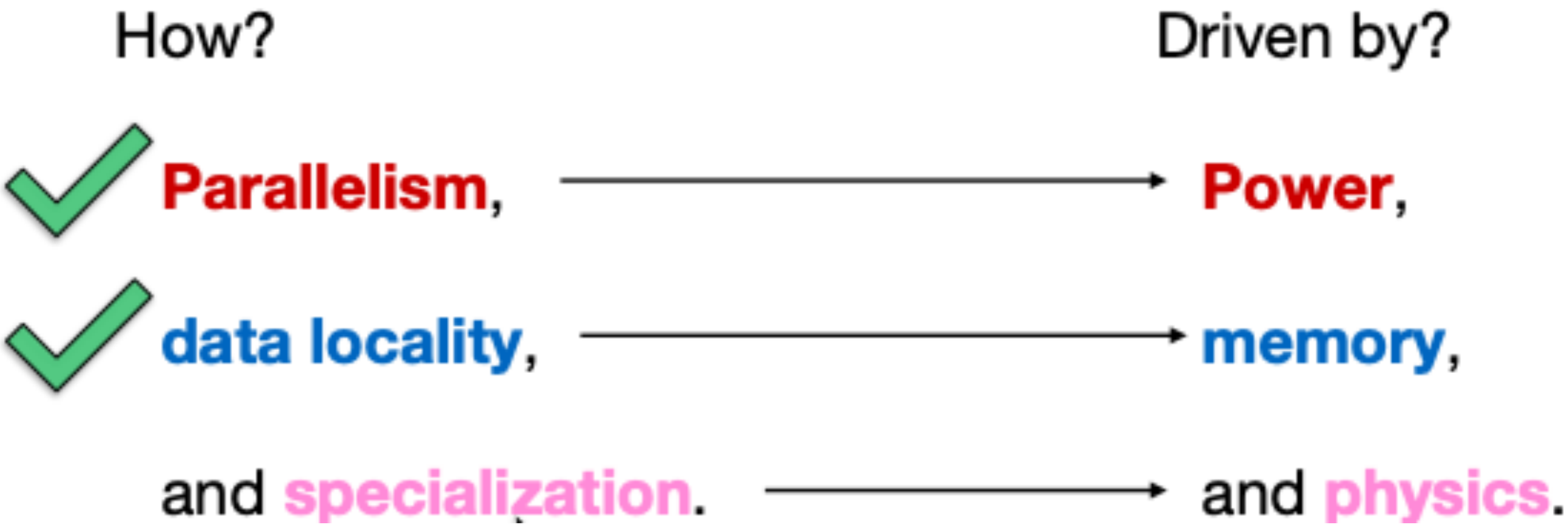


Parallelization alone is not enough due to the **processor-memory gap**, or the exponentially increasing gap between CPU and memory speeds [Hennessy & Patterson 2011].

The **memory wall**, or the **gap between memory latency and latency of floating-point operations**, is increasing exponentially over time!

This slide was taken or inspired from Helen Xu (Georgia Tech)

Motivating Trends

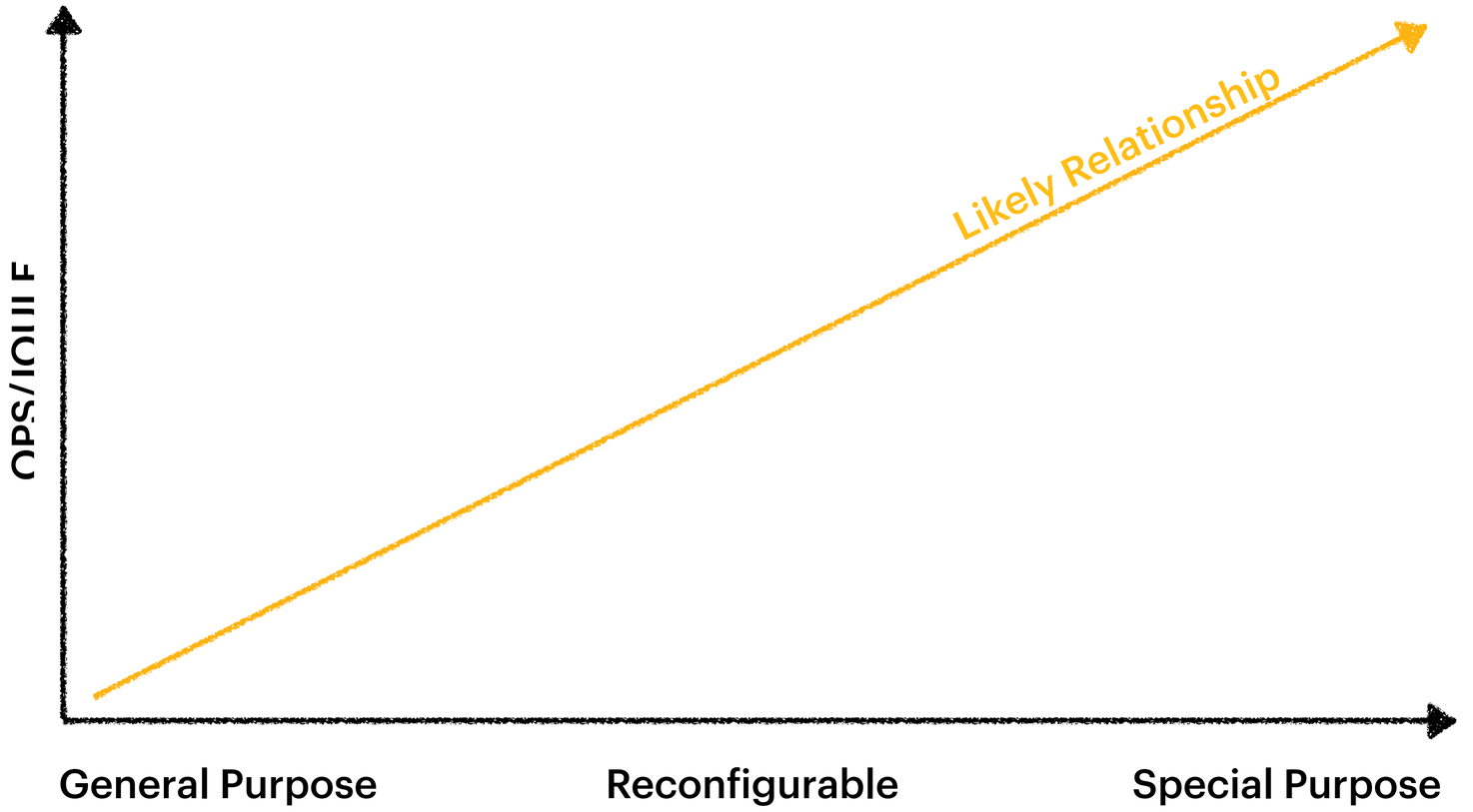


This slide was taken or inspired from Rich Vuduc (Georgia Tech)

Hardware Specialization: End Game for Moore's Law

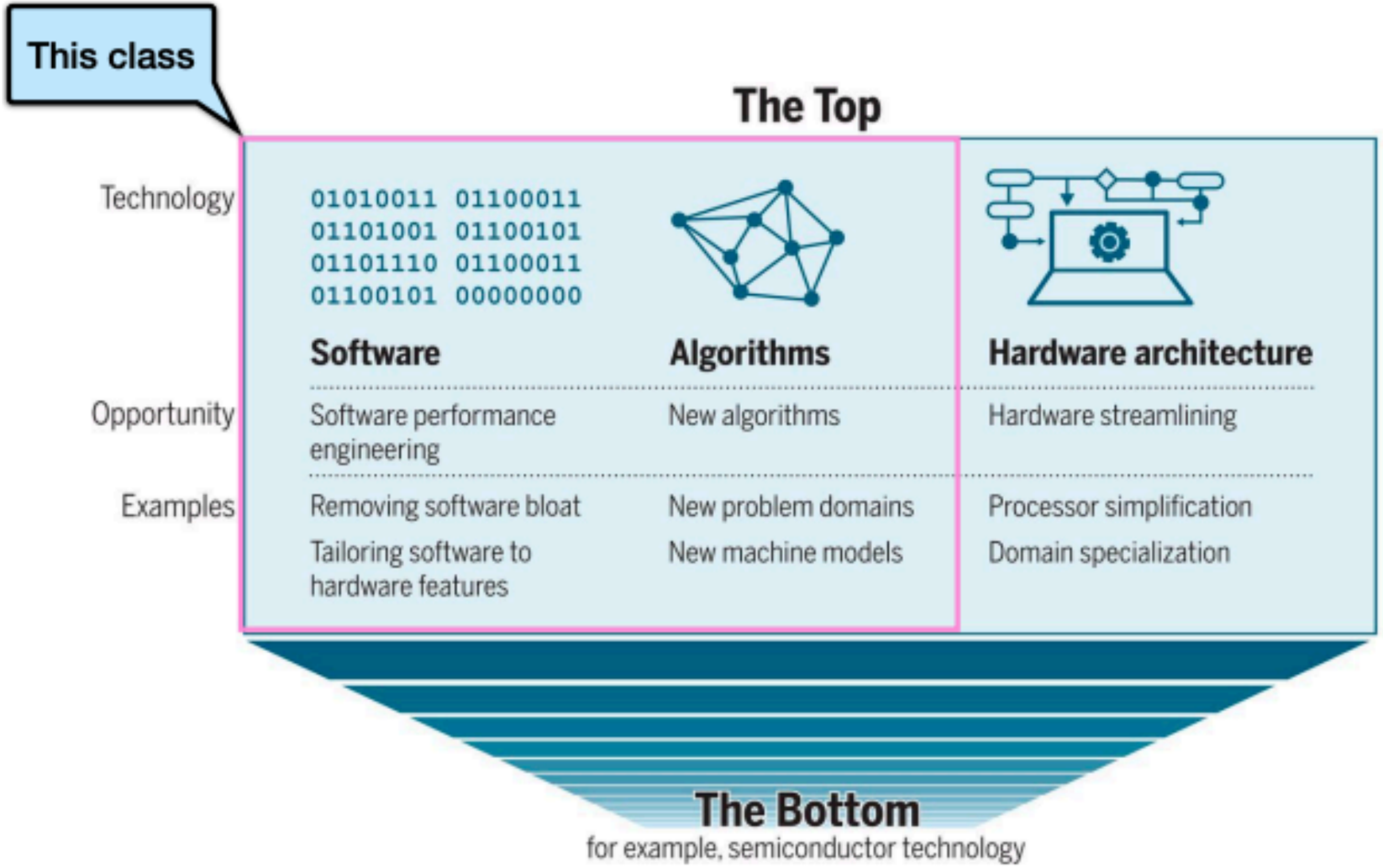
End of "Relaxed Programming"

Moore: the law that taught performance programmers to relax



Plot by Kathy Yelick

Performance Gains after Moore's Law Ends



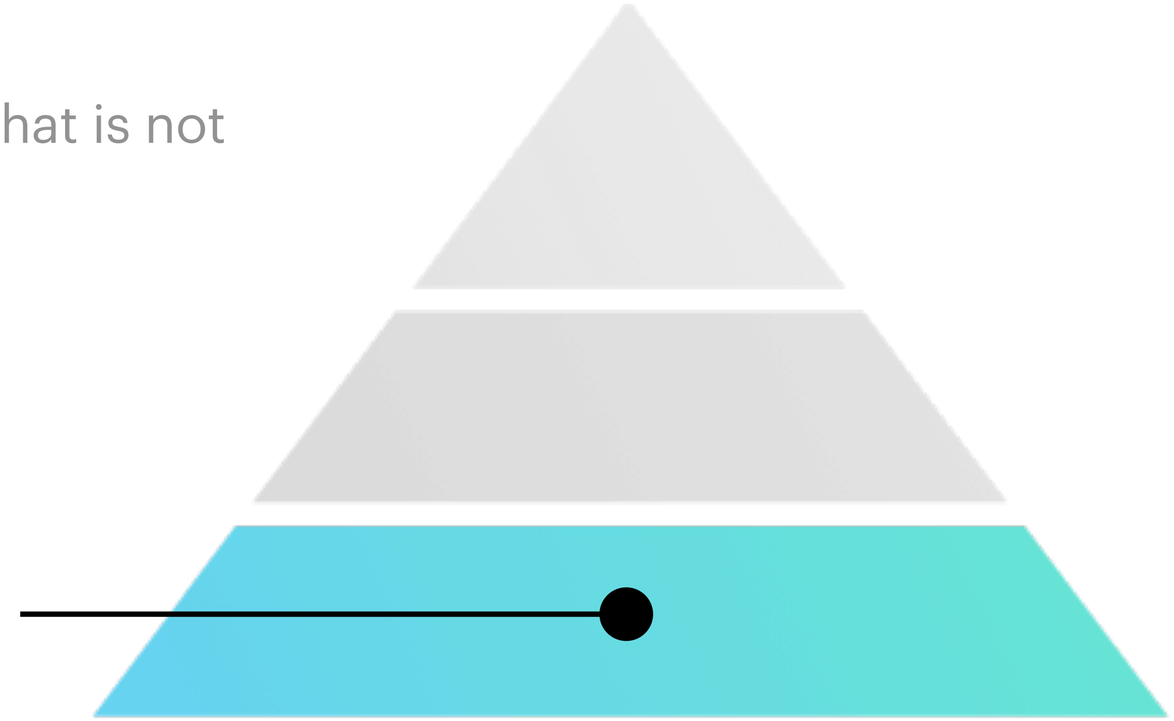
From "There's plenty of room at the Top: What will drive computer performance after Moore's law?" by Leiserson et al. (Science, 2020

Parallel Computing Pyramid

Plan for today

- Compare what is parallel computing and what is not
- Define performance and its historical trend
- **Course logistics and HW0**
- Define and use basic performance metrics

Basics: architecture, parallel concepts, locality and parallelism



Lectures Plan

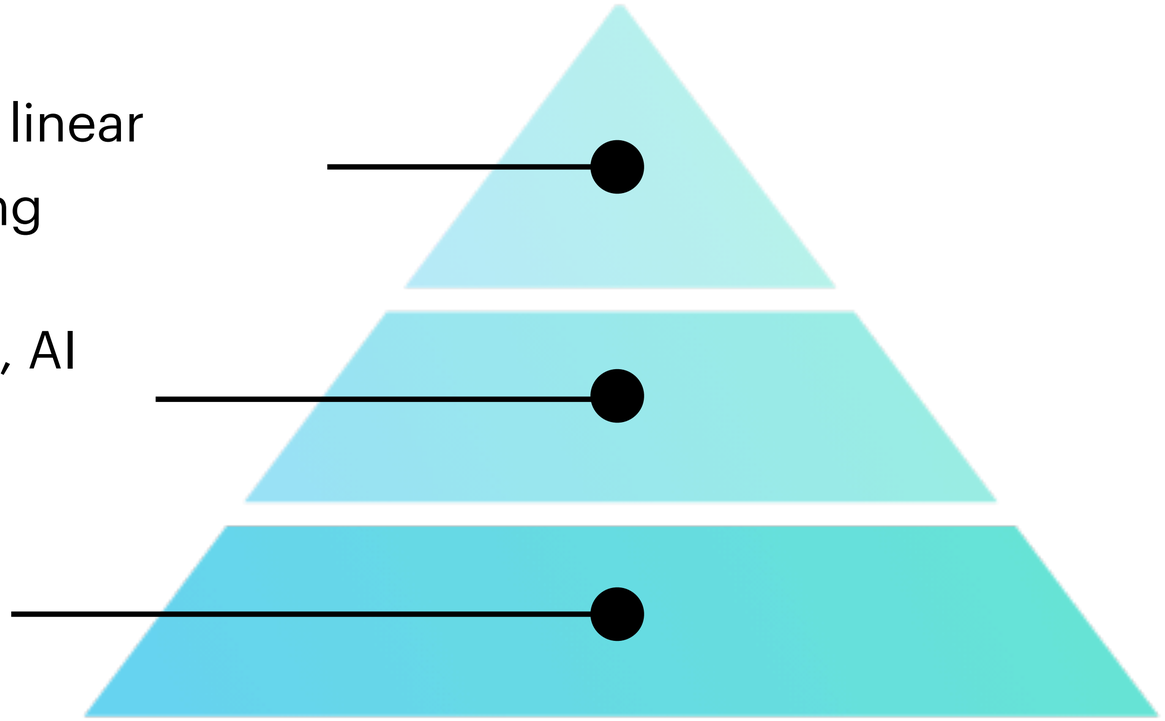
Parallel Computing Pyramid

A detailed schedule can be found **here** on **the** course webpage**!**

Patterns and Applications: dense and sparse linear algebra, graph partitioning, and load balancing

Technology: OpenMP, MPI, CUDA, NVSHMEM, AI accelerators

Basics: architecture, parallel concepts, locality and parallelism



Course Staff

Instructor: Professor Guidi

OH: Tu 3-4 pm in 437 Gates Hall

Contact: gguidi@cornell.edu

TA: Giridhar Balachandran

OH: TBD

Contact: gb555@cornell.edu

TA: Julian Bellavita

OH: Fr 4-5 pm in TBD

Contact: jb2695@cornell.edu

TA: Anish Bhupalam

OH: TBD

Contact: ab2352@cornell.edu

TA OHs will start **next week** while the instructor OH will start on February 3

Course Logistics in a Nutshell

- Course material on course website, **read syllabus carefully**
- **NO** midterm or final exam
- 1 non-coding/introductory assignment to be done **individually** (HW0)
- 6 coding assignments to be done **individually** (evaluated on Perlmutter)
  - **15-20 minutes recitation** by the head TA for each homework at the end of the lecture on the day they are released (the slides will be posted afterward)
- 1 final project to be done **individually** or **in small groups** (max 3 people)

Carefully read syllabus and posts on **EdDiscussion**

Course Logistics in a Nutshell

- In general, **two weeks** per coding homework
- **Partial credit** to submit some (predefined) progress after 1 week
- The **lowest score** assignment betwen HW1-6 will be dropped
- If you're on the waitlist and want to take the class, please just follow along as if you were enrolled. I expect everybody to be able to enroll (see syllabus for late enrollment).
- **Prerequisites:** CS3410 or equivalent, C/C++ programming
- There's **no textbook**, please see readings on the course website
- The time requirements is dependent on each student's background and experience—it's a 4-credit course so ~12 hours/week is expected

Course Logistics in a Nutshell

- The assignments are always **due 11:59 pm EST** on the due date of the assignment
- **1 second late = 1 day late** (please see slip days and late policy in the syllabus)

Plagiarism

- The assignments must be all of **your own writing / code**
- You may **not** copy source code from other groups, the web, or GenAI (see GenAI policy in the syllabus)
- GenAI is **strictly prohibited** in assignment and project **reports** but it can be treated as a “collaborator” on coding assignments and coding portion of the final project
- Plagiarism is considered a violation of Academic Integrity Policy

HW0: Hello, Perlmutter

Create a **NERSC account** and run the MPI (Message Passing Interface) ping-pong example from the demos

- **HW0** is available on the course website and GitHub
- **HW0** is due **2026-01-29** at 11:59 pm EST

Questions?

- For technical questions about the coursework, please use **EdDiscussion**. Do **not** email me or the TAs directly.
- If you have an emergency, please post privately on **EdDiscussion** so that it reaches the whole course staff.