CPSC 424/524
Parallel Programming Techniques
Fall 2018

Andrew Sherman
Department of Computer Science
Yale University

Monday/Wednesday 9:00am – 10:15am
WTS A51

What's this course about?

<u>High Performance (Technical) Computing (HPC)</u>

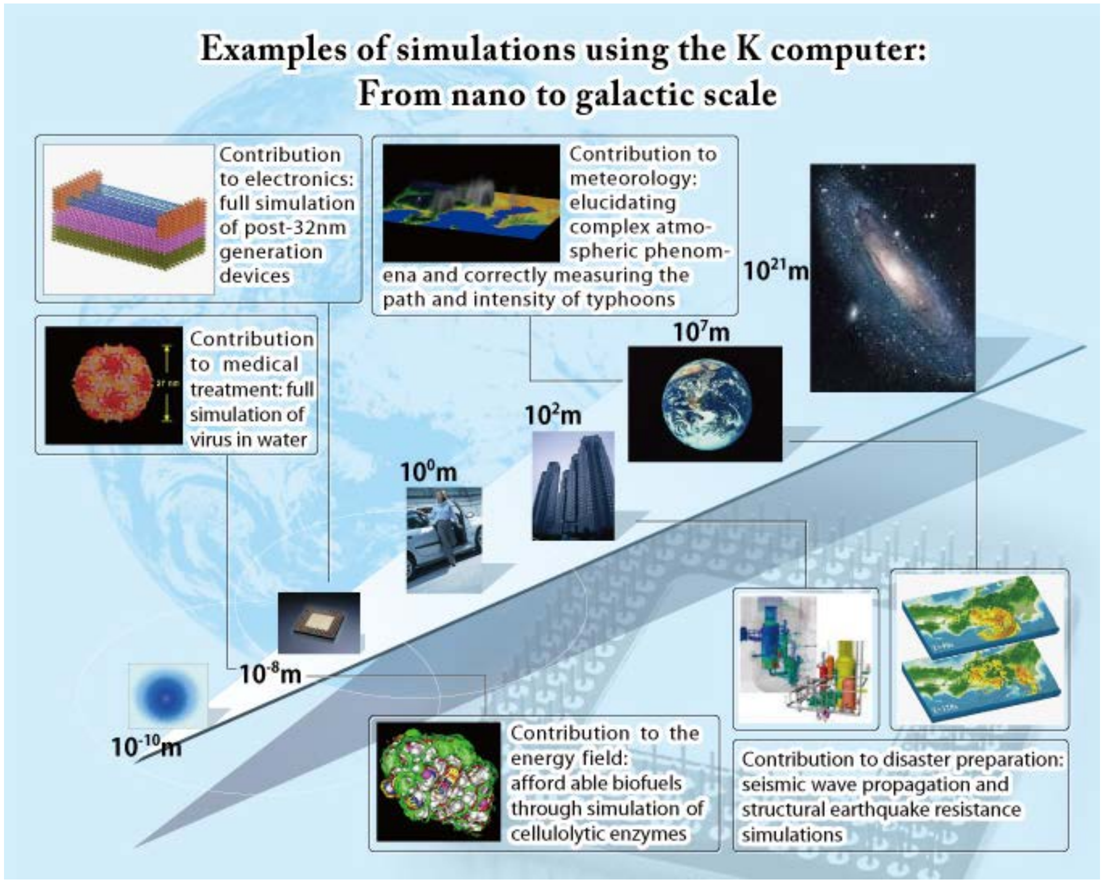
Using today's fastest computers ("supercomputers") to solve technical computing problems. This class will mainly focus on problems in science & engineering, but the techniques covered (mainly using parallel computing) are applicable to many fields.

Why is this interesting?

- Short answer: Better computational results
- More details:
  - Could solve the same problem faster:
    - Might be the key to making an application feasible (e.g., weather forecasts)
    - Could repeat a calculation with multiple parameter sets to find the best one
  - Could solve larger/more complex problems in the same time period
    - Might lead to better models that are more accurate and realistic

Why should you care about HPC?

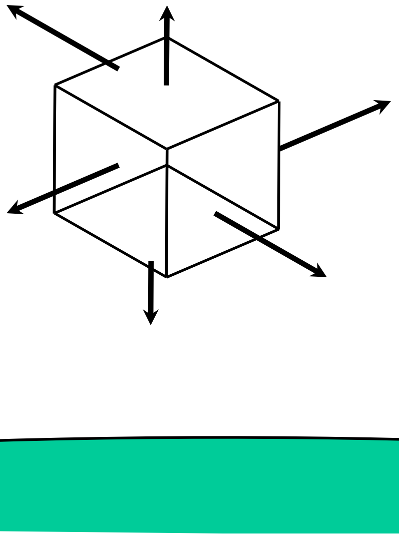
- Research: Broad range of research in science, engineering, and other fields
- Applications: Important "real-world" impact: weather, machine learning, AI, financial modeling, Internet search, data analysis, medicine, energy ...


Examples of simulations using the K computer:
From nano to galactic scale
Contribution to electronics: full simulation of post-32nm generation devices
Contribution to meteorology: elucidating complex atmospheric phenomena and correctly measuring the path and intensity of typhoons
Contribution to medical treatment: full simulation of virus in water
10^{21}m
10^{7}m
10^{2}m
10^{0}m
10^{-8}m
10^{-10}m
Contribution to the energy field: afford able biofuels through simulation of cellulolytic enzymes
Contribution to disaster preparation: seismic wave propagation and structural earthquake resistance simulations

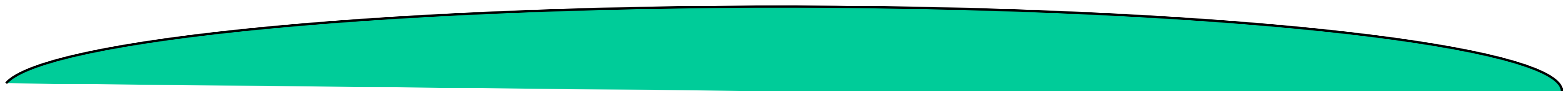
Riken AICS, 2011

Familiar Example: Weather Forecasting

Atmosphere modeled by dividing it into 3-dimensional cells.



Temperature, pressure, molecular composition, phases, etc.



Calculations involve determining what’s in a cell and how material evolves and moves, based on (or causing) differences/changes in temperature and pressure. The calculations for the cells are repeated many times to model the passage of time. (Moving ahead a small amount is called a “time step”.)

Why is Global Weather Forecasting Challenging?

- Suppose whole global atmosphere divided into cells of size .125 mile $\times$ .125 mile $\times$ .25 mile to a height of 12 miles (48 cells high)
  $\Rightarrow$ about 1.3×10^{11} cells.

- Suppose updating each cell uses ~200 arithmetic operations.
  $\Rightarrow$ For one time step, ~2.5×10^{13} arithmetic operations are needed.

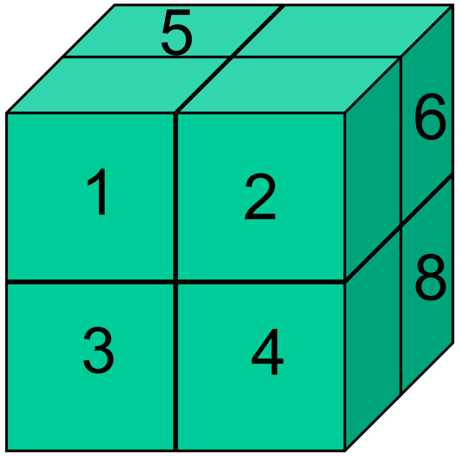
- To forecast the weather for 7 days using 1-minute intervals to track changes, a computer operating at 20 Gigaflops (2×10^{10} arithmetic operations/sec) on average would take ~1.25×10^{7} seconds.
  $\Rightarrow$ It would take over 20 weeks to simulate 7 days!

- To do this in 1 hour would require a computer ~3500 times faster
  $\Rightarrow$ Computer speed of ~70 Tflops (70×10^{12} arithmetic ops/sec)

Parallelism Makes Weather Forecasting Feasible

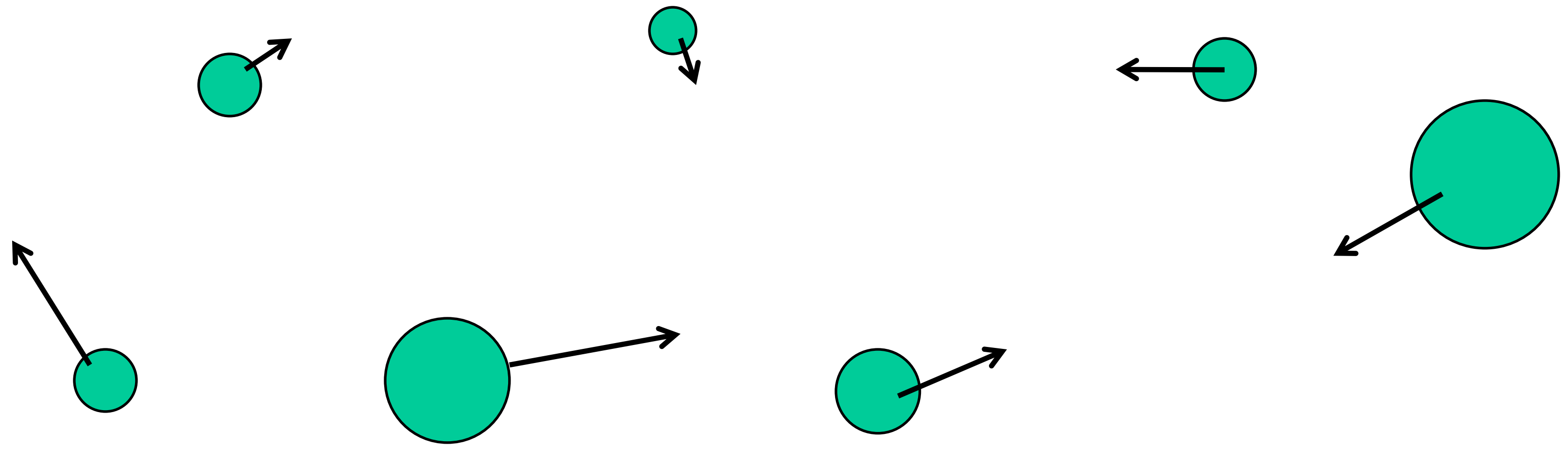
How can this sort of performance be achieved?

- Divide the problem among many individual processors (computers)



- But the computations in each cell depend on nearby cells, so now you have to deal with interprocessor communication, as well as with the computation. But with fast enough processors and a fast network, this can be made to work pretty well.

Another Example: Modeling Interacting Bodies

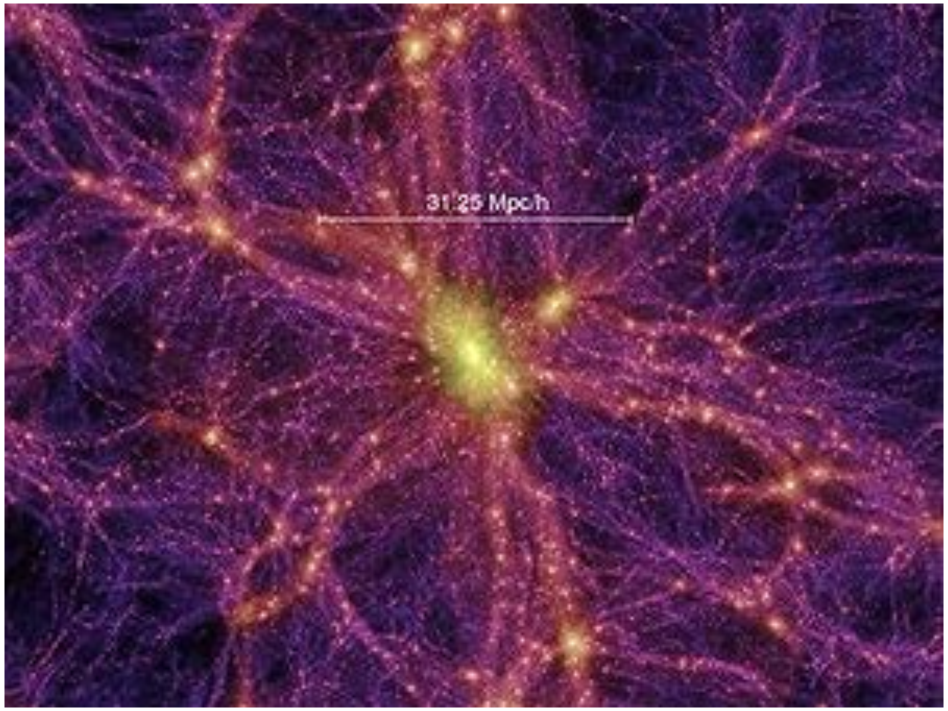


Each body is affected by each other body through forces. Movement of each body in a short time period (one time step) is predicted by evaluating the total instantaneous forces on each body, calculating body velocities, and moving the bodies through the time step. Many time steps are required.

Example: Gravitational N-Body Problem

Model positions and movements of bodies in space subject to gravitational forces from other bodies, using Newtonian physics.

Example: Cosmological Simulations



In 2005, the Millennium Simulation traced 2160^3, or just over 10 billion, “particles” (each representing ~1 billion solar masses of dark matter) in a cube of side ~2 billion light years.

Required over 1 month of time on an IBM supercomputer, and generated ~25 Terabytes of output. By analyzing the output, scientists were able to recreate the evolutionary history of ~20 million galaxies populating the cube.

Approaches to Modeling Many-Body Motion

Basic Idea: Start at some known configuration of the bodies, and use Newtonian physics to model their interactions and motions over a large number of timesteps

For each time step:

- Calculate the forces:
  - "**Brute Force" Algorithm**: With N bodies, N-1 forces to calculate for each body, or approx. $O(N^2)$ calculations (50% reduction for symmetry)
- Move the bodies to new locations
- Repeat

Challenges in Modeling Many-Body Motion

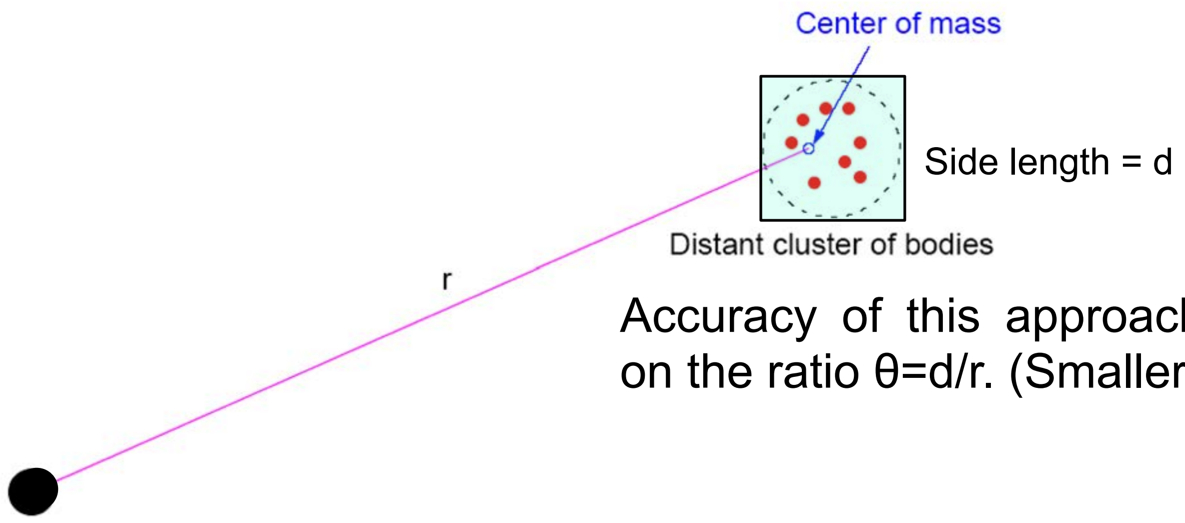
- A galaxy might have ~10^{11} stars. So one time step would require:
  - ~5×10^{21} force calculations using “brute force”

- Suppose that, using 1 computer, each force calculation takes 0.1 μsec (might be optimistic!). Then **1** time step takes:
  - Over 1.6×10^{7} years using “brute force”

- To make this computation feasible, you either need a MUCH better algorithm, or you need to find a way for many computers to cooperate to make each time step much faster, or both

Algorithmic Improvement: Clustering Approximation

Approximate the effect of a cluster of distant bodies by treating them as a single distant body with mass located at the center of mass of the cluster:

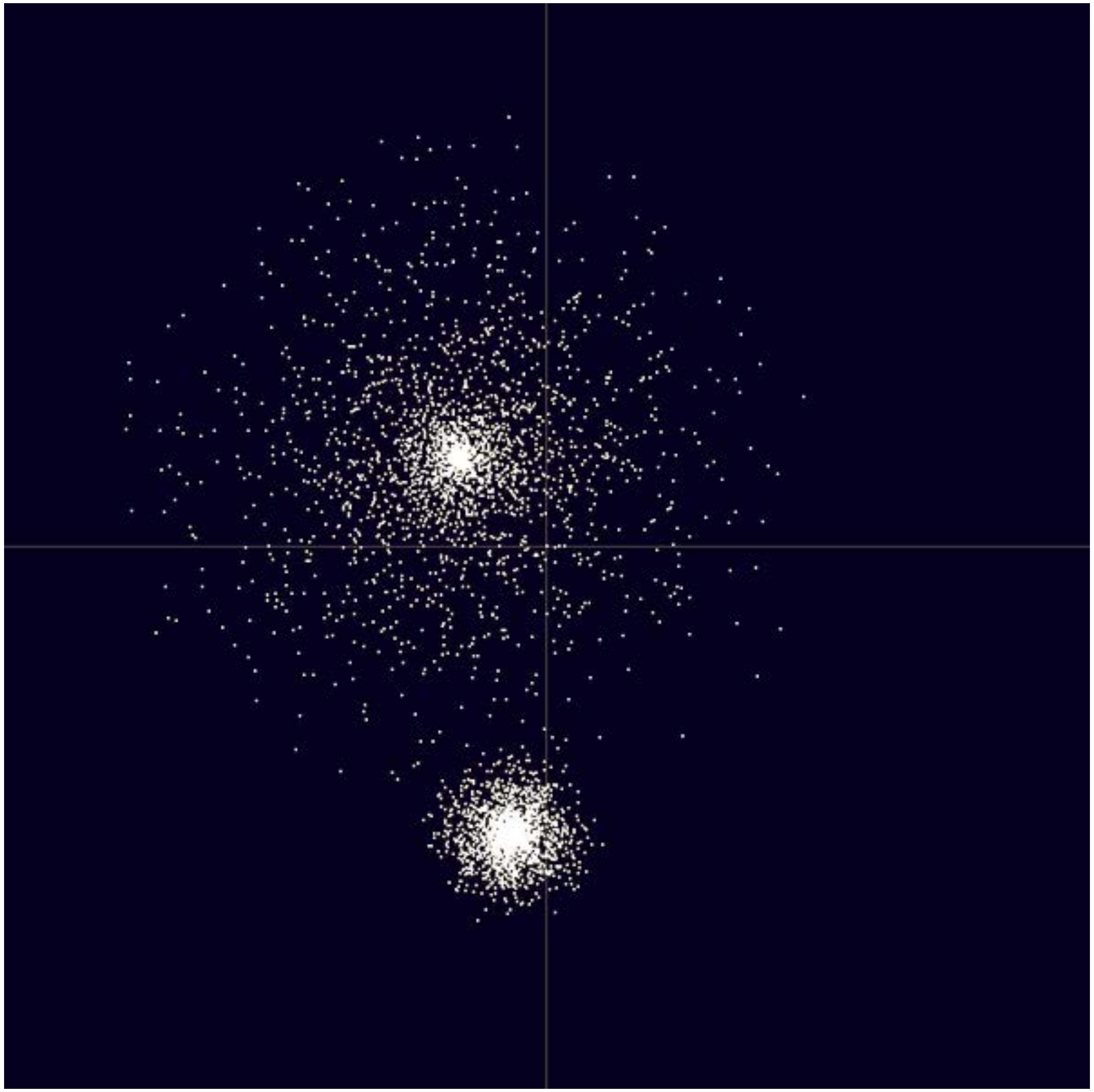


Accuracy of this approach depends on the ratio θ=d/r. (Smaller is better.)

This idea leads to $O(N \log_2 N)$ algorithms for N-Body problems. The approach has been "discovered" many times in many different fields, including as the Fast Multipole Method by Leslie Greengard and Vladimir Rokhlin at Yale.

In astrophysics, the idea underlies the Barnes-Hut algorithm, which reduces the serial runtime per timestep from 1.6×10^7 years to ~4 days. Further improvement can come from a "divide-and-conquer" parallel implementation based on adaptively dividing the cube into many sub-cubes.

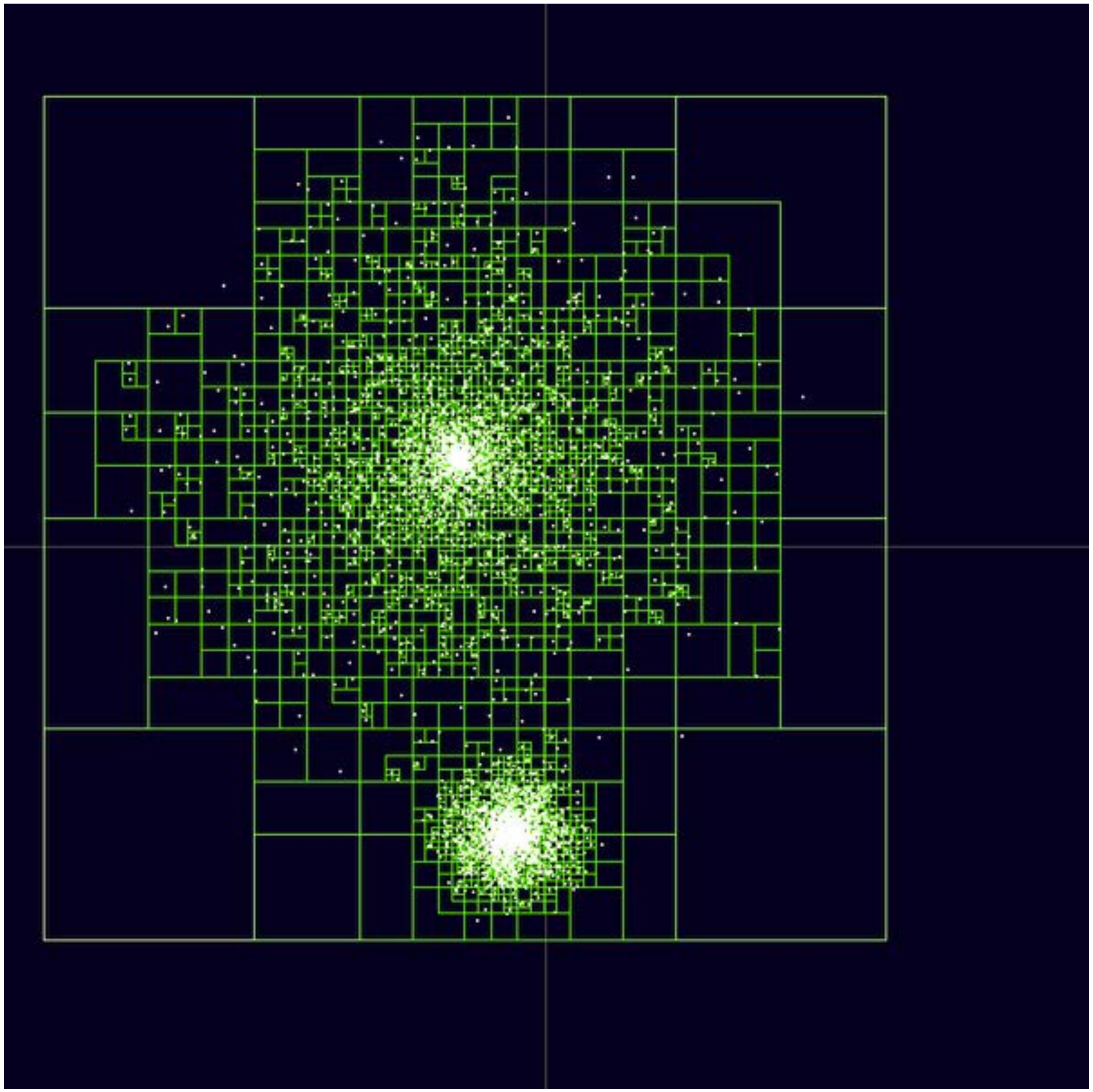
Barnes-Hut Example



Initial distribution of 5,000 bodies in 2 simulated galaxies

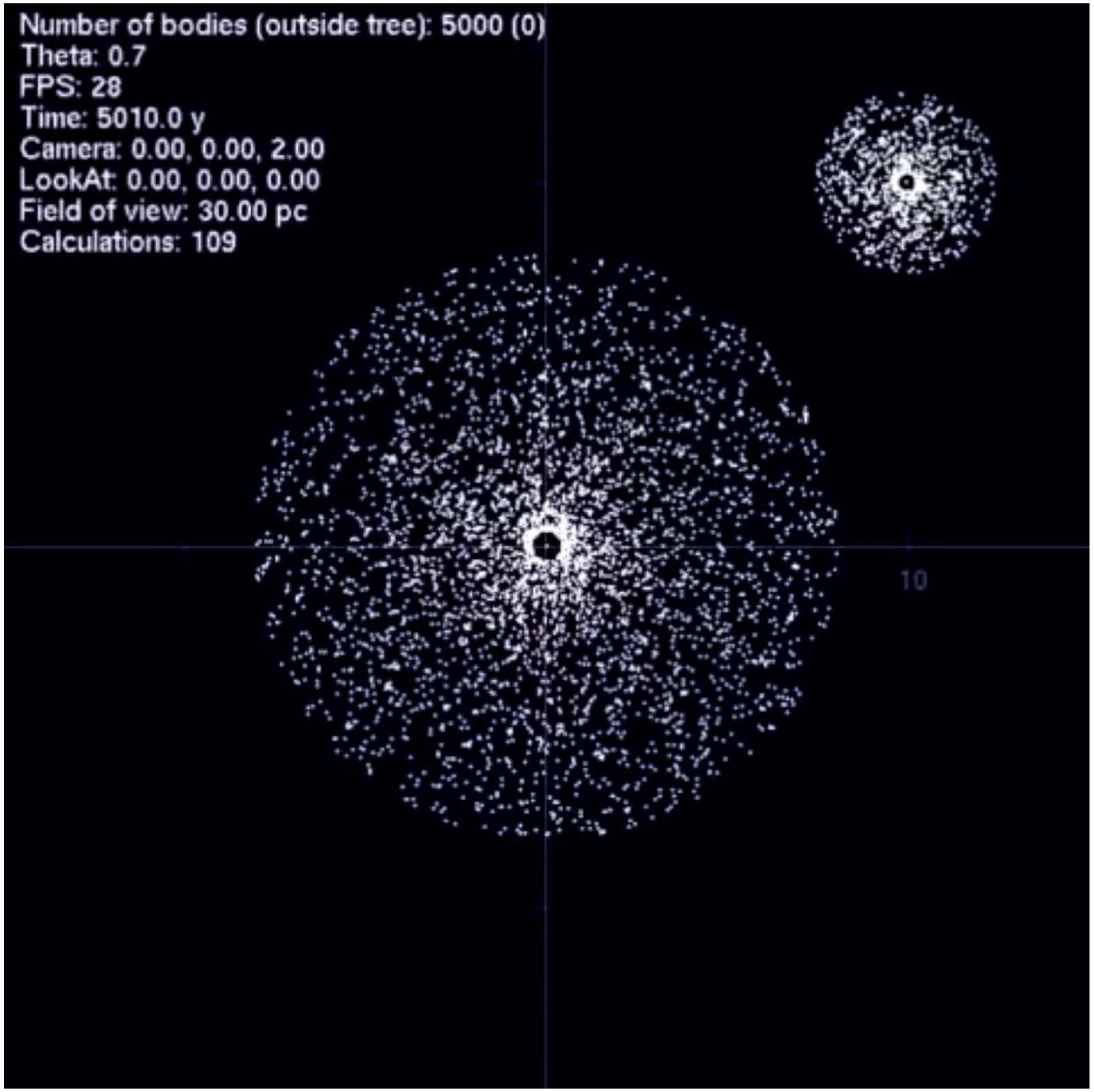
Source for this and other images and for video: Ingo Berg from Wikipedia. (http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation)

Barnes-Hut Full Partition



Shows full partition for 5,000 bodies, each in its own cell.
(Empty cells omitted.)

Colliding Galaxies Video



Source: http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation

A bit of history: HPC’s not really new!

- People have been developing and using “supercomputers” for a long time
- “Ancient” history: Supercomputers were very large monolithic computers
- Limited amounts of parallelism were incorporated in them.

IBM 7094
(c. mid-1960s)

CDC 7600 (c. 1970)
(36 MegaFlops Peak)

Cray 1 (c. 1976)
(250 MegaFlops Peak)

Your cell phone is surely much faster than these supercomputers:

Online reports claim 1.2 Gigaflops or more on an iPhone 7

Supercomputers Today

As of 2018:

- Today's supercomputers are highly parallel computers
- Most are networked "clusters" of many commodity processors
- Some use accelerators, such as special-purpose computers based on the graphics processing units (GPUs) designed for desktop video

Yale Omega Cluster (2009)
5632 cpus
57.8 Linpack TeraFlops

Sunway TaihuLight (2016)
10.6 million cores
93.0 Linpack PetaFlops
World's fastest (2016-2017)

ORNL Summit (2018)
4608 nodes
2.3 million cores
6 NVIDIA Volta GPUs/node
122.3 Linpack PetaFlops
World's fastest as of June 2018

Some reasons for *parallel* supercomputers

- Cost
  - Monolithic machines require huge investments by companies or by the government for use by a relative handful of consumers
  - Parallel machines can be built by connecting commodity parts (e.g., PCs or GPUs) whose cost is driven by huge standalone markets
- “Obvious” computational advantages
  - More processors ⇒ More *independent* computations per second
  - More memory ⇒ Less swapping & contention
  - More disks or other I/O devices ⇒ Faster aggregate I/O
- Good algorithmic fit to many problems
  - Many (most?) problems are “embarrassingly parallel” (e.g., Monte Carlo, parameter studies, neural nets, etc.)
  - “Divide-and-conquer”: often a useful approach that is naturally parallel
  - “Assembly Lines”: another naturally parallel way to solve problems

An even more important reason: Physics!

We've been living off of **Moore's Law** and **Dennard Scaling**.

What do these really say? What are the ramifications for HPC?

- Moore's Law ⇨ Transistors/chip double each 18-24 months at same cost
- Dennard Scaling ⇨ As transistors shrink, their power density stays constant
- Smaller transistors ⇨ Faster switching; higher clock speeds; constant power
- Nirvana!

- Higher power density ⇨ More power consumption per chip
- More power ⇨ More heat and higher temperature
- Higher temperature ⇨ Unreliability (And even melting!!! 😧)

This has led to a "power wall" limiting chip frequencies to ~3-4 GHz since 2006.

If we can't make individual processors faster simply by increasing clock speeds, how can we continue to increase performance in a given footprint?

Parallelism: To exploit increased transistor density (Moore's Law), the industry delivers many processors (cores) per chip, without increasing the clock speed.

So, how fast are today's supercomputers, anyway?

- In most cases, it depends on the application
- The standard comparison tool for technical computing is the "Linpack Benchmark" that looks at the time required to solve a set of linear equations:

  $$Ax = b$$

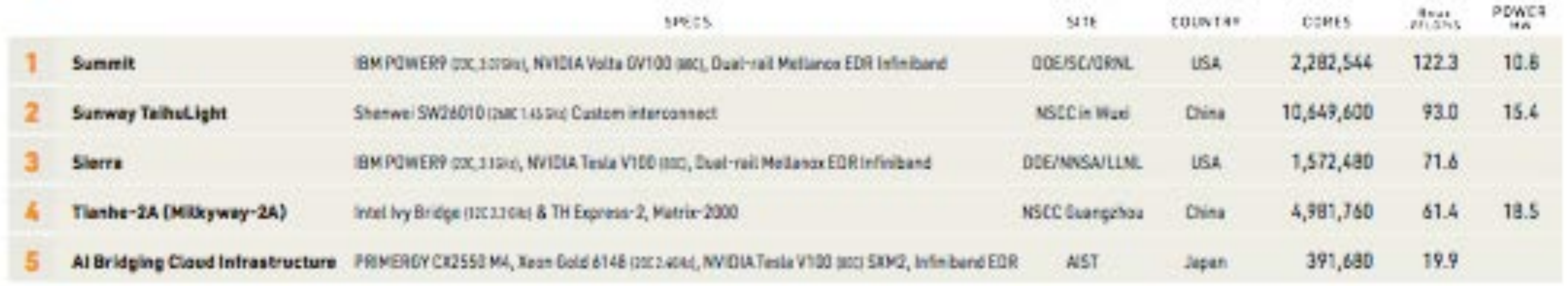
  for a random NxN matrix A and Nx1 vectors x and b. The benchmark score is the highest performance achieved for any value of N. (Often, the best N is the largest value for which the computation fits in memory on the machine.)
- **Top500 List:** (See top500.org.) Fastest 500 supercomputers ranked by Linpack Benchmark. Issued semiannually: Spring at ISC conf. in Europe; Fall at SC conf. in US. Now, there are also Green500 and Graph500 lists.
- Recent "World's Fastest Computers" on Top500 list:
  - 6/18-?: Summit (US, ORNL): 4608 nodes, 2.3 million cpus; 122.3 Linpack PFlops
  - 6/16-11/17: Sunway TaihuLight (China): 10.6 million cpus; 93.0 Linpack PFlops
  - 6/13-11/15: Tianhe-2 (China): 3.1 million cpus; 33.9 Linpack Petaflops
  - 11/12: Titan (USA, Cray XK7): 561 thousand cpus, 17.6 Linpack Petaflops

Most Recent Top500 List

TOP 500 The List.

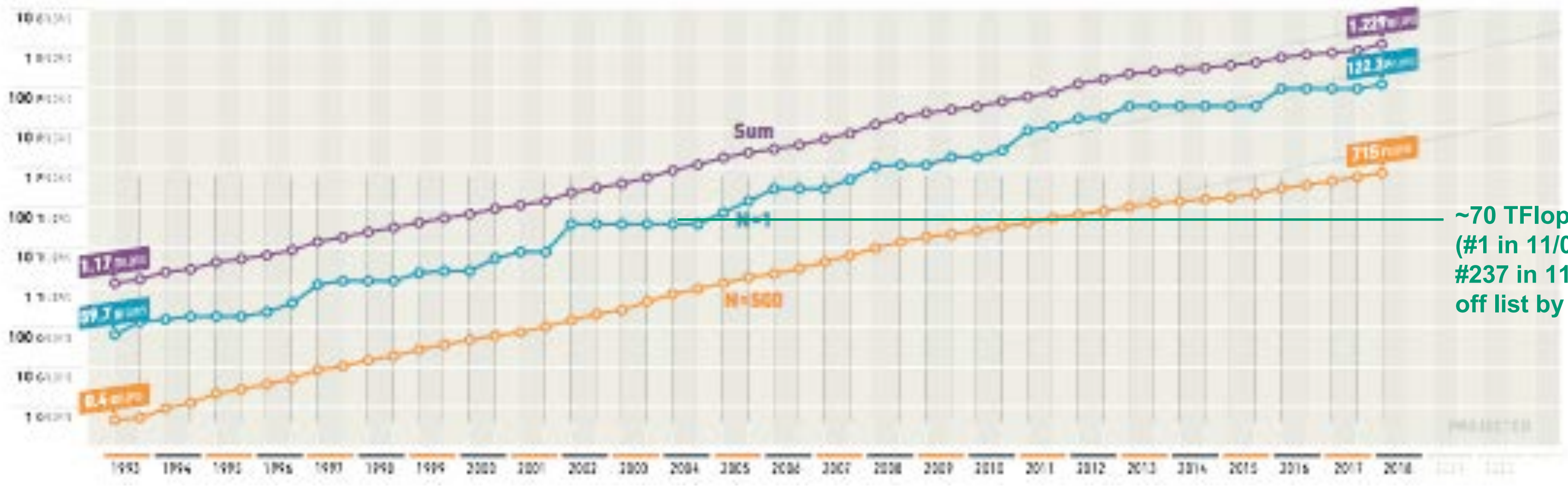
PRESENTED BY

FIND OUT MORE AT top500.org



		SPECS	SITE	COUNTRY	CORES	Rmax PFLOPS	POWER MW
1	**Summit**	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,282,544	122.3	10.6
2	**Sunway TaihuLight**	Shenwei SW26010 (260C 1.45 GHz) Custom interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
3	**Sierra**	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-rail Mellanox EDR Infiniband	DOE/NNSA/LLNL	USA	1,572,480	71.6	
4	**Tianhe-2A (Milkyway-2A)**	Intel Ivy Bridge (12C 2.2GHz) & TH Express-2, Matrix-2000	NSCC Guangzhou	China	4,981,760	61.4	18.5
5	**AI Bridging Cloud Infrastructure**	PRIMERGY CX2550 M4, Xeon Gold 6148 (20C 2.4GHz), NVIDIA Tesla V100 (80C) SXM2, Infiniband EDR	AIST	Japan	391,680	19.9	

PERFORMANCE DEVELOPMENT

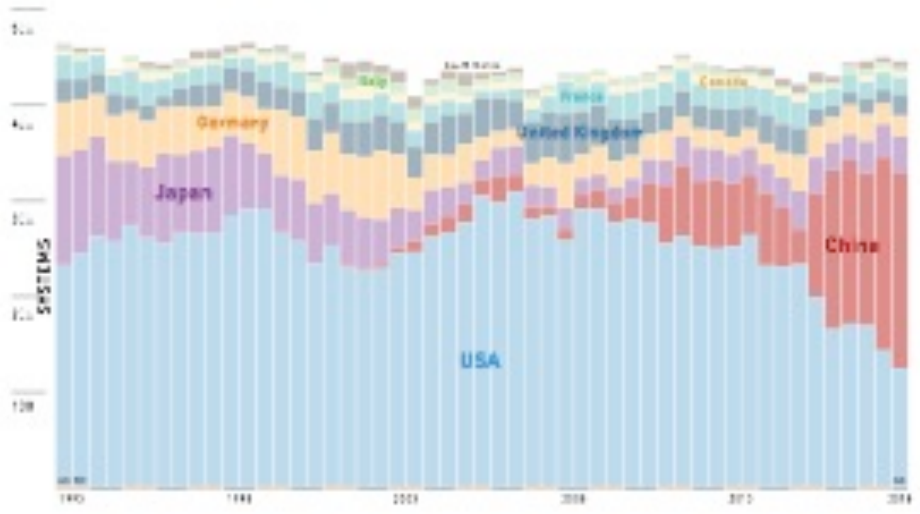


~70 TFlops
(#1 in 11/04;
#237 in 11/11;
off list by 11/12)

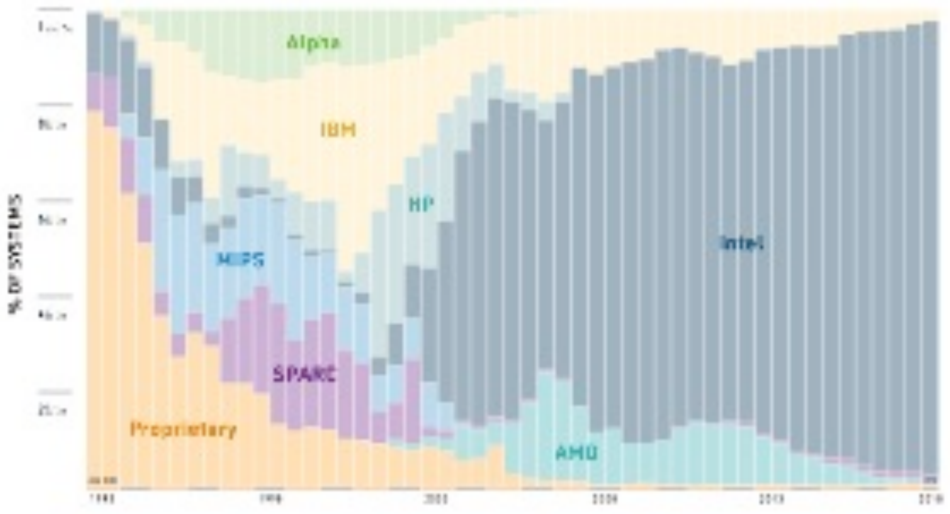
Source: www.top500.org

Top 500 Historical Performance Development

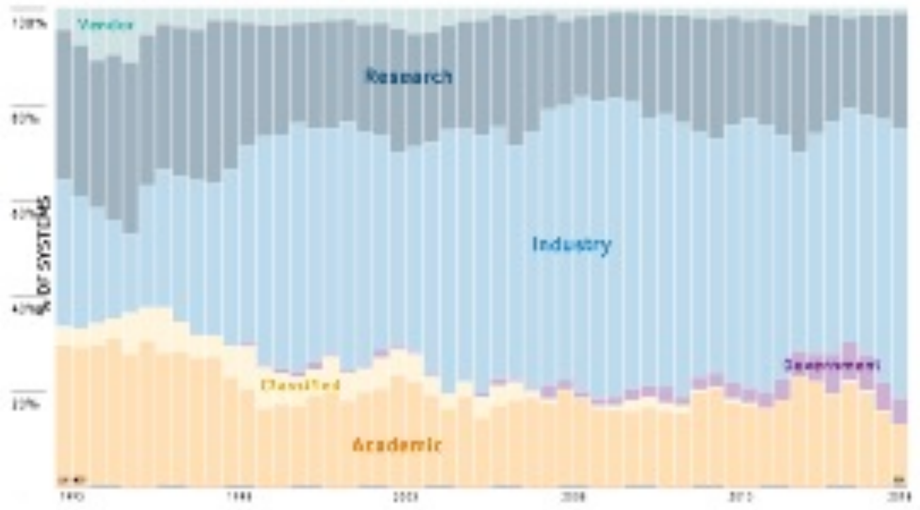
COUNTRIES



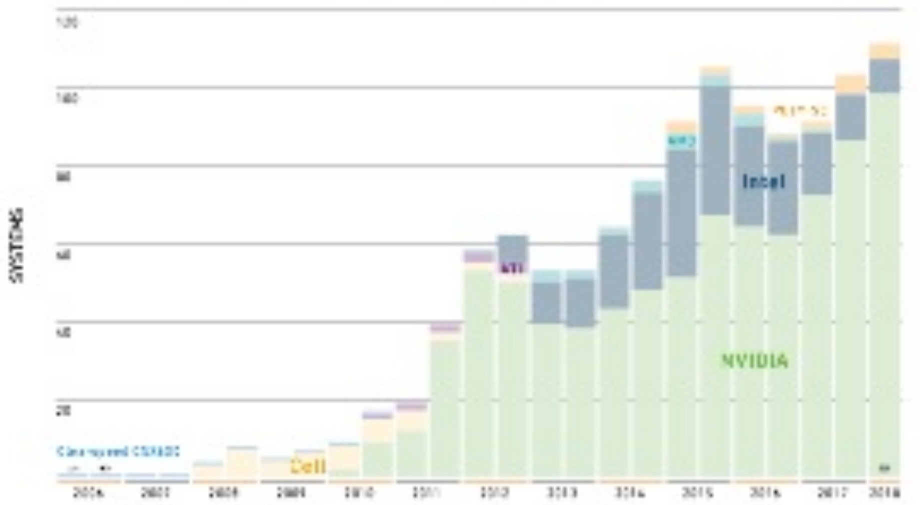
CHIP TECHNOLOGY



INSTALLATION TYPE



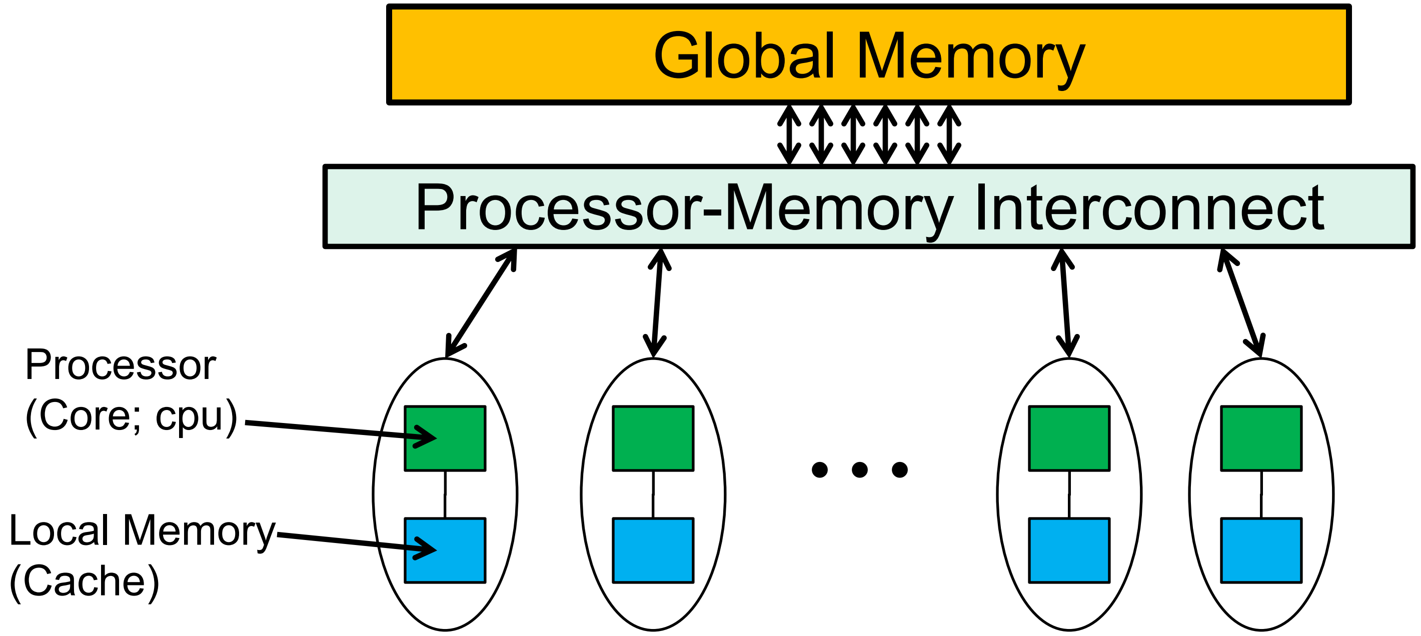
ACCELERATORS/CO-PROCESSORS



Source: www.top500.org

Types of Multiprocessor Parallel Computers - I

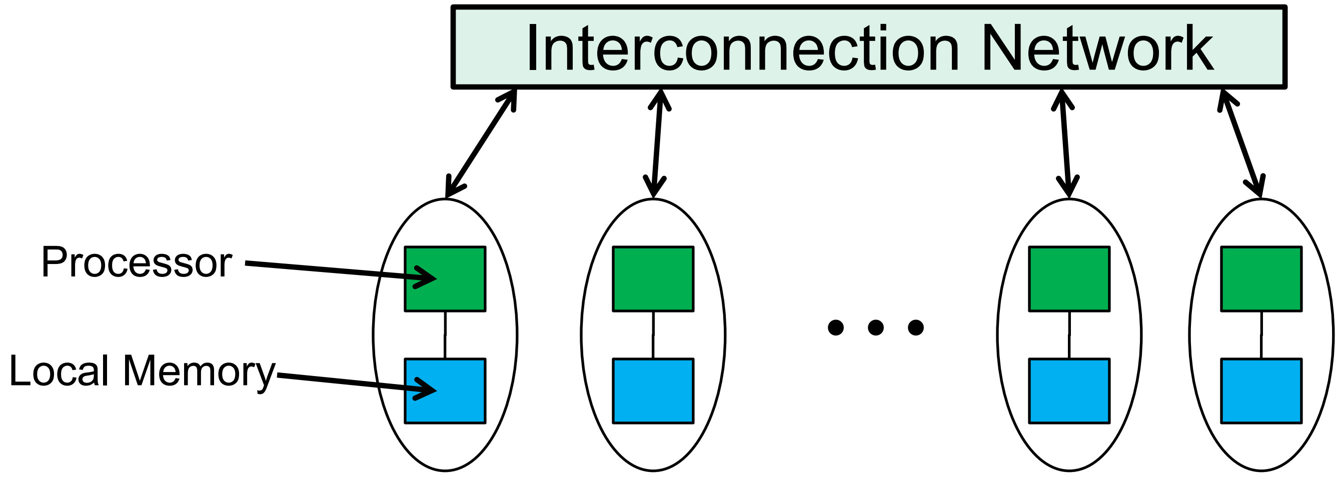
- **Shared Memory Multiprocessors (SMPs)**
  - Many independently programmable processors that access a global shared memory. Speed of access may vary based on data/proc. location



Examples: Individual multicore chips (with multi-level caches)
Separate chips (uniform or nonuniform memory access)
Your laptop, your TV, your cell phone, . . .

Types of Multiprocessor Parallel Computers - II

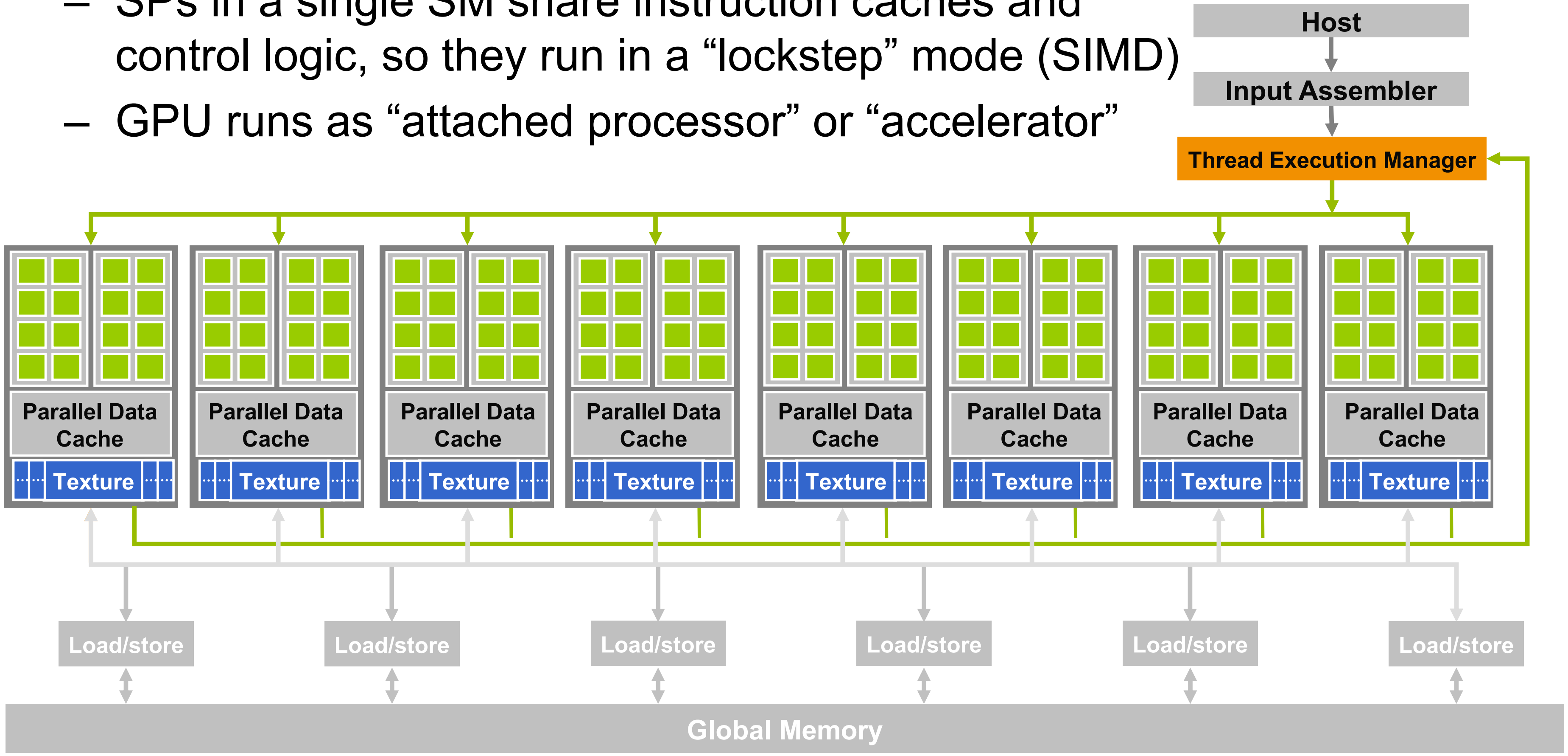
- **Distributed Memory Multicomputer**
  - Independent computers (“nodes”) connected through some sort of communication network with no global memory
  - Often, the nodes are SMPs



<u>Examples</u>: PCs/Workstations on a local-area network
Packaged “clusters” (such as Yale’s HPC clusters)

Types of Multiprocessor Parallel Computers - III

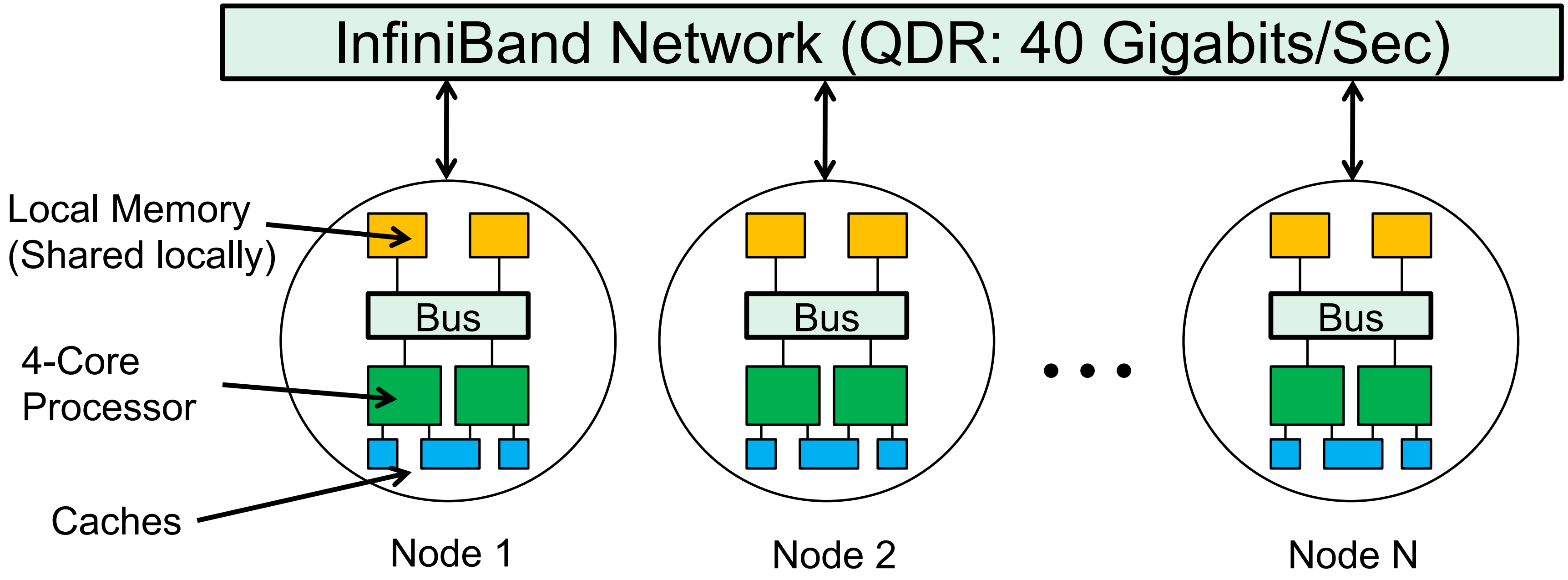
- **Many-Core Accelerators: Graphical Processing Unit (GPU)**
  - Huge number of heavily multithreaded "streaming processors" (SPs) organized into shared-memory "streaming multiprocessors" (SMs)
  - SPs in a single SM share instruction caches and control logic, so they run in a "lockstep" mode (SIMD)
  - GPU runs as "attached processor" or "accelerator"



Yale's HPC Clusters

- **Yale has 5 primary HPC clusters**
  - *Omega, Grace, Farnam, Ruddle, Milgram*
  - *Hybrid*: Independent shared-memory multicore compute nodes connected by a fast network. Some have GPUs on them.

Example: Omega (omega.hpc.yale.edu)


InfiniBand Network (QDR: 40 Gigabits/Sec)
Local Memory (Shared locally)
4-Core Processor
Caches
Bus
Bus
Bus
Node 1
Node 2
Node N

Course Content

- General Background (2-3 weeks overall)
  - Processor architecture (focusing on parallelism and memories)
  - Theoretical background on parallel computing
  - Debugging, benchmarking, performance measurement, tuning
- Programming Methodologies (2-3 weeks each)
  - Multi-threading (OpenMP, pthreads) for shared memory computers
  - Message-passing (MPI) for distributed memory clusters
  - Massive multi-threading (CUDA) for Nvidia GPUs
- Types of parallel computations (technical apps.) (2-3 weeks)
  - “Embarrassingly parallel”
  - Divide-and-conquer
  - Synchronous/data parallel
  - Linear algebra (e.g., matrix multiplication, solving linear equations)

Coursework and Assessment

- **Programming Assignments (~6)**: Roughly every 2+ weeks (55%).
- **Mid-term Exam**: In-class, probably Monday, October 15 (15%)
- **Final Exam or Final Project** (25%)
  - **Final Exam**: Friday, December 14, 2:00pm (All CPSC 424 students)
  - **Final Project:** Required for CPSC 524 students. Consult instructor re topic. Non-CS students are encouraged to select projects in their major fields.
    Due: Friday, December 14 by 5:30pm
- **Participation and instructor's judgment** (5%)
- **Class Schedule**: Monday & Wednesday, 9:00am – 10:15am, AKW 200
  - Class may meet during Reading Period (12/10 & 12/12), subject to change
- **Piazza**: Class site: https://piazza.com/yale/fall2018/cpsc42401cpsc52401/home
  - Sign up here: https://piazza.com/yale/fall2018/cpsc42401cpsc52401
  - To properly set privacy and other options for your Piazza account see: http://help.canvas.yale.edu/m/62761/l/964689-piazza-account-creation-and-modify-your-settings

Work Standards

Unless instructed otherwise, all submitted assignments must be your own <u>individual</u> work. Neither copied work nor work done in groups will be permitted or accepted without prior permission.

However….

You may discuss <u>questions about assignments and course material</u> with anyone. You may always consult with the instructor or TFs/ULAs on any course-related matter. You may seek <u>general advice and debugging assistance</u> from fellow students, the instructor, TFs/ULAs, Piazza conversations, and Internet sites.

<u>However, the work you submit must be entirely your own</u>. If you do benefit substantially from outside assistance, then you must acknowledge the source and nature of the assistance. (In rare instances, your grade for the work may be adjusted appropriately.) It is acceptable and encouraged to use outside resources to <u>*inform* your approach</u> to a problem, but <u>*plagiarism is unacceptable*</u> in any form and under any circumstances. If discovered, plagiarism will lead to adverse consequences for all involved.

Contact Information

<u>Instructor</u>

Andrew Sherman

160 St. Ronan Street

andrew.sherman@yale.edu

203-436-9171

Office Hours TBD

<u>Student Assistants</u>

TBD

TBD

TBD

Course Resources - I

- Primary Text:
  - *Introduction to High Performance Computing for Scientists and Engineers*, by Georg Hager & Gerhard Wellein (CRC Press 2011, ISBN 978-1-4398-1192-4; ~$60; Online: http://www.crcnetbase.com/isbn/9781439811931)
- Supplementary Texts
  - *Programming Massively Parallel Processors, 2nd or 3rd Ed.*, by David B. Kirk and Wen-mei W. Hwu (Morgan Kaufmann 2013/2016, ISBN 978-0-12-415992-1/978-0-12-811986-0; ~$56; Online: http://proquest.safaribooksonline.com/9780124159921)
  - *CUDA Documentation*, online at the CUDA Zone on the Nvidia website (https://developer.nvidia.com/category/zone/cuda-zone)
  - *High Performance Computing: Modern Systems and Practices*, by Thomas Sterling, Matthew Anderson, and Maciej Brodowicz (Morgan Kaufmann 2018, ISBN 978-0-12-420158-3); ~$90; Online: https://www.sciencedirect.com/book/9780124201583/high-performance-computing.

Course Resources - II

- Other Material:
  - *Using MPI: Portable Parallel Programming with the Message-Passing Interface, 2nd Edition*, by William Gropp, Ewing Lusk, and Anthony Skjellum. (MIT Press, 1999) (Ebook available with Yale NetID login at the following site: http://site.ebrary.com/lib/yale/docDetail.action?docID=10225306)
  - *An Introduction to Parallel Programming,* by Peter S. Pacheco (Morgan Kaufmann 2011, ISBN 978-0-12-374260-5; ~$60; Online: http://proquest.safaribooksonline.com/9780123742605)
  - *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, 2nd Edition*, by Barry Wilkinson and Michael Allen. (Pearson Education, Inc., 2005).

IMPORTANT NOTE: some of the course slides and other materials are based on materials provided by the authors of the listed texts and references, or by other HPC experts.

Yale HPC Resources

- HPC/Research Computing Website:
  - https://research.computing.yale.edu
- Getting Started Page
  - https://research.computing.yale.edu/support/hpc/getting-started
- Problems using the clusters (e.g., accounts, node failures, etc.)
  - Email to hpc@yale.edu
  - Not for class-specific questions!
- Information about Omega (cluster used for this class)
  - https://research.computing.yale.edu/support/hpc/clusters/omega