



High Performance/Parallel Computing

**Andrew Sherman
Senior Research Scientist in Computer Science**

**Yale Center for Research Computing
Department of Computer Science**

**National Nuclear Physics Summer School
June 25, 2018**

What is High Performance Computing (HPC)?

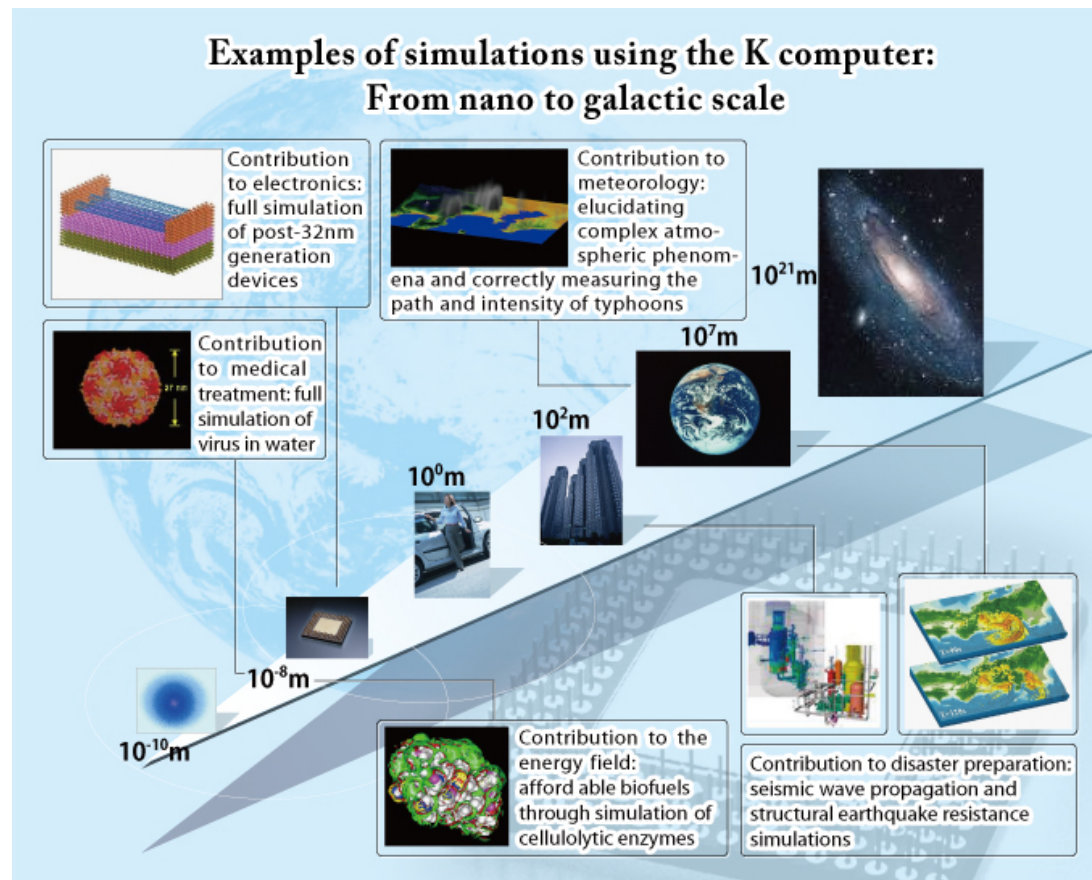
Using today's fastest computers ("supercomputers") to solve technical computing problems (mostly in science and engineering). Often computations involve parallel computing.

Why is HPC interesting to scientists and engineers?

- 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 amount of time
 - Might lead to better models that are more accurate and realistic

Why should you care about HPC?

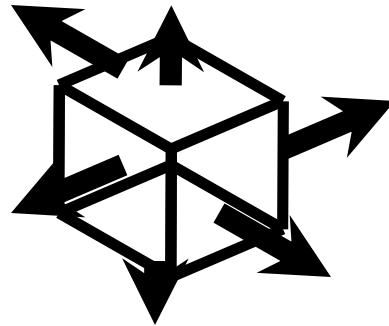
- **Research:** Broad range of research in science, engineering, and other fields
- **Applications:** Important “real-world” applications: weather, data analysis, AI, personalized medicine, machine learning ...



Riken AICS, 2011

Familiar Example: Weather Forecasting

Atmosphere modeled by dividing it into 3-dimensional cells.



Temperature,
pressure,
composition, etc.



Calculations for each cell repeated many times to model passage of time.

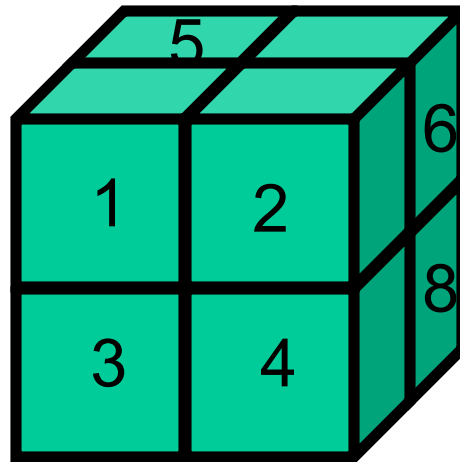
Why is Global Weather Forecasting Challenging?

- 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 each cell update uses ~ 200 arithmetic operations.
 \Rightarrow For one time step, $\sim 2.5 \times 10^{13}$ arithmetic operations are needed.
- To forecast the weather for 7 days using 1-minute intervals to track changes, a computer operating at 20 Gigafllops (2×10^{10} arithmetic operations/sec) on average would take $\sim 1.25 \times 10^7$ 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

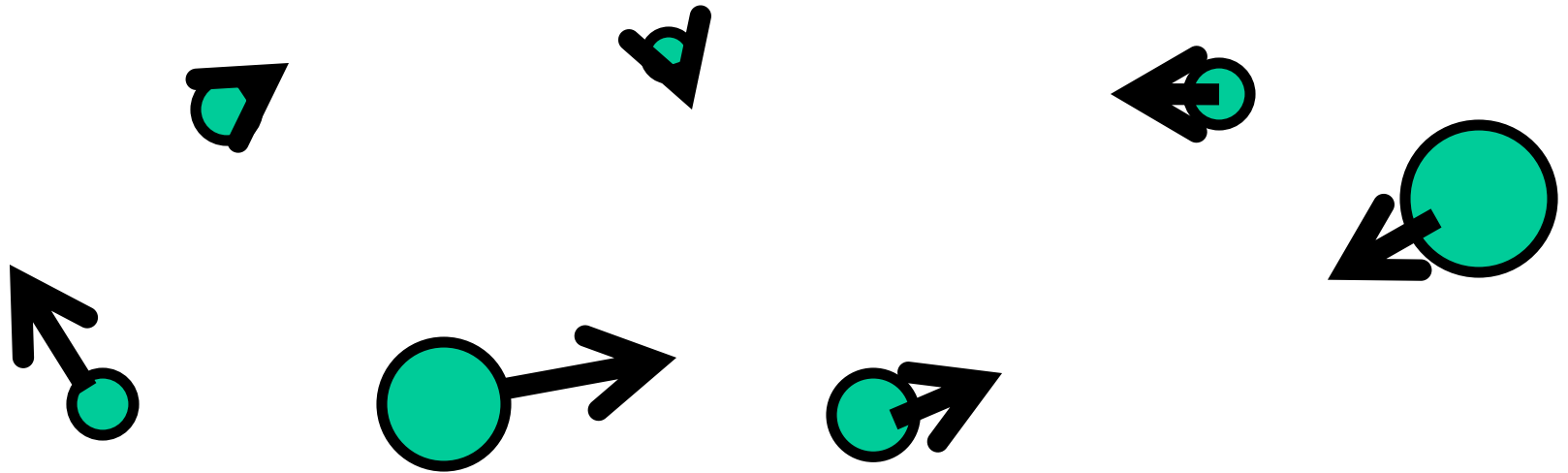
How can this sort of performance be achieved?

- 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

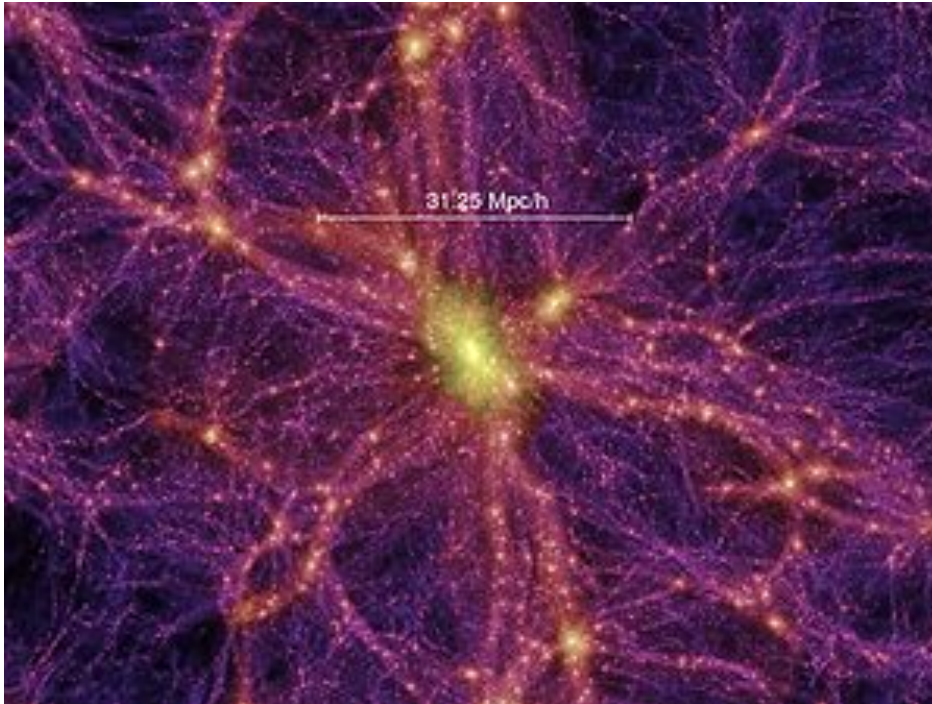


Each body is affected by each other body through forces. Movement of each body in a short time period (a “time step”) is predicted by evaluating the total instantaneous forces on each body, calculating body velocities, and moving the bodies through the time step. Many time steps are required.

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

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

For each time step:

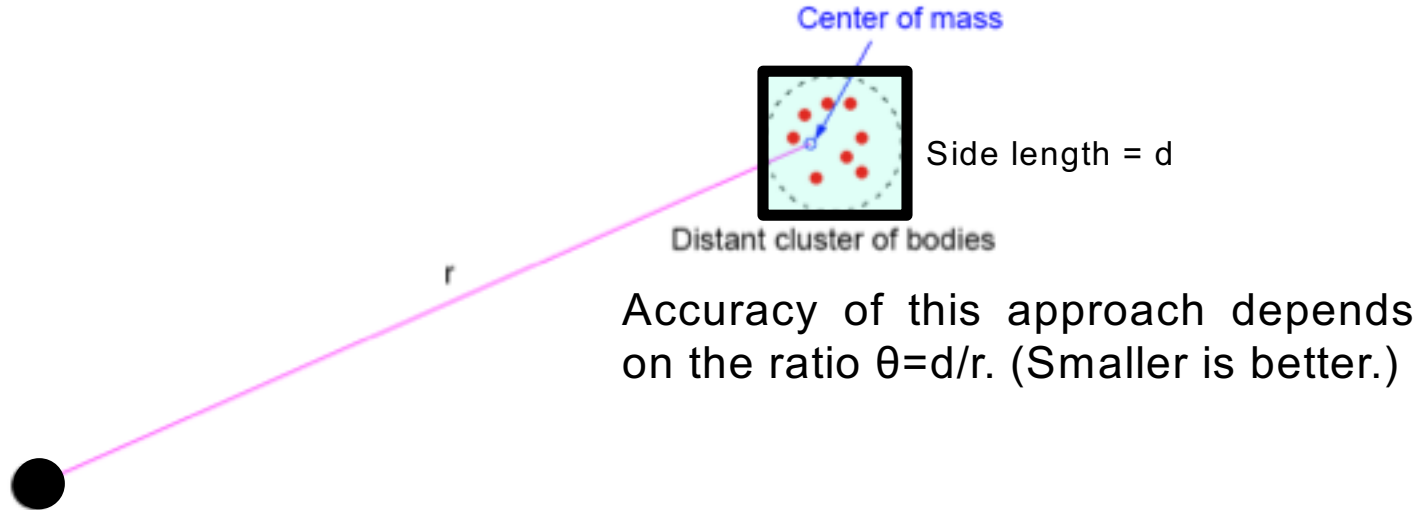
- Calculate the forces:
 - “**Brute Force**” **Algorithm**: With N bodies, $N-1$ forces to calculate for each body, or approx. $O(N^2)$ calculations (50% reduction for symmetry)
- Move the bodies to new locations
- Repeat

Challenges in Modeling Many-Body Motion

- A galaxy might have $\sim 10^{11}$ stars. So one time step would require:
 - $\sim 5 \times 10^{21}$ force calculations using “brute force”
- Suppose that, using 1 computer, each force calculation takes $0.1 \mu\text{sec}$ (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

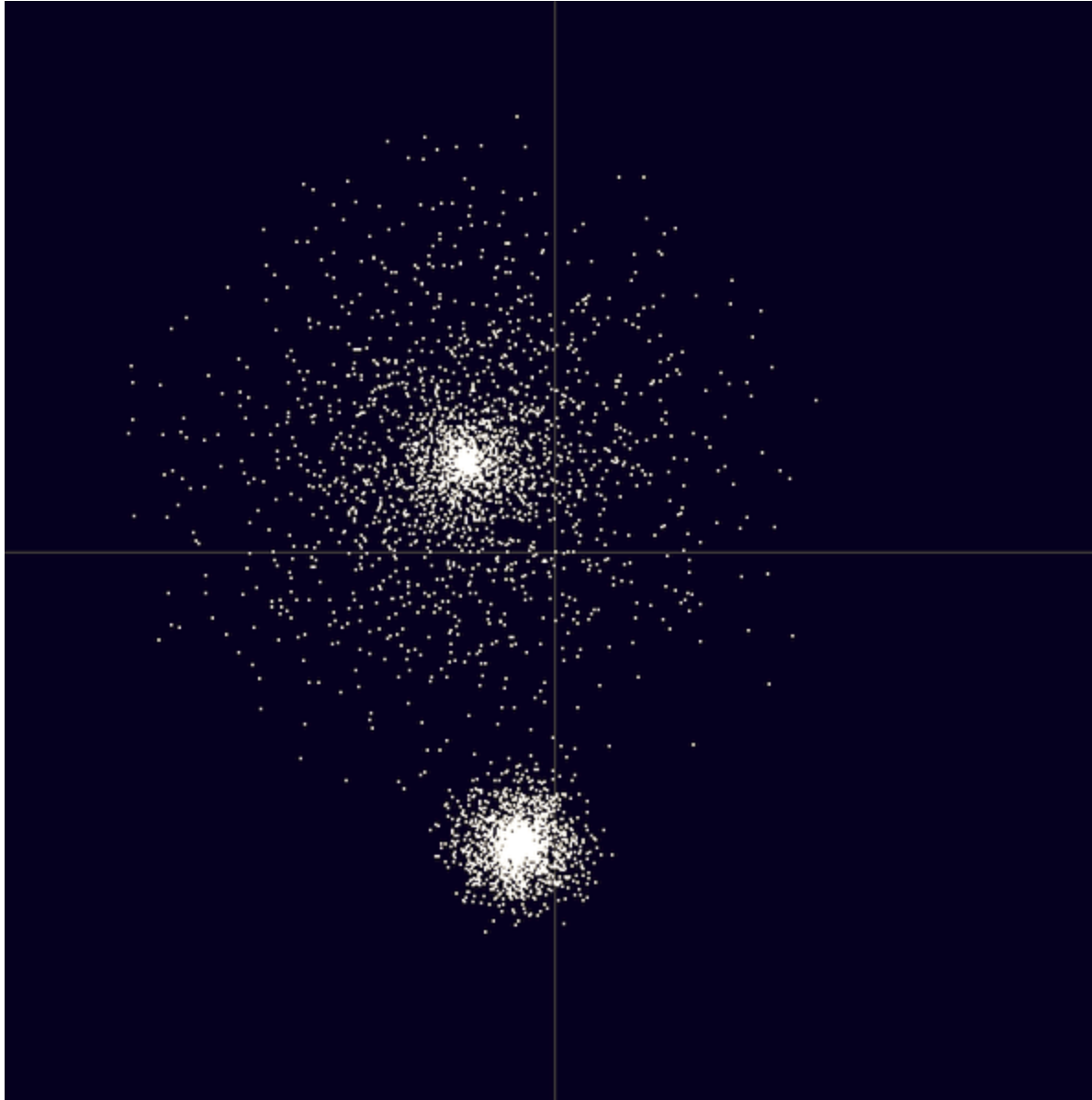
Approximate the effect of a cluster of distant bodies by treating them as a single distant body with mass located at the center of mass of the cluster:



This idea leads to $O(N \log_2 N)$ algorithms for N-Body problems. The approach has been “discovered” many times, 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

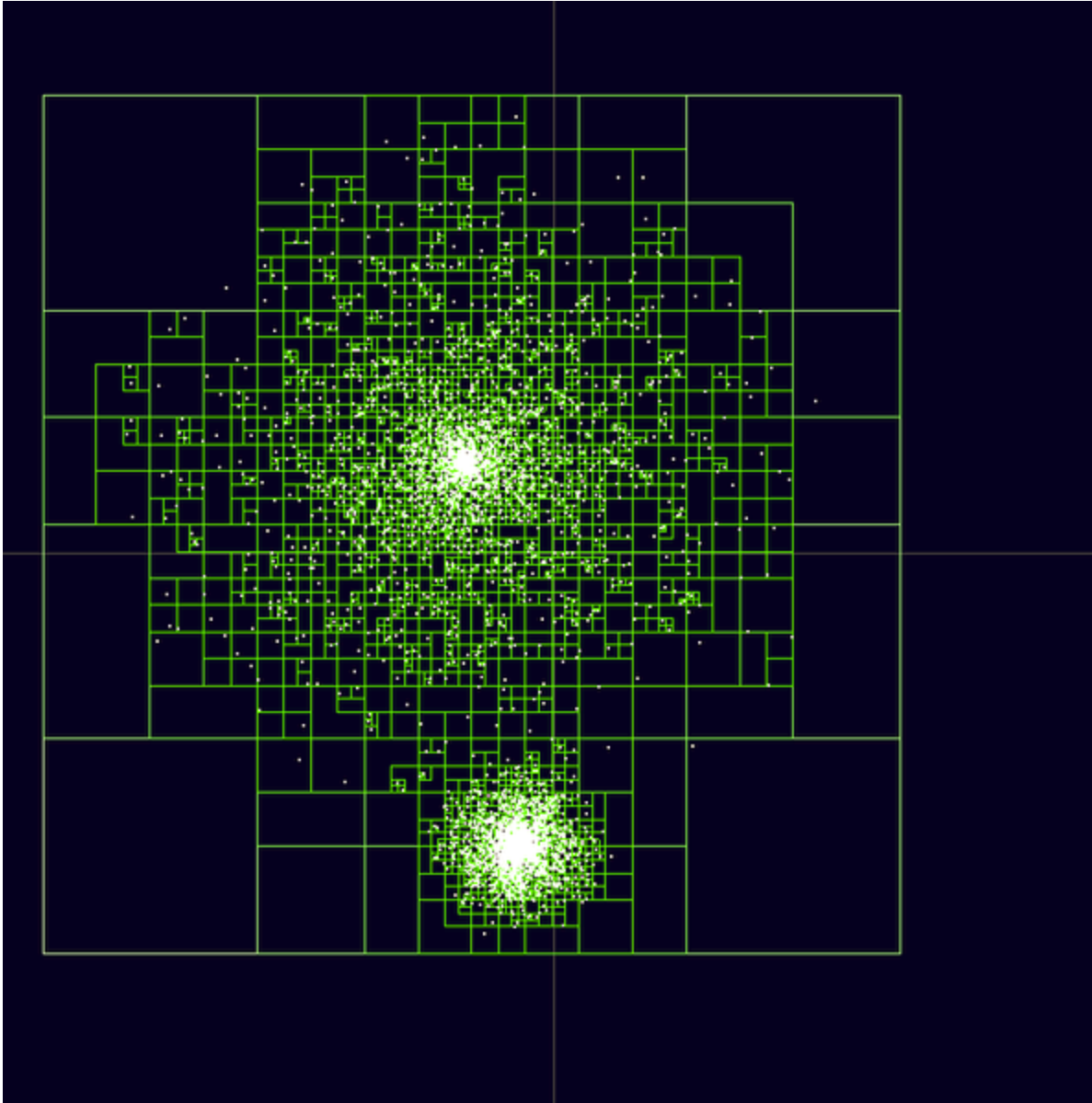


Initial distribution of
5,000 bodies in 2
simulated galaxies

Source for this and
other images and for
video: Ingo Berg from
Wikipedia.

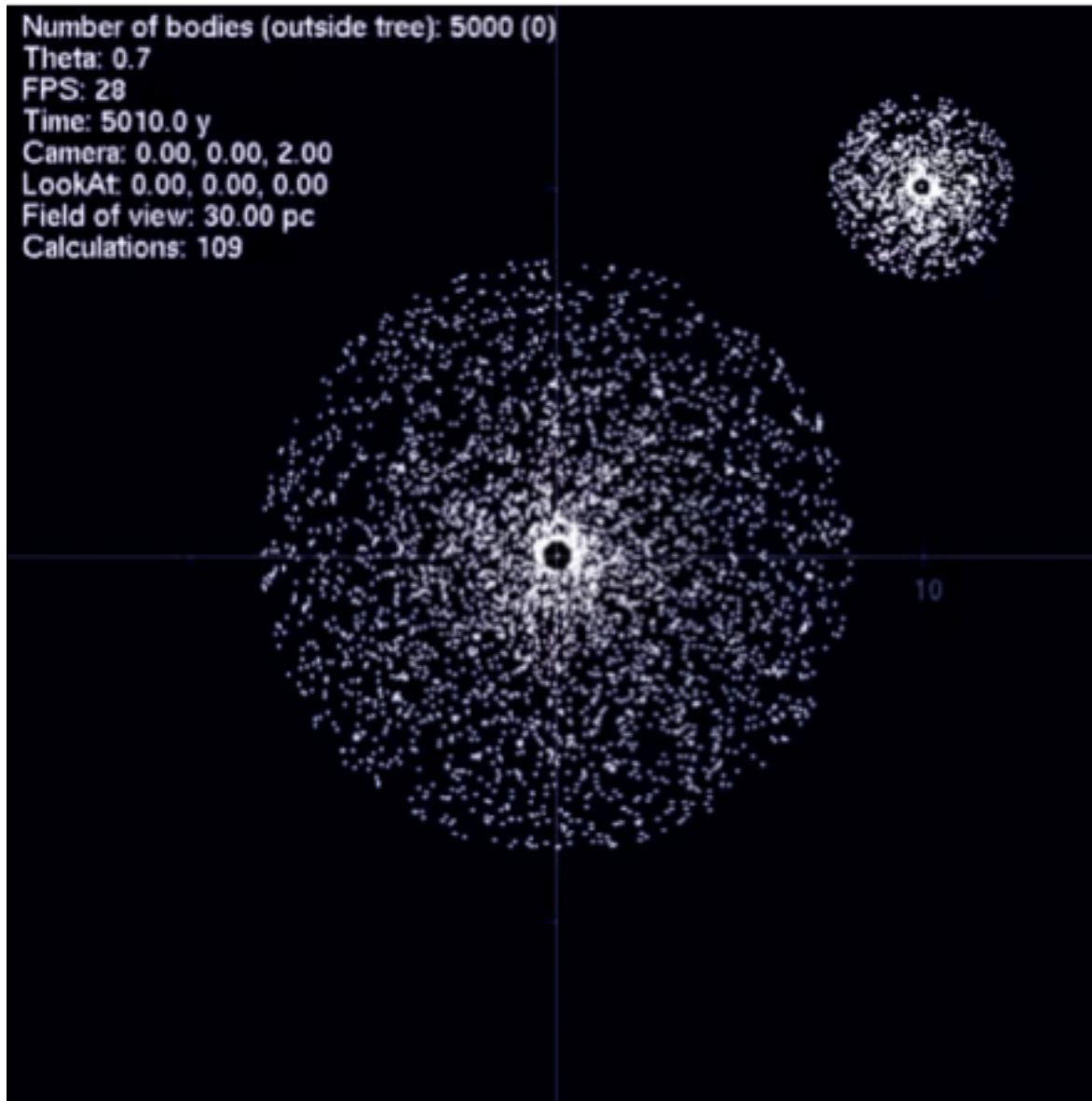
(http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation)

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 \Rightarrow More *independent* computations per second
 - More memory \Rightarrow Less swapping & contention
 - More disks or other I/O devices \Rightarrow Faster aggregate I/O
- Good algorithmic fit to many problems
 - Many (most?) problems are “embarrassingly parallel” (e.g., Monte Carlo, parameter studies, 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

This has led to a “power wall” limiting chip frequencies to ~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 $N \times N$ matrix A and $N \times 1$ 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 www.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

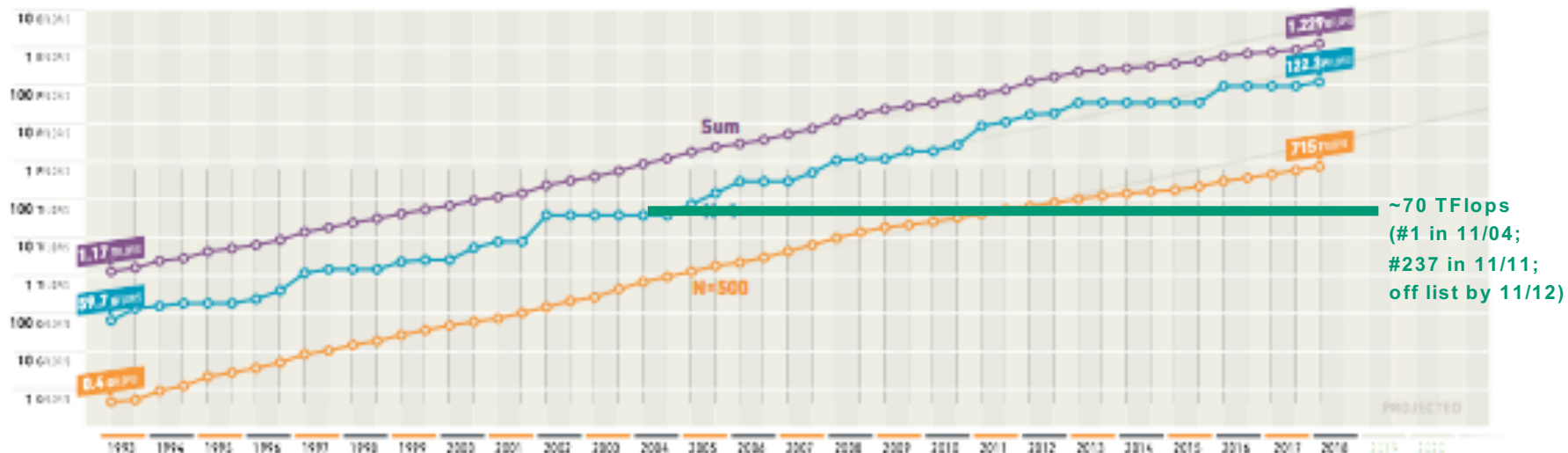


FIND OUT MORE AT
top500.org



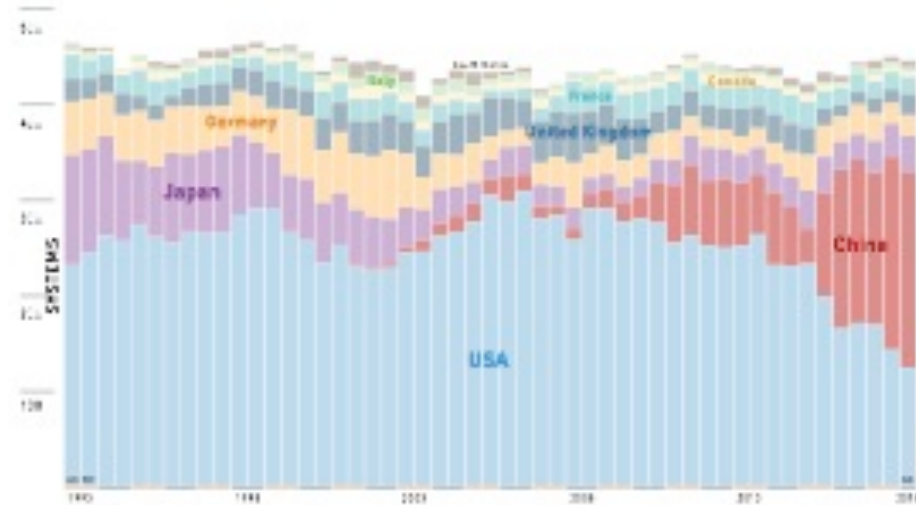
		SPEC5	SITE	COUNTRY	CORES	Peak FLOPS	POWER kW
1	Summit	IBM POWER9 (20C 1.078s), NVIDIA Volta GV100 (80C), Dual-rail Mellanox EDR Infiniband	ODE/SC/ORNL	USA	2,282,544	122.3	10.8
2	Sunway TaihuLight	Shenwei SW26010 (268C 1.458s) Custom Interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
3	Sierra	IBM POWER9 (20C 1.12s), NVIDIA Tesla V100 (80C), Dual-rail Mellanox EDR Infiniband	ODE/MNSA/LLNL	USA	1,572,480	71.6	
4	Tianhe-2A (Milkyway-2A)	Intel Ivy Bridge (12C 2.10s) & 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 2.40s), NVIDIA Tesla V100 (80C) SRM2, Infiniband EDR	AIST	Japan	391,680	19.9	

PERFORMANCE DEVELOPMENT

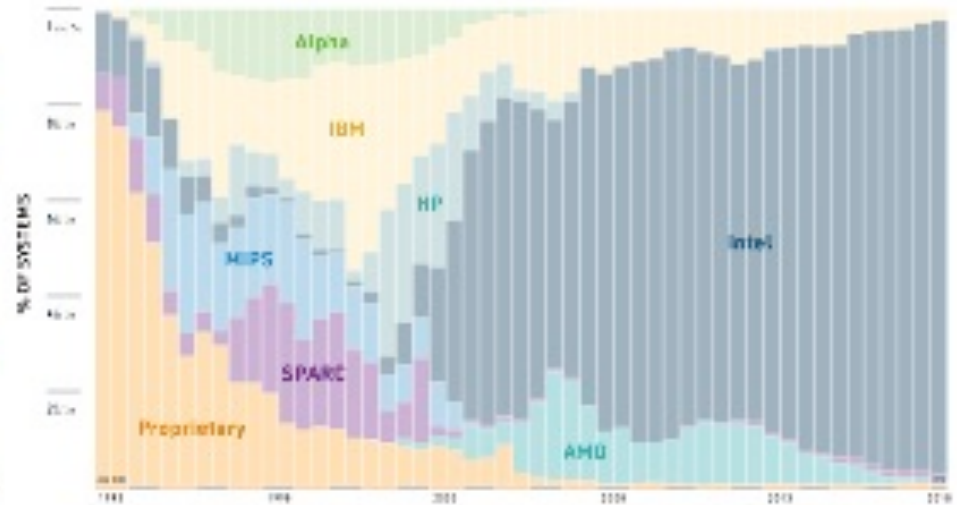


Top 500 Historical Performance Development

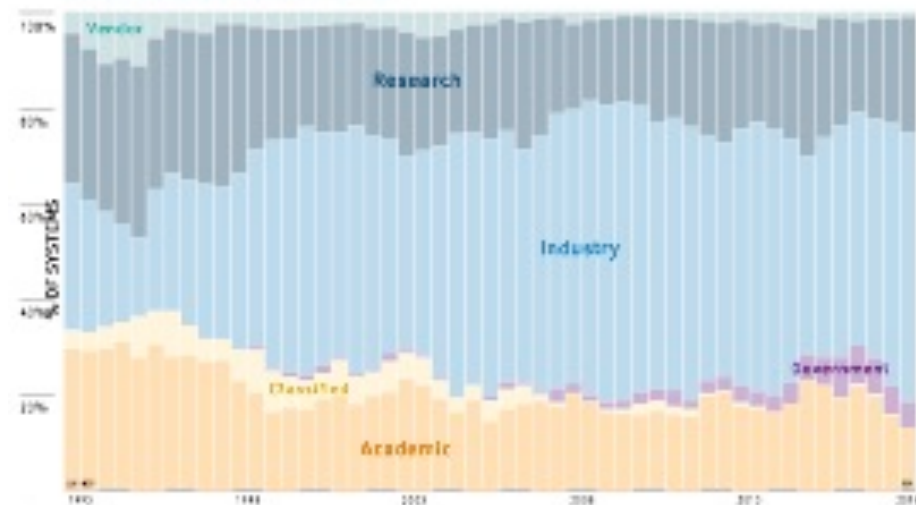
COUNTRIES



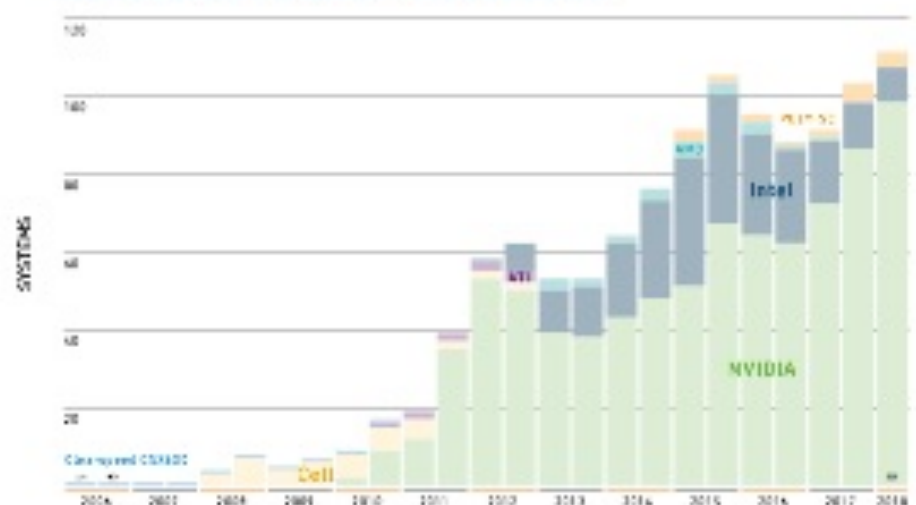
CHIP TECHNOLOGY



INSTALLATION TYPE



ACCELERATORS/CO-PROCESSORS



Getting Started on Grace

- MacOS or Linux:
 - research.computing.yale.edu/support/hpc/user-guide/connect-macos-and-linux
- Windows:
 - research.computing.yale.edu/support/hpc/user-guide/connect-windows
- Steps:
 1. Install software (if needed)
 2. Create an ssh keypair and upload it to:
gold.hpc.yale.internal/cgi-bin/sshkeys.py
 3. `ssh netid@grace.hpc.yale.edu`

Getting Started with the MPI Exercise

- `rsync -a ~ahs3/exercise .`
- `ls exercise`

This should produce output similar to:

```
build-run-mpi.sh  Makefile  rwork.o  task.c
```

MPI “Hello World” Program – Initialization Section

```
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <unistd.h>
#include "mpi.h"

main(int argc, char **argv ) {
    char message[100];
    int i,rank, size, type=99;
    int worktime, sparm, rwork(int,int);
    double wct0, wct1, total_time, cput;

    MPI_Status status;

    MPI_Init(&argc,&argv); // Required MPI initialization call

    MPI_Comm_size(MPI_COMM_WORLD,&size); // Get no. of processes
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Which process am I?
```


MPI “Hello World” Program – Master Section

```
/* If I am the master (rank 0) ... */
if (rank == 0) {
    sparm = rwork(0,0); //initialize the workers' work times
    sprintf(message, "Hello, from process %d.",rank); // Create message
    MPI_Barrier(MPI_COMM_WORLD); //wait for everyone to be ready
    wct0 = MPI_Wtime(); // set the start time; then broadcast data
    MPI_Bcast(message, strlen(message)+1, MPI_CHAR, 0, MPI_COMM_WORLD);
    MPI_Bcast(&sparm, 1, MPI_INT, 0, MPI_COMM_WORLD);

    /* Receive messages from the workers */
    for (i=1; i<size; i++) {
        MPI_Recv(message, 100, MPI_CHAR, i, type, MPI_COMM_WORLD, &status);
        sleep(3); // Proxy for master's postprocessing of received data.
        printf("Message from process %d: %s\n", status.MPI_SOURCE,message);
    }

    wct1 = MPI_Wtime(); // set the end time
    total_time = wct1 - wct0; // Get total elapsed time
    printf("Message printed by master: Total elapsed time is %f seconds.\n",
        total_time);
}
```

MPI “Hello World” Program – Worker Section

```
/* Otherwise, if I am a worker ... */
else {

    MPI_Barrier(MPI_COMM_WORLD); //wait for everyone to be ready

    /* Receive initial data from the master */
    MPI_Bcast(message, 100, MPI_CHAR, 0, MPI_COMM_WORLD);
    MPI_Bcast(&sparm, 1, MPI_INT, 0, MPI_COMM_WORLD);

    worktime = rwork(rank, sparm); // Simulate some work

    /* Create and send return message */
    sprintf(message, "Hello from process %d after working for %d seconds.",
            rank, worktime);

    MPI_Send(message, strlen(message)+1, MPI_CHAR, 0, type, MPI_COMM_WORLD);
}

MPI_Finalize(); // Required MPI termination call
}
```

MPI “Hello World” Program – Slurm Script I

```
#!/bin/bash

# THIS SECTION CONTAINS INSTRUCTIONS TO SLURM

#SBATCH --partition=nnpss
#SBATCH --ntasks=4 # Set number of MPI processes
#SBATCH --ntasks-per-node=2 --ntasks-per-socket=1 # Set procs per socket/node
#SBATCH --cpus-per-task=1 # set number of cpus per MPI process
#SBATCH --mem-per-cpu=6100mb # set memory per cpu
#SBATCH --job-name=HELLO_WORLD
#SBATCH --time=5:00

# THIS SECTION MANAGES THE LINUX ENVIRONMENT

# The module load command sets up the Linux environment to use
# specific versions of the Intel compiler suite and OpenMPI.
module load Langs/Intel/15 MPI/OpenMPI/2.1.1-intel15

# echo some environment variables
echo $SLURM_JOB_NODELIST
```

MPI “Hello World” Program – Slurm Script II

```
# THIS SECTION BUILDS THE PROGRAM
```

```
# Do a clean build
```

```
make clean
```

```
# My MPI program is named task
```

```
make task
```

```
# THIS SECTION RUNS THE PROGRAM
```

```
# Run the program several times using 2 nodes with 1 MPI process per socket.
```

```
# The run time for the runs may differ due to the built-in randomization.
```

```
mpirun -n 4 --map-by socket -display-map ./task
```

```
mpirun -n 4 --map-by socket -display-map ./task
```

```
mpirun -n 4 --map-by socket -display-map ./task
```

```
mpirun -n 4 --map-by socket -display-map ./task
```