

CSI 702

High Performance Computing

Dr. John Wallin

ST I, Room 109

703-993-3617

jwallin@gmu.edu

<http://www.scs.gmu.edu/~jwallin/c702s06>

Dr. Brett Berlin

My Interests

- observations and simulations of colliding galaxies
- numerical methods
- high velocity impacts
- high performance computing

Prerequisites

Fluency with one of the following computer languages: FORTRAN, FORTRAN90, C, or C++ (CSI 603, 604 or equivalent), Fluency with the Unix operating system (CSI 601,602 or equivalent). CSI 700 (Numerical Methods) and CSI 701 (Foundations of Computational Science) OR Permission of Instructor.

Translation:

All homework will be written in C, C++, Fortran 90, or Fortran. You are also expected to know how to use matlab and the basic numerical methods taught in CSI 700 and the techniques, methods, and standards taught in CSI 701. Your codes must also compile and work on the Linux machines in the SCS lab and on the other machines selected by the instructors.

A Mini-Quiz

1. How do you create, change, or move directories?
2. How do you delete, rename, or move a file?
3. How do you use tar and gzip to compress and backup a directory?
4. What file and directory permissions are required to set up a website on your account?
5. In which directory do you normally place your web-pages?
6. How do you modify your path?
7. How do you find the location of a binary that you wish to execute?

8. Have you Used Matlab to simulate real-world problems, such as a traveling salesman problem?
9. Have you ever used regular expressions to do searches?

A Word About Textbooks

Steve McConnell's book *Code Complete* is required for this class. It describes how to write high quality software. The emphasis is on software construction, that is, writing readable and maintainable code. This is an excellent reference for both experienced and inexperienced programmers.

Heath's book *Scientific Computing: An Introductory Survey* is required for the class and is also used in CSI 700 and CSI 701. In this class, we cover the last sections of the book, including ODE's, PDE's, FFT's. We also review selected other sections. This book is a good overview of numerical methods, focusing on algorithm rather than formal proofs.

Why Do Scientist Use Computers

- experiments are impossible
- experiments are too expensive
- equations too difficult to be solved analytically
- experiments don't provide enough insight or accuracy
- data sets too complex to be analyzed by hand

Computers bridge the gap between experiments and theory

The Atanasoff- Berry Computer

The earliest electronic digital computer was built in the basement of the Physics Department at Iowa State University by Atanasoff and Berry in 1937- 1942.

It was a special purpose machine that was used to solve a 27x27 element linear equation. Even though its programming was limited to a single task, it contained all the elements (storage, digital logic, base-2) of modern machines.

Although this seems like a trivial problem now, solving this type of matrix problem is extremely difficult without a computer.

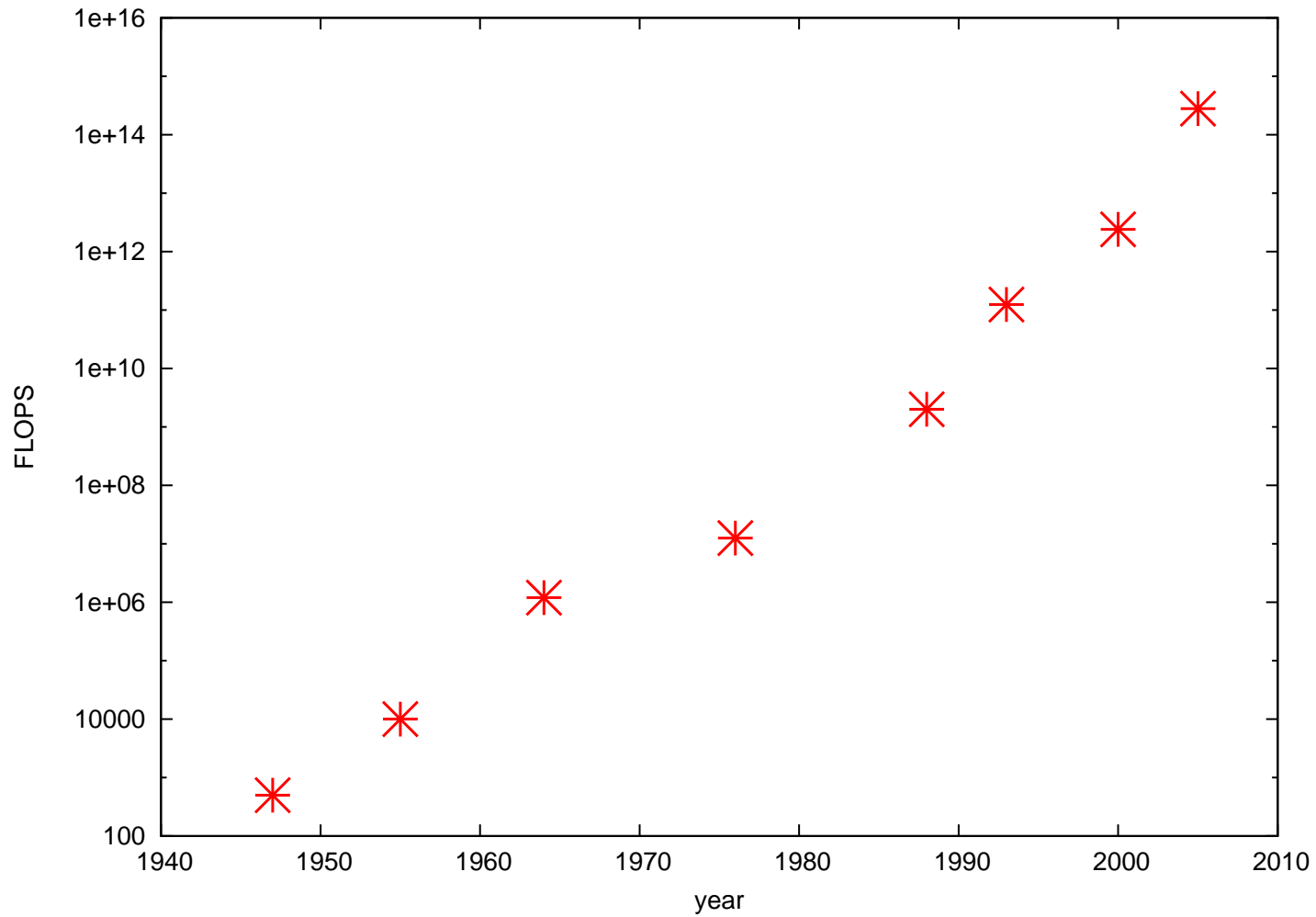
Supercomputer Speeds

(Taken from

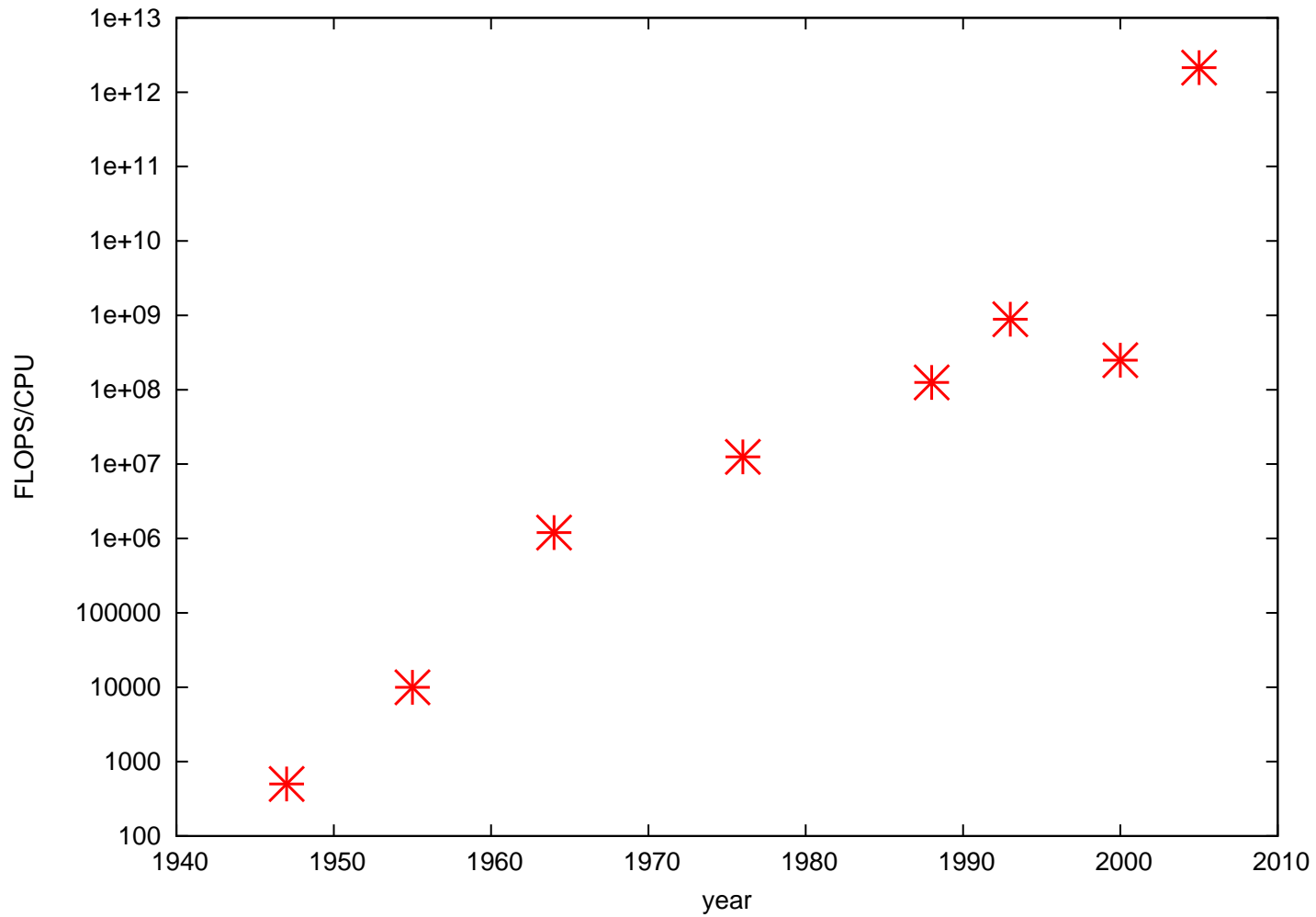
<http://home.earthlink.net/~mrob/pub/computer-history.html>
and <http://www.top500.org/main/archive.php>)

Year	Computer	speed	CPU
1947	Eniac	500 FLOPS	1
1955	IBM 704	10 kFLOPS	1
1964	CDC 6600	1.2 MFLOPS	1
1976	Cray 1	12.5 MFLOPS	1
1988	Cray Y-MP	2 GFLOPS	16
1993	Fujitsu Numerical Wind Tunnel	124.5 GFLOP	140
2000	ASCI Red (Sandia)	2.4 TFLOP	9632
2005	BlueGene/L (DOE/NNSA/LLNL)	280 TFLOP	131,072

Historical Trends in SuperComputing (1)



Historical Trends in SuperComputing (2)



The Drive toward High Performance Computing

- resolution
- dimensions
- physical realism

Resolution

When we increase the resolution we are using to solve a problem, computational time increases as well.

The increase in CPU time is usually much worse than a linear increase with the number of computational cells.

The Euler Equations

Consider the Euler equations. The size of the time step is limited by the Courant condition

$$\delta t = \frac{\delta x}{\min(v_i, c_i)}$$

where δx is the grid size, v_i is the bulk fluid velocity, and c_i is the local sound speed.

If we double the resolution, we decrease δx by a factor of two AND half the size of the time-step.

This means we need four times the CPU time to solve the same physical problem with twice the spatial resolution.

N-body Methods

The first N-body simulations included only a few hundred particles. Since every particle exerts a force on every other particle, the order of calculations goes as $O(n^2)$.

There are about 100 billion stars in our galaxy, not including the dark matter and gas. Modern cosmological simulations usually try to simulate the volume that contains 10,000 or more galaxies.

The current state of the art cosmological simulation has 10 billion stars.

Dimensions

Adding a physical dimension to a simulation greatly increases the cost of solving physical problems.

Early models were typically done in only one dimension. Most physical models are now done easily in two dimensions, but it is still computationally very expensive to do three dimensional simulations.

Even going from a two to three dimensional problem with the same physical resolution changes the cost from $O(n^2)$ to $O(n^3)$ where n is the grid size along one spatial dimension.

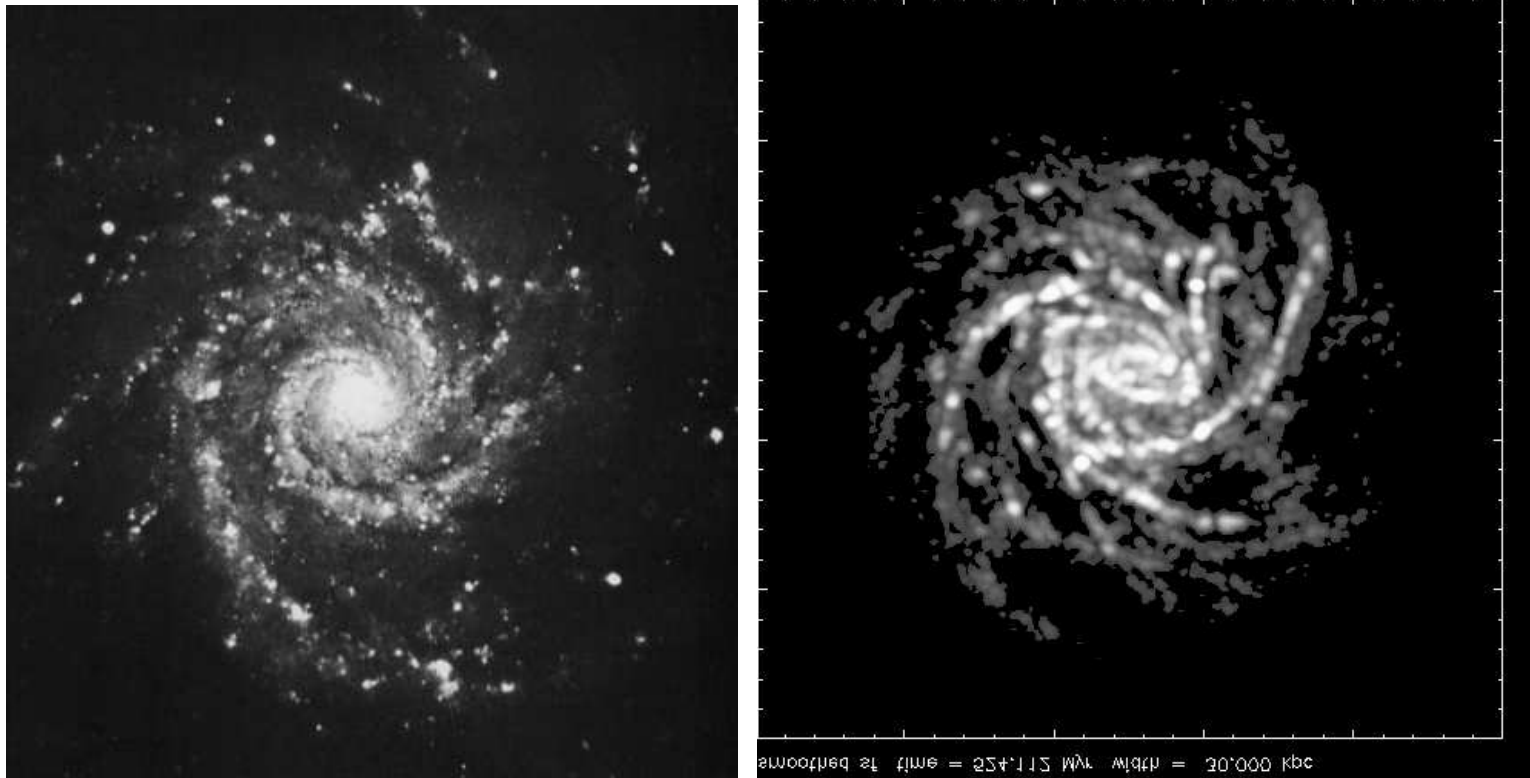
Physical Realism

Any set of equations is an approximation to physical reality. However, there are always choices in how much physics to include in any particular simulations.

If you take the example of galaxies, we can characterize different physical effects by their relative importance in changing the overall structure of the galaxy

Similar problems occur across Computational Science.

Galaxy Dynamics



Observation vs Simulation

Galaxy Dynamics

- large scale gravitational encounters
- internal gravitational forces
- gas dynamics
- formation of stars from gas
- feedback from star formation back into the gas
- active galactic nuclei

All programs approximate reality, but the better the physical model, the closer the results are to the real world.

Are Algorithms Important?

Which is more important:

- An efficient Algorithm
- A fast computer

$O(n^2)$ **to** $O(n \log n)$

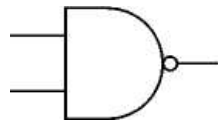
- The FFT algorithms changed the computational cost of calculation Fourier Transforms from $O(n^2)$ to $O(n \log n)$
- The particle mesh method and the hierarchical tree code changed the cost of solving n-body methods from $O(n^2)$ to $O(n \log n)$
- Assume it takes 1 second to calculate the Fourier transform of a given size. Approximately how much longer will it take to calculate the Fourier transform of a problem one thousand times larger?

Building Fast Computers

- The underlying basis for all computers is digital logic circuits - this has not changed since 1937
- All CPU's are based AND, OR, and NOT circuits
- If you can build faster digital circuits, you can increase the clock speed of your machine.
- You can also alter the design of your machine execute more simultaneous instructions

The Cray 1

The NAND Gate



Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	1

All digital logic circuits you need can be built from NAND gates.

If you build a faster NAND gate, the world will beat a path to your door.

Early Parallel Computers

- The development of parallel computers was predicated on the creation of computer networks.
- In the mid-1980's, high performance computing began moving toward parallelism
- Major computer companies began to create and sell multiprocessor machines.

SIMD Machines

- Early parallel computers executed a single instruction on all their CPU's. Each CPU held a different set of data, so they are called “Single Instruction, Multiple Data” machines.
- The instructions that could be executed in parallel were fairly limited, the individual CPU's were not very powerful, and the networking was slow, but there were usually a LOT of CPU's in the grid.
 - Thinking Machine CM-2 - 4k to 64k processors
 - MasPar - 16k processors

Limitations of SIMD Machines

- The biggest limitation was that the architectures limited the types of problems that could be run on these systems.
 - Problems that could not be decomposed to the size of the CPU array didn't work well.
 - Communication was limited both in speed and in cost.
- Not all programs could be mapped to this type of programming paradigm.

The Arrival of MIMD Machines

- In about 1993, computer companies started moving toward having more powerful nodes connected with networks.
- Each node was a fully functional computer.
- The nodes could all execute their own instructions on their own data - Multiple Instructions, Multiple Data.
- Communications was handled by message passing
- Each machine was constructed completely with proprietary hardware, operating systems, and software

Early MIMD Reinventing the Wheel

- Every major computer company hired a vast set of hardware and software specialists
 - operating systems needed to be written
 - networking protocols needed to be created
 - network switches needed to be built
 - compilers and languages(!) needed to be developed
- creating a computer from aluminum, copper and silicon was very expensive
- Since machines were built by different vendors
 - Users needed to rewrite their programs when a new machine arrived

- OS were unstable, tools were unreliable
- The development costs for new machines was HUGE
- By the late 1990's, most builders of large computers were in deep financial trouble.

Cluster Based Computing

- In 1994, Donald Becker and Thomas Sterling created the first commodity based cluster computer at GSFC working under a USRA contract.
- Beowulf clusters became wildly popular. Numerous organizations started putting together their own cluster based computers.
- The cost per calculation for Beowulf clusters was MUCH lower than what was available from large computer vendors.

Cluster Based Computing

The new direction for parallel computing is in cluster based computing. The essential characteristics are:

- multiple commercial off the shelf (COTS) machines (Intel, AMD or Apple Macintosh)
- no special operating system or compilers (usually Linux)
- high speed but COTS networking cards and routers connecting separate boxes
- standard message passing library handling communications through RSH or SSH
- The common name for these machines is Beowulf clusters.

Performance Characteristics of Beowulf Clusters

These are general characteristics of Beowulf clusters. Consider them rules of thumb rather than certainties.

- individual nodes are moderate end PC boxes
- communications are usually routed through a single router or switch
- most communications are fairly slow with high to moderate latency
- System performance depends heavily on the type of applications and the user base

Message Passing Libraries

With the creation of MIMD machines, message passing libraries had to be created to allow general communication between computational nodes.

The original libraries used were proprietary, and sold only with parallel machines. Every company wanted to have “new and better” features than every other company. This competition led to completely machine dependent programming. Every new parallel machine required a complete rewrite of the message passing sections. It is not very surprising that parallel computing has been a commercial failure... at least until recently.

Two main standards have emerged for message passing libraries:

- PVM

- MPI

