High-Performance Scientific Computing

Instructor: Randy LeVeque
TA: Grady Lemoine

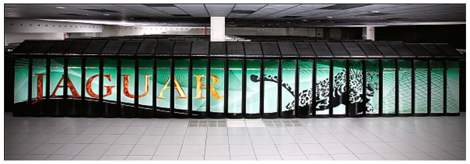
Applied Mathematics 483/583, Spring 2011

http://www.amath.washington.edu/~rjl/am583

"World's fastest computers"

http://top500.org

Roadrunner (Los Alamos)
122,400 cores



Jaguar (Oak Ridge)
224,162 cores

Outline of today's lecture

- Goals of this course, strategy for getting there
- Mechanics of homeworks
- Computer/software requirements
- Brief overview of computational science and challenges

Overview

High Performance Computing (HPC) generally means heavy-duty computing on clusters or supercomputers with 100s to million(s) of cores.

Our focus is more modest, but we will cover much background material that is:

- Essential to know if you eventually want to do HPC
- Extremely useful for any scientific computing project, even on a laptop.

Focus on scientific computing as opposed to other computationally demanding domains, for which somewhat different tools might be best.

Focus and Topics

Efficiently using single processor and multi-core computers

- Basic computer architecture, e.g. floating point arithmetic, cache hierarchies, pipelining
- Using Unix (or Linux, Mac OS X)
- Language issues, e.g. compiled vs. interpreted, object oriented, etc.
- Specific languages: Python, Fortran 90/95
- Parallel computing with OpenMP, MPI, IPython

Efficient programming as well as minimizing run time

- Version control: Mercurial (hg),
- Makefiles, Python scripting,
- Debuggers

Strategy

So much material, so little time....

- Concentrate on basics, simple motivating examples.
- Get enough hands-on experience to be comfortable experimenting further and learning much more on your own.
- Learn what's out there to help select what's best for your needs.
- Teach many things "by example" as we go along.

Lecture notes

- html and pdf versions at (green = link in pdf file)
  http://www.amath.washington.edu/~rjl/am583
- Written using Sphinx: Python-based system for writing documentation. Learn by example!!
- Source for each file can be seen by clicking on "Show Source" on right-hand menu.
- Source files are in class hg repository. You can clone the repository and run Sphinx yourself to make a local version.

```
$ hg clone http://bitbucket.org/.../uwamath583s11
$ cd uwamath583s11/sphinx
$ make html
$ firefox _build/html/index.html
```


Lecture slides

Slides from lectures will be linked from the Slides section of the class notes.

Generally in 3 forms, including one with space for taking notes.

With luck they will be posted at least 2 hours before class if you want to print and bring along.

Note: Slides will contain things not in the notes, lectures will also include hands-on demos not on the slides.

Prerequisites

Some programming experience in some language,
e.g., Matlab, C, Java.

You should be comfortable:

- editing a file containing a program and executing it,
- using basic structures like loops, if-then-else, input-output,
- writing subroutines or functions in some language

You are not expected to know Python or Fortran.

Some basic knowledge of linear algebra, e.g.:

- what vectors and matrices are and how to multiply them
- How to go about solving a linear system of equations

Some comfort level for learning new sofware and willingness to dive in to lots of new things.

Homeworks

There will be 6 homeworks, plus a take-home final "exam".

Electronic submission: via Mercurial (in order to get experience using Mercurial!)

Homework assignments will be in the notes.

Main goal: introduce many topics and get some hands-on experience with each.

Homework #1

Homework #1 is in the notes.

Tasks:

- Make sure you have a computer that you can use with
  - Unix (e.g. Linux of Mac OSX),
  - Python 2.5 or higer,
  - Mercurial

  See next slide.
- Use Mercurial (hg) to clone the class repository and set up your own repository.
- Copy a Python script from one to the other and run it, putting the output in a second file.
- Commit these files and push them to your repository for us to see.

Computer/Software requirements

You will need access to a computer with a number of things on it, see the section of the notes on Downloading and Installing Software.

Note: Unix is often required for scientific computing.

Windows: Many tools we'll use can be used with Windows, but learning Unix is part of this class.

Options:

- Install everything you'll need on your own computer,
- Install VirtualBox and use the Virtual Machine (VM) created for this class.
- Use a Linux machine in the Applied Mathematics department (via ssh).

TA and Office Hours

TA: Grady Lemoine

See the Class Catalyst Page for contact info, updated hours.

Office hours in Guggenheim 406

Monday, Tuesday, Friday 1:30 – 2:30

There is also a Discussion Board on the Class Catalyst Page, feel free to post **(and answer!)** questions about getting things to work.

Survey

Please take the survey found on the Class Catalyst Page to let us know about your background and computing plans.

As soon as possible.

Computational Science (and Engineering)

Often called the third pillar of science, complementing the traditional pillars of theory and experiment.

Direct numerical simulation of complex physics / biology / chemistry is possible.

Typically requires solving very large systems of mathematical equations.

Computational Science (and Engineering)

Unknowns represent values of some physical quantities, e.g.,

- (x, y, z) locations and velocities of individual atoms in a molecular dynamics simulation,
- (x, y, z) locations and velocities of individual stars in a cosmology simulation, e.g. galaxy formation.
- Density, pressure, velocities of a fluid at billions of points in a fluid dynamics simulation,
- Stress, strain, velocity of a solid at billions of points in a solid mechanics simulation.

Note: $1000 \times 1000 \times 1000$ grid has 1 billion grid points.

Need 8 gigabytes to store one variable at all grid points.

Computational Science and Engineering

A few examples of large scale problems for motivation.

Currently, that often means Tera-scale or Peta-scale.

Next comes Exa-scale.

How fast are computers?

Kilo	= thousand (10^3)
Mega	= million (10^6)
Giga	= billion (10^9)
Tera	= trillion (10^{12})
Peta	= 10^{15}
Exa	= 10^{18}

Processor speeds usually measured in Gigahertz these days.

Hertz means “machine cycles per second”.

One operation may take a few cycles.

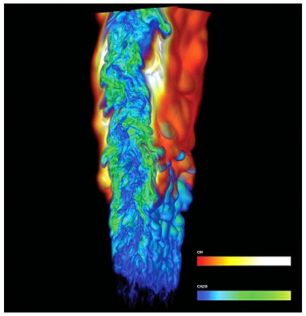
So a 1 GHz processor can do
$> 100,000,000$ operations per second.

Exascale is a billion times more than Gigascale.
(More speed and/or data.)

Combustion

Goal: Developing more fuel efficient and cleaner combustion processes for petroleum and alternative fuels.

Sample computation at Oak Ridge National Laboratory:



Ethylene combustion (simple!)

More than 1 billion grid points,
$\Delta x = \Delta y = \Delta z =$15 microns

4.5 million processor hours on Jaguar’s 31,000 cores

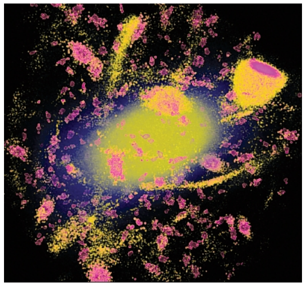
Generated > 120 terabytes of data

http://www.scidacreview.org/0902/html/news1.html

Milky Way’s dark matter halo

Goal: Understand nature of the universe.

Sample computation at Oak Ridge National Laboratory:



1.1 billion particles of dark matter,

simulated for 13.7 billion years

> 1 million processor hours on Jaguar (3000 cores)

http://www.scidacreview.org/0901/html/bt.html

How fast are computers?

Kilo	= thousand (10^3)
Mega	= million (10^6)
Giga	= billion (10^9)
Tera	= trillion (10^{12})
Peta	= 10^{15}
Exa	= 10^{18}

Processor speeds usually measured in Gigahertz these days.

Hertz means “machine cycles per second”.

One operation may take a few cycles.

So a 1 GHz processor can do
$> 100,000,000$ operations per second.

Exascale is a billion times more than Gigascale.
(More speed and/or data.)

How long does it take to solve a linear system?

Solving an $n \times n$ linear system $Ax = b$ requires $\approx \frac{1}{3}n^3$ flops.
(Using Gauss elimination for a dense matrix.)

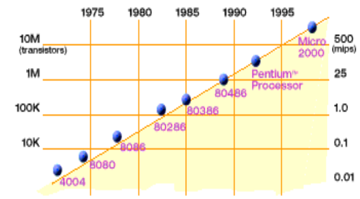
On a 100 MFlops system:

n	flops	time
10	3.3×10^2	0.0000033 seconds
100	3.3×10^5	0.0033 seconds
1000	3.3×10^8	3.33 seconds
10000	3.3×10^{11}	333 seconds = 5.5 minutes
100000	3.3×10^{14}	333333 seconds = 92.5 hours
1000000	3.3×10^{17}	92500 hours = 105 years

Assuming data transfer is not a problem!
It is a problem: It's often the bottleneck, not compute speed!
$10^6 \times 10^6$ matrix has 10^{12} elements $\implies$ 8 terabytes.

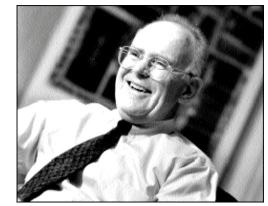
Moore's Law

Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years
Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

01/17/2007 CS267-Lecture 1 4

Increasing speed

Moore's Law: Processor speed doubles every 18 months.
$\implies$ factor of 1024 in 15 years.

Going forward: Number of cores doubles every 18 months.

Top: Total computing power of top 500 computers

Middle: #1 computer

Bottom: #500 computer

http://www.top500.org

More Limits: How fast can a serial computer be?

1 Tflop/s, 1 Tbyte sequential machine

r = 0.3 mm

- Consider the 1 Tflop/s sequential machine:
  - Data must travel some distance, r, to get from memory to CPU.
  - To get 1 data element per cycle, this means 10^{12} times per second at the speed of light, $c = 3 \times 10^8$ m/s. Thus $r < c/10^{12} = 0.3$ mm.
- Now put 1 Tbyte of storage in a 0.3 mm x 0.3 mm area:
  - Each bit occupies about 1 square Angstrom, or the size of a small atom.
- No choice but parallelism

01/17/2007 CS267-Lecture 1 9

Slide Source: Kathy Yellick

Take away messages

- Massively parallel machines are needed for Petascale or Exascale (millions or billions of cores).
- But also, all machines going to be multicore soon, with lots of cores.
- If you want to continue seeing benefiting from hardware improvements, you need to know something about parallel computing.