22S:295 Seminar in Applied Statistics
High Performance Computing in Statistics

Luke Tierney
Department of Statistics & Actuarial Science
University of Iowa

August 30, 2007
Please sign up for the course so we can get you computer accounts.

You will need an account on the cluster as well as an account on the math sciences network.

The accounts should be available by the second class meeting.

The class web page is

www.stat.uiowa.edu/~luke/classes/295-hpc

There are some pointers on computing and resources available on that page.

Class notes will also be posted there.

If you are not yet familiar with Linux or R you should become familiar with them soon.
A rough outline of what we might cover:

- Some background on HPC.
- Overview of the Statistics cluster.
- The snow package.
- Some tools for monitoring parallel applications.
- Other R parallel computing frameworks.
- Overview of PVM and MPI.
- Overview of OpenMP.
- Parallel linear algebra libraries.
- Batch scheduling on the cluster.
- Overview of Grid computing.
- ...
Why High Performance/Parallel Computing?

- Many computations are almost instantaneous on desktop computers.
- Some computations are beyond a single desktop computer: they
  - take too long
  - need too much memory
  - need too much disk storage
- Using multiple computers in an organized way is one solution.
- Using multiple processors on a single computer can also help.
- Doing this can be hard and/or expensive.
- Good tools can help.
- Before getting in too deep be sure to ask:
  - is the computation really needed?
An Important Number: $2^{32} = 4,294,967,296$

- For many years computers used a *32-bit architecture*:
  - Standard computer integer types are usually restricted to 32 bits, usually to the ranges $[-2^{31}, 2^{31} - 1]$ or $[0, 2^{32}]$.
  - The maximal amount of memory a process can address (the address space) is $2^{32}$ bytes, or 4 GB.
  - Using files more than 4G in size can be tricky.

- Larger amounts of memory can essentially only be used by working with multiple computers.

- More recently 64-bit architectures have become available:
  - C `int` and FORTRAN `integer` are usually still 32 bit for backward compatibility.
  - C `long` is usually 64 bit (except Win64) and supported in hardware.
  - Maximal address space is $2^{64} \approx 10^{19} = 10^7$TB = $10^4$PB = 10EB.
Early supercomputers were very fast single processors (1960s).

Single (1970s) and multiple (4–16, 1980s) vector processors.

Multiple standard processors with shared or distributed memory (1990s).

Beowulf clusters (mid 1990s):
- multiple (more or less) commodity computers
- reasonably fast dedicated communications network
- distributed memory (unavoidable for 32-bit)

Distributed shared memory systems
- can use 64-bit Beowulf-style hardware
- software, hardware, or combination
- Can provide illusion of single multi-processor system
- Sometimes called NUMA (Non-Uniform Memory Architecture)
- Still mostly proprietary and expensive
Some Recent Developments

- Moore’s law: number of transistors on a chip doubles every 18 months.
- Until recently this has meant speed increase at about the same rate.
- Recently speeds have remained flat — limiting factors:
  - heat
  - power consumption
- Additional transistors have been used for
  - integrating graphics, networking chipsets
  - multiple cores — 2 or 4 logical processors on a single chip
- Realistic 3D graphics have driven multiple processors on a chip:
  - Some nVIDIA cards have 128 (specialized) cores for $\sim$ $500$.
  - Sony/Toshiba/IBM Cell processor for PS 3 has one standard PowerPC Element (PPE) and 8 Synergistic Processing Elements (SPE).
  - Special libraries and methods are needed to program these.
  - Experimental interfaces from high level languages are available for some (Python, R for nVIDIA).
- Parallel programming is likely to become essential even for desktop computers.
Writing correct, efficient sequential programs is hard.
Writing correct, efficient parallel programs is harder.
Good tools can help:
- Low level tools:
  - sockets for distributed memory programming
  - threads for shared memory computing
- Intermediate level tools:
  - PVM, MPI message-passing libraries for distributed memory
  - OpenMP for shared memory
- Higher level tools:
  - simple systems like snow for R distributed memory
  - parallelized libraries (distributed or shared memory)
  - parallelizing compilers (mostly shared memory)

Some problems are easy to parallelize.
Some problems at least seem inherently sequential:
- pseudo-random number generation
- Markov chain Monte Carlo
A simple model says a computation runs $N$ times faster when split over $N$ processors.

More realistically, a problem has a fraction $S$ of its computation that is inherently sequential and $1 - S$ that can be parallelized.

Amdahl’s law:

$$\text{Maximum Speedup} = \frac{1}{S + \frac{1 - S}{N}}$$

Problems with $S \approx 0$ are called *embarrassingly parallel*.

Some statistical problems are (or seem to be) embarrassingly parallel:

- computing column means
- bootstrapping

Others seem inherently sequential:

- pseudo-random number generation
- Markov chain Monte Carlo