How to speed up your R computation by vectorization and parallel programming

Lecture 3

1. What is parallel computing and why is it useful in statistical computing?
2. How do we set up or utilize a parallel computer/cluster
3. Introduce some parallel R packages and use parallel apply functions to do so-called "embarrassingly parallel

What is parallel computing and why is it useful in statistical computing?

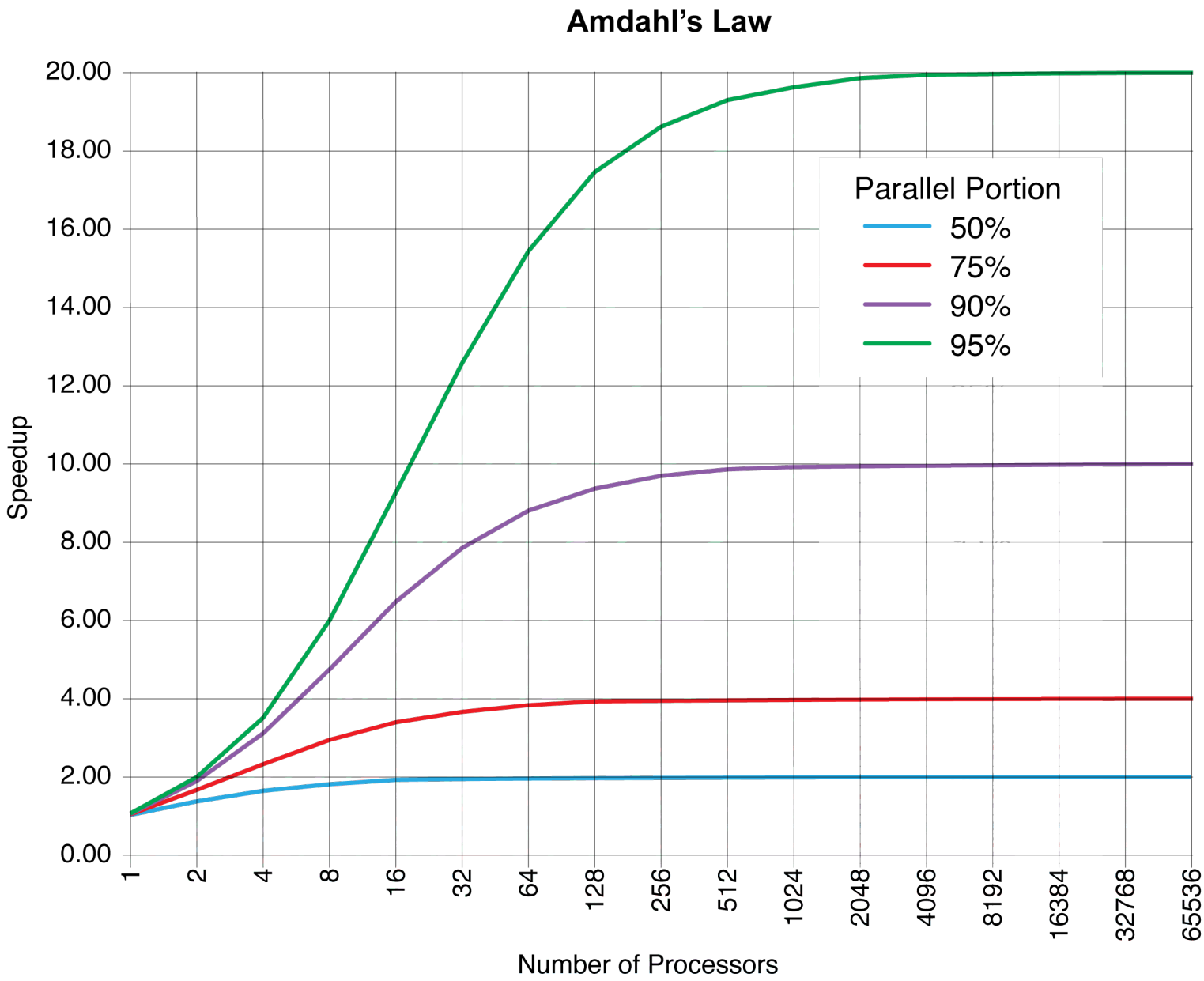
- Parallel computing is a form of computation in which many calculations are carried out **simultaneously**, operating on the principle that large problems can be divided into smaller ones
- In parallel (parallelism): smallers jobs are running concurrently
- Simply, a task can be broken into many small tasks that can be executed **simultaneously**

- If a task needs 10 hours of CPU time to finish, it still needs 10 hours of CPU time to finish whether it is running in parallel or not
- If the above task can be broken into 10 small tasks, each small task needs at least 1 hour of CPU time to finish
- If all small tasks are executed **simultaneously**, then the above task finishes about 1 hour of real time
- Except in rare circumstances, the above task should take over 1 hour of real time to finish after counting the overheads of splitting tasks and network communications
- Why do we need parallelism?
  - ⋆ R itself is written for serial computation. Its computation speed was determined by frequency scaling of a CPU until 2004
  - ⋆ However, increasing power consumption and heating by a CPU chip lead to the end of frequency scaling as the dominant computer architecture paradigm
  - ⋆ Moor's law (refer to CPU speed) is no longer true from serial computation point of view. Instead it is measured with multi-core chip and parallelism in mind
  - ⋆ To speed up R jobs, we need parallelism to utilize multi-core CPUs or cluster of computers

- Question: More CPUs, more speed-up?
  Answer: Speed-up follows Amdahl's law (and Gustafson's law)

  Let α be the fraction of running time a sequential program spends on non-parallelizable parts, then

$$\text{Maximum Speedup} = \frac{1}{\alpha}$$


Amdahl's Law
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Parallel Portion
50%
75%
90%
95%
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

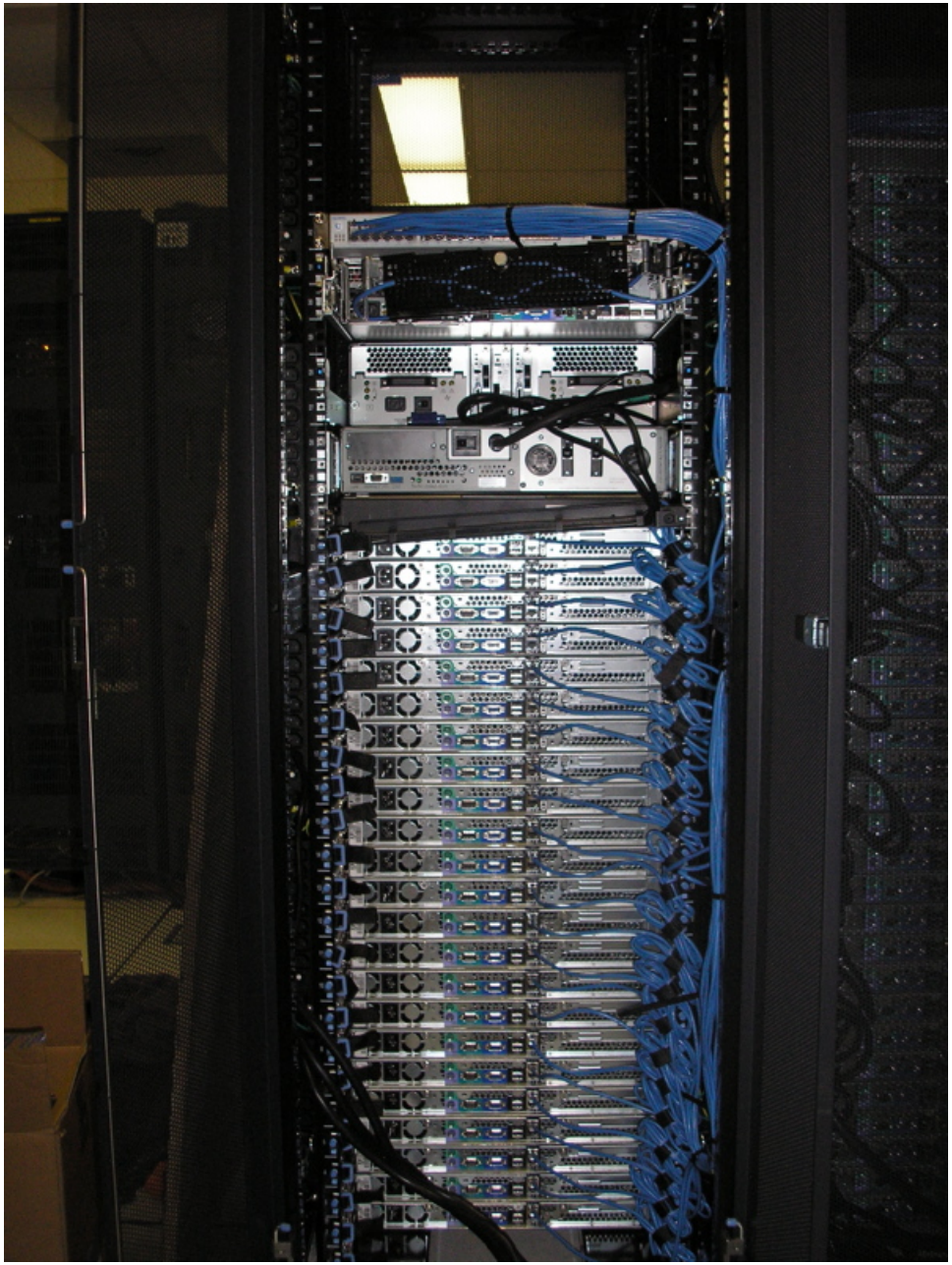
- Some parallel terminologies
  - Embarrassingly parallel
    - A task can be divided into many stand alone small tasks
    - Each small task can be executed independently one another
    - No communications between small tasks
  - Embarrassingly parallel applications are considered the easiest to parallelize
  - Many statistical modeling and simulation can be done in embarrassingly parallel
    - Monte Carlo simulation
    - (Double) bootstrap
    - Finance mathematics
    - Permutation tests
    - ...
  - Implicit parallelism: Jobs are automatically parallelized without users interferences
    - Some R packages can do it: pnmath, multicore, etc
    - There are many restrictions: portability, scalability?
  - Explicit parallelism: Allow or force the programmer to annotate his/per program indicate which parts should be executed as independent parallel tasks
    - R packages in this category: parallel, Rmpi, snow, and many others

    - Advantages: portability and scalability
- Other statistical computing can be done in parallel
    - Data mining
    - Graph or image analysis
    - Spatial stochastic modeling
    - ...

How do we set up or utilize a parallel computer/cluster?

- Multi-core PC (Windows, Mac OS X, Linux) is a parallel computer
    - No need to set up as far as hardware concerns unless multiple PCs are linked through network
    - Multi-core: 2 cores, 4 cores, 6 cores, ...
    - A workstation with 32 cores with 1 TB RAM
    - Do need proper software to support parallel programming/computing
- Beowulf Clusters
    - A collection of computers linked with high speed network

- A computer can have one CPU or multiple CPUs
- A computer can have either INTEL, AMD, or other vendors's CPUs
- Most used OS: Linux or other unix
- OS can be Windows or Mac OS X
- Advantages of using Linux
  - OS is free
  - Most parallel software is free and is properly configured
  - Very reliable; uptime can be many months
- Beowulf clusters are particularly well suited to implement process-level parallelism



  - SHARCNET (Shared Hierarchical Academic Research Computing Network)
    - http://www.sharcnet.ca
    - One of the largest cluster in Canada
    - Parallel R jobs can be submitted by using Rmpi
- Depending on budget, a cluster can be setup with relative short time
- Now we need software

Introduce some parallel R packages and use parallel apply functions to do so-called "embarrassingly parallel"

- Many parallel packages in R can be found through
  "CRAN Task View: High-Performance and Parallel Computing with R"
  http://cran.r-project.org/web/views/HighPerformanceComputing.html
- Rmpi and snow are two core parallel packages in R
- Many functions in snow are implemented in the parallel package
- The parallel package
  - R 2.14.0 or newer includes the parallel package as default

- ⋆ It includes several parallel versions of apply functions
- ⋆ It also comes a function to setup parallel RNG
- ⋆ Limitations:
  Runs only on shared memory systems;
  Implement embarrassing parallel only

- Usages of parallel package
  - ⋆ Find the number of cores: detectCores()
  - ⋆ Create a cluster (a collection of workers)
    cl= makeCluster(detectCores())
  - ⋆ Parallel lapply, sapply
    parLapply(cl, rep(1000000, 400), function(n)mean(rnorm(n)))
    parSapply(cl, rep(1000000, 400), function(n)mean(rnorm(n)))
  - ⋆ Use clusterExport to export all necessary R objects (data, functions) to workers
  - ⋆ Enable parallel RNG
    clusterSetRNGStream(cl)
  - ⋆ Enable parallel RNG with a specific seed
    clusterSetRNGStream(cl, iseed=123)
  - ⋆ Stop the cluster: stopCluster(cl)

- Adapt your codes (Monte Carlo) to use the parallel package
  - ⋆ Write your codes into multiple functions: f1,f2, ..., final_fun
  - ⋆ Index the final_fun:
    final_fun = function(i, data1=data1, data2=data2,...)
  - ⋆ Make sure the function final_fun is running properly
  - ⋆ Export all necessary functions and data
    clusterExport(cl, c("fun1","fun2"))
    clusterExport(cl, c("data1","data2"))
  - ⋆ Use one of parallel functions
    parSapply(cl, 1:N, final_fun)
  - ⋆ If the output of final_fun is a vector, then parSapply returns a matrix. Otherwise a list is return
- Make sure to activate parallel RNG with all Monte Carlo simulation
- Don't forget to shutdown the cluster: stopCluster(cl)