

# PARALLEL COMPUTING IN R

Erik Segur

# The Problem



- R natively only runs one process on one processor
  - ▣ R does not utilize the Dual-Core technology
- To cope with this limitation people would run several instances of R at the same time
- Computations were taking days, even weeks to finish

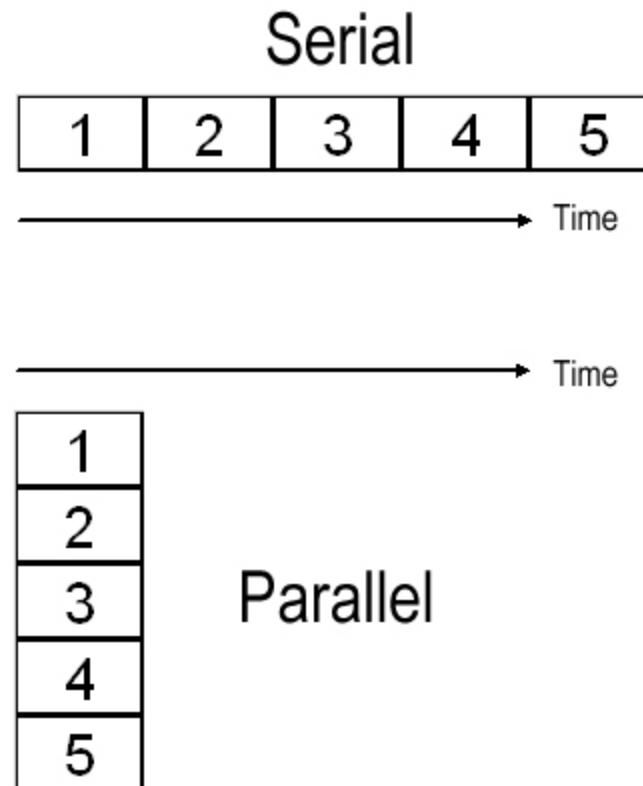
# The Solution



- Parallel Computing!
- What is parallel computing?
  - ▣ The simultaneous use of multiple processors to execute a problem
- This can be done on a single computer with more than one Processing Core or several computers linked together
- Hyper-Threading doesn't count, turn it off

# Serial vs. Parallel

- Serial: One iteration at a time
- Parallel: Multiple iterations at once, then collect results



# Time



- Parallel processing speeds up the calculations, by how much?

Single Processor	30 Processors
1 Minute	2 Seconds
1 Hour	2 Minutes
1 Day	1 Hour
1 Month	1 Day
1 Year	2 Weeks

# Time



- Ideally, the calculations would speed up  $X$  amount of times, where  $X = \#$  of processors
- In reality, time will vary
  - ▣ Calculations may vary in complexity
  - ▣ Different processor speeds
  - ▣ Current load on processor
  - ▣ Communication to other processors / computers
  - ▣ Managing and distributing load takes time
    - Master / Slave

# Master / Slave

- Master / Slave relationship where the number of iterations = number of processors.

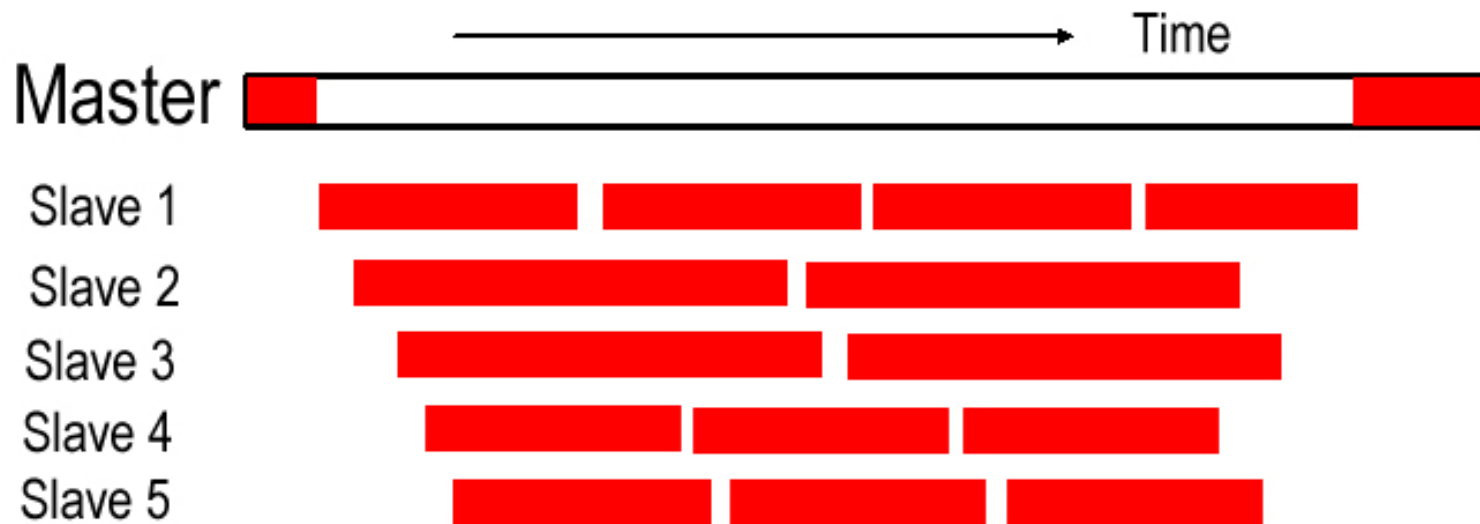


# Master / Slave



- What if...# of iterations  $>$  # of processors
- Load Balancing!
- With load balancing the next iteration will start as soon as a processor becomes available
- Without load balancing the program will wait for the slowest processor to complete its job before starting another set of iterations

# Load Balancing



# History



- Parallel processing has been around for awhile
  - Until recently it has only been for Linux / Unix
  - Recently developed for Windows! (last 6 months)
  - Main hang-up: Rmpi
  - Recent developments in new Windows Server 2008 High Performance Computing edition
  - Very new technology, still has its limits
    - 64 Bit Processors

# Setting Up Parallel Processing

- Software

- ▣ Deino MPI v1.1.0 (Do NOT get newest version)

- ▣ R

- Rmpi

- Snow

- Snowfall

- Rlecuyer

- Tutorial:

- [http://www.stt.msu.edu/links/Parallel\\_tutorial.pdf](http://www.stt.msu.edu/links/Parallel_tutorial.pdf)

# Packages

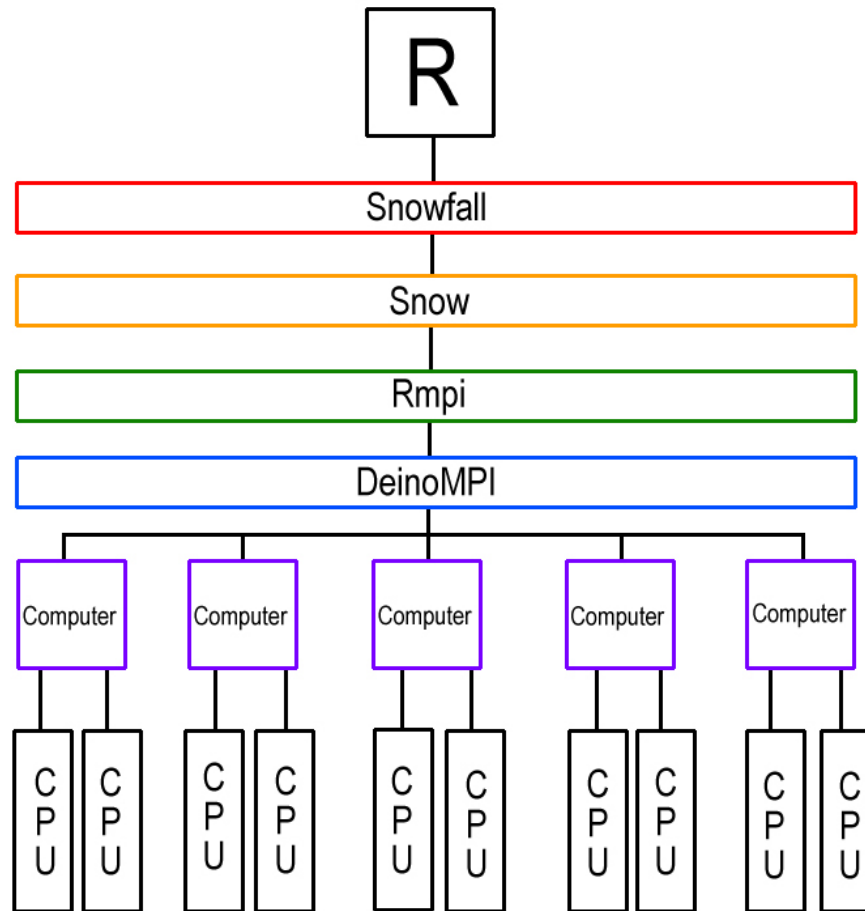


- DeinoMPI
  - ▣ Uses MPI (Message Passing Interface)
  - ▣ MPI allows several computers to talk to each other
  - ▣ Used in Computer Clusters and Super Computers
- Rmpi <http://www.stats.uwo.ca/faculty/yu/Rmpi/>
  - ▣ The main goal of Rmpi is to port low level MPI functions into R so that users do not have to know C or Fortran

# Packages

- Snow <http://www.sfu.ca/~sblay/R/snow.html>
  - ▣ snow implements an interface to low level mechanisms for creating a **virtual connection** between processes:
- Snowfall <http://cran.r-project.org/web/packages/snowfall/index.html>
  - ▣ Allows for easier programming to Snow package
- Rlecuyer
  - ▣ Random number generator for parallel systems
  - ▣ WHY?
    - Normal random number generator is based of system clock.
    - Multiple R processes can start at the same exact time and create identical “Random” numbers.

# Packages



# Setup and Install Demo



- [http://www.stt.msu.edu/links/Parallel\\_tutorial.pdf](http://www.stt.msu.edu/links/Parallel_tutorial.pdf)
- Must have R 2.8.1 or newer
- 32 bit Systems ONLY

# Writing Parallel Code in R

- Before you can start running your code you need to first initialize a few things
  - ▣ Load snowfall library ( enter `library(snowfall)` )
  - ▣ Initialize Snowfall Cluster
    - `sflnit()` function
      - `sflnit(parallel=TRUE, cpus=2, type="MPI",slaveOutfile="X.txt")`
        - SlaveOutfile: Location to store output from slaves
        - SlaveOutfile: HIGHLY inaccurate, but good for debugging
  - ▣ `sfSetMaxCPUs( number=100 )`
    - Needed ONLY for if # of CPU's >40

# Writing Parallel Code in R

## □ Loading packages

### □ `sfExport(package_name)`

- Packages needed to be loaded on slaves as well as the master

### □ `sfExport("variable_name")` or `sfExport("function_name")`

## □ Random Numbers

### □ If you need random numbers at all you will need this

- `sfClusterSetupRNGstream(seed=14)`
- Set the seed to whatever number you want

# Writing Parallel Code in R

## □ General format

```
main_function<-function(i)
{
  .... Do some calculations
  List(A=A,B=B,....)#this returns results
}
main_result<-sfClusterApplyLB(1:100,main_function)
```

- main\_result is a list of results from each iteration
- sfClusterApplyLB is the load balanced snowfall function that will distribute the work
- You can add system.time() to keep track of how long it took to complete the computations.
- For long computations, be sure to elevate explorer.exe in Task Manager

# Writing Parallel Code in R



- Clean Up!
  - EXTREMELY Important!!!
  - sfStop()
  - Simple Command, big impact
  - Shuts down cluster and cleans up remaining processes
  - If your code locks up, press the “Break” button in DeinoMPI and close RGui.exe through Task Manager.

# Writing Parallel Code in R



- Coding Example from Dr. Cui
  - ▣ Serial vs. Parallel

# Links



- Other useful information / links
- Setting up DeinoMPI / R tutorial
  - ▣ [http://www.stt.msu.edu/links/Parallel\\_tutorial.pdf](http://www.stt.msu.edu/links/Parallel_tutorial.pdf)
- Snowfall Reference Manual
  - ▣ <http://cran.r-project.org/web/packages/snowfall/snowfall.pdf>
- Snowfall Vignette
  - ▣ <http://cran.r-project.org/web/packages/snowfall/vignettes/snowfall.pdf>



Thanks for coming!