Data Analysis in R using High Performance Computing (HPC)

Tannistha Nandi, PhD (Bioinformatics)
Data Scientist, Research Computing Services
Bioinformatics National Team Lead, DRAC
Is R the right tool?

https://cran.r-project.org/
Does your workflow need HPC?

Scenario 1:
same R code needs to be run multiple times (usually on different input data)

Scenario 2:
computations need much more memory than what is available on your computer

Scenario 3:
Workflow needs GPU accelerator (code can make use of GPU).
HPC supports Command Line Interface

Graphical User Interface

Command line Interface

[tannistha.nandi@arc ~]$
[tannistha.nandi@arc ~]$ module avail R
-------------- /global/software/Modules/4.6.0/modulefiles ---------
R/3.5.3 R/3.6.2 R/4.2.6
[tannistha.nandi@arc ~]$ module load R/3.6.2
Loading R/3.6.2
  Loading requirement: lib/openblas/0.3.5-gnu lib/readline/6.3
[tannistha.nandi@arc ~]$
$R$
R version 3.6.2 (2019-12-12) -- "Dark and Stormy Night" Copyright (C) 2019
The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit) R is free software and comes with ABSOLUTELY NO Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R.
>
> 2 + 3
[1] 5
> sum(2,3)
[1] 5
> sum(2,3,4)
[1] 9
> x=c(2,3,4)
> sum(x)
[1] 9
>
```r
> x = c(2, 3, 3, 5, 5, 6, 6, 6, 7)
> hist(x)
> jpeg(file = "plot1.jpeg")
> hist(x, col = "darkgreen")
> dev.off()
```

```r
> ?hist

hist package:graphics R Documentation

Histograms

Description:

The generic function `hist` computes a histogram of the given data values. If `plot = TRUE`, the resulting object of class `"histogram"` is plotted by `plot.histogram`, before it is returned.

Usage:

`hist(x, ...)`
```
The generic function ‘hist’ computes a histogram of the given data values. If ‘plot = TRUE’, the resulting object of class ‘“histogram”’ is plotted by ‘plot.histogram’, before it is returned.

Usage:

hist(x, ...)

example(hist)

> help(hist)
hiHistograms

Description:

The generic function ‘hist’ computes a histogram of the given data values. If ‘plot = TRUE’, the resulting object of class ‘“histogram”’ is plotted by ‘plot.histogram’, before it is returned.

Usage:

hist(x, ...)

example(hist)

> example(hist)

hist> op <- par(mfrow = c(2, 2))
hist> hist(islands)
hist> utils::str(hist(islands, col = "gray", labels = TRUE))

List of 6
$ breaks : num [1:10] 0 2000 4000 6000 8000 10000 12000 14000 16000 18000
$ counts : int [1:9] 41 2 1 1 1 0 0 1
$ density : num [1:9] 4.27e-04 2.08e-05 1.04e-05 1.04e-05 1.04e-05 1.04e-05 ...
$ mids : num [1:9] 1000 3000 5000 7000 9000 11000 13000 15000 17000
$ xname : chr "islands"
$ equidist: logi TRUE
- attr(*, "class")= chr "histogram"

> quit()

Save workspace image? [y/n/c]: n
[tannistha.nandi@arc ~]$
[tannistha.nandi@arc ~]$ cat mycode.R
x=c(2,3,3,5,5,6,6,6,7)
jpeg(file="plot1.jpeg")
hist(x, col="darkgreen")
dev.off()

[tannistha.nandi@arc ~]$ Rscript mycode.R
[tannistha.nandi@arc ~]$ ls plot1.jpeg
plot1.jpeg
How to request for compute resources?

[tannistha.nandi@arc ~]$ salloc --mem=20G --time=01:00:00 --cpus-per-task=6
salloc: Granted job allocation 19501396
salloc: Waiting for resource configuration
salloc: Nodes fc29 are ready for job
[tannistha.nandi@fc29 ~]$ export PATH=~/software/R-4.2.3/bin:$PATH
[tannistha.nandi@fc29 ~]$ R

R version 4.2.3 (2023-03-15) -- 'Shortstop Beagle'
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> Simulation <- function(n){
+ ntests <- 10000
+ pop <- 1:365
+ anydups <- function(i)
+ any(duplicated(
+ sample(pop, n, replace=TRUE)))
+ sum(sapply(seq(ntests), anydups))/ntests
+ }

#SEQUENTIAL
> getDoParWorkers()
[1] 1
> system.time(s_result <- sapply(1:100, Simulation))
 user  system elapsed
15.334   0.001   15.395
# Loop with foreach

```r
foreach(n=1:100) %dopar% Simulation(n)
```

Parallel processing by registering a backend, some options are:

1. registerDoSEQ  
   default with ‘foreach’
2. registerDoParallel  
   register ‘DoParallel’ to be used with ‘foreach’
3. registerDoMC  
   register ‘doMC’ to be used with ‘foreach’
> library(doParallel)
> cluster1 <- makeCluster(2) # local cluster with 2 workers
> registerDoParallel(cluster1)
> getDoParWorkers()
[1] 2

> system.time(p1_result <- foreach(n=1:100) %dopar% Simulation(n))
  user  system elapsed
 0.044  0.009  8.859
> library(doMC)
> registerDoMC(6)
> getDoParWorkers()
[1] 6

> system.time(p2_result <- foreach(n=1:100) %dopar% Simulation(n))
    user  system elapsed
   17.153   0.351   3.027
Efficient parallelisation & communication overhead

Parallelisation doesn’t come for free:

There is a communication overhead in sending objects to and receiving objects from each parallel core.
<table>
<thead>
<tr>
<th>Province</th>
<th>Age</th>
<th>Salary</th>
<th>Product Purchased</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alberta</td>
<td>44</td>
<td>72000</td>
<td>No</td>
</tr>
<tr>
<td>Toronto</td>
<td>29</td>
<td>48000</td>
<td>Yes</td>
</tr>
<tr>
<td>Toronto</td>
<td>30</td>
<td>55000</td>
<td>No</td>
</tr>
<tr>
<td>Vancouver</td>
<td>38</td>
<td>61000</td>
<td>No</td>
</tr>
<tr>
<td>Vancouver</td>
<td>36</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Toronto</td>
<td>35</td>
<td>58000</td>
<td>Yes</td>
</tr>
<tr>
<td>Toronto</td>
<td>-</td>
<td>52000</td>
<td>No</td>
</tr>
<tr>
<td>Alberta</td>
<td>48</td>
<td>79000</td>
<td>Yes</td>
</tr>
<tr>
<td>Alberta</td>
<td>50</td>
<td>83000</td>
<td>No</td>
</tr>
<tr>
<td>Vancouver</td>
<td>37</td>
<td>63000</td>
<td>Yes</td>
</tr>
<tr>
<td>Toronto</td>
<td>35</td>
<td>58000</td>
<td>Yes</td>
</tr>
<tr>
<td>Toronto</td>
<td>54</td>
<td>52000</td>
<td>No</td>
</tr>
<tr>
<td>Alberta</td>
<td>36</td>
<td>79000</td>
<td>Yes</td>
</tr>
<tr>
<td>Alberta</td>
<td>89</td>
<td>48000</td>
<td>No</td>
</tr>
<tr>
<td>Vancouver</td>
<td>57</td>
<td>76000</td>
<td>Yes</td>
</tr>
</tbody>
</table>
#Import data in R

# import a tab delimited file (.txt)
data = read.table("Data.txt", sep="\t", header=T)

# import a comma delimited file (.csv)
data = read.csv("Data.csv", sep="","", header=T)

# Install and Load packages
#install.packages("readxl")
library("readxl")

# import an excel file (.xlsx)
data = read_excel("Data.xlsx", sheet =1) # specify the sheet by its index/ name
Getting ready for HPC

1. Command line editors like nano, vi, emacs

2. Knowledge of Linux

3. Job scheduler: SLURM
   (Simple Linux Utility for Resource Management)
Reference resources

Linux
http://linuxcommand.org/tlcl.php

R
https://cran.r-project.org/

Job scheduler
https://slurm.schedmd.com
Reach us if you need help to get started

support@hpc.ucalgary.ca

Happy to take your questions!!