Parallel Computing in R

ResBaz Dunedin Feb 8th 2017

Author: Tom Kelly, PhD Candidate, University of Otago

This lesson assumes you are familiar with repeating operations in loops e.g.:

```
for(ii in 1:10){
  print(ii^2)
}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
## [1] 64
## [1] 81
## [1] 100
```

Clearly for more iterations or more complex functions this could soon take a considerable amount of time:

```
max_n <- 100
vector <- rep(NA, max_n)
```

```
system.time({
  for(ii in 1:max_n){
    vector[ii] <- ii^2
  }
})
```

```
##    user  system elapsed
##       0       0       0
```

```
for(max_n in c(100, 1000, 10000, 100000)){
  print(max_n)
  print(system.time({
    for(ii in 1:max_n){
      vector[ii] <- ii^2
    }
  }))
}
```

```
## [1] 100
##    user  system elapsed
##   0.000   0.000   0.001
## [1] 1000
##    user  system elapsed
##   0.004   0.000   0.004
## [1] 10000
##    user  system elapsed
##   0.136   0.000   0.135
## [1] 1e+05
##    user  system elapsed
```

```
##  18.700   0.708  19.436
```

Apply functions

```
max_n <- 100
vector <- lapply(1:max_n, function(x) x^2) #create list
vector <- sapply(1:max_n, function(x) x^2) #create vector
vector
```

```
##   [1]     1     4     9    16    25    36    49    64    81   100   121
##  [12]   144   169   196   225   256   289   324   361   400   441   484
##  [23]   529   576   625   676   729   784   841   900   961  1024  1089
##  [34]  1156  1225  1296  1369  1444  1521  1600  1681  1764  1849  1936
##  [45]  2025  2116  2209  2304  2401  2500  2601  2704  2809  2916  3025
##  [56]  3136  3249  3364  3481  3600  3721  3844  3969  4096  4225  4356
##  [67]  4489  4624  4761  4900  5041  5184  5329  5476  5625  5776  5929
##  [78]  6084  6241  6400  6561  6724  6889  7056  7225  7396  7569  7744
##  [89]  7921  8100  8281  8464  8649  8836  9025  9216  9409  9604  9801
## [100] 10000
```

```
system.time({
  for(ii in 1:max_n){
    vector <- lapply(1:max_n, function(x) x^2) #create list
  }
})
```

```
##    user  system elapsed
##   0.008   0.000   0.007
```

```
system.time({
  for(ii in 1:max_n){
    vector <- sapply(1:max_n, function(x) x^2) #create vector
  }
})
```

```
##    user  system elapsed
##   0.008   0.000   0.008
```

```
for(max_n in c(100, 1000, 10000, 100000)){
  print(max_n)
  print(system.time({
    vector <- sapply(1:max_n, function(x) x^2) #create vector
  }))
}
```

```
## [1] 100
##    user  system elapsed
##       0       0       0
## [1] 1000
##    user  system elapsed
##   0.004   0.000   0.001
## [1] 10000
##    user  system elapsed
##   0.004   0.000   0.006
## [1] 1e+05
```

```
##    user  system elapsed
##   0.080   0.000   0.078
```

Vectorisation

```
max_n <- 100
vector <- 1:max_n
vector <- vector^2
vector
```

```
##   [1]     1     4     9    16    25    36    49    64    81   100   121
##  [12]   144   169   196   225   256   289   324   361   400   441   484
##  [23]   529   576   625   676   729   784   841   900   961  1024  1089
##  [34]  1156  1225  1296  1369  1444  1521  1600  1681  1764  1849  1936
##  [45]  2025  2116  2209  2304  2401  2500  2601  2704  2809  2916  3025
##  [56]  3136  3249  3364  3481  3600  3721  3844  3969  4096  4225  4356
##  [67]  4489  4624  4761  4900  5041  5184  5329  5476  5625  5776  5929
##  [78]  6084  6241  6400  6561  6724  6889  7056  7225  7396  7569  7744
##  [89]  7921  8100  8281  8464  8649  8836  9025  9216  9409  9604  9801
## [100] 10000
```

```
for(max_n in c(100, 1000, 10000, 100000)){
  print(max_n)
  print(system.time({
      vector <- 1:max_n
      vector <- vector^2
  }))
}
```

```
## [1] 100
##    user  system elapsed
##       0       0       0
## [1] 1000
##    user  system elapsed
##       0       0       0
## [1] 10000
##    user  system elapsed
##       0       0       0
## [1] 1e+05
##    user  system elapsed
##   0.000   0.000   0.001
```

Parallelisation

If iterations independent:

```
library("snow") #simple network of workstations
cl <- makeSOCKcluster(2) # number of cores
#makeMPIcluster(2)
#makeCluster(2)
clusterExport(cl, list=ls())
```

```
max_n <- 100
vector <- parLapply(cl, 1:max_n, function(x) x^2) #create list
vector <- unlist(vector)
vector
```

```
##   [1]     1     4     9    16    25    36    49    64    81   100   121
##  [12]   144   169   196   225   256   289   324   361   400   441   484
##  [23]   529   576   625   676   729   784   841   900   961  1024  1089
##  [34]  1156  1225  1296  1369  1444  1521  1600  1681  1764  1849  1936
##  [45]  2025  2116  2209  2304  2401  2500  2601  2704  2809  2916  3025
##  [56]  3136  3249  3364  3481  3600  3721  3844  3969  4096  4225  4356
##  [67]  4489  4624  4761  4900  5041  5184  5329  5476  5625  5776  5929
##  [78]  6084  6241  6400  6561  6724  6889  7056  7225  7396  7569  7744
##  [89]  7921  8100  8281  8464  8649  8836  9025  9216  9409  9604  9801
## [100] 10000
```

```
system.time({
  vector <- parLapply(cl, 1:max_n, function(x) x^2) #create list
  })
```

```
##    user  system elapsed
##    0.00    0.00    0.04
```

```
stopCluster(cl)
```

```
cl <- makeSOCKcluster(2) # number of cores
clusterExport(cl, list=ls())
for(max_n in c(100, 1000, 10000, 100000)){
  print(max_n)
  print(system.time({
    vector <- parLapply(cl, 1:max_n, function(x) x^2) #create list
  }))
}
```

```
## [1] 100
##    user  system elapsed
##   0.000   0.000   0.039
## [1] 1000
##    user  system elapsed
##    0.00    0.00    0.08
## [1] 10000
##    user  system elapsed
##   0.004   0.000   0.085
## [1] 1e+05
##    user  system elapsed
##   0.036   0.000   0.120
```

```
stopCluster(cl)
```

```
cl <- makeSOCKcluster(3) # number of cores
clusterExport(cl, list=ls())
for(max_n in c(100, 1000, 10000, 100000)){
  print(max_n)
  print(system.time({
    vector <- parLapply(cl, 1:max_n, function(x) x^2) #create list
  }))
```

```
}
```

```
## [1] 100
##    user  system elapsed
##   0.000   0.000   0.039
## [1] 1000
##    user  system elapsed
##   0.000   0.000   0.038
## [1] 10000
##    user  system elapsed
##   0.004   0.000   0.109
## [1] 1e+05
##    user  system elapsed
##   0.036   0.004   0.137
```

```
stopCluster(cl)
```

Note the increase in speed is near linear

However there is an "overhead time" to set up cluster (which only worth it for larger jobs):

```
system.time({
  cl <- makeSOCKcluster(2) # number of cores
  #makeMPIcluster(2)
  #makeCluster(2)
  clusterExport(cl, list=ls())
  stopCluster(cl)
})
```

```
##    user  system elapsed
##   0.008   0.008   0.295
```

```
for(max_n in c(100, 1000, 10000, 100000)){
  print(max_n)
  print(system.time({
    cl <- makeSOCKcluster(2) # number of cores
    clusterExport(cl, list=ls())
    vector <- parLapply(cl, 1:max_n, function(x) x^2) #create list
    stopCluster(cl)
  }))
}
```

```
## [1] 100
##    user  system elapsed
##   0.020   0.004   0.328
## [1] 1000
##    user  system elapsed
##   0.004   0.000   0.311
## [1] 10000
##    user  system elapsed
##   0.008   0.000   0.357
## [1] 1e+05
##    user  system elapsed
##   0.044   0.004   0.456
```

```
for(max_n in c(100, 1000, 10000, 100000)){
  print(max_n)
```

```
  print(system.time({
    cl <- makeSOCKcluster(3) # number of cores
    clusterExport(cl, list=ls())
    vector <- parLapply(cl, 1:max_n, function(x) x^2) #create list
    stopCluster(cl)
    }))
}
```

```
## [1] 100
##    user  system elapsed
##   0.032   0.000   0.468
## [1] 1000
##    user  system elapsed
##   0.008   0.000   0.403
## [1] 10000
##    user  system elapsed
##   0.012   0.000   0.473
## [1] 1e+05
##    user  system elapsed
##   0.048   0.000   0.549
```

Thus there are diminishing returns in practice (when increasing the number of cores) due to more communication between them.

Question Time

```
max_n <- 100

#for loop
for(ii in 1:max_n){
  print(ii^2)
}

#lapply
vector <- lapply(1:max_n, function(x) x^2) #create list

#parLapply on 2 cores
cl <- makeSOCKcluster(2) # number of cores
clusterExport(cl, list=ls())
vector <- parLapply(cl, 1:max_n, function(x) x^2) #create list
stopCluster(cl)
```

```
#setup data
dataset <- matrix(rnorm(600), 60, 10)
head(dataset)
```

```
##            [,1]       [,2]         [,3]       [,4]       [,5]       [,6]
## [1,]  1.1413315  0.1645909  1.108942426  0.0582424 -0.1739486  1.5194065
## [2,]  1.4609639 -0.1594043  0.002062562 -0.1940727  0.6411211 -0.4842080
## [3,] -1.3211921 -1.1386354  0.213566733 -0.9295126  0.8136654 -1.7675017
## [4,] -0.2000048 -0.4853933 -0.429627415  0.6031221 -0.2677124 -0.1203867
## [5,] -1.4850282 -0.2747699  0.321598555 -0.0928672 -0.7436777  0.8217351
## [6,] -1.7934968  0.0965008 -0.119501268 -0.1228585  0.7871398 -1.0379623
##            [,7]       [,8]       [,9]      [,10]
```

```
## [1,] -1.5578552  1.2137207  1.2260942 -0.8743838
## [2,]  1.5671589 -0.3422049  0.8340759 -0.4820724
## [3,] -1.2029268 -0.8956453 -0.7250044 -0.7217486
## [4,]  0.2680512  0.9097705  0.8618185  0.9671865
## [5,]  0.2464274 -0.8991501 -2.7523461 -1.4411381
## [6,]  1.9075332 -1.3704211  0.1214119 -0.3524388
```

```
#for loop:
mg_data <- matrix(NA, nrow(dataset)/3, ncol(dataset))
for(ii in 1:(nrow(mg_data))){
  mg_data[ii,] <- svd(dataset[(ii-1)*3+1:3,])$v[,1]
}

#make an lapply loop
mg_data <- matrix(NA, nrow(dataset)/3, ncol(dataset))
mg_data <-lapply(1:(nrow(mg_data)), function(ii){
  ##fill in function
})
mg_data <- t(as.data.frame(mg_data))

#make it parallel
cl <- makeSOCKcluster(2) # number of cores
clusterExport(cl, list=ls())
mg_data <- matrix(NA, nrow(dataset)/3, ncol(dataset))
mg_data <-parLapply(#here, #here, function(ii){
  #here
  })
mg_data <- t(as.data.frame(mg_data))
stopCluster(cl)
```

Answer Time

```
#make an lapply loop
mg_data <- matrix(NA, nrow(dataset)/3, ncol(dataset))
mg_data <-lapply(1:(nrow(mg_data)), function(ii){
  svd(dataset[(ii-1)*3+1:3,])$v[,1]
})
mg_data <- t(as.data.frame(mg_data))

#make it parallel
cl <- makeSOCKcluster(2) # number of cores
clusterExport(cl, list=ls())
mg_data <- matrix(NA, nrow(dataset)/3, ncol(dataset))
mg_data <-parLapply(cl, 1:(nrow(mg_data)), function(ii){
  svd(dataset[(ii-1)*3+1:3,])$v[,1]
})
mg_data <- t(as.data.frame(mg_data))
stopCluster(cl)
```

Running scripts

```
source("loop_script.R")
```

```
system("Rscript loop_script.R") #run bash command
library("data.table")
mg_data <- fread("mg_data.csv", data.table = F)
head(mg_data)
dir()
```

Passing arguments to Rscripts

```
system("Rscript loop_script_Rstudio_generic.R 3 100")
library("data.table")
mg_data <- fread("mg_data.csv", data.table = F)
head(mg_data)
```

```
##                                                                                 V1
## 1  c.0.506699418020483...0.628642300044541..0.269359874051679..0.31722711379329..
## 2                 c..0.0160327360042453...0.0479648773183941..0.576142479549336..
## 3                 c.0.710626095980894...0.0514131962100698...0.0857515889715776..
## 4 c..0.0040048643388877..0.4157881113965...0.140939414537173..0.164031752176782..
## 5                   c..0.320390541699733..0.0571739174659095..0.125784645013791..
## 6                   c.0.628690326988676...0.0932604355332615...0.14230957721699..
##             V1          V2          V3          V4          V5          V6
## 1  0.506699418 -0.62864230  0.26935987  0.31722711 -0.09939905  0.19329916
## 2 -0.016032736 -0.04796488  0.57614248 -0.05344534  0.24657722  0.08262376
## 3  0.710626096 -0.05141320 -0.08575159 -0.20155917 -0.25877671 -0.21074018
## 4 -0.004004864  0.41578811 -0.14093941  0.16403175 -0.13213437  0.29947882
## 5 -0.320390542  0.05717392  0.12578465  0.64297947  0.11177030  0.18846565
## 6  0.628690327 -0.09326044 -0.14230958 -0.27776117 -0.28505808 -0.14546889
##           V7          V8         V9         V10
## 1  0.1850576 -0.25827899  0.1558035  0.04902000
## 2 -0.3632397  0.51909889  0.3920613  0.19974966
## 3  0.1005776 -0.08570897 -0.0828292 -0.55559737
## 4  0.3966724 -0.22155712 -0.1991548  0.65351943
## 5  0.1180351 -0.21493680 -0.3247831 -0.50120912
## 6  0.2952607 -0.50849627  0.2047601 -0.09248959
```

```
dim(mg_data)
```

```
## [1] 20 11
```

```
dir()
```

```
##  [1] "challenge_answer.R"            "challenge_question.R"
##  [3] "loop_script_generic_array.sl"  "loop_script_generic.R"
##  [5] "loop_script_generic.sh"        "loop_script_generic.sl"
##  [7] "loop_script.R"                 "loop_script_Rstudio_generic.R"
##  [9] "mg_data_3core_100N.csv"        "mg_data_3core_10N.csv"
## [11] "mg_data_3core_15N.csv"         "mg_data_3core_20N.csv"
## [13] "mg_data_3core_2N.csv"          "mg_data_3core_30N.csv"
## [15] "mg_data_3core_3N.csv"          "mg_data_3core_4N.csv"
## [17] "mg_data_3core_5N.csv"          "mg_data_3core_80N.csv"
```

```
## [19] "mg_data.csv"                    "R_parallel_Demo.html"
## [21] "R_parallel_Demo.R"              "R_parallel_Demo.Rmd"
## [23] "R_parallel_Demo.Rproj"
```

```
for(ii in c(10, 20, 30)){
  system(paste("Rscript loop_script_Rstudio_generic.R 3", ii))
}
dir()
```

```
##  [1] "challenge_answer.R"            "challenge_question.R"
##  [3] "loop_script_generic_array.sl"  "loop_script_generic.R"
##  [5] "loop_script_generic.sh"        "loop_script_generic.sl"
##  [7] "loop_script.R"                 "loop_script_Rstudio_generic.R"
##  [9] "mg_data_3core_100N.csv"        "mg_data_3core_10N.csv"
## [11] "mg_data_3core_15N.csv"         "mg_data_3core_20N.csv"
## [13] "mg_data_3core_2N.csv"          "mg_data_3core_30N.csv"
## [15] "mg_data_3core_3N.csv"          "mg_data_3core_4N.csv"
## [17] "mg_data_3core_5N.csv"          "mg_data_3core_80N.csv"
## [19] "mg_data.csv"                   "R_parallel_Demo.html"
## [21] "R_parallel_Demo.R"             "R_parallel_Demo.Rmd"
## [23] "R_parallel_Demo.Rproj"
```

Passing arguments to bash

```
bash loop_script_generic.sh 3 80
ls
```

```
## arguments passed to bash
## 3 80
## 1st argument
## 3 cores
## remaining arguments
## 80 samples
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "80"
##  int 80
## challenge_answer.R
## challenge_question.R
## loop_script_generic_array.sl
## loop_script_generic.R
## loop_script_generic.sh
## loop_script_generic.sl
## loop_script.R
## loop_script_Rstudio_generic.R
## mg_data_3core_100N.csv
## mg_data_3core_10N.csv
## mg_data_3core_15N.csv
## mg_data_3core_20N.csv
```

```
## mg_data_3core_2N.csv
## mg_data_3core_30N.csv
## mg_data_3core_3N.csv
## mg_data_3core_4N.csv
## mg_data_3core_5N.csv
## mg_data_3core_80N.csv
## mg_data.csv
## R_parallel_Demo.html
## R_parallel_Demo.R
## R_parallel_Demo.Rmd
## R_parallel_Demo.Rproj
```

```
bash loop_script_generic.sh 3 {2..5} 10 15
ls
```

```
## arguments passed to bash
## 3 2 3 4 5 10 15
## 1st argument
## 3 cores
## remaining arguments
## 2 3 4 5 10 15 samples
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "2"
##  int 2
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "3"
##  int 3
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "4"
##  int 4
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "5"
##  int 5
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
```

```
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "10"
##  int 10
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "15"
##  int 15
## challenge_answer.R
## challenge_question.R
## loop_script_generic_array.sl
## loop_script_generic.R
## loop_script_generic.sh
## loop_script_generic.sl
## loop_script.R
## loop_script_Rstudio_generic.R
## mg_data_3core_100N.csv
## mg_data_3core_10N.csv
## mg_data_3core_15N.csv
## mg_data_3core_20N.csv
## mg_data_3core_2N.csv
## mg_data_3core_30N.csv
## mg_data_3core_3N.csv
## mg_data_3core_4N.csv
## mg_data_3core_5N.csv
## mg_data_3core_80N.csv
## mg_data.csv
## R_parallel_Demo.html
## R_parallel_Demo.R
## R_parallel_Demo.Rmd
## R_parallel_Demo.Rproj
```

Running scripts in the background

```
nohup Rscript loop_script_Rstudio_generic.R 3 100 &
nohup bash loop_script_generic.sh 3 2..5 10 15 &
```

```
## arguments passed to bash
## 3 2..5 10 15
## 1st argument
## 3 cores
## remaining arguments
## 2..5 10 15 samples
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
```

```
## [5] "--args"
## [6] "3"
## [7] "2..5"
##  int NA
## Error in rnorm(60 * sample_size) : invalid arguments
## Calls: matrix -> rnorm
## Execution halted
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "100"
##  int 100
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "10"
##  int 10
## [1] "/usr/lib/R/bin/exec/R"
## [2] "--slave"
## [3] "--no-restore"
## [4] "--file=loop_script_Rstudio_generic.R"
## [5] "--args"
## [6] "3"
## [7] "15"
##  int 15
```

Running R on the cluster

```
sbatch loop_script_generic.sl 3 105
sbatch loop_script_generic_array.sl 3 35 45 55
```

Remote access

```
rsh pan #remote shell
rcp loop_script_generic_array.sl pan:/projects/uoo00010/test_dir #remote copy
```

"secure shell"

```
ssh pan
scp loop_script_generic_array.sl pan:/projects/uoo00010/test_dir
```

sync (move changes)

```
rsync -u loop_script_generic_array.sl pan:/projects/uoo00010/test_dir
```

ssh alias

```
cat ~/.ssh/config
ssh simon.kelly@login.uoa.nesi.org.nz
exit
ssh -XY pan
exit
```

```
ssh biochembioinfo.otago.ac.nz
exit
ssh bio
exit
```

Supplementary Resources

NeSI Set Up (Wiki) for Otago Users

https://github.com/dannybaillie/NeSI

Install R packages on NeSI

https://github.com/TomKellyGenetics/install.nesi

Notes for this guide

https://github.com/TomKellyGenetics/R-Parallel-Lesson