parLapply in real life

PARALLEL PROGRAMMING IN R

Nabeel Imam
Data Scientist

datacamp

Let's meet the workers

```
cluster <- makeCluster(4)
```

```
clusterEvalQ(cluster, {
  id <- Sys.getpid()
  print(
    paste("Hello, my worker ID is", id)
  )
})
```

```
[[1]]
[1] "Hello, my worker ID is 425108"

[[2]]
[1] "Hello, my worker ID is 425129"

[[3]]
[1] "Hello, my worker ID is 425150"

[[4]]
[1] "Hello, my worker ID is 425171"
```

Filtering data in parallel

```
print(file_list)
```

```
 [1] "./health/Afghanistan.csv"
 [2] "./health/Albania.csv"
 [3] "./health/Algeria.csv"
 [4] "./health/American Samoa.csv"
 [5] "./health/Andorra.csv"
...
```

Filtering data in parallel

```
filterCSV <- function (csv) {
  read.csv(csv) %>%
    dplyr::filter(!is.na(health_exp_pc))
}

cl <- makeCluster(4)
ls_df <- parLapply(cl, file_list, filterCSV)
stopCluster(cl)
```

```
Error in checkForRemoteErrors(val) :
  first error: could not find function "%>%"
```

clusterEvalQ to the rescue

Load a package on the cluster

```
cl <- makeCluster(4)
clusterEvalQ(cl, library(dplyr))

ls_df <- parLapply(cl, file_list, filterCSV)
stopCluster(cl)
```

Load multiple packages on the cluster

```
clusterEvalQ(cl, {
  library(dplyr)
  library(stringr)
})
```

```
[[1]]
       Country health_exp_pc Year
1  Afghanistan      81.27103 2002
2  Afghanistan      82.45785 2003
3  Afghanistan      89.47005 2004
...

[[2]]
   Country health_exp_pc Year
1  Albania       300.2757 2001
2  Albania       314.3254 2002
3  Albania       343.9442 2003
...
```

Filtering with conditions

```
# Function with an argument for starting year
filterCSV <- function (csv, min_year) {

  read.csv(csv) %>%
    dplyr::filter(!is.na(health_exp_pc),

                  # Filter data for min_year and onwards
                  Year >= min_year)
}

selected_year <- 2010 # Value to be supplied to min_year
```

Filtering with conditions

```
cl <- makeCluster(4)
clusterEvalQ(cl, library(dplyr))
clusterExport(cl, "selected_year",
              envir = environment())

ls_df <- parLapply(cl, file_list, filterCSV,
                   min_year = selected_year)
stopCluster(cl)
```

- Export `selected_year` to cluster
- Export from current environment

- Supply `selected_year` to `min_year`

Filtering with conditions

```
[[1]]
       Country health_exp_pc Year
1  Afghanistan      143.6695 2010
2  Afghanistan      143.0915 2011
3  Afghanistan      151.9180 2012
...
  Country health_exp_pc Year
1 Albania      451.8820 2010
2 Albania      485.5835 2011
3 Albania      529.6322 2012
...
```

Cluster hygiene checklist

- Determine the number of cores
- Create appropriate cluster
  - PSOCK for compatibility on all systems
  - FORK for Linux or Mac (and speed!)
- Load any libraries needed
- Export any variables needed
- Supply exported variable to the named argument
- Stop the cluster once done

```
n_cores <- detectCores() - 2

cluster <- makeCluster(n_cores)




clusterEvalQ(cluster, library(crucial_package))
clusterExport(cluster, "variable_we_need")

parLapply(cluster, ls_inputs, our_function,
          named_argument = variable_we_need)

stopCluster(cluster)
```

Let's practice!

PARALLEL PROGRAMMING IN R

The parallel apply family

PARALLEL PROGRAMMING IN R

Nabeel Imam
Data Science

datacamp

Weight gain during pregnancy

```
print(ls_weights)
```

```
$AK
  [1] 30 33 50 30 26 99 52 28 24 40 30 99 40
 [14] 20 20 39 25 30 20 33 99 54 30 99 30 38
...
$AL
   [1] 60 32 30 21 30 27 28 32 18 28 30 24 59
  [14] 30 19 30 15 17 18 53 26  5 25 41 50 20
...
$AR
  [1] 31 40 30 99 27 36 30 35 25 19 10 25 40
 [14] 33 37 12 34 99 43 26 28 99 37 30 26  6
...
```

[1] Natality Data from CDC Births

FORKing with mclapply

```
boot_mean <- function (weights) {

  est <- rep(0, 1e4)

  for (i in 1:1e4) {
    b <- sample(weights, replace = T)
    est[i] <- mean(b)
  }

  return(est)
}
```

```
ls_boot <- mclapply(ls_weights, boot_mean,
                    mc.cores = 4)
```

- Faster by 20 to 40%
- FORK cluster generated in the background
  - No cluster set up required
  - Does not work on Windows

Balance the load with parLapplyLB

```
lapply(ls_weights, length)
```

```
  AK    AL    AR    AZ    CA    CO    CT   DC    DE    FL    GA    HI    IA  ...
 260  1503   816  1649 14869  1286  1151  520   295  4693  2824   510   891  ...
```

[1] Designed by Freepik

Balance the load with parLapplyLB

```
microbenchmark(
  "parLapply" = {
    cl <- makeCluster(4)
    ls <- parLapply(cl, ls_weights, boot_mean)
    stopCluster(cl)
  },
  "parLapplyLB" = {
    cl <- makeCluster(4)
    ls <- parLapplyLB(cl, ls_weights, boot_mean)
    stopCluster(cl)
  },
  times = 10
)
```

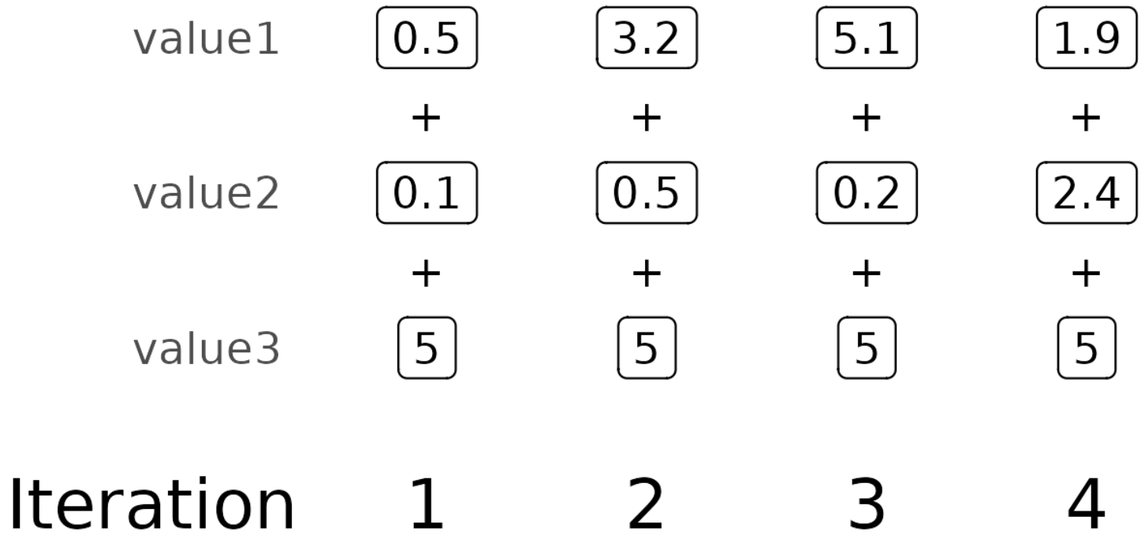
```
Unit: seconds
        expr      min     mean      max neval
  parLapply 15.77250 16.38548 16.83822    10
parLapplyLB 14.55609 15.49170 17.26866    10
```

- Average (mean) execution time drops from 16.4 seconds to 15.5 seconds (~ 5% reduction)

Multiple arguments to loop over

```
add <- function (x, y, z) x + y + z


value1 <- c(0.5, 3.2, 5.1, 1.9)
value2 <- c(0.1, 0.5, 0.2, 2.4)
value3 <- 5
```



clusterMap

```
cl <- makeCluster(4)

clusterMap(cl, add,
           x = value1,
           y = value2,
           z = value3) # Single value

stopCluster(cl)
```

```
print(value3)
```

```
[1] 5
```

```
[[1]]
[1] 5.6

[[2]]
[1] 8.5

[[3]]
[1] 10.2

[[4]]
[1] 8.4
```

clusterMap vs. parLapply

```
clusterMap(cl,          # Cluster
           add,         # Function
           x = value1, # Multiple inputs
           y = value2,
           z = value3) # Could be single values to recycle
```

```
parLapply(cl,                  # Cluster
          input,               # One input
          fun,                 # Function
          arg_name = static) # Exported static variables
                               # supplied to named argument
```

Weight gained with clusterMap

```
print(ls_weights)
```

```
$AK
  [1] 30 33 50 30 26 99 52 28 24 40 30 99 40
...
```

```
print(ls_plur)
```

```
$AK
  [1] 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1
  ...
```

```
boot_dist <- function (weights, pluralities) {
  est <- rep(0, 1e4)
  ratio <- weights/pluralities
  for (i in 1:1e4) {
    b <- sample(ratio, replace = T)
    est[i] <- mean(b)
  }
  return(est)
}

cl <- makeCluster(4)
result <- clusterMap(cl, boot_dist,
                     weights = ls_weights,
                     pluralities = ls_plur)

stopCluster(cl)
```

Total number of births

```
print(mat_births)
```

```
     1   2   3   4   5   6   7   8   9  10  11  12
AK   7  13   8   8   5   9  12  19  16  16   7  16
AL  55  58  79  66  48  63  69  81  74  76  61  82
AR  28  28  41  37  40  40  42  52  40  44  33  38
AZ  69  70  70  71  68  61  73  81  76  74  87  62
CA 603 599 659 601 669 719 675 687 660 698 703 633
CO  50  55  72  56  63  56  56  70  58  48  53  58
CT  51  51  57  58  59  41  51  46  54  56  58  52
DC  25  18  14  40  22  19  24  25  20  26  20  25
...
```

Row and column operations in parallel

```
cl <- makeCluster(4)

# parCapply for column operations
parCapply(cl, mat_births, sum)



# parRapply for row operations
parRapply(cl, mat_births, sum)

stopCluster(cl)
```

- Column-wise sum

```
 1    2    3    4    5    6    7
4258 3966 4437 4345 4371 4482 4614
 8    9   10   11   12
4738 4686 4483 4427 4247
```

- Row-wise sum

```
AK   AL   AR   AZ   CA   CO   CT
136  812  463  862 7906  695  634
DC   DE   FL   GA   HI   IA   ID
278  151 2519 1512  254  472  237
...
```

Let's practice!

PARALLEL PROGRAMMING IN R

Using foreach

PARALLEL PROGRAMMING IN R

Nabeel Imam
Data Scientist

datacamp

A new loop

Native for loop in R

```
numbers <- 1:1e6

sqroots <- rep(0, length(numbers))

for (i in 1:length(numbers)) {
  sqroots[i] <- sqrt(numbers[i])
}
```

The foreach loop

```
numbers <- 1:1e6

library(foreach)

sqroots <- foreach(i = numbers) %do% {
  sqrt(i)
}
```

Parallel loops

```
numbers <- 1:1e6

sqroots <- foreach(i = numbers) %do% {
  sqrt(i)
}
```

```
cl <- makeCluster(4)

library(doParallel)
registerDoParallel(cl)

sqroots <- foreach(i = numbers
                   # The parallel operator
                   ) %dopar% {
  sqrt(i)
}

stopCluster(cl)
```

Top engineering universities

```
print(uni_list)
```

```
 [1] "./uni_data/Argentina.csv"
 [2] "./uni_data/Australia.csv"
 [3] "./uni_data/Austria.csv"
 [4] "./uni_data/Azerbaijan.csv"
 [5] "./uni_data/Bahrain.csv"
 [6] "./uni_data/Bangladesh.csv"
 [7] "./uni_data/Belarus.csv"
 [8] "./uni_data/Belgium.csv"
 [9] "./uni_data/Bolivia.csv"
[10] "./uni_data/Bosnia and Herzegovina.csv"
...
```

```
cl <- makeCluster(4)
registerDoParallel(cl)

ls_df <- foreach(csv = uni_list) %dopar% {
  read.csv(csv)
}

stopCluster(cl)
```

Collecting results with foreach

```
cl <- makeCluster(4)
registerDoParallel(cl)
ls_df <- foreach(csv = uni_list) %dopar% {
  read.csv(csv)
}
stopCluster(cl)
```

```
[[1]]
 location                  institution score
Argentina  Universidad de Buenos Aires  68.9
...
[[2]]
 location                    institution score
Australia  Australian National University  82.1
...
```

```
cl <- makeCluster(4)
registerDoParallel(cl)

df_uni <- foreach(csv = uni_list,
                  .combine = "rbind") %dopar% {
  read.csv(csv)
}
stopCluster(cl)
```

```
   location                      institution score
1 Argentina     Universidad de Buenos Aires  68.9
2 Argentina  Universidad Católica Argentina  33.3
3 Argentina     Universidad de Palermo (UP)  29.1
...
```

Read, filter, and combine

```
library(dplyr)

n_unis <- 3

# Empty list
ls_df <- list()

for (i in 1:length(uni_list)) {
  # Read, filter, collect in empty list
  ls_df[[i]] <- read.csv(uni_list[[i]]) %>%
    top_n(n_unis, total_score)

}
# Combine the list into one
combined_df <- Reduce("rbind", ls_df)
```

foreach for the win

```
n_unis <- 3

cl <- makeCluster(4)
registerDoParallel(cl)

df_top3 <- foreach(csv = uni_list,
                   .packages = "dplyr",
                   .export = "n_unis",
                   .combine = "rbind") %dopar% {

  read.csv(csv) %>%
    top_n(n_unis, score)
}

stopCluster(cl)
```

```
 location                    institution score
Argentina     Universidad de Buenos Aires  68.9
Argentina  Universidad Católica Argentina  33.3
Argentina     Universidad de Palermo (UP)  29.1
Australia  Australian National University  82.1
Australia     The University of Melbourne  81.6
Australia        The University of Sydney  79.6
  Austria             University of Vienna  50.6
  Austria     Technische Universität Wien  45.7
...
```

Let's practice!

PARALLEL PROGRAMMING IN R

Advanced foreach operations

PARALLEL PROGRAMMING IN R

Nabeel Imam
Data Scientist

datacamp

The case of the enormous CSV

```
df <- read.csv("very_large.csv")
```

```
Error: cannot allocate vector of size 6286.7 Mb
In addition: There were 12 warnings (use warnings() to see them)
```

Iterators are it

```
library(iterators)
it <- ireadLines("very_large.csv")
nextElem(it)
```

```
"\"\",\"Date\",\"Price\",\"Company\""
```

- Read first line in CSV

```
nextElem(it)
```

```
[1] "\"1\",\"2015-01-02\",312.579987,\"Amazon\""
```

- And the next

```
nextElem(it)
```

```
[1] "\"2\",\"2015-01-05\",307.01001,\"Amazon\""
```

- And so on...

Iterators with foreach

```
foreach(line = ireadLines("very_large.csv"),
        .combine = "rbind") %do% {

  if (grepl("Tesla", line)) return(line)

}
```

```
result.2  "\"2011-01-03\",5.368,\"Tesla\""
result.3  "\"2011-01-04\",5.332,\"Tesla\""
result.4  "\"2011-01-05\",5.296,\"Tesla\""
result.5  "\"2011-01-06\",5.366,\"Tesla\""
result.6  "\"2011-01-07\",5.6,\"Tesla\""
...
```

%dopar% the CSV

```
cl <- makeCluster(4)
registerDoParallel(cl)

foreach(line =  ireadLines("very_large.csv"),
        .combine = "rbind") %dopar% {

  if (grepl("Tesla", line)) {
    return(
      strsplit(gsub("\"", "", line), ",")[[1]]
    )
  }
}

stopCluster(cl)
```

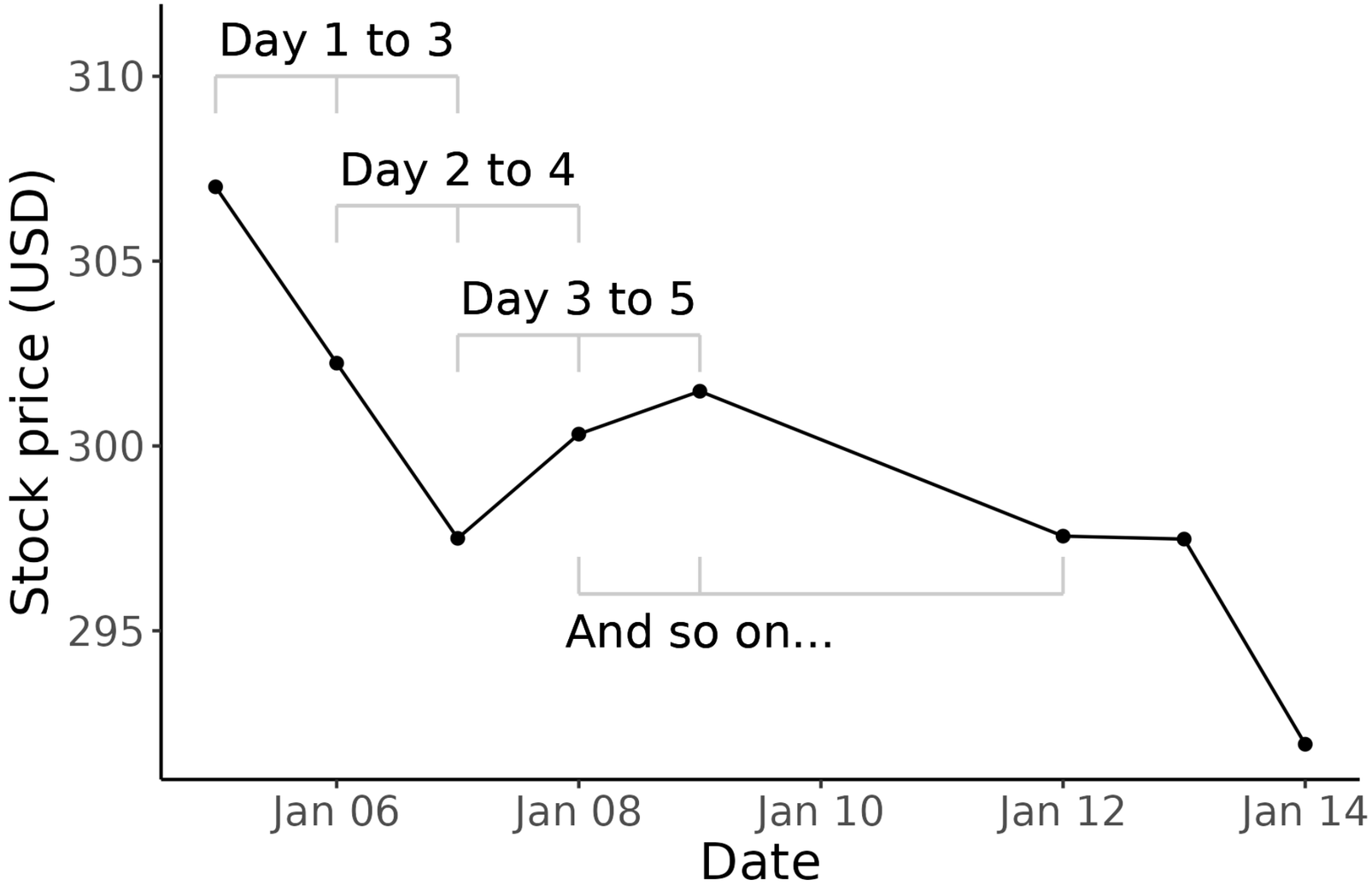
```
          [,1]         [,2]    [,3]
result.2  "2011-01-03" "5.368" "Tesla"
result.3  "2011-01-04" "5.332" "Tesla"
result.4  "2011-01-05" "5.296" "Tesla"
result.5  "2011-01-06" "5.366" "Tesla"
result.6  "2011-01-07" "5.6"   "Tesla"
...
```

Stock prices

```
head(df_stocks)
```

```
            Amazon   Apple Facebook   Google Microsoft  Netflix  Tesla Uber Walmart Zoom
2015-01-02 312.58 27.8475    78.58 527.5616     46.66 49.15143 44.574   NA   86.27   NA
2015-01-05 307.01 27.0725    77.98 521.8273     46.37 49.25857 42.910   NA   85.72   NA
2015-01-06 302.24 26.6350    77.23 513.5900     46.38 47.34714 42.012   NA   85.98   NA
2015-01-07 297.50 26.8000    76.76 505.6118     45.98 47.34714 42.670   NA   86.78   NA
2015-01-08 300.32 27.3075    76.74 496.6265     46.75 47.12000 42.562   NA   89.21   NA
2015-01-09 301.48 28.1675    78.20 503.3780     47.61 47.63143 41.784   NA   90.32   NA
...
```

Three-day moving average



Three-day moving average for Tesla stock

```
df_tesla <- df_stocks %>%
  dplyr::select(Tesla)

n_rows <- nrow(df_tesla) - 2
```

```
cl <- makeCluster(4)
registerDoParallel(cl)

tesla_ma3 <- foreach(row = 1:n_rows) %dopar% {
  mean(df_tesla$Tesla[row:(row + 2)])
}

stopCluster(cl)
```

```
print(tesla_ma3)
```

```
[[1]]
[1] 43.16533

[[2]]
[1] 42.53067

[[3]]
[1] 42.41467

[[4]]
[1] 42.33867
...
```

Moving average for all columns

```
head(df_stocks)
```

```
            Amazon   Apple Facebook   Google Microsoft  Netflix  Tesla Uber Walmart Zoom
2015-01-02 312.58 27.8475    78.58 527.5616     46.66 49.15143 44.574   NA   86.27   NA
2015-01-05 307.01 27.0725    77.98 521.8273     46.37 49.25857 42.910   NA   85.72   NA
2015-01-06 302.24 26.6350    77.23 513.5900     46.38 47.34714 42.012   NA   85.98   NA
2015-01-07 297.50 26.8000    76.76 505.6118     45.98 47.34714 42.670   NA   86.78   NA
2015-01-08 300.32 27.3075    76.74 496.6265     46.75 47.12000 42.562   NA   89.21   NA
2015-01-09 301.48 28.1675    78.20 503.3780     47.61 47.63143 41.784   NA   90.32   NA
```

- Loop over all columns
- Within each column iteration, loop over rows

Nested loops

```
n_rows <- nrow(df_stocks) - 2
n_cols <- ncol(df_stocks)

cl <- makeCluster(4)
registerDoParallel(cl)

# Iterate over columns
moving_avg <- foreach(col = 1:n_cols,
                      .combine = "cbind") %:%
# Iterate over rows
              foreach(row = 1:n_rows,
                      .combine = "c") %dopar% {
# Average from day n to n + 2 (3-days)
    mean(df_stocks[row:(row + 2), col])
}
stopCluster(cl)
```

- Get dimensions to iterate over
- Set up the cluster
- Nest loops with `%:%`
- Use `%dopar%` for the inner-most loop; supply `c()` to combine results into a vector

Moving averages

```
head(moving_avg)
```

```
     result.1 result.2 result.3 result.4 result.5 result.6 result.7 ...
[1,] 307.2767 27.18500 77.93000 520.9930 46.47000 48.58571 43.16533 ...
[2,] 302.2500 26.83583 77.32334 513.6764 46.24333 47.98428 42.53067 ...
[3,] 300.0200 26.91417 76.91000 505.2761 46.37000 47.27143 42.41467 ...
[4,] 299.7667 27.42500 77.23333 501.8721 46.78000 47.36619 42.33867 ...
[5,] 299.7867 27.87500 77.59333 497.8631 47.26000 47.28048 41.65200 ...
[6,] 298.8400 28.05833 77.75667 498.1457 47.33333 46.91428 41.01933 ...
...
```

Let's practice!

PARALLEL PROGRAMMING IN R