Lab 2: Parallel Computing

Introduction to Statistical Computing

Elena Conderana & Sergio Cuenca

Contents

This lab is to be done outside of class time. You may collaborate with one classmate, but you must identify yourself and his/her name above, in the author's field, and you must submit **your own** lab as this completed .Rmd file.

Installing and loading packages

In order to perform the exercise in this practice you should install and load the `doParallel` package.

```
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 4.3.3
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 4.3.3
```

```
## Loading required package: iterators
```

```
## Warning: package 'iterators' was built under R version 4.3.3
```

```
## Loading required package: parallel
```

Q1. Is parallelization worth it?

1a. First, let's check how many cores do you have in your computer. Create a new variable `no_cores` equal to the number of cores minus 1.

```
no_cores <- parallel::detectCores()-1
```

1b. Register the cores and prepare the clusters using the `registerDoParallel()` and `makeCluster()` function. This will allow to parallelize code in the following code chunks.

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

1c. Now, you have the following function which calculates the prime numbers from `1` to `n`. Use the `microbenchmark` package to check which is faster to calculate the prime numbers when n goes from 10 to 10000: `lapply`, a `for` loop, `parLapply` or a `foreach` loop. The `lapply` and the `for` loop have been written for you. Which function is faster?

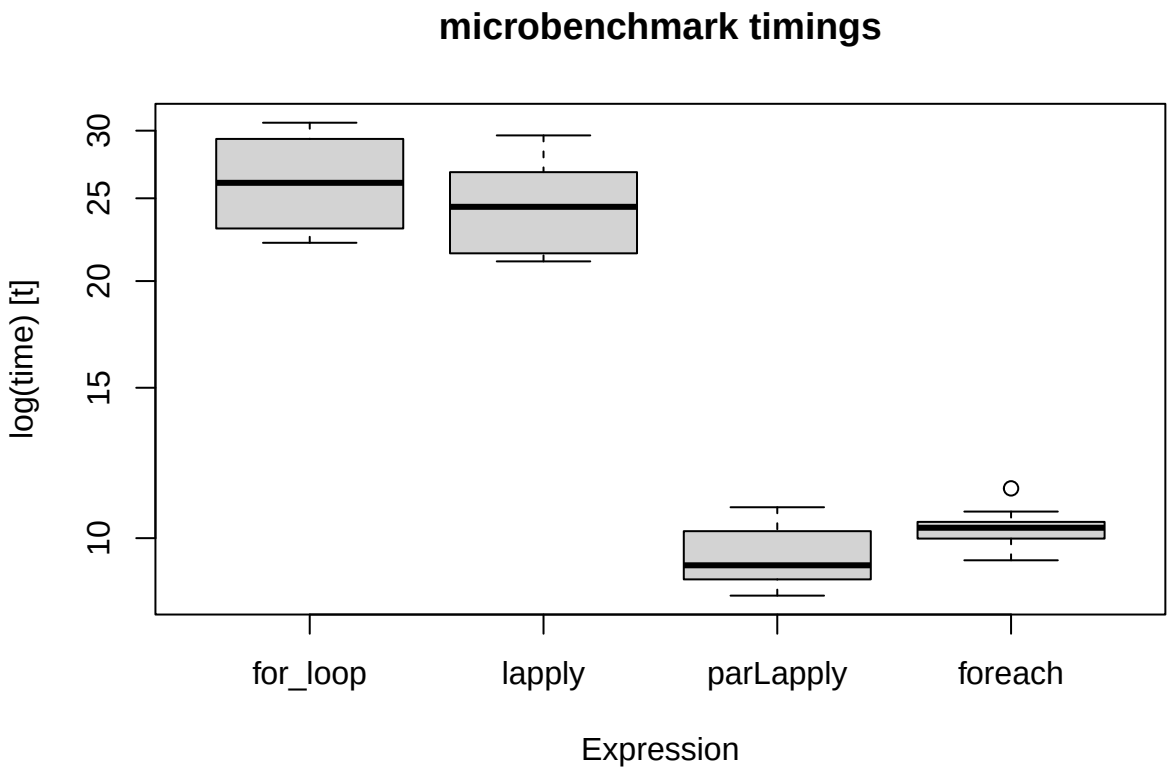
```
library(microbenchmark)
```

```
## Warning: package 'microbenchmark' was built under R version 4.3.3
```

```r
getPrimeNumbers <- function(n) {
   n <- as.integer(n)
   if(n > 1e6) stop("n too large")
   primes <- rep(TRUE, n)
   primes[1] <- FALSE
   last.prime <- 2L
   for(i in last.prime:floor(sqrt(n)))
   {
      primes[seq.int(2L*last.prime, n, last.prime)] <- FALSE
      last.prime <- last.prime + min(which(primes[(last.prime+1):n]))
   }
   which(primes)
}
n_vec <- 10:10000
lapplyPrimeNumbers <- function(n_vec) {
  result <- lapply(n_vec, getPrimeNumbers)
}
forPrimeNumbers <- function(n_vec) {
  result <- list()
  for (n in n_vec) {
    result[[n]] <- getPrimeNumbers(n)
  }
}

parLapplyPrimeNumbers <- function(n_vec, cl) {
  clusterExport(cl, "getPrimeNumbers")
  result <- parLapply(cl, n_vec, getPrimeNumbers)
}

foreachPrimeNumbers <- function(n_vec) {
  result <- foreach(n = n_vec, .export = "getPrimeNumbers") %dopar% {
    getPrimeNumbers(n)
  }
}

bench_results <- microbenchmark(
  for_loop = forPrimeNumbers(n_vec),
  lapply = lapplyPrimeNumbers(n_vec),
  parLapply = parLapplyPrimeNumbers(n_vec, cl),
  foreach = foreachPrimeNumbers(n_vec),
  times = 10
)

print(bench_results)
```

```
## Unit: seconds
##       expr       min        lq     mean    median       uq      max neval
##   for_loop 22.173259 23.046202 26.24267 26.065240 29.33905 30.64889    10
##     lapply 21.090017 21.552563 24.63915 24.431811 26.82444 29.61922    10
##  parLapply  8.565109  8.947913  9.53242  9.294351 10.19034 10.86899    10
##    foreach  9.420767  9.988525 10.26304 10.284520 10.44830 11.43652    10
```

```r
boxplot(bench_results)
```



1d Remember to use stop the clusters in `cl` using the `stopCluster` function.

```r
stopCluster(cl)
```

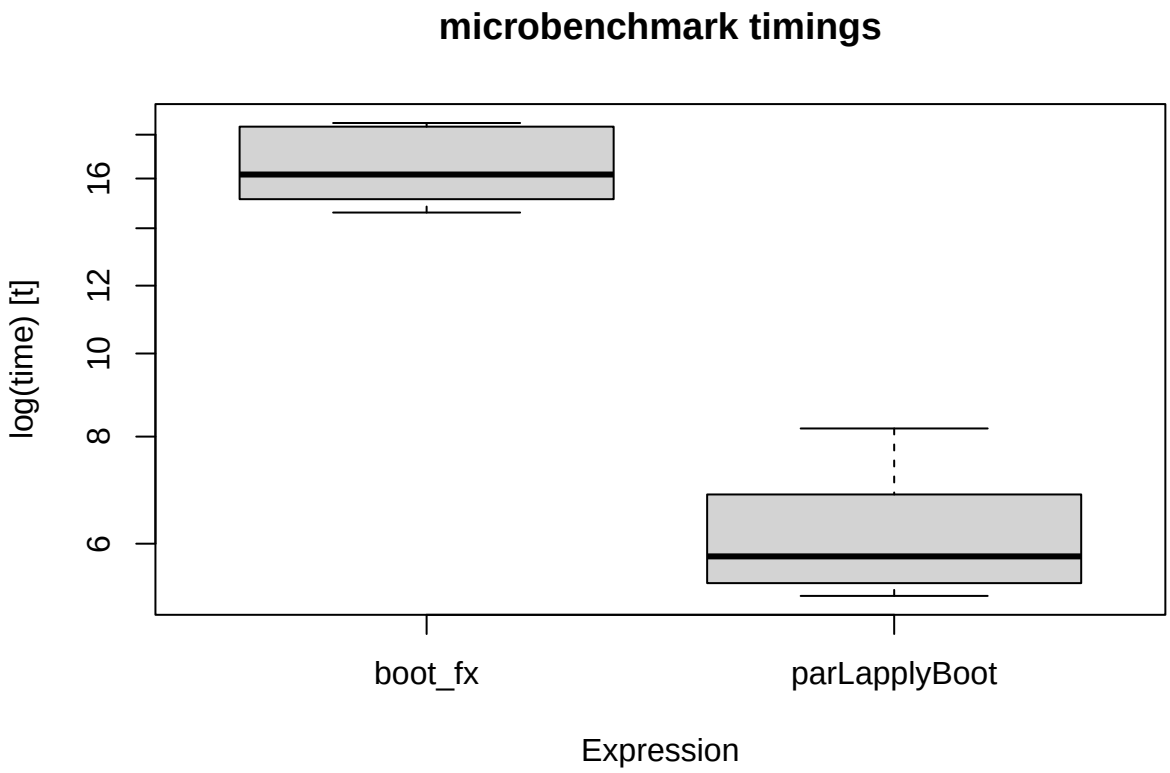
Challenge 01. Search around your computer for a sequential code that might be parallelized. Using the `doParallel` package, parallelize the code and calculate the speedup. If you cannot find any code to parallelize, use the following code:

```r
x <- iris[which(iris[,5] != "setosa"), c(1,5)]
trials <- seq(1, 10000)
boot_fx <- function(trial) {
  ind <- sample(100, 100, replace=TRUE)
  result1 <- glm(x[ind,2]~x[ind,1], family=binomial(logit))
  r <- coefficients(result1)
  res <- rbind(data.frame(), r)
}

cl <- makeCluster(no_cores)
clusterExport(cl, c("boot_fx", "x"))

parLapplyBoot <- function(trial, cl) {
  result <- parLapply(cl, trial, boot_fx)
}
```

```r
bench_results_boot <- microbenchmark(
  boot_fx = lapply(trials, boot_fx),
  parLapplyBoot = parLapplyBoot(trials, cl),
  times = 10
)

print(bench_results_boot)
```

```
## Unit: seconds
##           expr       min        lq      mean    median        uq       max
##        boot_fx 14.602741 15.135870 16.547065 16.173434 18.392419 18.573014
##  parLapplyBoot  5.215137  5.397126  6.181511  5.799208  6.848456  8.176862
##  neval
##     10
##     10
```

```r
boxplot(bench_results_boot)
```




```r
stopCluster(cl)
```