Derived functions from read. table () have a lot of convenient features, but it seems like there is a lot of steps in the implementation that would slow things down. The gain in performance of reading 29 CSV files (about 2.2 GB) shows quite different picture. While the parallelization process does bring some improvement considering the ‘user time’, i.e. the CPU time charged for the process execution at the machine level, the ‘elapsed time’, i.e. the ‘real’ elapsed time since the process was started doesn’t show much difference. Let’s go through it.

**Data**

list\_of\_datasets <- list.files(pattern = "\*.csv")

list\_of\_datasets

[1] "despesas\_contratadas\_candidatos\_2018\_AC.csv"

[2] "despesas\_contratadas\_candidatos\_2018\_AL.csv"

[3] "despesas\_contratadas\_candidatos\_2018\_AM.csv"

[4] "despesas\_contratadas\_candidatos\_2018\_AP.csv"

[5] "despesas\_contratadas\_candidatos\_2018\_BA.csv"

[6] "despesas\_contratadas\_candidatos\_2018\_BR.csv"

[7] "despesas\_contratadas\_candidatos\_2018\_BRASIL.csv"

[8] "despesas\_contratadas\_candidatos\_2018\_CE.csv"

[9] "despesas\_contratadas\_candidatos\_2018\_DF.csv"

[10] "despesas\_contratadas\_candidatos\_2018\_ES.csv"

[11] "despesas\_contratadas\_candidatos\_2018\_GO.csv"

[12] "despesas\_contratadas\_candidatos\_2018\_MA.csv"

[13] "despesas\_contratadas\_candidatos\_2018\_MG.csv"

[14] "despesas\_contratadas\_candidatos\_2018\_MS.csv"

[15] "despesas\_contratadas\_candidatos\_2018\_MT.csv"

[16] "despesas\_contratadas\_candidatos\_2018\_PA.csv"

[17] "despesas\_contratadas\_candidatos\_2018\_PB.csv"

[18] "despesas\_contratadas\_candidatos\_2018\_PE.csv"

[19] "despesas\_contratadas\_candidatos\_2018\_PI.csv"

[20] "despesas\_contratadas\_candidatos\_2018\_PR.csv"

[21] "despesas\_contratadas\_candidatos\_2018\_RJ.csv"

[22] "despesas\_contratadas\_candidatos\_2018\_RN.csv"

[23] "despesas\_contratadas\_candidatos\_2018\_RO.csv"

[24] "despesas\_contratadas\_candidatos\_2018\_RR.csv"

[25] "despesas\_contratadas\_candidatos\_2018\_RS.csv"

[26] "despesas\_contratadas\_candidatos\_2018\_SC.csv"

[27] "despesas\_contratadas\_candidatos\_2018\_SE.csv"

[28] "despesas\_contratadas\_candidatos\_2018\_SP.csv"

[29] "despesas\_contratadas\_candidatos\_2018\_TO.csv"

**Function**

library(pbapply)

library(parallel)

library(doParallel)

library(dplyr)

#' Reads a list of datasets

#' @param x A list of datasets (names of datasets are strings)

#' @param func A function, the read function to use to read the data

#' @param parallelize Parallelize the code

#' @param ... Further arguments passed to func

readListFaster <- function(x, func, ..., parallelize = FALSE, rbind = FALSE){

stopifnot(length(x) > 0)

read\_and\_assign <- function(dataset, func){

dataset\_name <- as.name(dataset)

dataset\_name <- func(dataset, ...)

}

if (parallelize) {

message("Reading in data in parallel")

clusters <- parallel::detectCores() %>%

parallel::makeCluster()

doParallel::registerDoParallel(clusters)

output <- invisible( # invisible is used to suppress the unneeded output

pbapply::pblapply(x,

read\_and\_assign,

func = func,

...,

cl = clusters)

)

parallel::stopCluster(clusters)

} else if (!parallelize) {

output <- invisible(

pbapply::pblapply(x,

read\_and\_assign,

func = func)

)

}

# Remove what's after the "." at the end of the data set names and what's before any / for url files.

x <- stringr::str\_replace\_all(x,".\*/|\\..\*", "")

names(output) <- x

if (rbind) {

dplyr::bind\_rows(output)

} else {

output

}

}

**Results**

**Without Paralelize**

system.time(loaded\_datasets1 <- readListFaster(list\_of\_datasets,

func = read.csv2,

parallelize=FALSE,

rbind = TRUE))

user system elapsed

229.852 15.221 264.272

**With Paralelize**

system.time(loaded\_datasets2 <- readListFaster(list\_of\_datasets,

func = read.csv2,

parallelize=TRUE,

rbind = TRUE))

user system elapsed

74.463 24.082 261.827

It’s important to realise that every bit of optimisation matters. But it would not help us if the outcome data frames were different, don’t you agree? Luckily all 174 million records do match.

table(loaded\_datasets1 == loaded\_datasets2)

TRUE

174053476