Práctica 3: Teoría de colas

Dulce Esperanza Carrasco Castillo 1445183

12 de febrero de 2019

1. Objetivo

Conocer que la rapidez de ejecución de tareas en paralelo depende de la cantidad de núcleos que realmente están trabajando y también del orden en el que se estén ejecutando las tareas [**?**].

2. Descripción

En esta práctica se midió el tiempo que se tarda realizar ciertas tareas, se utilizó la combinación de números primos y no primos en diferente proporción y se varió la cantidad de núcleos trabajando. Primero se verificó la cantidad de núcleos con los que cuenta el dispositivo en el que se trabajó usando el comando: detectCores(), se creó un vector para los números primos y otro para los números no primos (misma cantidad de números de cada uno) 1, después se hizo otro vector para combinar los números primos y no primos y que tomara los números aleatoriamente. Para variar la proporción se disminuía la cantidad de números a un vector(primos) y se le agregaban al otro(noprimos) para mantener la misma cantidad de números totales. Se corrieron para 3,2 y 1 núcleo. Se repitió el proceso para números de 7 y 9 dígitos.

3. Resultados

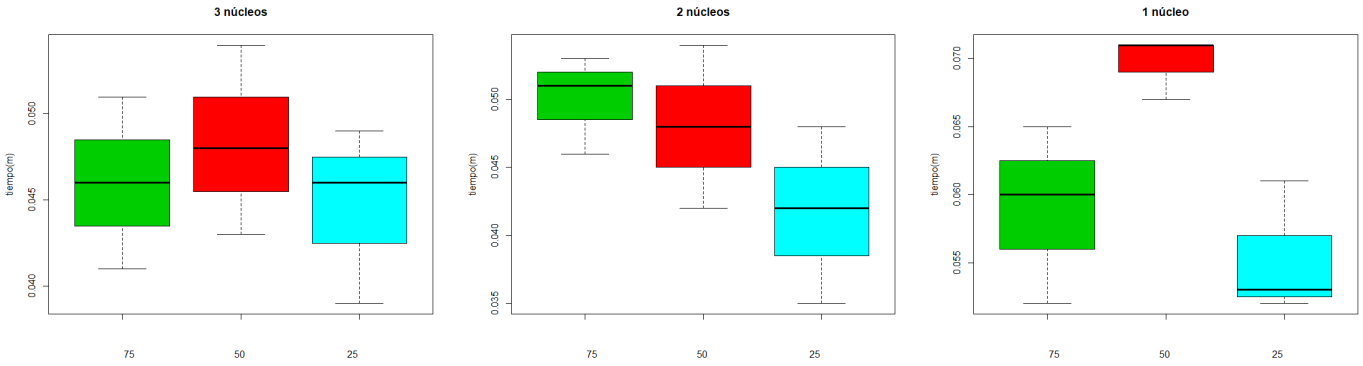
Se observa en la figura 2 que al usar la mayor cantidad de núcleos los tiempos de ejecución no varían mucho pero si se lleva más tiempo cuando está en una proporción 50/50, de manera que al disminuir la cantidad de primos o noprimos disminuye el tiempo de ejecución. Con dos núcleos, el trabajo se reparte de manera más ordenada y con un núcleo se puede ver que los números primos hacen la tarea más pesada.

En la figura 3 se utilizan números de 7 dígitos, se puede observar que al usar la mayor cantidad de núcleos los tiempos de ejecución son menores y va aumentando conforme se usan menos núcleos para realizar la tarea. Al usar menor cantidad de núcleos los números primos hacen la tarea más pesada y el tiempo de ejecución es mayor.

En la figura 4 se utilizaron números con 9 dígitos, se puede ver que los tiempos son más grandes que los tiempos con 5 y 7 dígitos, y hay mucha diferencia entre los tiempos que se tarda cada combinación, lo que vuelve a confirmar que donde hay mayor cantidad de números primos, mayor es el tiempo que se tarda en realizar la tarea, por lo que los números primos hacen la tarea pesada.

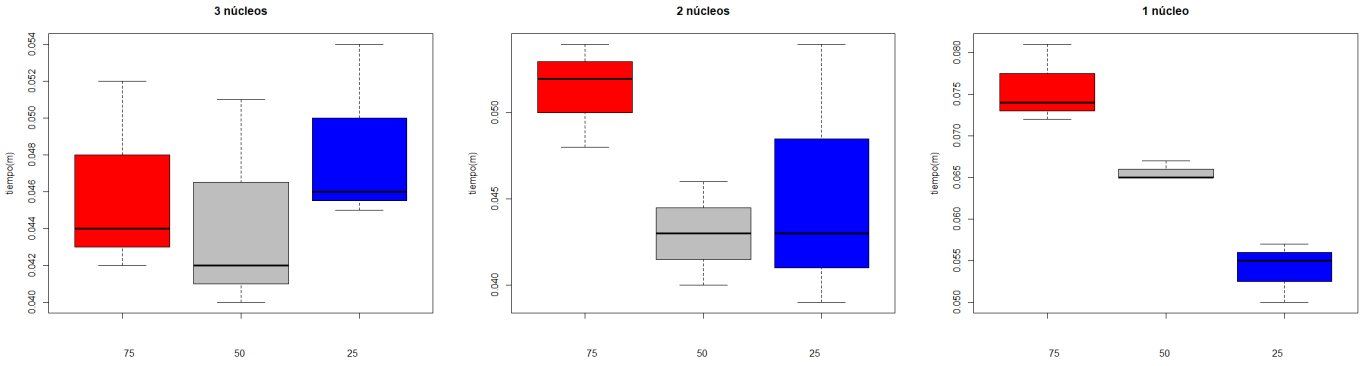
```r
primo <- function(n) {
  if (n == 1 || n == 2) {
    return(TRUE)
  }
  if (n %% 2 == 0) {
    return(FALSE)
  }
  for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
    if ((n %% i) == 0) {
      return(FALSE)
    }
  }
  return(TRUE)
}

noprimos=c(valores)
mispri=c(valores)

combinacion <- sort(sample(c(mispri, noprimos)))
original <- combinacion
invertido <- rev(combinacion)
replicas <- 10
suppressMessages(library(doParallel))
registerDoParallel(makeCluster(detectCores() - 3))
ot <-  numeric()
it <-  numeric()
at <-  numeric()
for (r in 1:replicas) {
  ot <- c(ot, system.time(foreach(n = original, .combine=c) %dopar% primo(n))[3]) # de menor a mayor
  it <- c(it, system.time(foreach(n = invertido, .combine=c) %dopar% primo(n))[3]) # de mayor a menor
  at <- c(at, system.time(foreach(n = sample(original), .combine=c) %dopar% primo(n))[3]) # orden aleatorio
}
stopImplicitCluster()
summary(ot)
summary(it)
summary(at)
```

Figura 1: Código



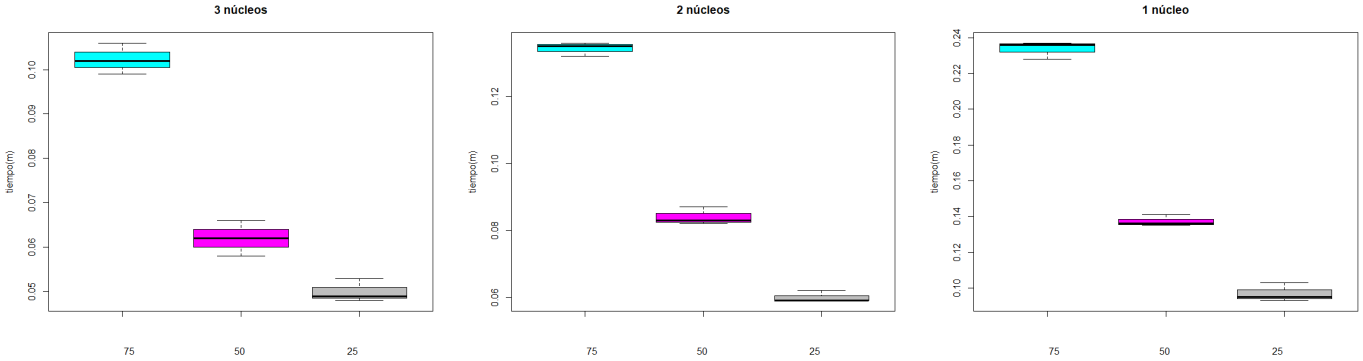
(a) (b) (c)

Figura 2: 5 dígitos



(a) (b) (c)

Figura 3: 7 dígitos



(a) (b) (c)

Figura 4: 9 dígitos