Reporte práctica 8
Modelo de urnas

1. INTRODUCCIÓN

En la práctica se simula un sistema en dónde se dan los fenómenos de coalescencia y fragmentación de partículas, además las partículas se pueden unir para formar cúmulos y estos cúmulos se pueden fragmentar. Esto puede servir en la práctica de laboratorio como para lograr predecir qué cantidad de partículas quedarán atrapadas en un filtro de cierta apertura de poro.

2. SIMULACIÓN

Se tiene una cierta cantidad de partículas *n* y el tamaño de los *k* cúmulos que existen siguen una distribución normal.

Determinamos un valor crítico de tamaño de cúmulos para poder unirse, a tamaños mayores a este valor crítico los cúmulos tienden a fragmentarse en dos pedazos no vacíos con tamaños distribuidos uniformemente al azar. Para observar la distribución de tamaños se convierten a frecuencias y evitar tener que lidiar con cúmulos individuales, ya que pueden ser muchos de un mismo tamaño.

Después los cúmulos que quieran unirse serán apuntados para uniones con la función *unirse*. Se forman pares al azar entre los que se quieran unir. Posteriormente se romperán los cúmulos propensos a la fragmentación mediante la función *romperse*.

Para la tarea base se solicita paralelizar el código eficientemente, para lograr esto se optó por las funciones que se realizan gran cantidad de veces y se pueden hacer independientemente, como *romperse*, *unirse* y *juntarse*. Para esto se utilizaron tres *forech* para realizar en paralelo estas funciones. Se modificó el código reemplazando el *for* existente por el *foreach* como se muestra:

```
#####################################inicia romperse
  #for (i in 1: dim(freq) [1]) {# fase de rotura
  f1<-function () {
    cumulos <- integer ()
    urna <- freq[i,]
    if (urna$tam > 1) {# no tiene caso romper si no se puede
    cumulos <- c (cumulos, romperse (urna$tam, urna$num))
    } else {
```

```
      cumulos <- c (cumulos, rep(1, urna$num))
    }
    return(cumulos)
  }
  cumulos=foreach (i=1: dim(freq) [1],.combine =c)%dopar%f1()

  ######################################termina romperse
```

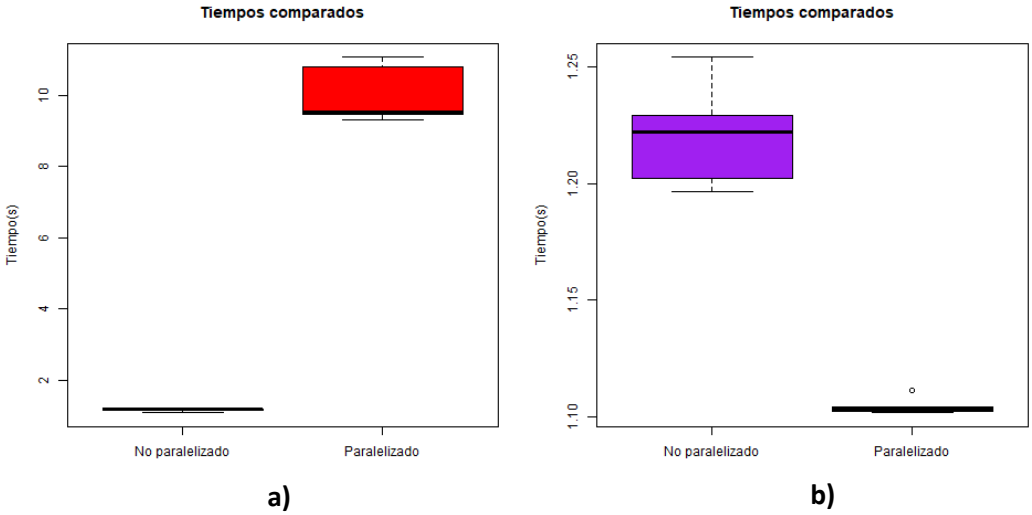
```
####################################incia unirse
  #for (i in 1: dim(freq) [1]) {# fase de union
   f2<-function () {
     cumulos <- integer ()
     urna <- freq[i,]
    cumulos <- c (cumulos, unirse (urna$tam, urna$num))
  return (cumulos)
  }
  cumulos=foreach (i=1: dim(freq) [1], .combine =c)%dopar%f2()

  ######################################termina unirse
```

```
####################################inicia juntarse
       # for (i in 1: floor (nt / 2)) {
      f3<-function () {
        suma <- juntarse[2*i-1] + juntarse[2*i]
        return(suma)
        }
    par=foreach (i=1: floor(nt / 2), .combine=c)%dopar%f3()
       cumulos<-c (positivos, par)
  ######################################termina juntarse
```

Al correr el código con el valor en *k* ya establecido no se observó una mejora en el tiempo de ejecución por lo que se procedió a aumentar el número de *k* a 100,000 en dónde ya se nota una disminución en tiempo.

Para realizar cinco réplicas de cada código y poder realizar el *boxplot* correspondiente se realizó un código en dónde mediante un *for* se fija la cantidad de réplicas y mediante *source* son llamados los dos programas.

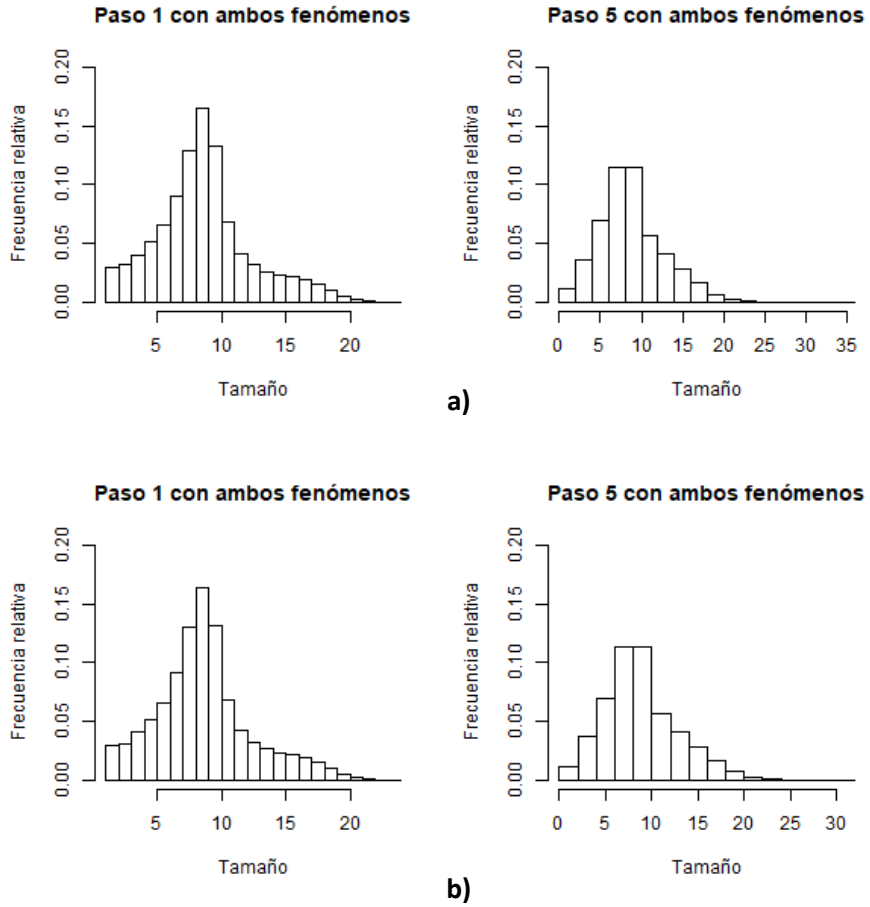
3. RESULTADOS



a) **b)**

Figura 1. Gráfica caja-bigote de 5 réplicas de cada código (paralelizado y no paralelizado) utilizando una *k* =10,000 (a) y *k* =100,000 (b).

Los resultados obtenidos muestran una diferencia de tiempos de ejecución entre el programa paralelo y no paralelo; aunque en el caso de la *k=10,000* no existe una mejora en el tiempo en paralelo (figura 1a) para *k=100,000* se reduce considerablemente el tiempo de ejecución (figura 1b).



a)

b)

Figura 2. Distribución de frecuencias en paso uno y cinco para el código no paralelizado (a) y el paralelizado (b)

Para verificar que no se modificó la distribución de las frecuencias de tamaños en el código paralelizado se muestran las frecuencias en el paso uno y cinco de ambos códigos (paralelizado y no paralelizado), figura 2. Se observa que no existe diferencia notable entre las frecuencias de ambos códigos en los pasos inicial y final.

4. RETOS

- Reto 1

En el reto uno se pide que se varíe el valor de *k* y que *n* sea igual a *30k*, para esto se utilizó la herramienta *source* para llamar los códigos y correrlos con tres diferentes *k* (100,000, 150,000 y 200,000) en cinco réplicas cada *k*. Lo anterior mediante dos *for* uno controlando el número de réplicas y el otro controla el valor de *k*.

Se grafica mediante la librería *ggplot2* y se dividen los valores de *k* entre mil para hacer cifras más sencillas y fáciles de visualizar en la gráfica.

```
Tiempos<-data.frame()
Tnp<-numeric ()

Tp<-numeric ()
for (corrida in 1:5) {
for (k in seq (100000,200000,50000)) {

    source ('~/GitHub/SimulacionComputacional/P8/codigobase.R',
encoding = 'UTF-8')

  Tnp <- cbind ("original", tiempo, k, corrida)

  Source ('~/GitHub/SimulacionComputacional/P8/prueba2paralelo.R',
encoding = 'UTF-8')

  Tp <- cbind ("paralelo",tiempo, k, corrida)

  Tiempos<-  rbind (Tiempos,Tnp, Tp)
  }

}
```

Sabiendo que, si existe una diferencia en los tiempos, se procedió a realizar la prueba estadística de *t Student* para calcular los *p-value* y verificar si la diferencia es o no significativa

```
load("C:/Users/ede_p/OneDrive/Documentos/GitHub/SimulacionComputacional/P8/datosR1.RData")

colnames(Tiempos)<-c("Tipo","Tiempo","k","corrida")
Tiempos$Tiempo<- as.numeric(levels(Tiempos$Tiempo))[Tiempos$Tiempo]
Tiempos[Tiempos$Tiempo>30,4]<-Tiempos[Tiempos$Tiempo>30,4]/60

for (k in c(100,150,200)){
  tstparalelo<-Tiempos [Tiempos$k==k& Tiempos$Tipo=="paralelo",]
  tstoriginal<-Tiempos [Tiempos$k==k & Tiempos$Tipo=="original",]

  datospara<-tstparalelo$Tiempo
  datosorig<-tstoriginal$Tiempo

  pval<-t.test(datosorig,datospara)
  print(pval)
}
```

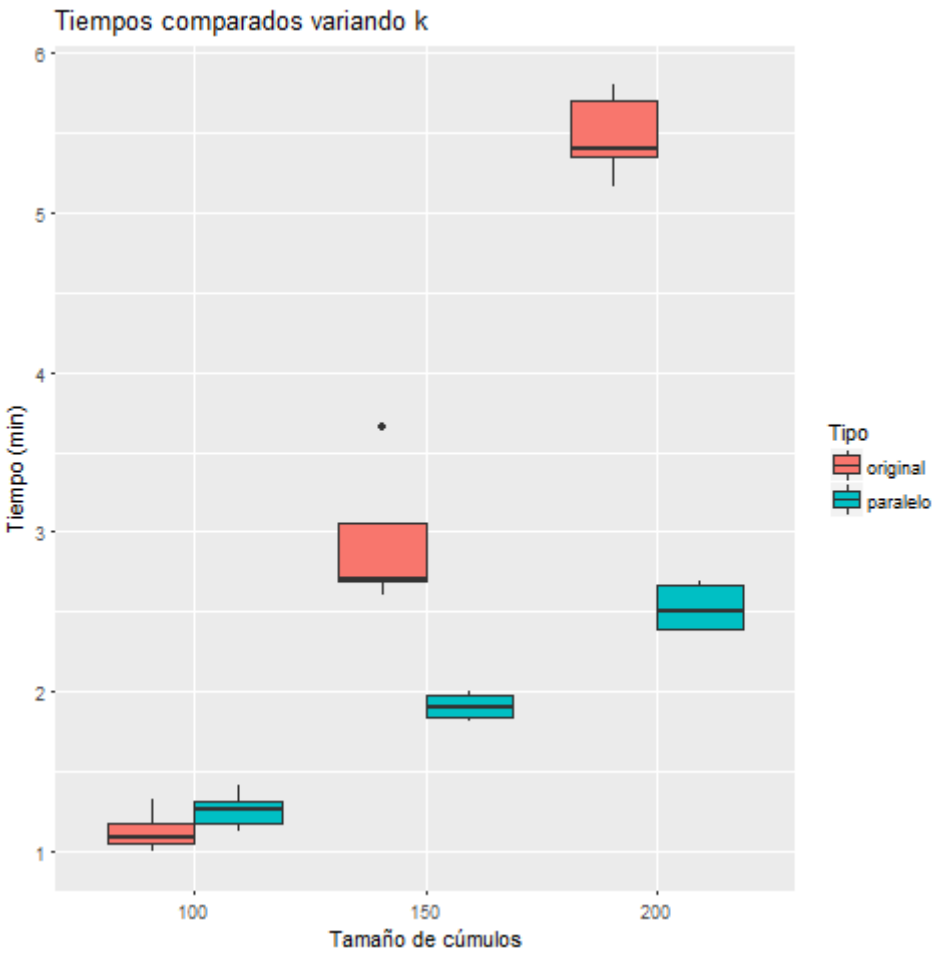


Figura 3. Boxplot de los tiempos con cinco corridas de los códigos con tres valores diferentes de *k* (100,000; 150,000 y 200000) y *n=30k*.

En la figura 3 se muestra que al cambiar a *n=30k* la mejoría en tiempo se observa hasta *k* en 150,000; al aumentar su valor cada vez resulta que el código en paralelo se ejecuta más rápido y se aleja más del tiempo de ejecución del código original.

Tabla 1. *Valores de p-value obtenidos por la prueba t Student para tres valores de k*

Valores de *k*	p-value	Significancia
100,000	0.1157	No significativo
150,000	5.277×10^{-3}	Altamente significativo
200,000	4.142×10^{-7}	Altamente significativo

Para validar estadísticamente la diferencia en tiempos de ejecución (figura 3) se procedió a realizar la prueba *t* Student para tres valores de *k* resultando en diferencias altamente significativas para dos de los valores, cuando el código paralelo es más rápido que el original y no es significativa la diferencia en donde el código en paralelo se ejecuta en mayor tiempo, como se resume en la tabla 1.

5. CONCLUSIONES

Para valores de *k* menores a 100,000 en el código secuencial es más eficiente que el paralelizado.

Se logró paralelizar el código sin cambiar en gran medida la distribución de frecuencias comprado con el código original.

Al cambiar *n=30k*, a partir de 150,000 para *k* se observa que el código en paralelo disminuye el tiempo de ejecución con una diferencia altamente significativa.

Entre más grande sea el valor de *k* el código en paralelo resulta más eficiente.

Al modificar el valor en *n*, varía un poco el tiempo de ejecución y se tiene que aumentar el valor en *k* para observar la mejora con el código en paralelo.