Práctica 8: modelo de urnas

Gabriela Sánchez Y.

Introducción

Se simula un fenómeno de coalescencia y fragmentación, donde partículas se unen para formar cúmulos y éstos se pueden volver a descomponer en fragmentos menores.

Supóngase que se tiene una cantidad total de n partículas y que al inicio el tamaño de los k cúmulos existentes sigue la distribución normal.

Además, se supone que la mediana de los tamaños iniciales corresponde al tamaño crítico c: cúmulos menores a c solamente pueden pegarse uno al otro y quedarse como son, pero tamaños $\geq c$ pueden además fragmentarse. La fragmentación es tal que, si un cúmulo se rompe, siempre resulta en dos pedazos no vacíos, cuyos tamaños se determinan uniformemente al azar.

La probabilidad de rotura se modela con la curva sigmoidal

$$\frac{1}{1+e^{\frac{c-x}{d}}},$$

donde d es un factor arbitrario que suaviza la curva. La probabilidad de unión queda dictaminada por la distribución exponencial

$$e^{-\frac{x}{c}}.$$

El archivo `p8or.R` es una versión de la simulación no paralelizada.

Tarea

- Paralelizar tanto como resulte eficientemente posible en la simulación y medir cuánto tiempo se logra ahorrar.

La simulación en `p8or.R` avanza por dos fases en cada iteración. Primero todos aquellos cúmulos que quieren fragmentarse, se fragmentan. Luego, con los cúmulos existentes después de esta primera fase, se revisan los que quieren unirse, para así formar pares uniformemente al azar entre ellos.

Ésta secuencia nos permite paralelizar las fases de rotura y unión.

Para la fase de rotura y unión se crean las funciónes `romperse` y `unirse` respectivamente:

```
romperse <- function(i) {
  urna <- freq[i,]
  if (urna$tam > 1) {
    romper <- round((1/(1 + exp((c - urna$tam)/d)) * urna$num))
    resultado <- rep(urna$tam, urna$num - romper)
    if (romper > 0) {
      for (cumulo in 1:romper) {
        t <- 1
        if (urna$tam > 2) {
          t <- sample(1:(urna$tam-1), 1)
        }
        resultado <- c(resultado, t, urna$tam - t)
      }
    }
    assert(sum(resultado) == urna$tam * urna$num)
    return(resultado)
  } else {
    return(rep(1, urna$num))
  }
}

unirse <- function(i) {
  urna <- freq[i,]
  unir <- round((-urna$tam/c) * urna$num)
  if (unir > 0) {
    division <- c(rep(- urna$tam, unir),
                  rep(urna$tam, urna$num - unir))
    assert(sum(abs(division)) == urna$tam * urna$num)
    return(division)
  } else {
    return(rep(urna$tam, urna$num))
  }
}
```

Cuadro 1: Tiempos de ejecución

Secuencial	Paralelo
0.037	1.272
0.249	1.272
0.234	1.259
0.240	1.246
0.231	1.231
0.243	1.244
0.225	1.256
0.224	1.264
0.226	1.228
0.224	1.240

En el archivo `p8par.R` se encuentra la versión paralelizada de la simulación.

Se mide el tiempo de ejecución en paralelo y secuencial con $k = 1000$, $n = 100000$ y cinco pasos en el tiempo para 10 réplicas. En este caso, la versión paralela resulta ser más tardada, tal y como se muestra en el cuadro 1.

Primer reto

- Estudiar si el ahorro en el tiempo al paralelizar es estadísticamente significativo para diferentes valores de k, teniendo $n = 30k$.

Se varía $k = 10^i$ con $i = 1, 2, \ldots, 5$. Para cada valor se tienen cinco réplicas y se miden los tiempos de ejecución secuencial y paralelo. Los resultados se muestran en la figura 1.

Según los resultados obtenidos, para ciertos valores de k el tiempo de ejecución secuencial es menor que el paralelo, lo que podría deberse a que las operaciones que se relizan en la simulación son cálculos sencillos.

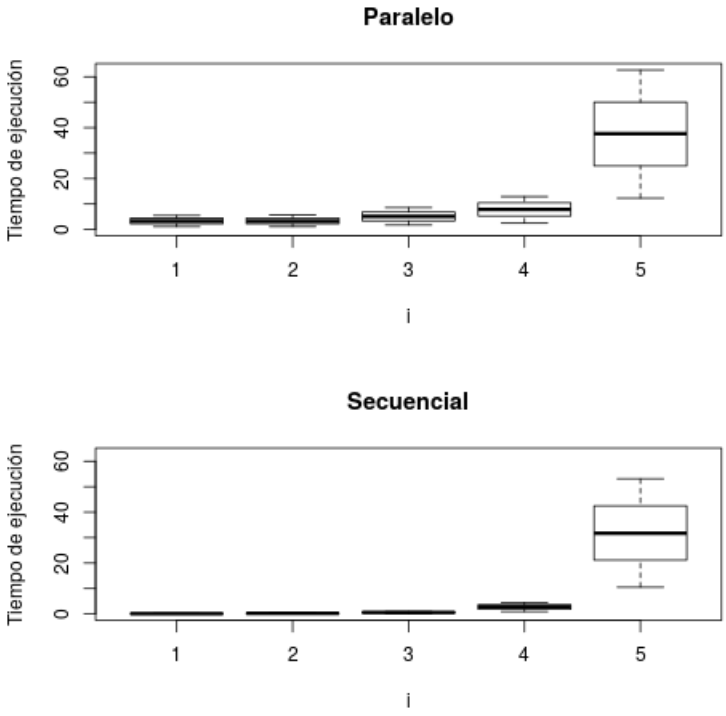

Figura 1: Tiempos de ejecución para distintos valores de k.