Reporte paralelización de algoritmos
Programación y Análisis de Algoritmos

Edgar Steven Baquero Acevedo

Diciembre 10, 2020

1 Algunas consideraciones preliminares

Sobre los recursos computacionales

Se considera necesario considerar algunas propiedades físicas del computador donde se compila y ejecutan los algoritmos:

- **OS:** `Kernel: 5.8.18-1-MANJARO x86_64 bits: 64 Desktop: GNOME 3.38.1 Distro: Manjaro Linux`
- **CPU:** `Info: Dual Core model: Intel Core i5-3317U bits: 64 type: MT MCP`
- **Dispositivos:**
  - `vendor: PNY model: CS900 240GB SSD size: 223.57 GiB` (Ruta del sistema operativo)
  - `vendor: SanDisk model: SSD i100 24GB size: 22.37 GiB` (Memoria Swap)
  - `Memory: 7.48 GiB` (Memoria RAM)
  - `Speed: 862 MHz min/max: 800/2600 MHz Core speeds (MHz): 1: 921 2: 1450 3: 1048 4: 1117` (Núcleos)

Sobre los presets de ejecución

También se considera necesario saber sobre las configuraciones antes de la compilación de los algoritmos. Entre otras, mencionamos las siguientes:

- El número de threads que se usa en cada ejecución es 4, pues es el número de núcleos de procesamiento mencionados anteriormente.
- Se ejecutó en un promedio de 50 repeticiones con distintos tamaños para cada algoritmo (suma, resta, multiplicación e inversa) en paralelo y en secuencial.
- Los tamaños de matrices que se consideraron están entre 50×50 y 5000×5000, mientras que para la multiplicación e inversa de matrices, se consideraron tamaños comprendidos entre 10×10 y 500×500.
- Sobre la anterior entrega, se desarrolló sobre un paradigma orientado a objetos, considerando el correcto encapsulamiento de los atributos, que no fueron considerados correctamente en la anterior entrega, a pesar de su correcta ejecición.
- Con el fin de hacer el estudio y las gráficas simples, se consideraron matrices cuadradas en todos los casos. Sin embargo, cabe aclarar que el menú de uso considera matrices de cualquier tamaño.

2 Una revisión parcial de los algoritmos

Revisamos algunos aspectos que nos ayudan a entender mejor cómo se paralelizaron los algoritmos.

Suma

Los hilos que se consideran para la suma de las matrices, usan como hecho que se pueden particionar por bloques. Gráficamente, podemos ver cómo funciona esto para dos matrices genéricas $A, B \in M_{m\times n}(\mathbb{R})$:

$$A = \begin{pmatrix}\text{Hilo A}_1 & \text{Hilo A}_2 & \text{Hilo A}_3 & \text{Hilo A}_4\end{pmatrix}$$

$$B = \begin{pmatrix}\text{Hilo B}_1 & \text{Hilo B}_2 & \text{Hilo B}_3 & \text{Hilo B}_4\end{pmatrix},$$

que es análogo a considerar una partición por bloques de la forma:

$$\begin{pmatrix}\text{Hilo}_1 & \text{Hilo}_2 \\ \text{Hilo}_3 & \text{Hilo}_4\end{pmatrix},$$

y mucho más sencillo de implementar.

Resta

Se considera una partición por bloques igual a la que se mostró para la suma.

Multiplicación

Supongamos que el producto matricial $A \cdot B = C$ está bien definido, así, la partición que se considera para paralelizar el algoritmo de suma, está dada por:

$$A \cdot B = \begin{pmatrix} \text{Hilo A}_1 \\ \text{Hilo A}_2 \\ \text{Hilo A}_3 \\ \text{Hilo A}_4 \end{pmatrix} \cdot B = \begin{pmatrix} \text{Hilo A}_1 \cdot B \\ \text{Hilo A}_2 \cdot B \\ \text{Hilo A}_3 \cdot B \\ \text{Hilo A}_4 \cdot B \end{pmatrix} = \begin{pmatrix} \text{Hilo C}_1 \\ \text{Hilo C}_2 \\ \text{Hilo C}_3 \\ \text{Hilo C}_4 \end{pmatrix} = C,$$

que de nuevo, es equivalente a considerar un producto matricial por bloques, y es mucho más sencillo de implementar.

Inversa

Para este caso, la paralelización considerada se usa sobre la copia de datos entre las matrices. Por ejemplo, el algoritmo implementado, considera una expansión de una matriz, cuyos datos tienen que ser copiados de nuevo en esta matriz extendida. Así, es pertinente procurar optimizar los algoritmos en su copiado, pues la reducción por *Gauss-Jordan* requiere el uso completo de la matriz.

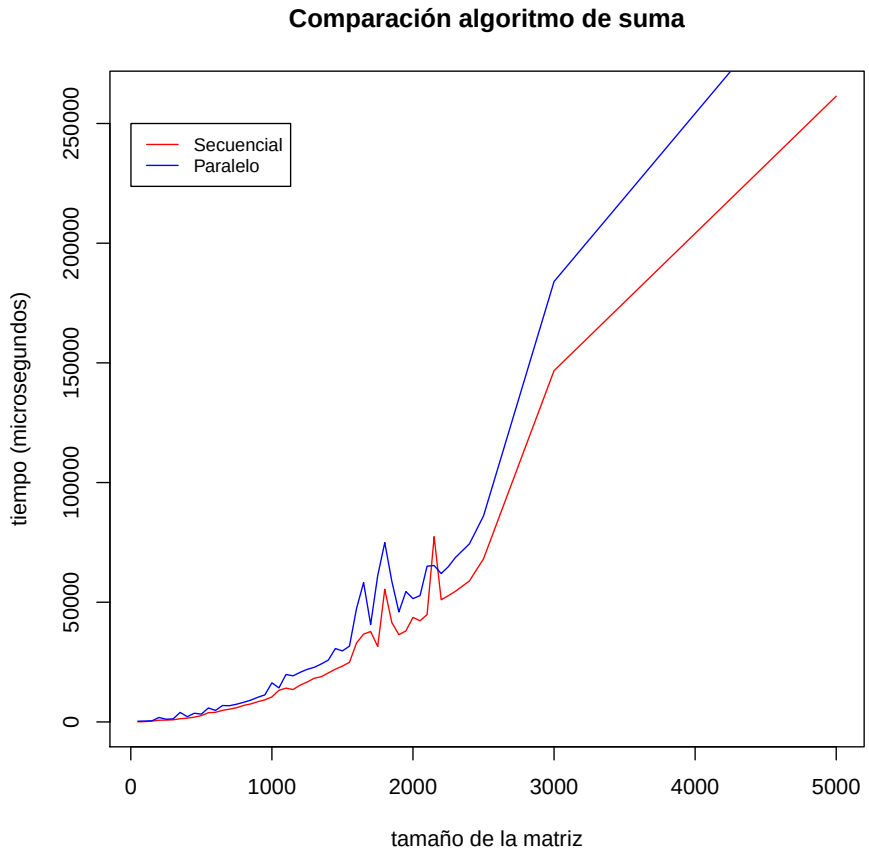
3 Comparaciones

Con el fin de hacer más ameno el entendimiento de los procesos de ejecución paralelos, usamos el sofware `R` para recopilar y visualizar los datos generados, los cuales contienen los tiempos en formato `csv` para distintos tamaños de matrices.

Suma

```
suma = read.csv("/run/media/ed4st/Data/Documentos/Master/semestre1/ProgAnalgo/tareas/proyect
plot(suma$size, suma$seq_time, type = "l",
     col = "red", main = "Comparación algoritmo de suma",
     xlab = "tamaño de la matriz", ylab = "tiempo (microsegundos)")
lines(suma$size, suma$par_time, col = "blue")

legend(1, 250000, legend=c("Secuencial", "Paralelo"),
       col=c("red", "blue"), lty = 1:1, cex=0.8)
```

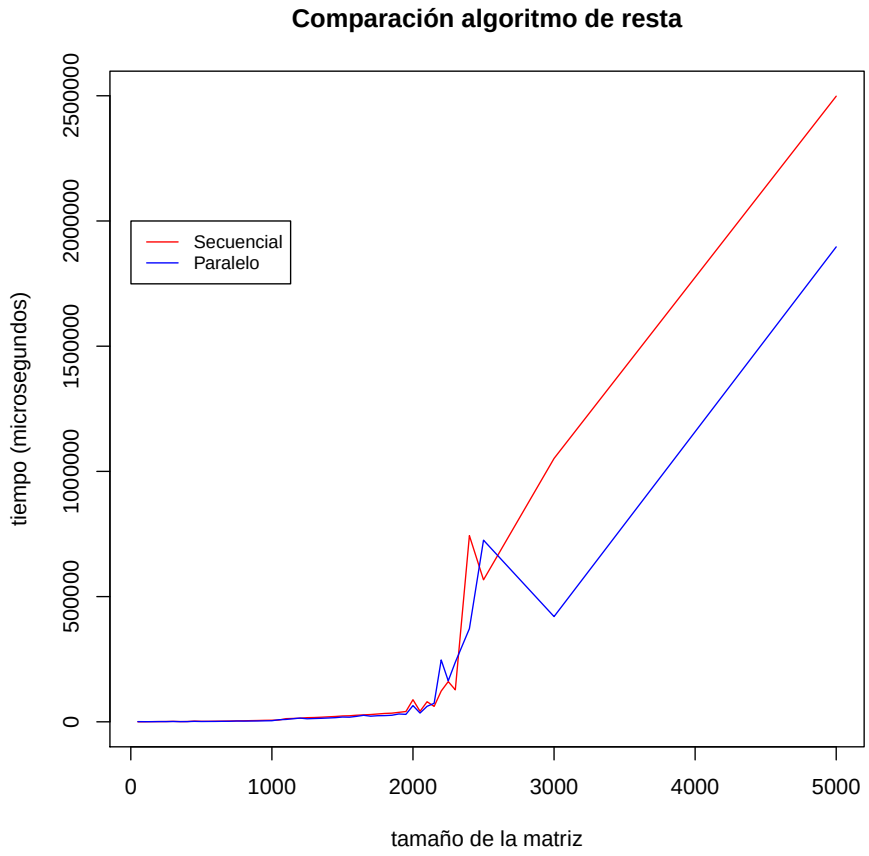


Como se puede observar, en este caso, el algoritmo secuencial sigue siendo sustancialmente mejor que el paralelo. Es muy raro el caso donde el paralelo supera al secuencial.

Resta

```
resta = read.csv("/run/media/ed4st/Data/Documentos/Master/semestre1/ProgAnalgo/tareas/proyec
plot(resta$size, resta$seq_time, type = "l",
     col = "red", main = "Comparación algoritmo de resta",
     xlab = "tamaño de la matriz", ylab = "tiempo (microsegundos)")
lines(resta$size, resta$par_time, col = "blue")

legend(1, 2000000, legend=c("Secuencial", "Paralelo"),
       col=c("red", "blue"), lty = 1:1, cex=0.8)
```

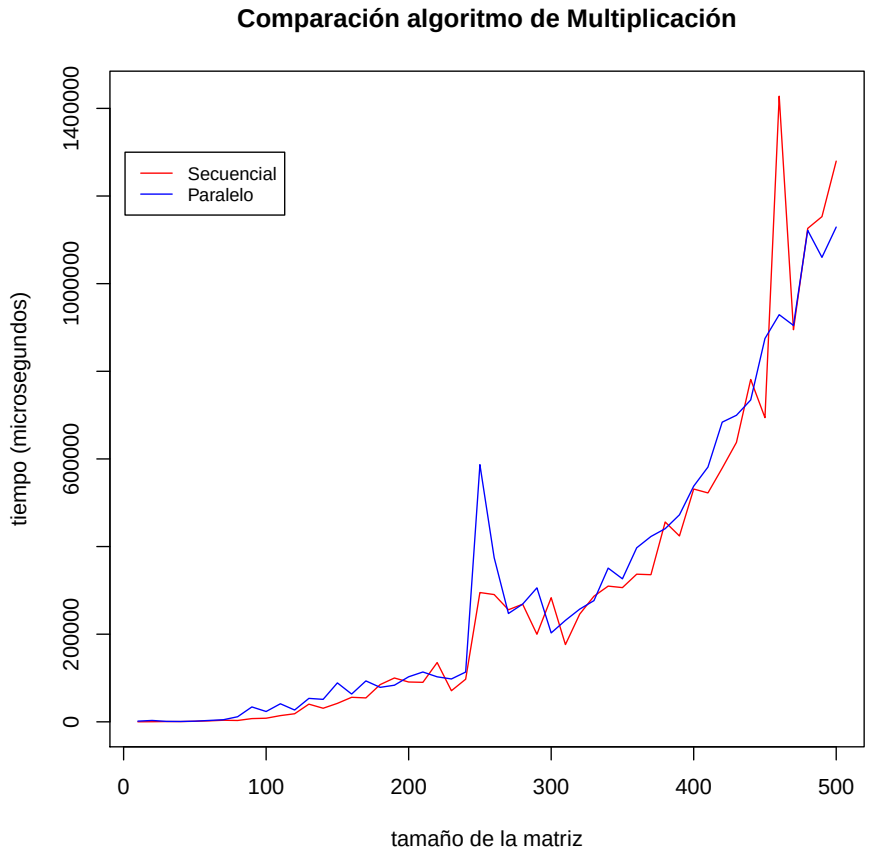


En este caso, es posible detallar que el algoritmo paralelo mejora su tiempo en algunas ejecuciones.

Multiplicación

```
multiplicacion= read.csv("/run/media/ed4st/Data/Documentos/Master/semestre1/ProgAnalgo/tarea
plot(multiplicacion$size, multiplicacion$seq_time, type = "l",
     col = "red", main = "Comparación algoritmo de Multiplicación",
     xlab = "tamaño de la matriz", ylab = "tiempo (microsegundos)")
lines(multiplicacion$size, multiplicacion$par_time, col = "blue")

legend(1, 1300000, legend=c("Secuencial", "Paralelo"),
       col=c("red", "blue"), lty = 1:1, cex=0.8)
```

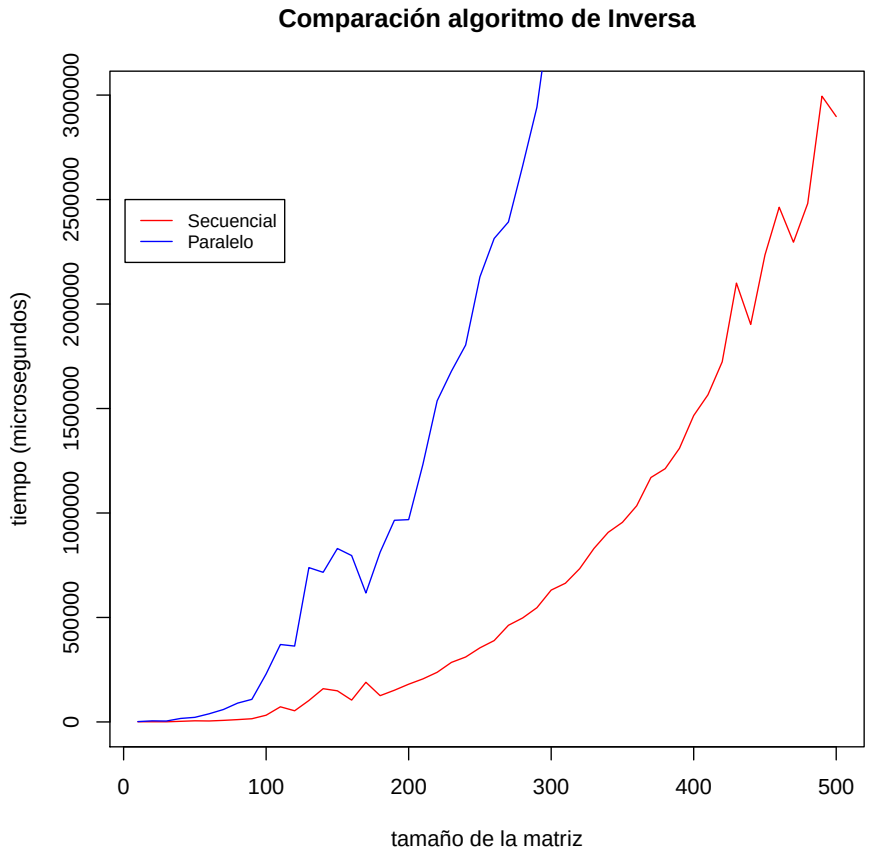


Notamos una tendencia a mejorar el algoritmo paralelo a medida que el tamaño de la matriz se hace mayor.

Inversa

```
inversa = read.csv("/run/media/ed4st/Data/Documentos/Master/semestre1/ProgAnalgo/tareas/proy
plot(inversa$size, inversa$seq_time, type = "l",
     col = "red", main = "Comparación algoritmo de Inversa",
     xlab = "tamaño de la matriz", ylab = "tiempo (microsegundos)")
lines(inversa$size, inversa$par_time, col = "blue")

legend(1, 2500000, legend=c("Secuencial", "Paralelo"),
       col=c("red", "blue"), lty = 1:1, cex=0.8)
```



Es posible ver que indudablemente, el algoritmo secuencial es mejor que el algoritmo paralelo.

4 Conclusiones

Pudimos obsevar gráficamente cómo se comportan las implementaciones planteadas. Para los algoritmos paralelos de suma y resta, que en principio son iguales, se vio una pequeña diferencia en los tiempo de ejecución para tamaños grandes; esto es posible, ya que la sobrecarga del operador binario de suma y resta, con el que cuenta el compilador, es un operador que se implementa sobre operaciones bit a bit.

Sobre el algoritmo de multiplicación, se puede observar una mejoría en los tiempos de ejecución, esto, puesto que los hilos representan una optimización en los distintos núcleos físicos del procesador.

Sobre el algoritmo de inversa, vemos que se debería intentar paralelizar otras secciones planteadas en el algoritmo de inversa secuencial. Razón de los resultados mostrados, se debe también a incorrecta repartición de los límites de los hilos, ya que, como la paralelización, que en principio se consideró de libre escogencia, no fue la mejor. Otra razón, con menor peso, fue la falta de tiempo para explorar otras opciones de paralelización del mismo.

Cabe resaltar también, que parte del trabajo realizado, se hizo con base en lo visto en clase, y dado que el tema fue introductorio, no se alcanzó a observar otras funcionalidades que ofrece la librería `pthread`.