Implementación de multiplicación de matrices con concurrencia

Presentado por: Juan Camilo Rojas Cortés

Explicación del algoritmo:

El algoritmo al que era necesario aplicarle concurrencia es el de multiplicación de matrices. Este proceso consiste en generar una matriz resultante en la que cada celda sea el producto punto de la fila de su subíndice en la primera matriz y la columna de su subíndice en la segunda matriz. Para generar este producto punto es necesario recorrer completamente una fila y una columna de la matriz para hacer la suma y la multiplicación, repitiéndose este proceso para cada una de las celdas de la matriz resultante. Por ese motivo, este es un algoritmo con una complejidad de n^3, siendo n las dimensiones de la matriz en el caso de que se trabaje con matrices cuadradas. Por ejemplo, si se multiplican 2 matrices de 5X5, el algoritmo secuencial deberá hacer 125 iteraciones, aumentando cúbicamente el número de estas conforme se incrementan las dimensiones de la matriz.

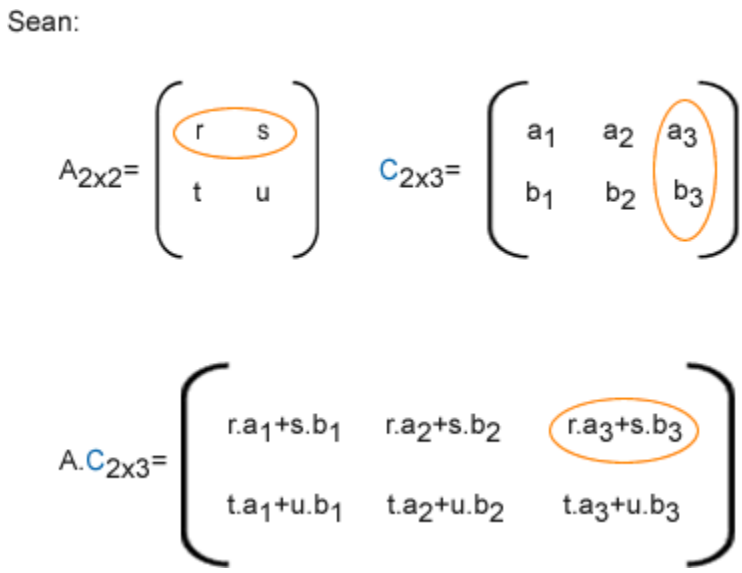

Figura 1. Representación gráfica de la multiplicación de matrices

Algoritmo paralelo:

Para el presente trabajo, se optó por hacer una paralelización con hilos. En este caso, con el objetivo de disminuir la porción de código que se ejecuta de forma secuencial, se le pasa como parámetro a cada hilo un número de fila. De esa forma, de manera secuencial solamente se realizan n iteraciones, siendo n el número de filas y columnas de la matriz. Así, a cada hilo le corresponde hacer n^2 iteraciones, pues cada uno de estos se encarga de calcular los resultados para una fila de la matriz. En esta implementación, el número de hilos tiene relación directa con las dimensiones de la matriz. En una matriz de NXN, el algoritmo generará N hilos.

Análisis de tiempos y speed up:

Las pruebas realizadas a la paralelización del presente algoritmo se hicieron en una máquina con las siguientes características:

- Memoria RAM: 4 GB.
- Procesador: AMD A8-7410 APU with AMD Radeon R5 Graphics × 4
- Nucleos de procesamiento: 4
- Sistema operativo: ubuntu 16.04 LTS de 64 bits.

Para las pruebas de ejecución, se tomaron 7 valores de n distintos (10, 100, 200, 500, 800, 1000 y 2000) y para cada uno de estos se hizo un total de 10 pruebas, evaluando los tiempos de ejecución en cada uno de los algoritmos, tanto serial como parlelos. A continuación se pueden ver estos resultados:

IMPLEMENTACIÓN SECUENCIAL (tiempo dado en segundos)							
Iteración	**10**	**100**	**200**	**500**	**800**	**1000**	**2000**
1	0,00001	0,008707	0,066717	1,696583	11,025918	23,05065	198,588116
2	0,000022	0,011321	0,063909	1,777635	11,217768	23,003422	198,731774
3	0,000011	0,007936	0,063588	1,55296	10,97958	22,790453	197,164891
4	0,000022	0,010614	0,063801	1,529325	11,007149	22,826133	196,296044
5	0,000021	0,008507	0,064053	1,914466	11,095712	22,893029	196,564942
6	0,000022	0,006972	0,059075	1,454648	10,027709	21,131893	185,382501
7	0,000022	0,009318	0,059398	1,553169	10,16939	21,132597	184,00702
8	0,000009	0,011053	0,061804	1,539698	10,355836	21,554958	185,116232
9	0,00001	0,007345	0,058828	1,5348	10,662544	21,563793	182,589572
10	0,000009	0,011026	0,059724	1,55803	10,313104	21,778125	184,630674
Promedio	0,0000158	0,0092799	0,0620897	1,6111314	10,685471	22,1725053	190,9071766

Tabla 1. Mediciones de tiempos para la implementación secuencial

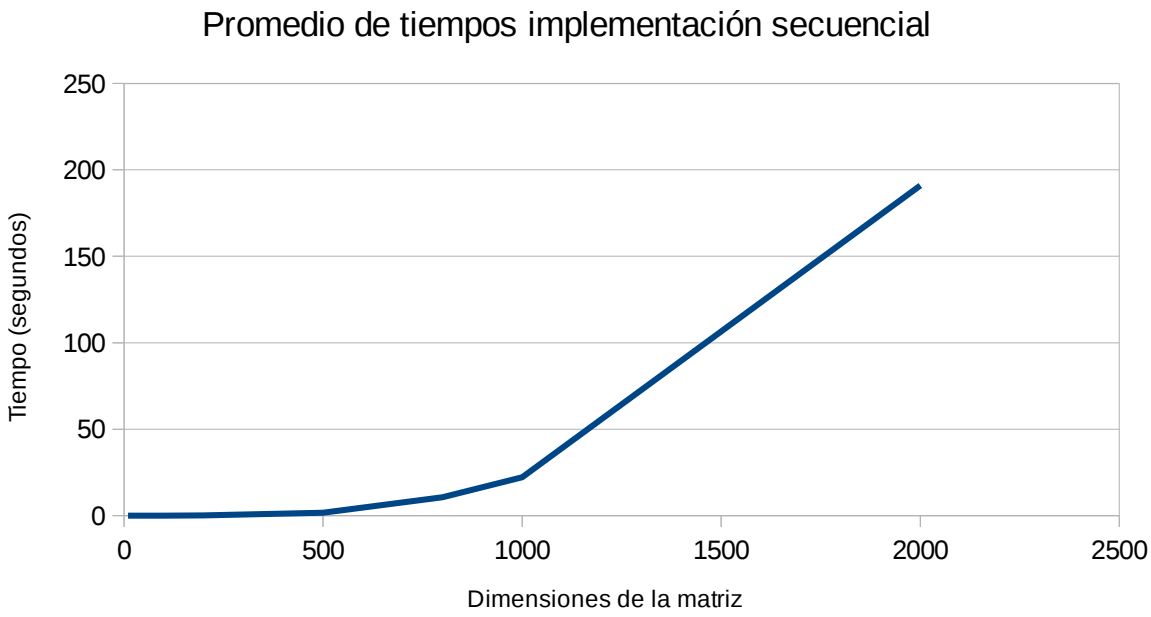

Gráfico 1. Promedio de tiempos para la implementaación secuencial

IMPLEMENTACIÓN CON HILOS (tiempo dado en segundos)							
Iteración	**10**	**100**	**200**	**500**	**800**	**1000**	**2000**
1	0,001228	0,014815	0,072391	1,445255	9,949138	19,311508	114,049306
2	0,001342	0,015114	0,077129	0,976123	7,98672	14,913088	192,238484
3	0,001283	0,014989	0,081865	1,476339	9,656639	16,30239	165,830074
4	0,001109	0,013836	0,077696	1,486744	8,821073	21,629171	162,933353
5	0,00133	0,015078	0,060715	1,36368	9,513284	20,571258	133,769969
6	0,001211	0,012456	0,07937	1,673618	6,640853	21,067466	180,908396
7	0,001132	0,015327	0,070366	1,175732	9,191771	20,349819	127,448398
8	0,00108	0,015689	0,077763	1,140972	7,236302	20,564291	191,596062
9	0,000388	0,010247	0,077225	1,014727	8,716826	20,231222	170,62311
10	0,000478	0,012974	0,075472	1,407769	9,655982	14,403707	170,374787
Promedio	0,00105810	0,0140525	0,0749992	1,3160959	8,7368588	18,934392	160,9771939

Tabla 2. Mediciones de tiempos para implementación con hilos

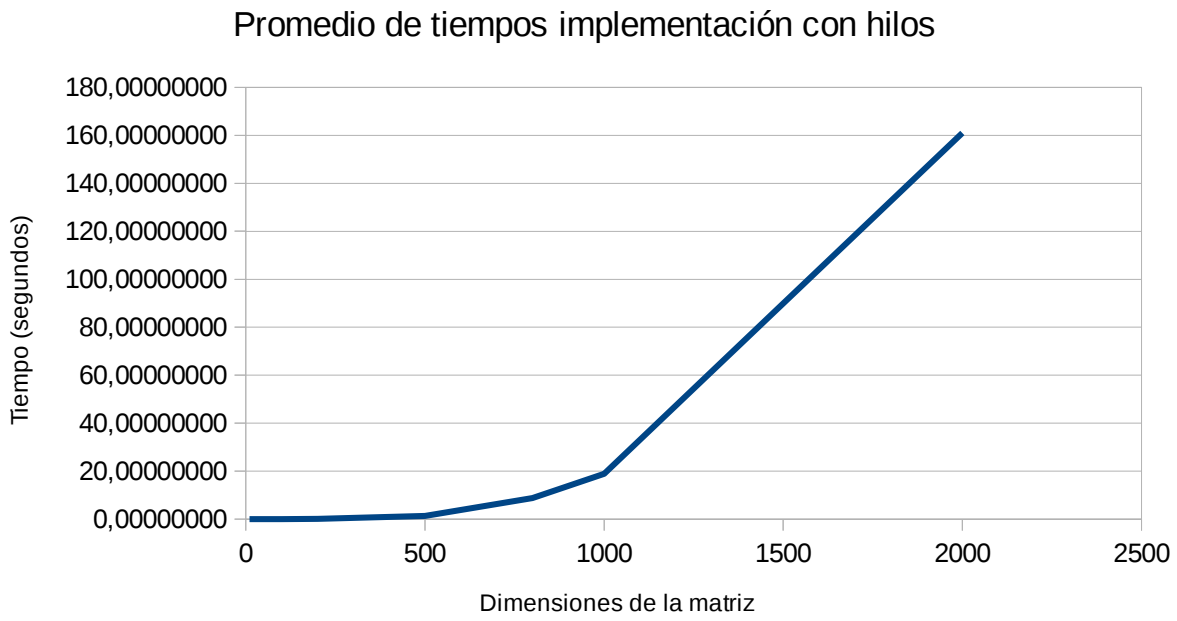

Gráfico 2. Promedio de tiempos para la implementación con hilos

Para tener una mayor claridad de los resultados comparativos entre ambas implementaciones, se realizó el cálculo del speed up rate. A continuación se pueden ver los resultados:

Tabla 3. Speed up rate

Dimensiones	Promedio implementación secuencial (segundos)	Promedio implementación con hilos (segundos)	Speed up rate
10	0,0000158	0,00105810	-0,985067574
100	0,0092799	0,0140525	-0,339626401
200	0,0620897	0,0749992	-0,1721285027
500	1,6111314	1,3160959	0,2241747733
800	10,685471	8,7368588	0,2230335003
1000	22,1725053	18,934392	0,1710175484
2000	190,9071766	160,9771939	0,1859268507

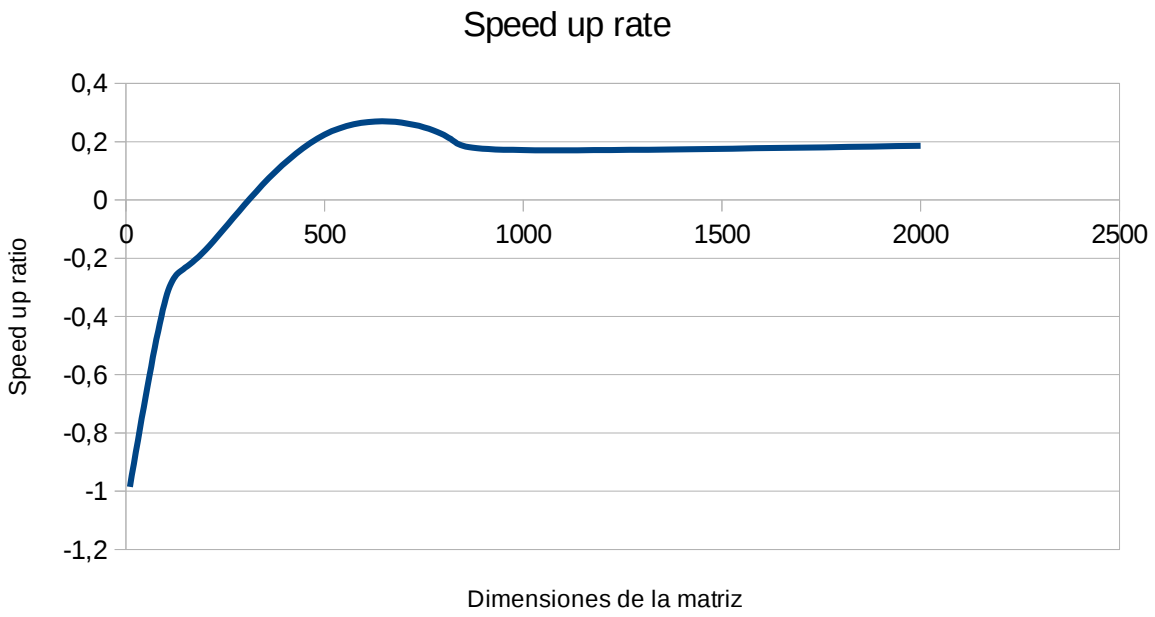

Gráfico 3. Speed up rate

Conclusiones:

Como se puede apreciar en los resultados presentados anteriormente, cuando se trabaja con un número reducido de filas y columnas, y por lo tanto de hilos, la eficiencia del algoritmo paralelo es menor a la del algoritmo secuencial. Esto puede ocurrir porque al ser pocas iteraciones, el procesador puede ejecutar el programa de forma serial rápidamente, mientras que el desgaste para el sistema operativo por la creación de hilos termina siendo superior. Sin embargo, como se puede apreciar en el Gráfico 3, cuando se supera un valor N de 300 (ver intersección de la línea con el eje x), el desempeño del algoritmo paralelo comienza a ser mejor que el del algoritmo secuencial, llegando a su pico máximo en los 500 hilos. El speedup disminuye al llegar a los 1000 hilos y empieza una tendencia creciente más lenta, llegando a ejecutarse la multiplicación de matrices aproximadamente un 20% más rápido que en la solución secuencial.