Multiplicación de matrices de forma secuencial y paralela

Presentado por:

Presentado a:
Ramiro Andrés Barrios Valencia

HPC: High Performance Computing
Ingeniería de Sistemas y Computación
Universidad Tecnológica de Pereira
2022

Tabla de contenido

Resumen

Por medio del presente documento se muestra el proceso realizado para llevar a cabo el algoritmo de multiplicación de matrices en el lenguaje de programación C, las multiplicaciones fueron realizadas de dos formas: secuencial y paralela, esto con el fin de analizar su desempeño tras ejecutarlo, variando el tamaño de las matrices a multiplicar y la disposición de los recursos del procesador a través de la librería pthread. Se han realizado pruebas ejecutando el algoritmo de forma secuencial, luego se implementaron hilos con la librería pthread y 32 hilos y por último se implementó procesos con forks. Con base en estas pruebas se han obtenido las respectivas conclusiones

Introducción

A lo largo de los años la computación ha ido evolucionando hasta el punto de poder resolver problemas de grandes dimensiones en un tiempo más reducido, un ejemplo de estos problemas es la multiplicación de matrices. A pesar de la brevedad para programar este algoritmo en algún lenguaje de programación, “se ha invertido mucho trabajo en hacer que los algoritmos de multiplicación de matrices sean eficientes. […] Se han diseñado muchos algoritmos diferentes para multiplicar matrices en diferentes tipos de hardware, incluidos sistemas paralelos y distribuidos, donde el trabajo computacional se distribuye en múltiples procesadores” (Qaz Wiki, 2020). En este caso se ha optado por resolver el problema programando dos versiones, una para darle solución de manera secuencial y otra para darle solución de manera paralela. Por otra parte, teniendo en cuenta la definición del speedup (o aceleración) “Herramienta de la Mejora continua que de forma metodológica busca incrementar la velocidad de procesamiento de las máquinas” (Progressa Lean, 2014) se pretende calcular este valor en base a los resultados arrojados tras la ejecución de ambas versiones del algoritmo programadas en el lenguaje C, utilizando los conceptos vistos durante las sesiones del curso HPC.

Finalmente se busca comprobar que, ejecutando el algoritmo con múltiples hilos, se hace posible reducir los tiempos de ejecución del algoritmo, beneficiando a las áreas donde este se aplica, tales como el álgebra lineal o la inteligencia artificial.

Marco conceptual

High Performance Computing

Cuando hablamos de HPC por sus siglas en inglés, o Computación de alto rendimiento en español, “hacemos referencia a un campo de la computación actual que da solución a problemas tecnológicos muy complejos y que involucran un gran volumen de cálculos o de coste computacional.” (López, 2017) Para nuestro caso, al aumentar la dimensión de las matrices a multiplicar, esto supone un alto costo computacional, el cual, según el análisis, se ve disminuido gracias al uso de herramientas que permiten, precisamente, disminuir el tiempo de ejecución.

Multiplicación matricial

Para implementar el algoritmo en el lenguaje C es necesario comprender previamente algunos conceptos, entre ellos, como es el proceso de multiplicar matrices de forma manual, el cual podemos aprender fácilmente gracias a vídeos explicativos y a diversos sitios web, tal como Producto de matrices.

Otros sitios en internet nos dan una idea más general de la programación de la multiplicación matricial, como por ejemplo el que se expone a continuación, que ha sido extraído del blog Análisis y diseño de algoritmos.

Dadas dos matrices A y B, tales que el número de columnas de la matriz A es igual al número de filas de la matriz B, es decir:

$$A := (a_{ij})_{m \times n} \text{ y}$$
$$B := (b_{ij})_{n \times p}$$

La multiplicación de A por B genera una nueva matriz resultado de la forma: $C = AB := (C_{ij})mxp$ donde cada elemento *Cij* está definido por:

$$c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}$$

Definición que permite acercarse más a una idea del código ya que de entrada plantea las tres variables a iterar para cada una de las celdas de la matriz resultado.

Complejidad Computacional

“La Teoría de la Complejidad estudia la eficiencia de los algoritmos en función de los recursos que utiliza en un sistema computacional (usualmente espacio y tiempo)” (Cortéz, 2004). Para este estudio, la complejidad computacional permite observar que en la multiplicación de matrices se gastan bastantes recursos de procesamiento en cuanto a tiempo, es decir, para matrices de dimensiones muy grandes, el procesador estará ocupado bastante tiempo resolviendo la multiplicación, lo que motiva aún más a programar de manera paralela el algoritmo, aprovechando al máximo los recursos del procesador.

Programación paralela

“La computación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente.” (Ferestrepoca, 2019) En nuestro caso de estudio se ejecutará el código de multiplicación de matrices de forma secuencial y de forma paralela, esto con el fin de comparar los resultados obtenidos por cada ejecución y de esta manera poder analizar la eficiencia de la programación paralela.

Hilos

“La diferencia fundamental entre hilos y procesos es que un proceso dispone de un espacio de memoria separado, mientras que un hilo no. De esta forma los hilos pueden trabajar directamente con las variables creadas por el proceso padre, mientras que los procesos hijos no pueden hacerlo.” (Montero & Antunez, 2011) Esta información nos facilita la comprensión del principio aplicado para esta implementación.

Speedup

El speedup representa la ganancia que se obtiene en la versión paralela de un programa con respecto a su versión secuencial. Para este estudio se ha tomado el speedup como:

$$Speedup = \frac{T*(n)}{T_p(N)}$$

Donde T*(n) hace referencia al tiempo de ejecución secuencial
y T_p (N) hace referencia al tiempo de ejecución paralelo

Marco Contextual

Características de la máquina

Se probó el algoritmo para la multiplicación de matrices generando matrices cuadradas de dimensiones 10, 100, 200, 400, 800, 1600 y 3200 respectivamente. Uno de los algoritmos fue ejecutado de manera secuencial y el otro utilizando hilos. Se analizaron los distintos métodos evaluando los recursos consumidos como el tiempo; se sacaron algunas gráficas para mostrar los resultados obtenidos.

Características del PC donde se realizaron las pruebas:

- Procesador: Intel(R) Core(TM) i7-10700
- Cantidad de núcleos: 8
- Cantidad de subprocesos/hilos: 16
- Frecuencia básica del procesador: 2.90GHz
- Frecuencia Turbo máxima: 4.80GHz
- Ram 16GB DDR4 @ 2400 MHz
- SSD: 240GB - R: 540 MB/s - W: 500 MB/s
- Sistema operativo: Ubuntu 20.04.4 LTS

Desarrollo

Para iniciar el desarrollo de esta actividad lo primero era generar matrices cuadradas con números random, por lo cual se usó la librería stdlib.h. El tamaño de las matrices cuadradas usadas fueron 10, 100, 200, 400, 800, 1600 y 3200. Con las matrices a multiplicar listas, se pudo aprovechar la implementación en forma secuencial, con hilos y por último con procesos para analizar el comportamiento del algoritmo.

El código en C de la multiplicación de matrices utilizando hilos se basa en el código Programación multithread en C (pthread.h) con el cual se comenzó

añadiendo las librerías de hilos pthread.h para crear, asignar y juntar las actividades que debe realizar cada hilo y la librería time.h para poder contabilizar el tiempo que demora la ejecución. Para trabajar con time.h de forma correcta, algunas partes del código se basan en el recurso 8 formas de medir el tiempo de ejecución en C / C ++. El código empieza asignando el número de hilos y las variables que almacenarán el tamaño de las filas y las columnas de las matrices que se usarán en la ejecución de la multiplicación. Se pide al usuario que digite el tamaño de la matriz para seguido a esto, con la función malloc reservar memoria para poder trabajar con hilos, posteriormente se procede a la lectura de las matrices y la creación de los hilos a través de funciones implementadas dentro del código. Una vez creados los hilos, se toma el tiempo a través de gettimeofday. Por otra parte, cuando ya está definida la forma en cómo se creará la cantidad de hilos y la unión del trabajo de cada uno, se hace la declaración de la función multiplicación la cual está diseñada para asignar las filas por multiplicar en cada uno de los hilos que la utilicen, además, dentro de esta función se realiza la operación de multiplicación de las matrices por medio de tres ciclos anidados. Finalmente, se escribe el resultado de la multiplicación de matrices hecho por cada hilo, se une a través de la función proporcionada por pthread.h para tener la matriz resultante completa y utilizando nuevamente la función gettimeofday se hace posible calcular el tiempo de ejecución entre la creación de hilos y la unión de sus resultados. La salida de este programa es únicamente el tiempo de ejecución. El código de la multiplicación de matrices secuencial usa las mismas librerías usadas con los hilos, excepto pthread.h, ya que esta vez la ejecución es secuencial. El código empieza definiendo las variables que almacenarán el tamaño de filas y columnas de la matriz que vamos a usar en la ejecución de la multiplicación, se declaran los arreglos bidimensionales para las matrices a multiplicar siendo estas DoubleA y DoubleB y el arreglo bidimensional Double C para el resultado. Se le pide al usuario que ingrese el tamaño de la matriz y se usa la función malloc para hacer la reserva de memoria para poder ejecutar y se procede a la lectura

de las matrices a través de la función read mat, la cual recibe la cantidad de filas y columnas de la matriz. Posteriormente se toma el tiempo con gettimeofday y se hace el llamado a la función matmul cuyos parámetros de entrada son las filas y columnas, que representan la dimensión de las matrices a multiplicar y dentro de esta función se realiza la operación de multiplicación de las matrices por medio de tres ciclos anidados y una vez termina esta función de ejecutarse, se toma el tiempo nuevamente con gettimeofday, con el fin de establecer la diferencia de tiempo entre el inicio y fin de la ejecución. Al igual que en el caso de los hilos, este programa secuencial imprime únicamente el tiempo. Luego de hacer la ejecución de cada multiplicación de matriz con diferentes tamaños y en el caso de los hilos con diferente número de hilos, obtenemos los resultados del tiempo de ejecución de cada uno de estos, lo que permite realizar los análisis y cálculos de la variable speedup para cada uno de los casos.

El código fuente resultante de la implementación descrita se encuentra registrado en el enlace: Caso de Estudio Matrices con las respectivas instrucciones para una ejecución correcta.

Pruebas

1. Resultados de ejecutar el algoritmo en forma secuencial (Utilizando uno de los cores del procesador):

Tabla 1: Resultados de la ejecución secuencial.

Secuencial							
Dimensiones	10	100	200	400	800	1600	3200
1	0.018	5.73	24.441	232.5	1975.131	21938.946	250373.07
3	0.003	2.934	24.02	227.079	1967.557	20599.225	253157.11
4	0.003	2.907	23.861	225.545	1974.101	20616.176	250368.16
5	0.003	2.944	23.557	229.237	1979.652	21110.895	250342.46
6	0.004	2.999	24.992	230.962	2009.117	23537.424	253920.33

7	0.003	2.958	23.821	225.605	1958.448	20618.942	249549.43
8	0.003	2.931	23.922	229.129	1952.08	20254.705	249288.25
9	0.003	2.902	24.049	229.668	1933.098	20586.635	249015.46
10	0.003	2.871	23.744	226.326	1981.072	21841.5	257478.95
11	0.003	2.987	23.69	229.432	1952.122	20596.536	246725.31
Promedio	0.0046	3.2163	24.0097	228.5483	1968.2378	21170.0984	251021.853

Fuente: Elaboración propia.

2. Resultados de ejecutar el algoritmo con dos hilos: A partir de aquí aparece la fila de speedup, en la cual está representada la cantidad de veces que se mejora la ejecución al correr el código con varios hilos.

Tabla 2: Resultados de la ejecución con 2 hilos.

2 Hilos							
Dimensiones	10	100	200	400	800	1600	3200
1	0.282	6.433	12.159	115.249	972.375	10293.911	121942.44
3	0.088	1.536	12.077	111.255	975.409	10113.194	120706.24
4	0.118	1.618	12.252	118.352	980.131	10492.231	121788.45
5	0.089	1.559	12.054	114.215	978.408	10311.015	125212
6	0.093	1.556	11.874	113.721	971.952	10200.231	120736.84
7	0.108	1.547	11.904	114.538	975.786	10505.258	120937.14
8	0.097	1.492	12.68	116.968	978.488	11420.478	123595.98
9	0.095	1.527	12.203	151.472	972.907	10238.908	123750.82
10	0.068	1.639	12.706	113.722	965.054	10361.337	121698.47
11	0.248	4.385	11.628	115.367	967.918	10268.703	120704.58
Promedio	0.1286	2.3292	12.1537	118.4859	973.8428	10420.5266	122107.296
Speedup	0,0358	1,3809	1,9755	1,9289	2,0211	2,0316	2,0557

Fuente: Elaboración propia.

3. Resultados de ejecutar el algoritmo con cuatro hilos:

Tabla 3: Resultados de la ejecución con 4 hilos.

4 Hilos							
Dimensiones	10	100	200	400	800	1600	3200
1	0.364	5.426	6.289	57.019	489.973	5830.477	60699.015
3	0.125	0.913	5.962	56.654	489.433	5698.888	60752.61
4	0.159	0.883	6.701	57.068	499.9	5676.881	62203.354
5	0.088	0.839	6.055	57.33	507.075	5690.481	61037.997
6	0.116	0.898	6.02	56.441	489.604	5352.826	60757.978
7	0.12	0.908	6.47	58.222	504.971	5450.684	60820.1
8	0.15	0.799	5.974	57.101	493.854	5473.308	61212.802
9	0.119	0.85	6.492	57.671	501.353	5452.587	60831.442
10	0.109	0.85	5.987	58.25	504.271	5416.464	60799.567
11	0.176	0.861	5.975	58.112	509.9	5218.986	60961.887
Promedio	0.1526	1.3227	6.1925	57.3868	499.0334	5526.1582	61007.6752
Speedup	0,0301	0,24316	0,38772	0,3983	3,9441	3,8309	4,1146

Fuente: Elaboración propia.

4. Resultados de ejecutar el algoritmo con ocho hilos:

Tabla 4: Resultados de la ejecución con 8 hilos.

8 Hilos							
Dimensiones	10	100	200	400	800	1600	3200
1	0.796	2.974	7.082	47.177	275.484	2684.944	31783.242
3	0.241	0.801	6.349	49.965	306.366	3053.39	34945.748
4	0.247	0.788	6.302	46.938	344.132	3020.742	34327.25
5	0.246	0.854	5.614	47.09	313.025	3043.606	34311.088
6	0.202	0.829	6.011	30.377	377.278	3016.857	34317.201
7	0.252	0.785	6.247	53.699	356.92	3053.899	34300.87
8	0.259	0.87	7.053	35.977	389.397	3038.882	34518.506
9	0.129	0.795	6.223	28.758	366.561	3048.365	34166.439

10	0.183	0.85	6.929	36.26	350.525	3059.393	34440.425
11	0.211	0.894	6.301	46.991	346.487	3057.196	34724.674
Promedio	0.2766	1.044	6.4111	42.3232	342.6175	3007.7274	34183.5443
Speedup	0,0166	3,08075	3,74502	5,4001	5,7447	7,0386	7,3434

Fuente: Elaboración propia.

5. Resultados de ejecutar el algoritmo con dieciséis hilos:

Tabla 5: Resultados de la ejecución con 16 hilos.

16 Hilos							
Dimensiones	10	100	200	400	800	1600	3200
1	1.092	2.932	4.623	38.678	257.617	2734.106	31383.818
3	0.212	0.799	4.429	34.61	304.383	3122.906	34328.862
4	0.62	0.643	4.335	47.574	300.815	3079.328	34406.629
5	0.66	0.577	4.447	36.338	310.682	3199.897	33862.441
6	0.495	0.63	4.448	37.691	309.367	3002.412	33723.185
7	0.565	0.62	4.665	44.901	298.673	3164.429	33763.094
8	0.583	0.695	4.81	31.489	321.032	3116.4	33727.408
9	0.3	0.663	4.48	37.501	298.294	3092.956	33560.834
10	0.383	0.733	5.732	50.93	296.364	3104.987	34317.148
11	0.243	0.609	4.51	36.603	297.174	3116.698	33457.007
Promedio	0.5153	0.8901	4.6479	39.6315	299.4401	3073.4119	33653.0426
Speedup	0,0089	3,61341	5,16571	5,7668	6,5731	6,8881	7,4591

Fuente: Elaboración propia.

6. Resultados de ejecutar el algoritmo con treinta y dos hilos:

Tabla 6: Resultados de la ejecución con 32 hilos.

32 Hilos							
Dimensiones	10	100	200	400	800	1600	3200

1	2.3	0.594	4.461	32.725	251.487	2743.595	31039.227
3	0.649	0.669	4.324	35.551	270.44	3223.351	31901.562
4	0.863	0.582	4.06	33.769	284.826	3238.35	34641.341
5	0.775	0.689	3.955	29.639	300.889	3181.566	30481.089
6	0.993	0.615	3.89	31.17	299.599	3166.91	32474.235
7	0.644	0.593	3.722	31.885	311.487	3275.085	30723.089
8	0.89	0.572	4.002	30.501	305.219	3197.587	32524.382
9	0.674	0.582	3.68	31.486	297.031	3202.984	30536.386
10	0.4	0.628	4.516	31.461	307.336	3164.068	30179.338
11	0.509	0.592	3.793	31.088	303.007	3162.537	30591.229
Promedio	0.8697	0.6116	4.0403	31.9275	293.1321	3155.6033	31509.1878
Speedup	0,0053	5,25883	5,94255	7,1584	6,7145	6,7087	7,9666

Fuente: Elaboración propia.

7. Resultados de ejecutar el algoritmo con procesos:

Tabla 7: Resultados de la ejecución por procesos.

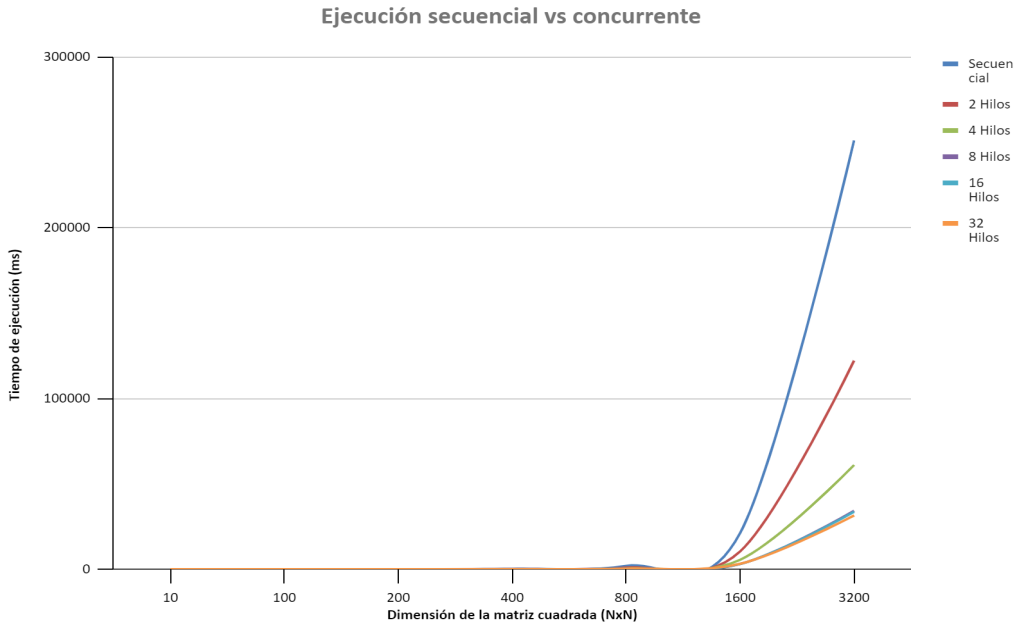
procesos							
Dimensiones	10	100	200	400	800	1600	3200
1	1.962	9.5	31.119	118.535	562.545	3812.245	69651.141
3	0.411	9.12	31.942	147.937	703.068	4364.176	61346.905
4	0.37	8.982	34.154	145.705	726.484	4504.307	82582.497
5	2.558	48.046	95.955	128.875	619.725	4177.667	55933.158
6	0.402	8.875	31.959	139.856	649.702	4178.688	56587.394
7	0.411	9.295	36.425	139.843	650.792	4319.874	56743.088
8	0.418	8.721	35.589	139.501	646.55	4170.255	56079.317
9	0.427	8.926	33.901	140.703	647.038	4210.422	55447.565
10	0.397	8.702	32.05	141.08	655.189	4162.569	56950.866
11	0.397	8.704	32.997	142.463	651.493	4196.765	57444.272
Promedio	0.7753	12.8871	39.6091	138.4498	651.2586	4209.6968	60876.6203

Speedup	0,0059	0,24958	0,60617	1,6508	3,0222	5,0289	4,1235

Fuente: Elaboración propia.

Ahora, para complementar el análisis de los datos y tener una vista un poco más amplia, se han utilizado las tablas para hacer gráficas comparativas que nos permitan visualizar la eficiencia de trabajar con hilos.

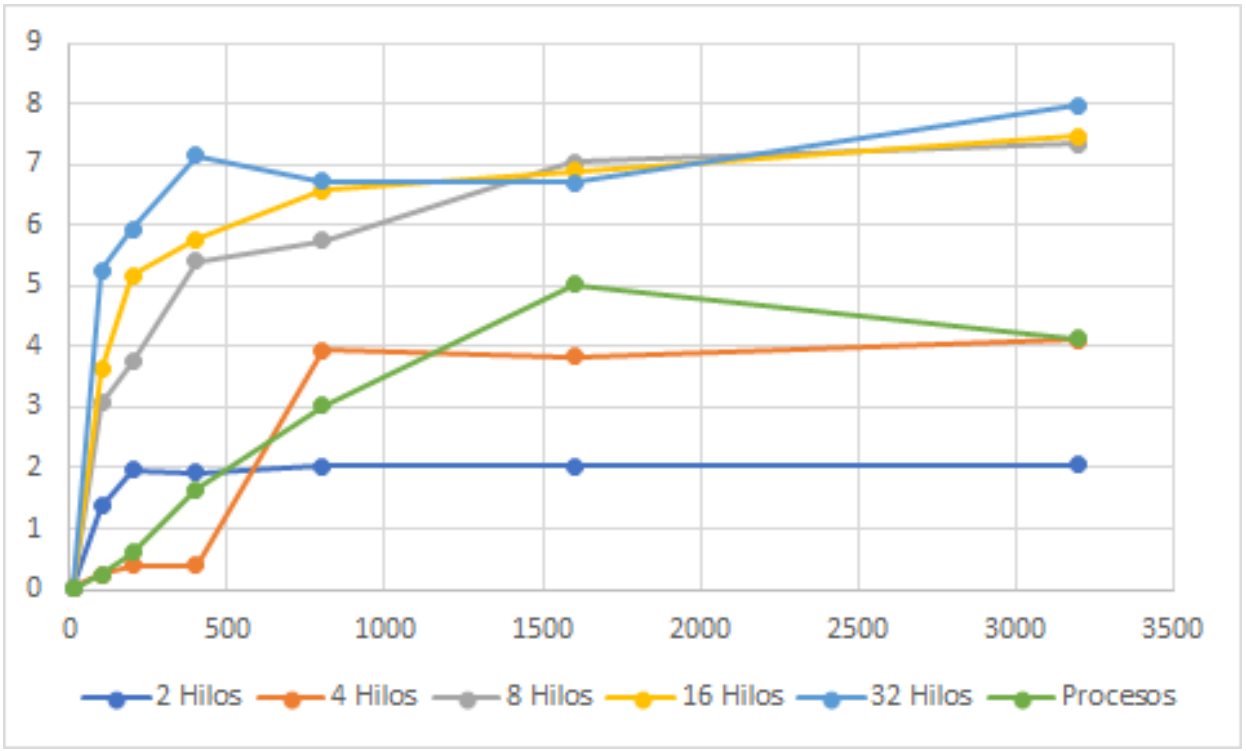
Gráfico 1. Tiempo de ejecución en función de las dimensiones de la matriz.



Fuente: Elaboración propia.

A continuación se muestra una gráfica comparativa entre entre todos los speed up de cada método de paralelización usado, es bueno recordar que el speedup es la división de el tiempo que se tiene en la ejecución secuencial, sobre el tiempo obtenido al paralelizar, por lo que en la gráfica no se incluye en speed up del método secuencial ya que este es una constante igual a 1.

Gráfico 2. Valor de Speed Up en función de las dimensiones de la matriz.



Fuente: Elaboración propia.

Conclusiones

1. Luego de analizar los datos obtenidos, se puede concluir que el uso de hilos es muy necesario para trabajar con matrices de dimensiones superiores a 1200, ya que a partir de este tamaño el tiempo de ejecución en secuencial es mucho mayor.
2. Los valores de speedup de las ejecuciones con hilos, aumentan proporcionalmente al aumento de los hilos utilizados
3. Usar más de ocho hilos es innecesario, debido a que la diferencia entre los speedup obtenidos con las ejecuciones de 16 y 32 hilos, es muy pequeña.
4. Usar procesos no es tan recomendable, ya que el speed up que se obtiene con pocas o altas dimensiones, no sobresale demasiado frente a paralizaciones superiores a 4 hilos
5. Analizando la gráfica 1, se nota un pico cerca a las ejecuciones con matrices de dimensiones 800, esto, según suposición nuestra, debido al proceso de asignación de memoria que es necesario hacer para poder trabajar con matrices de dimensión superior a 830 aproximadamente

Bibliografía

Cortéz, A. (2004). TEORÍA DE LA COMPLEJIDAD COMPUTACIONAL Y TEORÍA DE. *Revista De Investigación De Sistemas E Informática, 1*(1), 102-105. Recuperado de: https://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/view/3216

Ferestrepoca. (2019). Recuperado de: http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html

López, A. d. (2017, 11 30). *Encamina.* From Por una nube sostenible: https://blogs.encamina.com/por-una-nube-sostenible/introduccion-al-high-performance-computing/

Montero, L. H., & Antunez, R. R. (2011, 07). *Parallel programming: definitions, mechanisms and trouble.* From https://www.researchgate.net/publication/274960405_Parallel_programming_definitions_mechanisms_and_trouble

Progressa Lean. (2014, 11 2). *Progressa Lean*. From https://www.progressalean.com/que-es-speed-up/

Qaz Wiki. (2020). *Qaz Wiki*. From https://es.qaz.wiki/wiki/Matrix_multiplication_algorithm