Programación Concurrente y Paralela

Practica 1 y 3

Hernán Iglesias Ramos

INTRODUCCION

En esta práctica, lo que se va a hacer es realizar un producto matricial completo de diferentes maneras, primero se realiza de manera secuencial, tanto con la matriz original como con la transpuesta, y luego diferentes maneras de dividir las matrices para mejorar el uso de memoria y/o paralelizar el sistema para ver las diferentes mejoras en el tiempo de ejecución y ver cuál es la solución más optima. En el caso de que dividamos la matriz en diferentes bloques, el tamaño del bloque que he decidido escoger es de 50 (parámetro blk será igual a 50).

Análisis teórico

Análisis a priori de los diferentes sistemas de ejecución sobre los que se probaran los algoritmos implementados.

TPP_{DP} (máxima velocidad teórica)

Para calcular el TPP_{dp} de los tres procesadores de los que disponemos usaremos una formula vista en clase:

$$TPP_{dp} = chasis * \ \ nodos_{chasis} * sockets_{nodo} * cores\ _{sockect} * clock_{GHz} * \frac{nºflop}{ciclo}$$

$$TPP_{dp} = sockets * cores\ _{sockect} * clock_{GHz} * (AVX\ O\ FMA)$$

A continuación, procedemos a calcular el TPP_{dp} de todos los nodos de cálculo en los cuales se ejecutarán los algoritmos:

- Intel Core i3-2100 CPU @ 3,10GHz con 2 núcleos.

$$TPP_{dp} = 1 * 1 * 1 * 2 * 3{,}10 * 8 = 49{,}6\ Gflop$$

- Intel Xeon E5620 @ 2,40GHz con 4 núcleos y 2 procesadores.

$$TPP_{dp} = 1 * 1 * 2 * 4 * 2{,}40 * 8 = 153{,}6\ Gflop$$

- AMD Ryzen 7 3700x @ 3,60GHz con 8 núcleos.

$$TPP_{dp} = 1 * 1 * 8 * 3{,}60 * 16 = 460{,}8\ Gflop$$

Podemos ver que el procesador con mayor capacidad de cálculo es el Ryzen, seguido del Xeon y del I3

Tiempo por FLOP (teórico mínimo)

El tiempo por Flop teórico mínimo se calcula como la inversa de TPP_{dp}.

$$t_c = \frac{1}{TPP_{dp}}$$

- Intel Core TM I3-2100 @ 3,10GHz con 2 núcleos.

$$t_c = \frac{1}{49{,}6\ Gflop} = 2{,}016 * 10^{-11}$$

- Intel Xeon E5620 @ 2,40GHz con 4 núcleos y 2 procesadores.

$$t_c = \frac{1}{153{,}5\ Gflop} = 6{,}510 * 10^{-12}$$

- AMD Ryzen 7 3700x @ 3,60GHz con 8 núcleos.

$$t_c = \frac{1}{460{,}8\ Gflop} = 2{,}170 * 10^{-12}$$

Complejidad temporal y espacial

La complejidad espacial, se calcula sabiendo los tamaños de las matrices que utilizamos.

Siendo las matrices, A[m * k] , B[k * n] y C[m * n].

La complejidad temporal, será el resultado de todo el conjunto, es decir, de la complejidad espacial y la parte a transponer también.

Python

Complejidad temporal = n*m*k

MyDEGMM

Complejidad espacial:

Matriz C=m*n

Matriz A=m*k

Matriz B=k*n

Total=m*n+m*k+k*n

Complejidad temporal: T(n)=3+m*(2n*4k)tc

MyDEGMMT

Complejidad espacial:

Matriz C=m*n

Matriz A=m*k

Matriz B=k*n

Matriz Bt=n*k

Total= m*n+m*k+k*n+n*k

Complejidad temporal:

Parte de transponer: T(n)=(k/p) *n

Total: T(n,p)=(k/p)*n+m*(n*(2+4k/p))tc

MyDEGMMB (version paralela)

Complejidad espacial:

Matriz C=m*n

Matriz A=m*k

Matriz B=k*n

Matriz Bt=n*k

Total= m*n+m*k+k*n+n*k

Complejidad temporal:

Parte de transponer: T(n)=(k/p)*n

Total: T(n,p)=(k/p)*n+4+(m/p)*n+((m/p)*(n*k)/blk)+(m/p)*(2n*4k)tc

MyDEGMMB (versión secuencial)

Complejidad espacial:

Matriz C=m*n

Matriz A=m*k

Matriz B=k*n

Matriz Bt=n*k

Total= m*n+m*k+k*n+n*k

Complejidad temporal:

Parte de transponer: T(n)=k*n

Total: T(n)=k*n+ 4+m*n+(m*(n*k))+m*(2n*4k)tc

SPPED-UP TEÓRCIO

Intel I3 – tiene 2 núcleos, con lo que el SpeedUp será 2.

Xeon – tiene 2 procesadores, y por cada procesador 4 núcleos, tendrá un SpeedUp de 8.

Ryzen – tiene 8 núcleos, por lo que el SppedUp será 8.

Esto quiere decir que el speedup será siempre el mismo, constante. Algo que en los tiempos experimentales nos dice que no es cierto, ya que el tiempo teórico no contempla diferentes variables.

(Los tiempos en las tablas, están en formato Científico).

TABLAS

Python

Python	***Maquina : I3***					
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico**
1000	1001	999	1,00E+09	2,02E-11	2,02E-02	7,51E-02
2000	2001	1999	8,00E+09	2,02E-11	1,62E-01	5,55E-01
3000	3001	2999	2,70E+10	2,02E-11	5,45E-01	1,85E+00

Python	***Maquina : Xeon***					
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico**
1000	1001	999	1,00E+09	6,51E-12	6,51E-03	4,52E-02
2000	2001	1999	8,00E+09	6,51E-12	5,21E-02	3,21E-01
3000	3001	2999	2,70E+10	6,51E-12	1,76E-01	8,85E-01

Python	***Maquina : Ryzen***					
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico**
1000	1001	999	1,00E+09	2,17E-12	2,17E-03	1,44E-02
2000	2001	1999	8,00E+09	2,17E-12	1,74E-02	6,22E-02
3000	3001	2999	2,70E+10	2,17E-12	5,86E-02	1,76E-01

MyDEGMM

MyDGEMM	*Maquina : I3*			*Secuencial*			
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1001	999	8,000E+09	2,02E-11	1,616E-01	2,16E+00	7,76E-01
2000	2001	1999	6,400E+10	2,02E-11	1,293E+00	1,86E+01	8,56E+00
3000	3001	2999	2,160E+11	2,02E-11	4,363E+00	5,48E+01	1,97E+01

MyDGEMM	*Maquina : Xeon*			*Secuencial*			
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1001	999	8,000E+09	6,51E-12	5,208E-02	9,50E-01	2,67E-01
2000	2001	1999	6,400E+10	6,51E-12	4,166E-01	8,81E+00	4,08E+00
3000	3001	2999	2,160E+11	6,51E-12	1,406E+00	2,79E+01	1,09E+01

MyDGEMM	*Maquina : Ryzen*			*Secuencial*			
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1001	999	8,000E+09	2,17E-12	1,736E-02	3,44E-01	7,76E-02
2000	2001	1999	6,400E+10	2,17E-12	1,389E-01	2,77E+00	9,09E-01
3000	3001	2999	2,160E+11	2,17E-12	4,687E-01	9,46E+00	3,07E+00

MyDGEMM		*Maquina: I3*		*Parelelo*			
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1001	999	4,000E+09	2,02E-11	8,080E-02	1,88E+00	3,37E-01
2000	2001	1999	3,200E+10	2,02E-11	6,464E-01	1,48E+01	3,09E+00
3000	3001	2999	1,080E+11	2,02E-11	2,182E+00	5,05E+01	1,20E+01

MyDGEMM	*Maquina: Xeon*			*Parelelo*			
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1001	999	4,000E+09	6,51E-12	2,604E-02	9,25E-01	9,53E-02
2000	2001	1999	3,200E+10	6,51E-12	2,083E-01	7,48E+00	2,46E+00
3000	3001	2999	1,080E+11	6,51E-12	7,031E-01	2,59E+01	7,93E+00

MyDGEMM	*Maquina: Ryzen*			*Parelelo*			
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1001	999	1,001E+09	2,17E-12	2,17E-03	3,49E-01	2,67E-02
2000	2001	1999	8,004E+09	2,17E-12	1,74E-02	2,83E+00	7,43E-01
3000	3001	2999	2,710E+10	2,17E-12	5,88E-02	9,60E+00	2,69E+00

MyDEGMMT

MyDGEMMT		*Maquina : Ryzen*			*Paralelo*		
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1001	999	5,021E+08	2,17E-12	1,09E-03	6,69E-01	1,04E+00
2000	2001	1999	4,009E+09	2,17E-12	8,70E-03	5,07E+00	4,23E+00
3000	3001	2999	1,352E+10	2,17E-12	2,93E-02	1,69E+01	9,01E+00

MyDEGMMB

MyDGEMMB	*Maquina: Ryzen*				*Secuencial*		
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1000	1000	9,00E+09	2,17E-12	1,95E-02	2,54E+00	1,77E-01
2000	2000	2000	7,20E+10	2,17E-12	1,56E-01	2,03E+01	1,48E+00
3000	3000	3000	2,43E+11	2,17E-12	5,27E-01	6,86E+01	5,03E+00

MyDGEMMB	*Maquina : Ryzen*				*Paralelo*		
m	**n**	**k**	**nº flop**	**tc**	**teorico**	**empirico O0**	**empiricoO3**
1000	1000	1000	1,003E+09	2,17E-12	2,18E-03	4,11E-01	3,61E-02
2000	2000	2000	8,021E+09	2,17E-12	1,74E-02	2,79E+00	2,20E-01
3000	3000	3000	2,707E+10	2,17E-12	5,87E-02	9,84E+00	7,72E-01

Cálculos de las diferentes tablas.

Los valores de m, n y k siempre serán {1000, 1001, 999}, {2000,20001, 1999}, {3000, 30001, 2999} respectivamente, salvo en la función de bloques en la cual será m=n=k.

El valor de número de flops, lo obtenemos de usar T(n) o T(n,p), según el tipo de función que estemos usando, sustituyendo los valores de m, n, k y p.

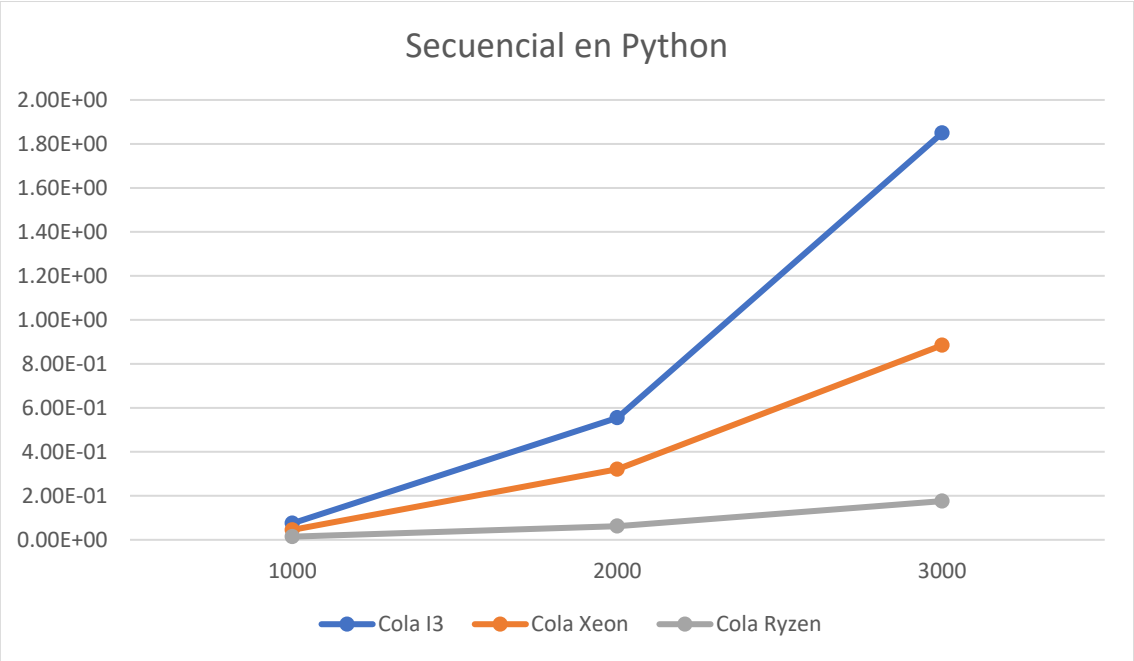
El t_c, como bien comenté antes, es la inversa de los Gigaflops que puede ejecutar cada procesador. Y el tiempo teórico, es el resultado del producto del número de flops y el t_c.

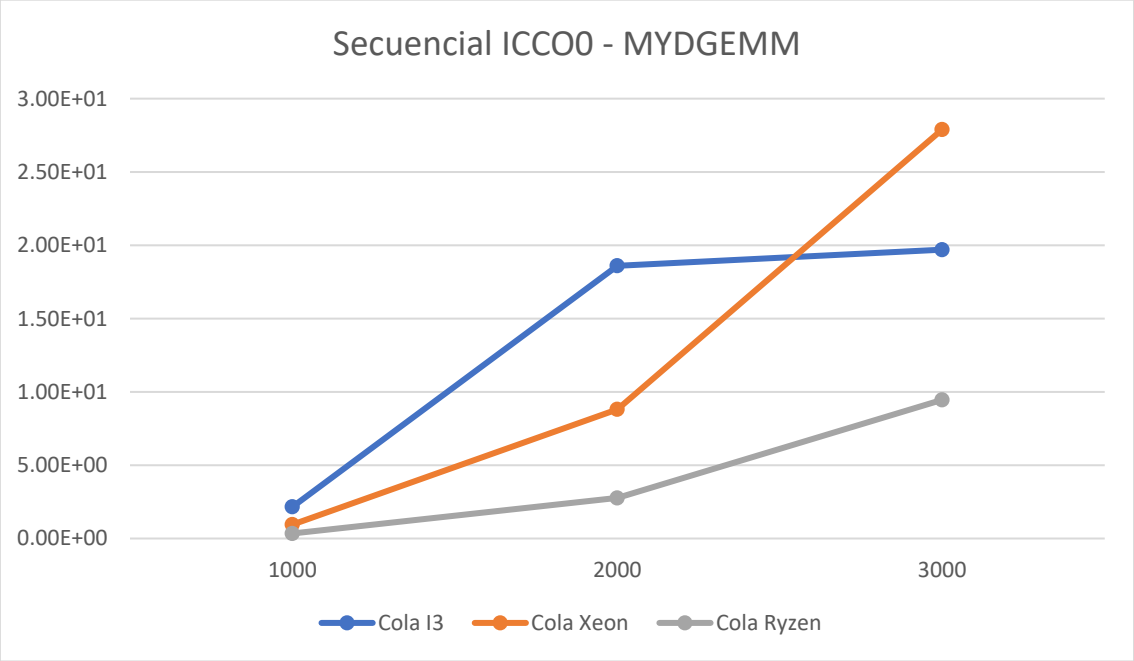
Observando los tiempos teóricos, vemos que, son como poco 10 veces mejores que los experimentales, algo que se podría deber a que el tiempo teórico contempla únicamente el número de instrucciones de cada función, y que l procesador solo ejecuta esa función, cuando en la realidad, está ejecutando diferentes cosas, hay tiempos entre el paso de información en la memoria y latencias.

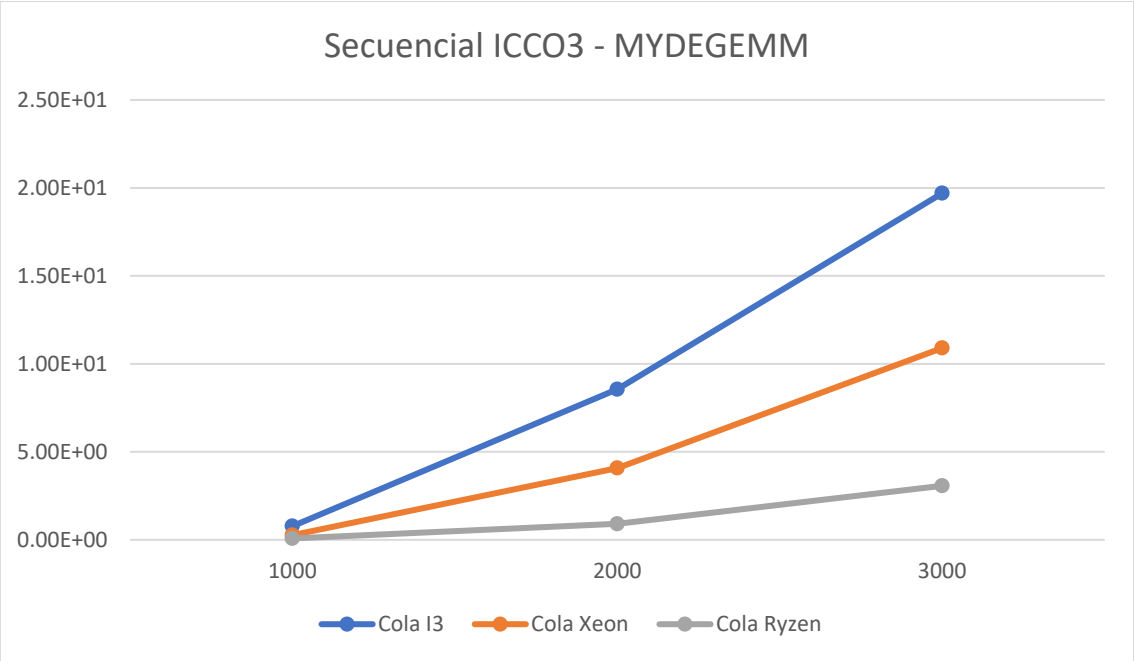
CONCLUSIONES

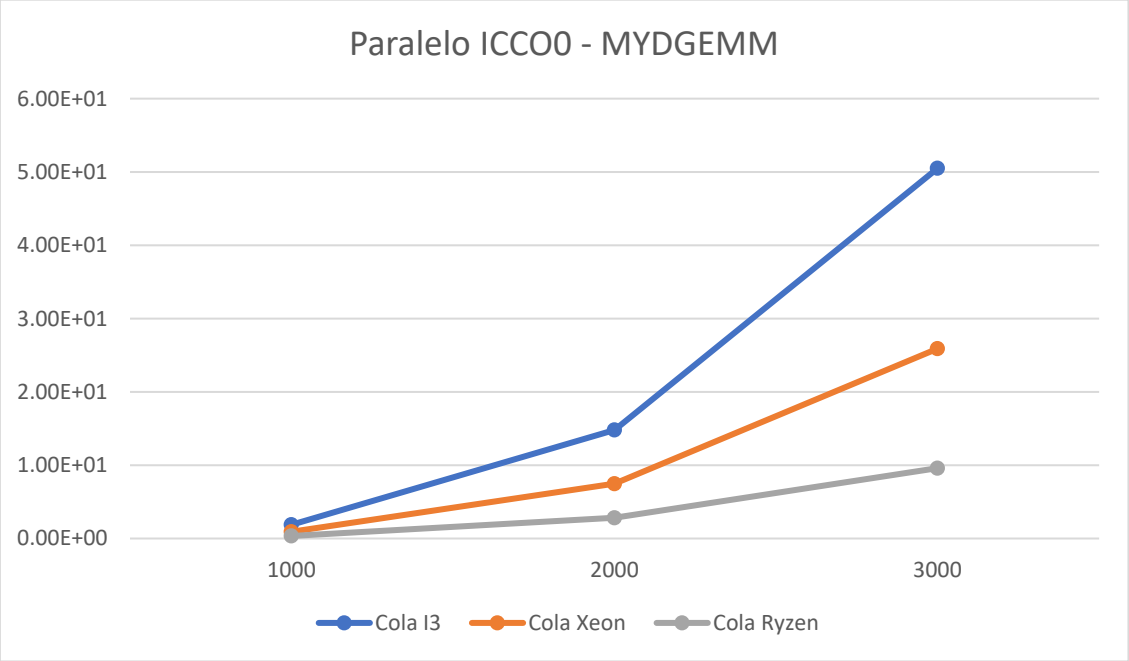
Hay que comentar que el compilador utilizado es el icc, ya que es mejor que el gcc, a causa de que los mejores tiempos para todos los algoritmos son generados por el compilador icc, se puede deber a que se emplean optimizaciones mejores que las empleadas por el gcc.

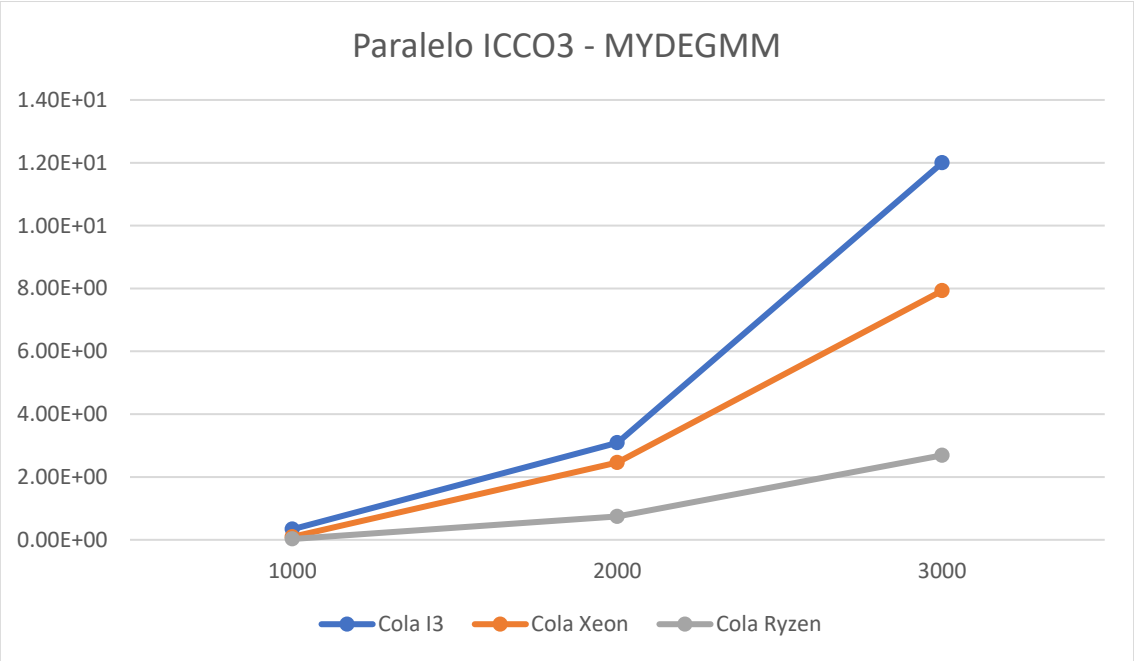
Y dentro de icc, distinguimos de icco0 e icco3, fijándonos en la recogida de tiempos vemos que resultan unos tiempos más bajos con la librería icco3, es decir, es más rápida.


Secuencial en Python
2.00E+00
1.80E+00
1.60E+00
1.40E+00
1.20E+00
1.00E+00
8.00E-01
6.00E-01
4.00E-01
2.00E-01
0.00E+00
1000
2000
3000
Cola I3
Cola Xeon
Cola Ryzen


Secuencial ICCO0 - MYDGEMM
3.00E+01
2.50E+01
2.00E+01
1.50E+01
1.00E+01
5.00E+00
0.00E+00
1000
2000
3000
Cola I3
Cola Xeon
Cola Ryzen


Secuencial ICCO3 - MYDEGEMM
2.50E+01
2.00E+01
1.50E+01
1.00E+01
5.00E+00
0.00E+00
1000
2000
3000
Cola I3
Cola Xeon
Cola Ryzen

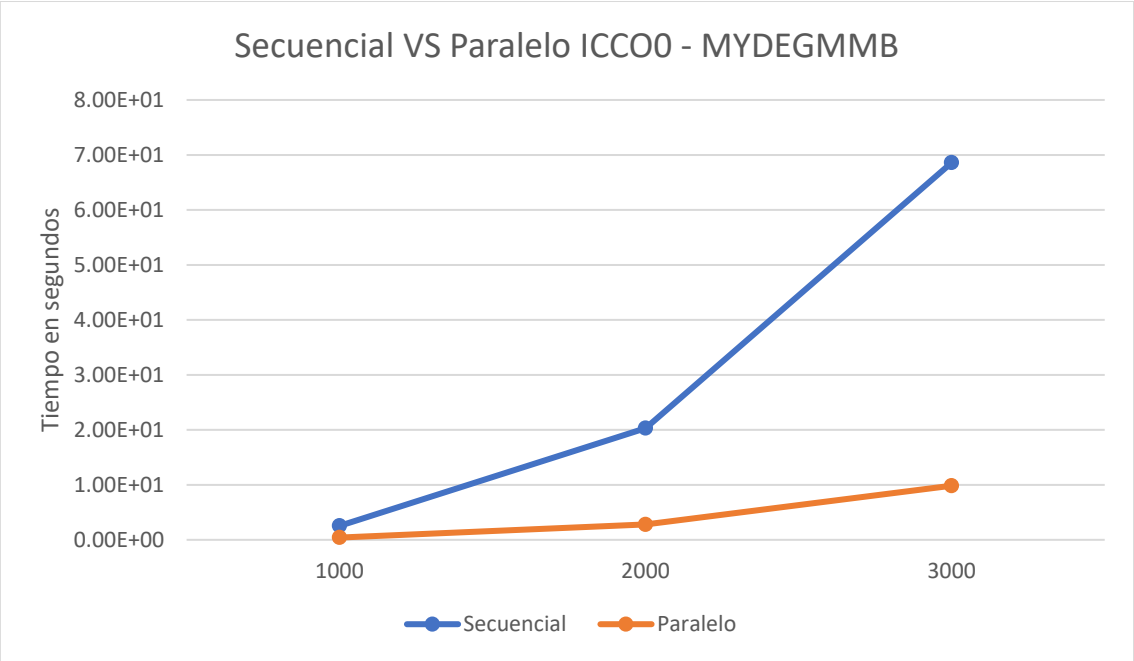

Paralelo ICCO0 - MYDGEMM
6.00E+01
5.00E+01
4.00E+01
3.00E+01
2.00E+01
1.00E+01
0.00E+00
1000
2000
3000
Cola I3
Cola Xeon
Cola Ryzen

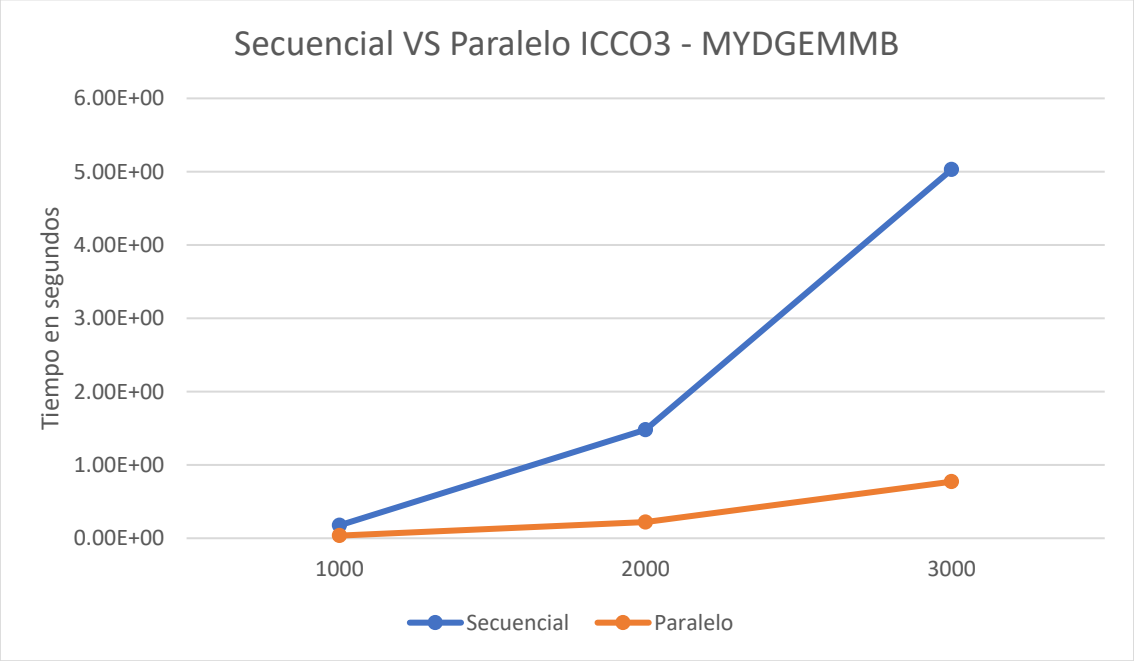


¿Qué encontramos de particular en esta serie de graficas que he realizado?

Vemos que tenemos solo gráficas, en las que actúan las tres máquinas, eso solo nos sucede en Python y en la funcion de MyDEGMM (tanto secuencial como en paralelo), y lo que nos encontramos es que según los cálculos realizados en el análisis teórico la cola más rápida es la GPU (es la que siempre tiene los tiempos más bajos). Sin embargo, la siguiente más rápida es la Xeon salvo en, secuencial de MyDEGMM (icco0), para cierta talla, es la cola I3, la cual es más rápida. Esto se debe a que no se hace un buen uso de los dos procesadores que tiene la cola Xeon, motivo por el cual es más rápida que la cola I3.

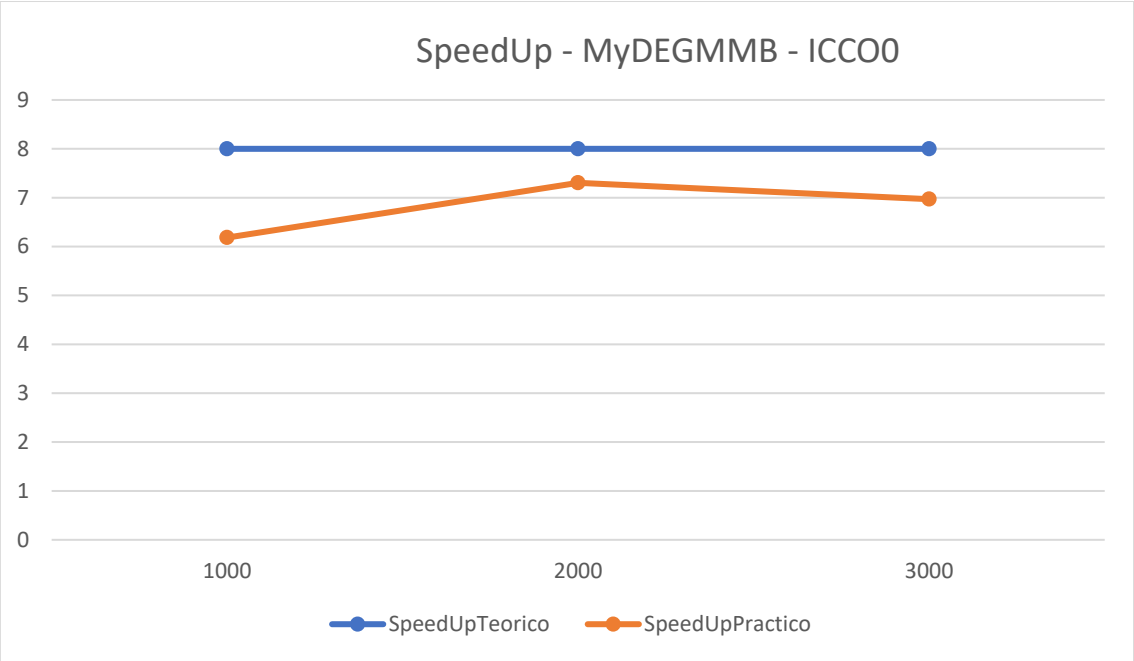
Ahora vamos a centrarnos solo en visualizar la cola GPU, al ser la más rápida. Vamos a ver las diferencias entre Secuencial y Paralelo, el estudio más importante

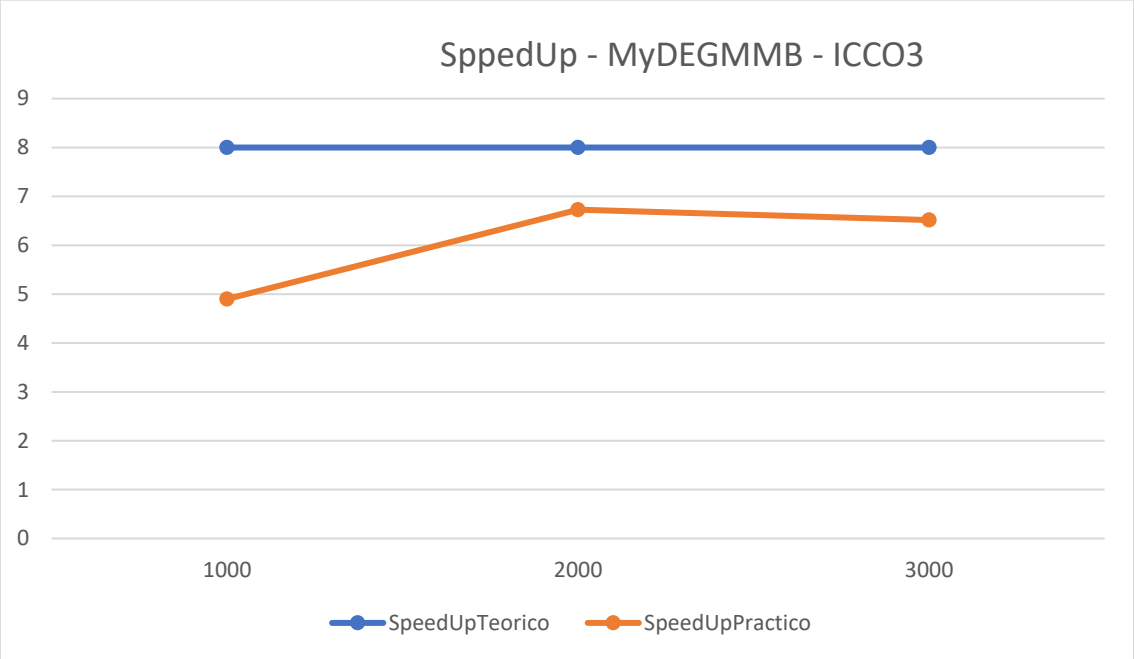




Lo que observamos al visualizar estas dos gráficas, es que la versión paralela resulta más optima y eficiente que la secuencial, teniendo un mayor grado de aprovechamiento del trabajo.

Esto se debe, a que, al dividir el problema en varios hilos, o tareas, nos permite que en cada instante se puedan estar llevando a cabo diferentes tareas (partes del código) lo cual no podrá suceder en la parte secuencial.





Muestra el speedup la paralelización del producto matricial por bloques, donde vemos que el speedup es sub-lineal, al estar por debajo del speedup teórico, puede ser a causa de la dependencia de datos y que en ocasiones no es posible dividir en subproblemas.

En resumen, el mejor tiempo de ejecución del producto matricial completo se obtiene cuando se realiza una implementación paralela con transpuesta y bloques, compilada con icc y ejecutada sobre la cola GPU.