Prácticas 1 y 3: Programación Concurrente y Paralela

Producto Matricial Completo: Secuencial – Paralelo

Isidro Benítez Zapico

Contenido

Introducción 3
TPPdp y Tiempo por Flop teórico mínimo 4
Speedup y Eficiencias teóricas 4
Complejidad Espacial 5
Complejidad Temporal 5
Tiempos Teóricos y Empíricos 6
Conclusiones 11

Introducción

La idea de este trabajo es aplicar los conocimientos de paralelismo con OpenMP para resolver un producto matricial completo, ya sea en una versión normal o utilizando la transpuesta de la matriz A. También se busca estudiar la complejidad, eficiencia y demás detalles de los diferentes algoritmos para ver sus diferencias. Aplicando técnicas y mejoras como Row-Major, flags de optimización de los diferentes compiladores además del uso de bloques y tareas.

TPPdp y Tiempo por Flop teórico mínimo

Para calcular el TPPdp de los diferentes nodos de cálculo se utilizará la siguiente formula:

$$TPP_{dp} = chasis * nodos_{chasis} * sockets_{nodo} * cores_{socket} * clock_{GHz} * \frac{nº flop}{ciclo}_{dp}$$

- i3-2100: 1*2*3.1*8 = 49.6 TPPdp (Gflop)
- Xeon CPU E5- 2603: 2*6*1.7*16 = 326.4 TPPdp (Gflop)
- Ryzen 7 3700X: 1*8*3.6*16 = 460.8 TPPdp (Gflop)

Gracias a los datos obtenidos podemos calcular los tiempos por flop teóricos mínimos:

$$Tc = 1/TPPdp$$

- I3 Secuencial = 1 / (49.6 * 10^9) / 2 = 4.0322E-11
- I3 Paralelo = 1 / (49.6 * 10^9) = 2.0161E-11
- Xeon Secuencial = 1 / (326.4 * 10^9) / 12 = 3.6764E-11
- Xeon Paralelo = 1 / (326.4 * 10^9) = 3.0637E-12
- Ryzen Secuencial = 1 / (460.8 * 10^9) / 8 = 1.7361E-11
- Ryzen Paralelo = 1 / (460.8 * 10^9) = 2.17013E-12

Speedup y Eficiencias teóricas

Si dividimos Secuencial entre Paralelo obtenemos los siguientes speedup teóricos:

- I3 = 4.0322E-11 / 2.0161E-11 = 2
- Xeon = 3,6764E-11 / 3.0637E-12 = 12
- Ryzen = 1.7361E-11 / 2.17013E-12 = 8

Con estos podemos calcular la eficiencia teórica aplicando la siguiente formula:

$$Eficiencia = SpeedUp/cores$$

- Intel I3 = (2 / 2) = 1
- Xeon = (12 / 12) = 1
- Ryzen = (8 / 8) = 1

La eficiencia teórica es 1, es decir el caso ideal por tanto la experimental deber ser ≤ 1.

Complejidad Espacial

La complejidad espacial no varía en función de si el algoritmo es secuencial o paralelo, sino que para calcularla hay que tener en cuenta el nº de estructuras usadas.

Para el problema C=βC+αAB la complejidad espacial sería n*m + m·k + n*k y la complejidad de transponer una matriz NxM sería de n*m.

Complejidad Temporal

La complejidad temporal es el tiempo que tarda el programa en ejecutarse, en este caso como el programa son varias funciones, dependiendo de que función o funciones ejecutemos, su coste temporal variará.

La complejidad temporal secuencial de cada problema es la siguiente:

- Producto matricial completo C=βC+αAB

$$MyDGEMM/T: T(n,m,k,p) = m * n * (2 + k) * tc$$

$$MyDGEMMB: T(n,m,k,p,blk) = n^2 * (\lambda + n) * tc$$

- Transpuesta de una matriz NxM: n*m.

La complejidad temporal en paralelo de cada problema es la siguiente:

MyDGEMM en paralelo es similar a MyDGEMMT por tanto se obtiene con la misma fórmula:

$$MyDGEMM/T: T(n,m,k,p) = \frac{m}{p} * n * (2 + k) * tc$$

En MyDGEMMB como n = m = k se simplifica la formula hasta obtener la siguiente

expresión:

$$MyDGEMMB: T(n,m,k,p,blk) = \frac{n^2 * (\lambda + n)}{p} * tc$$

Tiempos Teóricos y Empíricos

Utilizando El tiempo por Flop teórico mínimo y multiplicándolo por la complejidad temporal (añadiendo parámetros como la talla del problema y bloque), obtenemos los tiempos teóricos para cada método, variando en función del procesador y si esta paralelizado o no.

Aquí tenemos las tablas de MyDGEMM con todos sus tiempos para los diferentes procesadores:

I3 - MyDGEMM	N	M	K	Tc	nFlops	Tiempo Teórico	Python	O0	O3
Secuencial	1001	1000	999	4,03E-11	1,00E+09	0,04	0,10	2,55	0,67
	2001	2000	1999	4,03E-11	8,01E+09	0,32	0,72	20,38	5,71
	3000	3001	2999	4,03E-11	2,70E+10	1,09	2,40	68,89	19,10
Paralelo	1001	1000	999	2,02E-11	5,01E+08	0,01	0,10	0,39	0,14
	2001	2000	1999	2,02E-11	4,00E+09	0,08	0,41	2,73	0,80
	3000	3001	2999	2,02E-11	1,35E+10	0,27	1,35	9,12	2,92

Xeon - MyDGEMM	N	M	K	Tc	nFlops	Tiempo Teórico	Python	O0	O3
Secuencial	1001	1000	999	3,68E-11	1,00E+09	0,04	0,09	6,73	1,73
	2001	2000	1999	3,68E-11	8,01E+09	0,29	0,69	53,97	14,28
	3000	3001	2999	3,68E-11	2,70E+10	0,99	2,28	182,55	48,25
Paralelo	1001	1000	999	3,06E-12	1,67E+08	0,00	0,03	1,13	0,29
	2001	2000	1999	3,06E-12	1,33E+09	0,00	0,17	9,03	2,43
	3000	3001	2999	3,06E-12	4,50E+09	0,01	0,47	30,55	8,20

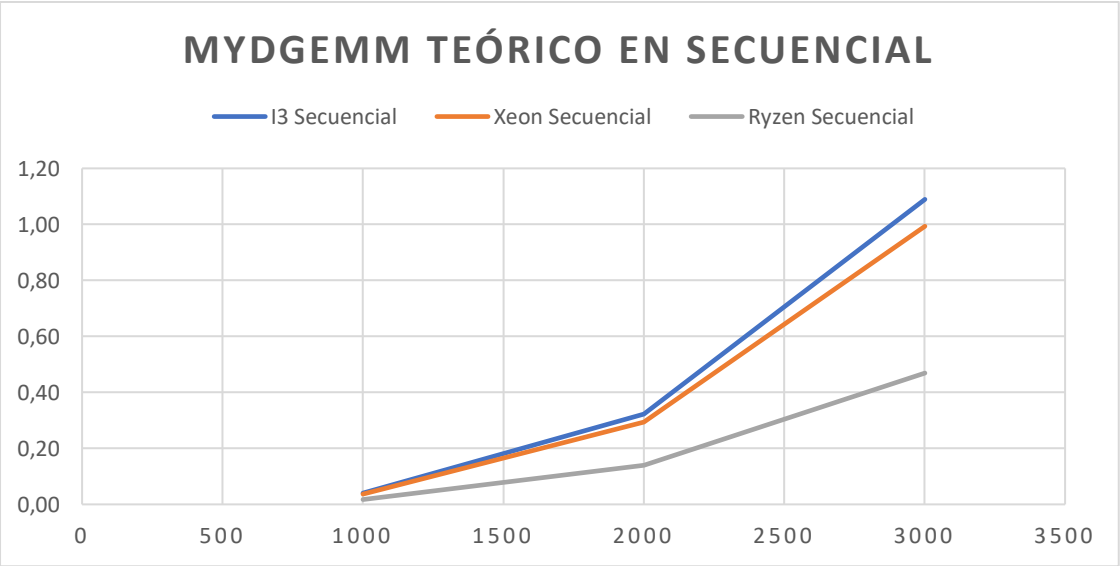
Ryzen - MyDGEMM	N	M	K	Tc	nFlops	Tiempo Teórico	Python	O0	O3
Secuencial	1001	1000	999	1,74E-11	1,00E+09	0,02	0,04	2,55	0,68
	2001	2000	1999	1,74E-11	8,01E+09	0,14	0,27	20,38	5,71
	3000	3001	2999	1,74E-11	2,70E+10	0,47	0,89	68,89	19,10
Paralelo	1001	1000	999	2,17E-12	1,25E+08	0,00	0,03	0,39	0,14
	2001	2000	1999	2,17E-12	1,00E+09	0,00	0,10	2,73	0,80
	3000	3001	2999	2,17E-12	3,38E+09	0,01	0,22	9,12	2,92

Y aquí las tablas de MyDGEMMB y MyDGEMMT Transpuesta para Ryzen:

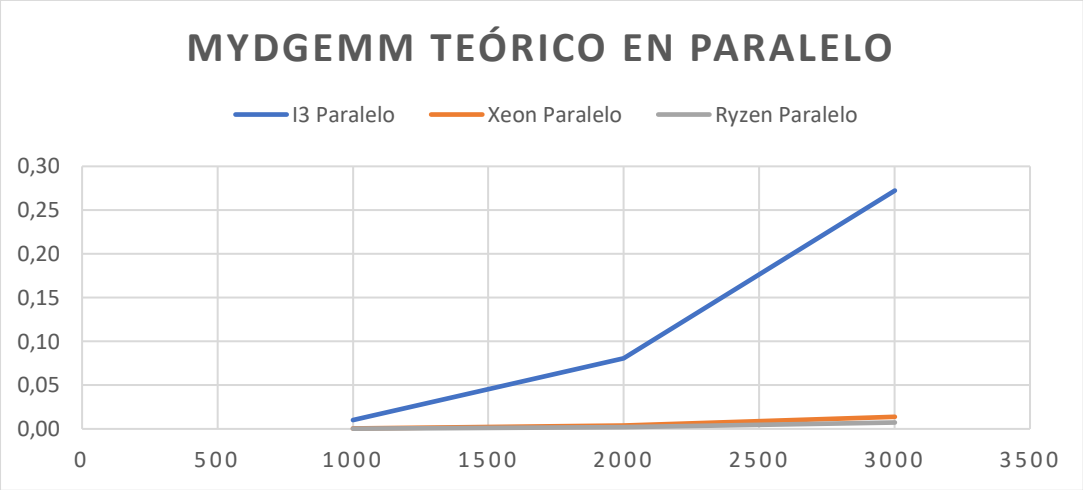
Ryzen - MyDGEMMB	N	M	K	Tc	nFlops	Tiempo Teórico	Python	O0	O3
Secuencial	1000	1000	1000	1,74E-11	1,02E+09	0,02	0,04	2,63	0,51
	2000	3000	2000	1,74E-11	1,22E+10	0,21	0,27	21,03	4,07
	3000	2000	3000	1,74E-11	1,84E+10	0,32	0,89	70,68	13,17
Paralelo	1000	1000	1000	2,1701E-12	1,28E+08	0,00	0,01	0,59	0,11
	2000	2000	2000	2,1701E-12	1,02E+09	0,00	0,09	3,28	0,73
	3000	3000	3000	2,1701E-12	3,44E+09	0,01	0,20	9,83	1,93

Ryzen - MyDGEMMT Transpuesta	N	M	K	Tc	nFlops	Tiempo Teórico	Python	O0	O3
Paralelo	1001	1000	999	1,74E-11	8,02E+03	0,00	0,01	0,44	0,12
	2001	2000	1999	1,74E-11	1,60E+04	0,00	0,09	3,27	0,80
	3000	3001	2999	1,74E-11	2,40E+04	0,00	0,17	10,98	2,81

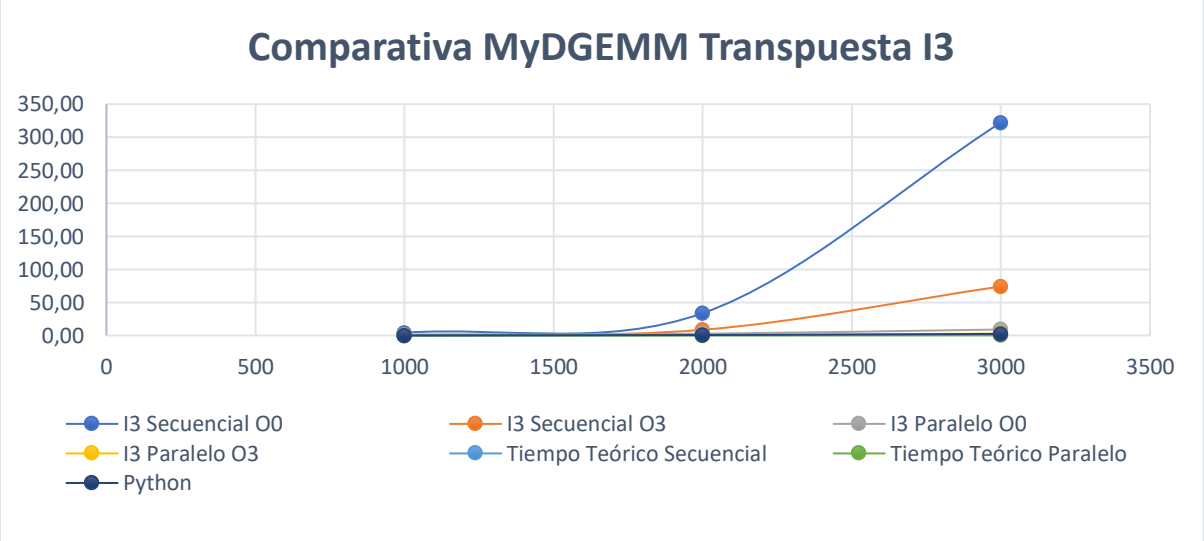
Podemos observar cómo en secuencial no hay diferencias notables entre I3 y Xeon, siendo el Ryzen un poco más rápido.

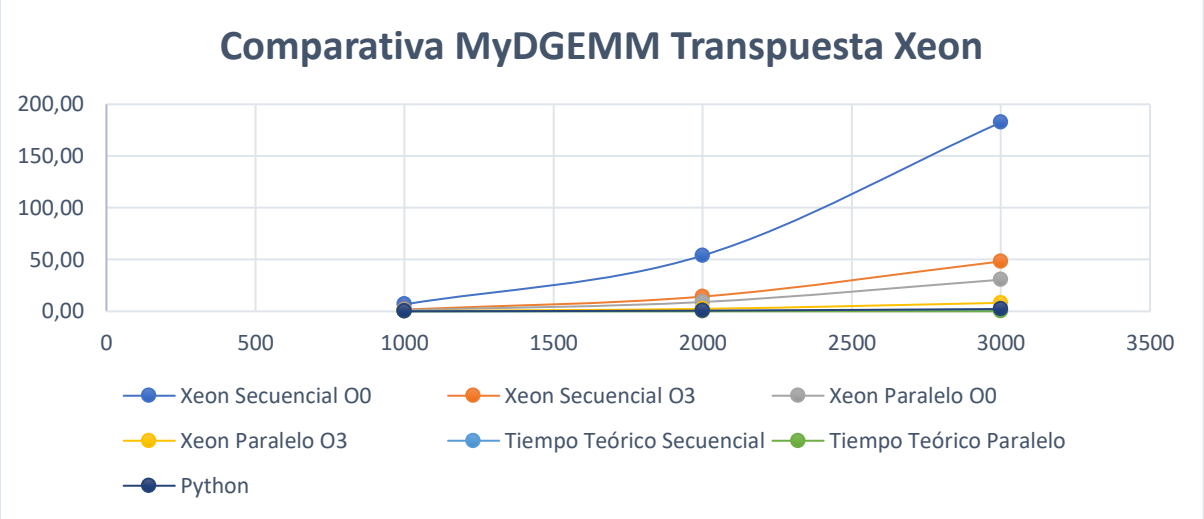


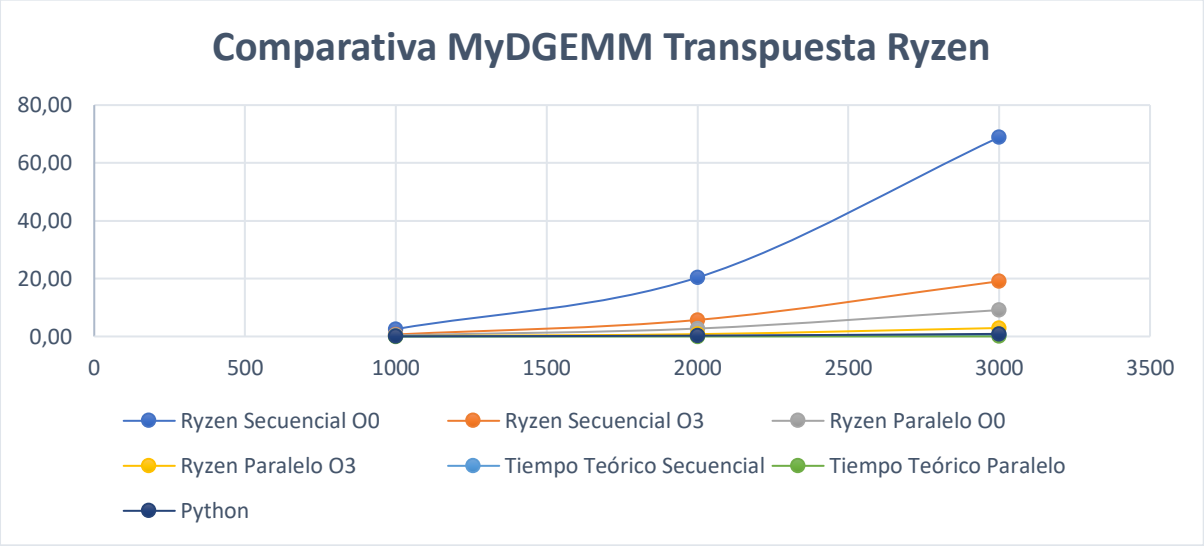
Mientras que, al utilizar los diferentes procesadores con todos sus núcleos, la diferencia aumenta significativamente, donde podemos concluir que el rendimiento es similar entre Ryzen y Xeon mientras que el I3 sale perjudicado al tener únicamente dos núcleos.



Si comparamos los tiempos de Python con los de C (tanto O0 como O3) y teóricos obtenemos gráficas como éstas:

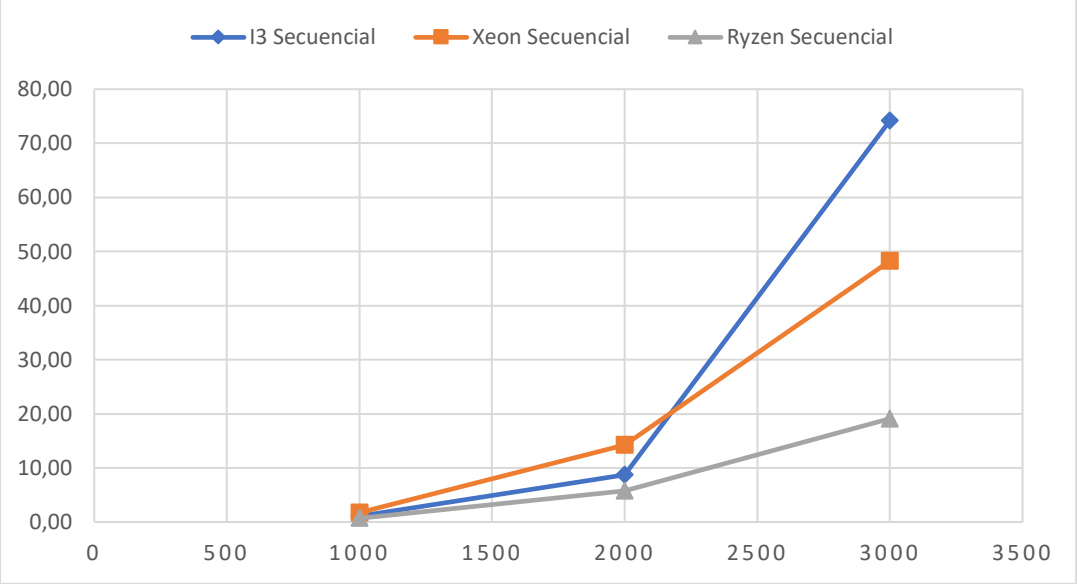




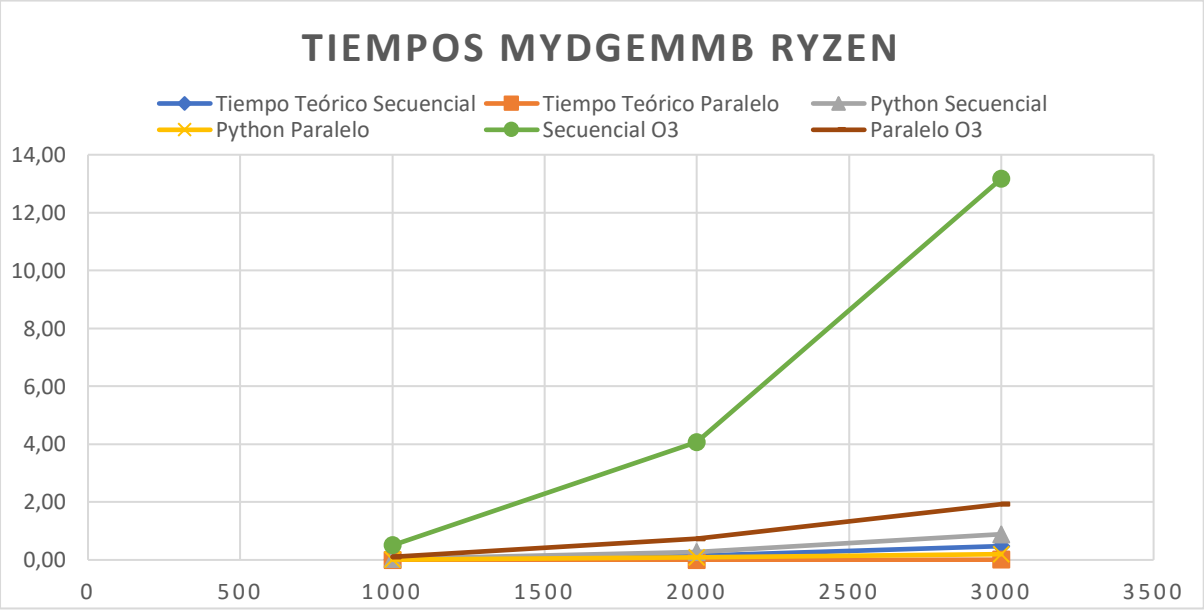


De estas gráficas podemos sacar varias conclusiones, el paralelismo es mucho más eficiente que las versiones secuenciales y las técnicas de optimización en el compilado ayudan a mejorar el rendimiento además se cumplen los limites teóricos calculados anteriormente.

Si comparamos las versiones secuenciales, podemos ver cómo, aunque al principio el I3 se muestra mejor que el Xeon, el resultado es que el I3 es el peor de todos, por delante va el Xeon y finalmente el Ryzen.



Finalmente, otro gráfico que resulta interesante es la comparativa entre tiempos Secuenciales, Paralelos teóricos y Python de MyDGEMMB.



Conclusiones

El paralelismo es esencial en la computación, demostrando tanto en el plano teórico como en el empírico que la utilización de múltiples CPU, la división de problemas en bloques o en tareas y la optimización de los compiladores disminuye significativamente el tiempo de procesamiento. Por lo que, aunque en los comienzos de la informática se utilizaba una sola CPU, los procesos eran secuenciales y no había ningún tipo de tiempo compartido, hoy en día esto es impensable. Ya que el paralelismo está presente en casi todos los ámbitos de la informática, el Sistema Operativo principalmente, renderizado de gráficos, cálculos matemáticos, interfaces gráficas, Inteligencia Artificial e infinidad de usos más…