66.26 Arquitecturas paralelas

Trabajo Práctico Final

Integrantes:

Alumno	padron
Llauró, Manuel Luis	95736
Blanco, Sebastian Ezequiel	98539

GitHub:
https://github.com/BlancoSebastianEzequiel/66.26-TP-Final

Índice

1. Objetivo **1**

2. Desarrollo teorico **2**
2.1. Speed up 2
2.2. Ley de Amdahl 2
2.3. Ley de Gustafson 3
2.4. Map-reduce 3

3. Implementacion **4**
3.1. Explicacion del modelo 4
3.2. Multiplicacion de matrices por bloques 4
3.2.0.1. Preprocesamiento 4
3.2.0.2. Mapeo 5
3.2.0.3. Reduccion 5
3.3. multiplicacion de matrices de elemento por fila 5
3.3.0.1. Preprocesamiento 5
3.3.0.2. Mapeo 5
3.3.0.3. Reduccion 5
3.4. Multiplicacion de matrices de columna por fila 6
3.4.0.1. Preprocesamiento 6
3.4.0.2. Mapeo 6
3.4.0.3. Reduccion 6
3.5. Forma de ejecucion 6
3.6. Datos sobre la computadora que se utilizó 6

4. Resultados **8**
4.1. Multiplicacion por bloques 8
4.1.0.1. Salida Amdahl 8
4.1.0.2. Salida Gustafson 9
4.2. Multiplicacion elemento por fila 11
4.2.0.1. Salida Amdahl 11
4.2.0.2. Salida Gustafson 12
4.3. Multiplicacion columna por fila 15
4.3.0.1. Salida Amdahl 15
4.3.0.2. Salida Gustafson 16

5. Conclusiones **19**

1. Objetivo

Se propone la verificación empírica de la ley de amdahl (trabajo constante) versus la ley de Gustafson (tiempo constante) aplicada a un problema de paralelismo utilizando el modelo de programación MapReduce.
Haremos una multiplicación de matrices (ambas de NxN) y se realizarán las mediciones de tiempo variando la cantidad de threads involucrados en el procesamiento.
Luego se realizarán las mismas mediciones manteniendo fija la cantidad de threads pero variando la dimensión de las matrices.

2. Desarrollo teorico

2.1. Speed up

Es la mejora en la velocidad de ejecución de una tarea ejecutada en dos arquitecturas similares con diferentes recursos.
La noción de speedup fue establecida por la ley de Amdahl, que estaba dirigida particularmente a la computación paralela. Sin embargo, la speedup se puede usar más generalmente para mostrar el efecto en el rendimiento después de cualquier mejora en los recursos.
De forma genérica se define como:

$$speed\_up = \frac{Rendimiento\_con\_mejora}{Rendimiento\_sin\_mejora} \tag{1}$$

En el caso de mejoras aplicadas a los tiempo de ejecución de una tarea:

$$speed\_up = \frac{T\_ejecucion\_sin\_mejora}{T\_ejecucion\_con\_mejora} \tag{2}$$

2.2. Ley de Amdahl

Utilizada para averiguar la mejora máxima de un sistema de información cuando solo una parte de éste es mejorado.
Establece que la mejora obtenida en el rendimiento de un sistema debido a la alteración de uno de sus componentes está limitada por la fracción de tiempo que se utiliza dicho componente.

Suponiendo que nuestro algoritmo se divide en una parte secuencial `s` u una parte paralelizable `p` y siendo `N` la cantidad de threads, entonces podemos decir que:

$$speed\_up = \frac{s+p}{s+\frac{p}{N}} \tag{3}$$

Amdahl establece un límite superior al speedup que puede obtenerse al introducir una mejora en un determinado algoritmo. Este límite superior está determinado por la porción de la tarea sobre la que se aplique la mejora. Entonces si tomamos la ecuacion anterior y calculamos el limite de la misma con `N` tendiendo a infinito tenemos:

$$speed\_up\_max = 1 + \frac{p}{s} \tag{4}$$

2.3. Ley de Gustafson

Establece que cualquier problema suficientemente grande puede ser eficientemente paralelizado. La ley de Gustafson está muy ligada a la ley de Amdahl, que pone límite a la mejora que se puede obtener gracias a la paralelización, dado un conjunto de datos de tamaño fijo, ofreciendo así una visión pesimista del procesamiento paralelo. Por el contrario la ley de Gustafson propone realizar mas trabajo con la misma cantidad de recursos, de esta manera aprovecho la paralelizacion para calcular mas cosas.

Entonces siendo `s` el tiempo de la ejecucucion de la seccion serie, siendo `p` el tiempo de la ejecucion de la seccion paralela y siendo `N` la cantidad de procesadores podemos calcular el speed up como:

$$speed\_up = \frac{s + p * N}{s + p} \tag{5}$$

2.4. Map-reduce

MapReduce es una técnica de procesamiento y un programa modelo de computación distribuida. El algoritmo MapReduce contiene dos tareas importantes.

Map toma un conjunto de datos y se convierte en otro conjunto de datos, en el que los elementos se dividen en tuplas (pares: clave, valor).

Reduce toma la salida de un mapa como entrada y combina los datos tuplas en un conjunto más pequeño de tuplas.

La principal ventaja de MapReduce es que es fácil de escalar procesamiento de datos en múltiples nodos.

De acuerdo a este modelo, basado en la programación funcional, la tarea del usuario consiste en la definición de una función map y una función reduce y definidas estas funciones, el procesamiento es fácilmente paralelizable, ya sea en una sola máquina o en un cluster.

3. Implementacion

3.1. Explicacion del modelo

La implementación del MapReduce para resolver el problema esta basado en el siguiente esquema:

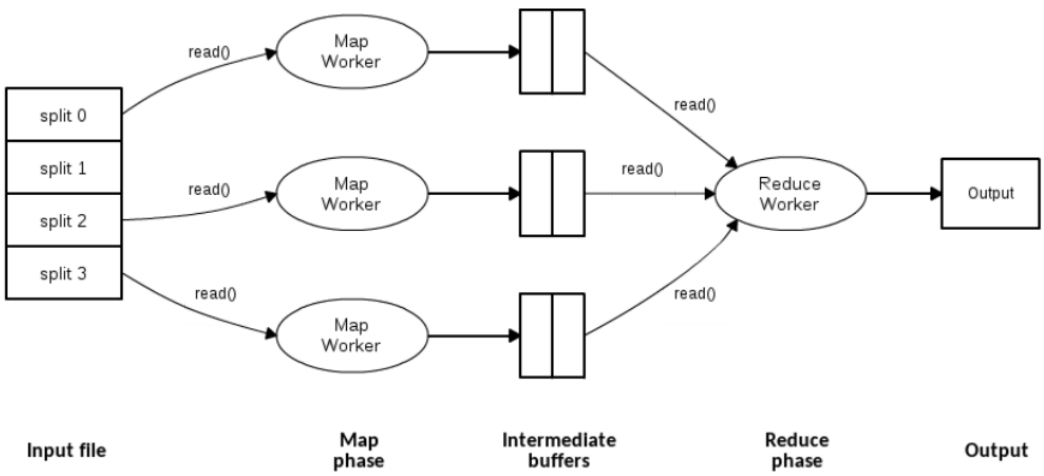

Figura 1: Esquema de un map reduce

En nuestro caso creamos una clase llamada `MapReduce` la cual usa una libreria de `python` llamada `multiprocessing` en donde usamos el modulo `pool` el cual ofrece un medio conveniente para paralelizar la ejecución de una función a través de múltiples valores de entrada, distribuyendo los datos de entrada a través de procesos (paralelismo de datos).

Entonces lo que hicimos fue instanciar dos `pool`, uno para hacer el map y el otro para el reduce de manera que el primero se le pasa como atributo la cantidad de worker en el cual se quiere paralelizar el problema y el segundo solo se usa uno de manera tal que la fase de reduce se la serie.

3.2. Multiplicacion de matrices por bloques

3.2.0.1 Preprocesamiento

Generamos una lista de tuplas donde cada una tiene la posicion `(r, c)` de un bloque de la matriz A, tiene el bloque en custion `a_block_rc`, y la fila numero `c` de bloques de la matriz B, quedando con este formato:
`(r, c, a_block_rc, b_block_c)`

3.2.0.2 Mapeo

Recibimos la posicion `r`, `c` del bloque `a`, el bloque `a` y una lista de bloques `b` que es la fila `c` de bloques en la matriz B.
Entonces multiplicamos el bloque `a` por cada bloque de la lista de bloques `b` y guardamos en un vector una tupla con una clave `r`, `c_b` donde `c_b` es el indice en la lista de bloques `b` y como valor guardamos la multiplicacion. Por cada multiplicacion, agregamos una de estas tuplas al vector de salida para luego devolver este.

3.2.0.3 Reduccion

Recibimos la posicion de un bloque de salida y una lista de multiplicaciones parciales de bloques. Se suman estas multiplicaciones parciales y se devuelve un vector con los valores resultantes del la multiplicacion. Pero por cada valor se calcula la posicion de salida del mismo en la matriz resultante y nos deshacemos de la posicion de los bloques

3.3. multiplicacion de matrices de elemento por fila

3.3.0.1 Preprocesamiento

Consiste en generar una lista de tuplas a partir de las dos matrices. Se itera por cada elemento (`a_ij`) de la matriz A y se guarda en cada tupla el numero de fila del elemento `a_ij`, el elemento `a_ij` y la fila `j` de la matriz B.

3.3.0.2 Mapeo

De esta manera, en la funcion map, obtenemos partes de esta lista de tuplas y devolvemos un par clave, valor donde la clave es la posicion de salida de la matriz resultante `(i, j)` y el valor es la multiplicacion del elemento `a_ij` contra cada elemento de la fila `j` de la matriz B

3.3.0.3 Reduccion

Obtenemos una posicion de salida y una lista de valores que resultaron de la multiplicacion que se hizo en el map. Entonces se suman las multiplicaciones parciales y se obtiene el valor en la posicion de salida de la matriz resultante

3.4. Multiplicacion de matrices de columna por fila

3.4.0.1 Preprocesamiento

Consiste en generar una lista de tuplas a partir de las dos matrices. Se guarda en cada tupla la columna `i` de la matriz A y la fila `i` de la matriz B

3.4.0.2 Mapeo

Recibimos una columna de la matriz A y una fila de la matriz B y por cada elemento de la columna `elem_a` lo multiplicamos por cada elemento de la fila `elem_b` obteniendo una matriz parcial de la multiplicacion. por cada multiplicacion guardamos en un vector una tupla con un par clave valor donde la clave es la posicion de salida de la matriz resultante y el valor es la multiplicacion anteriormente mencionada. Finalmente se devuelve el vector de tuplas.

3.4.0.3 Reduccion

Se recibe la posicion de salida de la matriz resultante y una lista de multiplicaciones parciales. Entonces se suman estas y se devuelve la posicion de salida y la suma.

3.5. Forma de ejecucion

Para el caso de Amdahl multiplicamos dos matrices de `10x10` con `1`, `2`, `4`, `8`, `16` y `32` threads.

Para el caso de gustafson se usan siempre 4 threads multiplicando dos matrices de `2x2`, `4x4`, `8x8`, `16x16`, `32x32` y `64x64`

Para realizar el calculo se debe ejecutar:

```
$ sh scripts/run.sh.
```

Luego para generar los graficos que vemos en el informe se debe ejecutar:

```
$ sh scripts/generate_output_data.sh
```

3.6. Datos sobre la computadora que se utilizó

El equipo sobre el que se realizarán las mediciones es una laptop con un procesador Intel core I7 que posee 4 nucleos a 2.7 Ghz, es decir, soporta hasta 4 threads en paralelo, con 16 Gb de memoria y corriendo sobre un sistema Linux.
Para averiguar estos datos en linux se ejecutaron los siguientes comandos:

- Cantidad de cores: `$ grep -c processor /proc/cpuinfo`

- Velocidad de reloj: `$ lscpu | grep GHz`
- Memoria RAM: `$ free -g`

4. Resultados

4.1. Multiplicacion por bloques

4.1.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	3008.300543	515.333891	100
1	2	1966.158152	531.873941	100
2	3	1695.144415	522.877455	100
3	4	1689.055204	535.524607	100
4	8	1534.405231	616.285801	100
5	16	1768.144131	526.465178	100
6	32	1777.097225	520.468235	100
7	64	1921.860695	585.740805	100
8	128	2399.235725	554.533005	100

Figura 2: Salida de los tiempos en serie y paralelo

De acuerdo a estos datos podemos calcular el speed up maximo, real y teórico.

	number_of_threads	theoretical_speed_up	real_speed_up	max_speed_up
0	1	1.000000	1.000000	6.837576
1	2	1.744819	1.648917	6.837576
2	3	2.321081	2.038762	6.837576
3	4	2.780187	2.322621	6.837576
4	8	3.953048	2.661462	6.837576
5	16	5.009769	3.602358	6.837576
6	32	5.782675	3.988811	6.837576
7	64	6.266037	4.072303	6.837576
8	128	6.539342	5.152428	6.837576

Figura 3: Speed up real, teorico y maximo segun la cantidad de threads

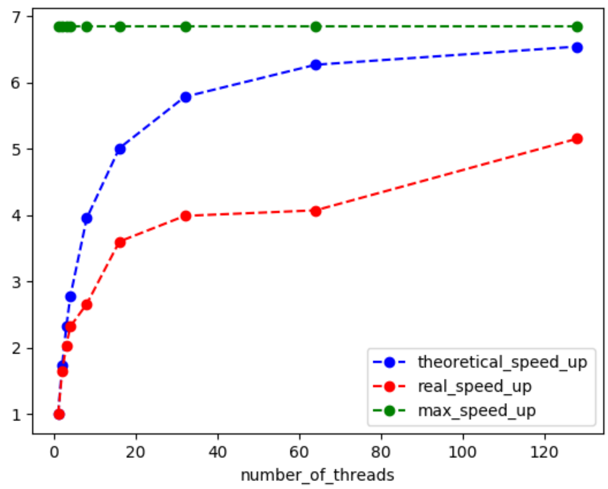

Figura 4: Grafico

Podemos observar que

4.1.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	30.057907	25.198221	2
1	4	54.799795	37.292719	4
2	4	66.013098	38.167953	16
3	4	378.149271	155.233383	64
4	4	1753.703117	528.430462	100
5	4	14348.142147	3509.501457	200
6	4	47013.806581	11095.981121	300

Figura 5: Salida de los tiempos en serie y paralelo con el error

Podemos ver que

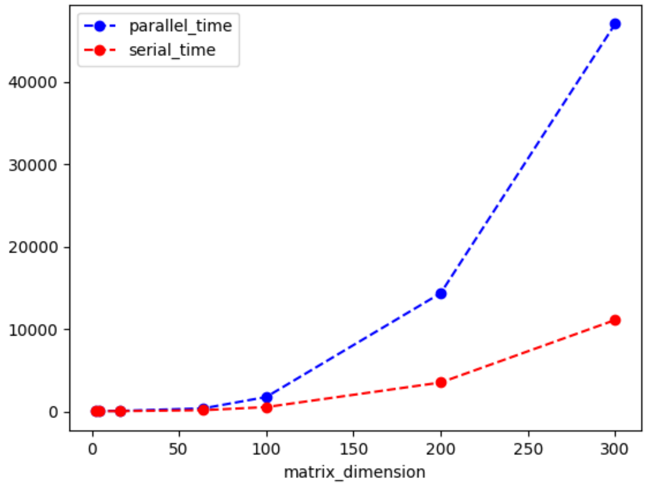

Figura 6: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

Luego a partir de estos datos podemos calcular el speed up y obtuvimos lo siguiente:

	matrix_dimension	speed_up
0	2	2.631923
1	4	2.785155
2	16	2.900915
3	64	3.126893
4	100	3.305347
5	200	3.410420
6	300	3.427154

Figura 7: Tabla de valores del speed up

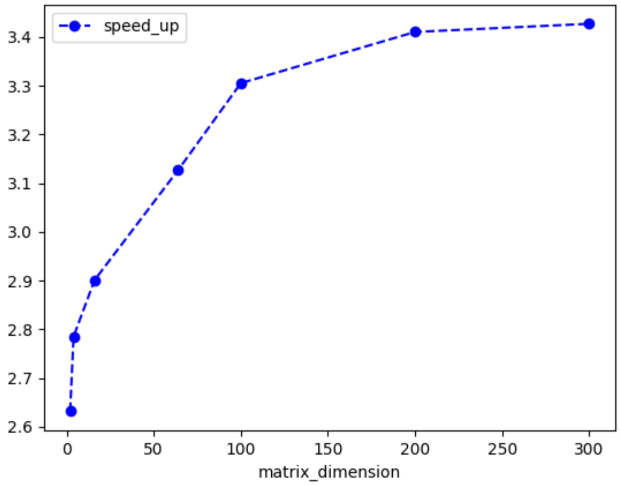

Figura 8: Grafico del speed up

4.2. Multiplicacion elemento por fila

4.2.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	2905.721664	553.509712	100
1	2	1904.639482	552.566290	100
2	3	1710.852623	548.820019	100
3	4	1679.252386	553.561211	100
4	8	1652.954102	574.209690	100
5	16	1629.258156	560.539484	100
6	32	1748.234510	552.772284	100
7	64	2290.686607	554.420948	100
8	128	2867.236614	547.203064	100

Figura 9: Salida de los tiempos en serie y paralelo

De acuerdo a estos datos podemos calcular el speed up maximo, real y teórico.

	number_of_threads	theoretical_speed_up	real_speed_up	max_speed_up
0	1	1.000000	1.000000	6.249631
1	2	1.724124	1.632819	6.249631
2	3	2.272695	2.019180	6.249631
3	4	2.702651	2.293890	6.249631
4	8	3.773467	2.852307	6.249631
5	16	4.705686	3.306013	6.249631
6	32	5.368864	3.788260	6.249631
7	64	5.775863	4.820477	6.249631
8	128	6.003414	5.994416	6.249631

Figura 10: Speed up real, teorico y maximo segun la cantidad de threads

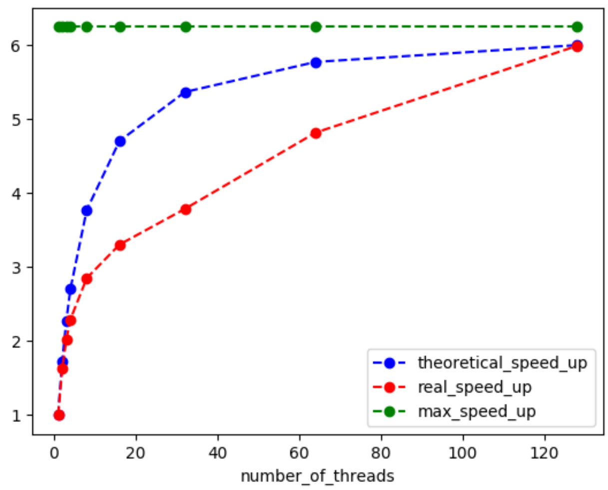

Figura 11: Grafico

Podemos observar que

4.2.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	13.072014	15.079498	2
1	4	36.913872	18.557072	4
2	4	41.501284	22.579193	16
3	4	473.168850	179.124594	64
4	4	1731.611729	547.322750	100
5	4	13785.633802	5024.935484	200
6	4	56481.339693	18792.214394	300

Figura 12: Salida de los tiempos en serie y paralelo con el error

Podemos ver que

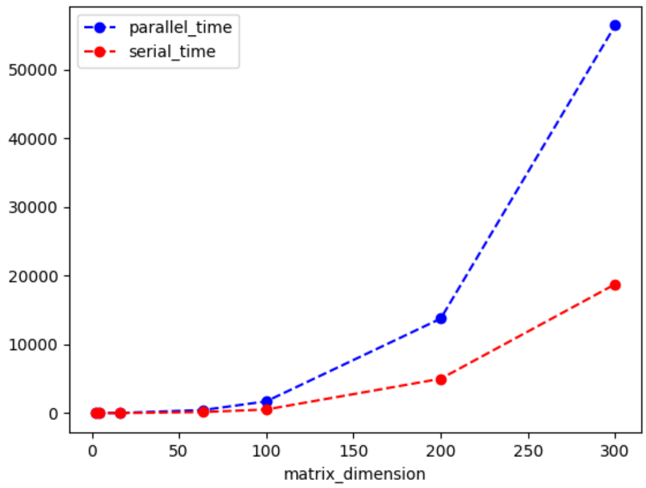

Figura 13: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

Luego a partir de estos datos podemos calcular el speed up y obtuvimos lo siguiente:

	matrix_dimension	speed_up
0	2	2.393035
1	4	2.996390
2	16	2.942930
3	64	3.176178
4	100	3.279502
5	200	3.198599
6	300	3.251043

Figura 14: Tabla de valores del speed up

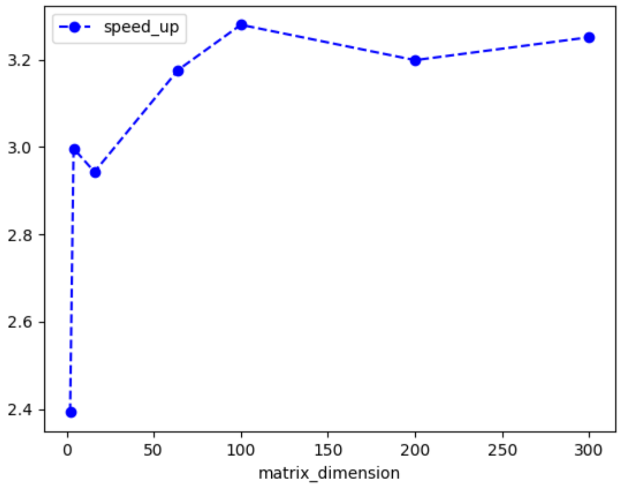

Figura 15: Grafico del speed up

4.3. Multiplicacion columna por fila

4.3.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	2949.785471	704.976082	100
1	2	2175.084352	694.887638	100
2	3	1950.680733	697.954893	100
3	4	1945.697308	740.370750	100
4	8	1945.789099	698.452950	100
5	16	1984.361887	772.732973	100
6	32	2246.499777	758.331776	100
7	64	2395.889282	716.163158	100
8	128	2757.127047	716.336489	100

Figura 16: Salida de los tiempos en serie y paralelo

De acuerdo a estos datos podemos calcular el speed up maximo, real y teórico.

	number_of_threads	theoretical_speed_up	real_speed_up	max_speed_up
0	1	1.000000	1.000000	5.184235
1	2	1.676597	1.610146	5.184235
2	3	2.164838	1.964598	5.184235
3	4	2.533766	2.189500	5.184235
4	8	3.403897	2.808015	5.184235
5	16	4.109532	3.074522	5.184235
6	32	4.584746	3.626681	5.184235
7	64	4.866096	4.129587	5.184235
8	128	5.020130	4.707378	5.184235

Figura 17: Speed up real, teorico y maximo segun la cantidad de threads

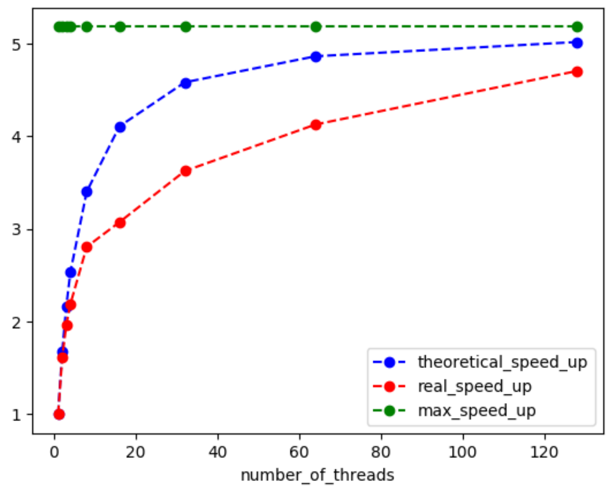

Figura 18: Grafico

Podemos observar que

4.3.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	33.318758	23.381710	2
1	4	54.231882	23.900509	4
2	4	57.114601	31.041861	16
3	4	571.743011	233.061075	64
4	4	1925.363302	767.220259	100
5	4	16716.236830	6059.144735	200
6	4	56208.539009	18043.642282	300

Figura 19: Salida de los tiempos en serie y paralelo con el error

Podemos ver que

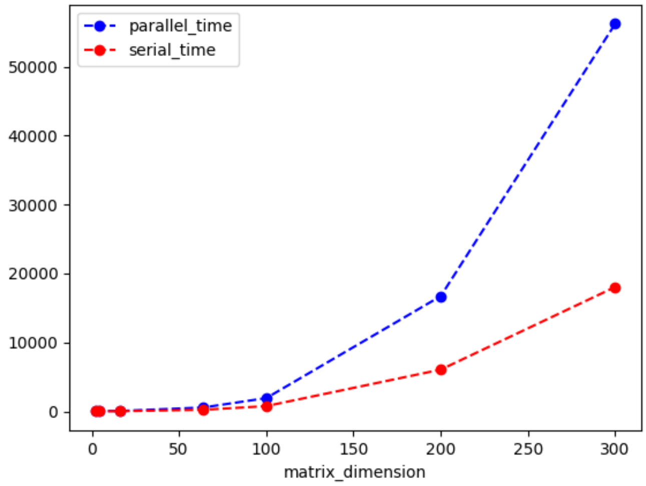

Figura 20: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

Luego a partir de estos datos podemos calcular el speed up y obtuvimos lo siguiente:

	matrix_dimension	speed_up
0	2	2.762883
1	4	3.082307
2	16	2.943633
3	64	3.131238
4	100	3.145185
5	200	3.201882
6	300	3.270985

Figura 21: Tabla de valores del speed up

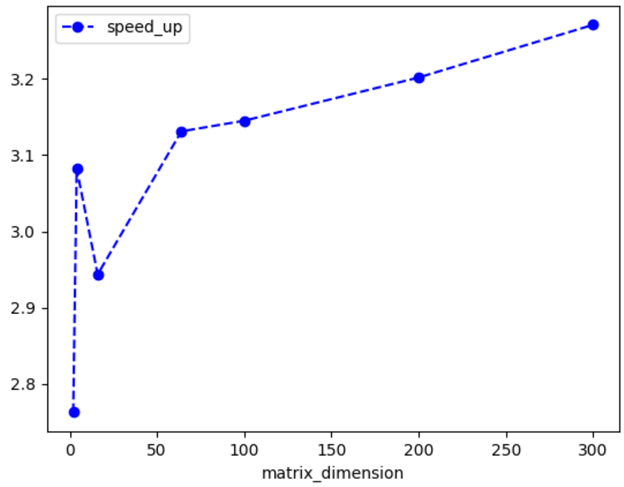

Figura 22: Grafico del speed up

5. Conclusiones

Podemos decir que obtuvimos resultados esperados pero se tuvieron que hacer varias corridas y ajustar ciertos numeros para entender por que llegamos a estos resultados.
Primero para que las curvas de speed up de Amdahl nos den bien habia que tener en cuenta que la dimension de las matrices tenia que ser lo suficientemente grandes como para tener un becnhmark rasonable, pero tambien hay un limite superior para el cual no superamos la capacidad de los procesadores. Luego, una vez mapeados los datos, se aprovecho el uso de los multiprocesadores para reordenar los datos de manera tal que la parte de `reduce` pueda leerlos. Es decir, cuando los ordenamos lo hacemos dividiendo el trabajo en partes donde cada una la realiza cada procesador. De esta manera obtenemos una organizacion tipo arbol donde evitamos un cuello de botella.
Y segundo se coloco un `sleep` de medio segundo (que no afecto el calculo del tiempo paralelo-serie transcurrido) para evitar que cualquier trabajo que no sea puramente vinculado a la CPU afecte nuestro porgrama (como por ejemplo I/O)

Finalmente podemos decir que hay que tener en cuenta que hay otros programas corriendo en las cuatro CPU que tiene la computadora en la cual se probo este programa y que dependiendo del tamaño de informacion que manejamos, podemos tener un cuello de botella ya sea por intercambios de memoria o por exceso de memoria. Entonces se tuvieron que hacer varias corridas analizando el trafico de informacion mediante el comando `gnome-system-monitor` donde filtrando los procesos y solo viedno los de `python` pudimos ver el uso de cada CPU y graficos al respecto.