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

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:

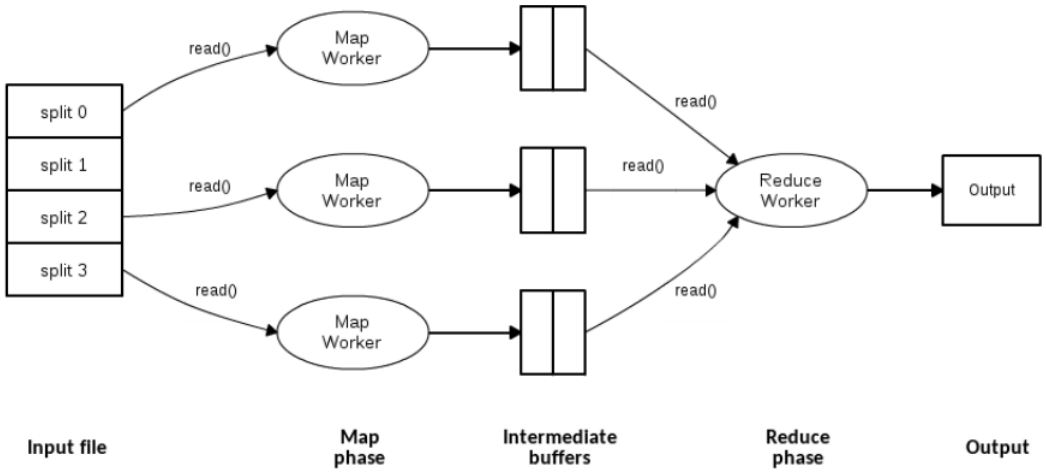

Figura 1: Esquema de un map reduce

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	51077.219009	19300.437450	300
1	2	32189.337730	19619.884968	300
2	3	26976.829290	19688.769341	300
3	4	26194.507837	20019.755840	300
4	8	26637.074471	23729.829788	300
5	16	30390.383005	19324.795723	300
6	32	25784.678221	23436.945438	300
7	64	26647.127628	23362.513304	300
8	128	28719.156027	19576.766253	300

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	3.646428
1	2	1.569562	1.450647	3.646428
2	3	1.937381	1.627054	3.646428
3	4	2.194519	1.739446	3.646428
4	8	2.740020	1.861342	3.646428
5	16	3.128902	2.342382	3.646428
6	32	3.367900	2.030367	3.646428
7	64	3.501634	2.103112	3.646428
8	128	3.572564	2.439048	3.646428

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

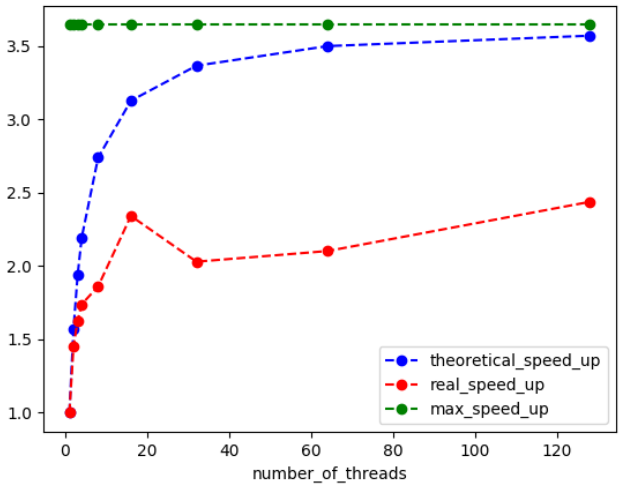

Figura 4: Grafico

Podemos observar que

4.1.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	49.705505	16.244411	2
1	4	54.050922	16.224146	4
2	4	54.918051	34.598827	8
3	4	79.056740	18.027782	16
4	4	75.916052	27.108431	32
5	4	236.617804	99.617720	64
6	4	751.295090	307.744265	100
7	4	7378.479958	6003.620625	200
8	4	27626.085281	24217.552423	300
9	4	67275.854349	46028.301239	400

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

Podemos ver que

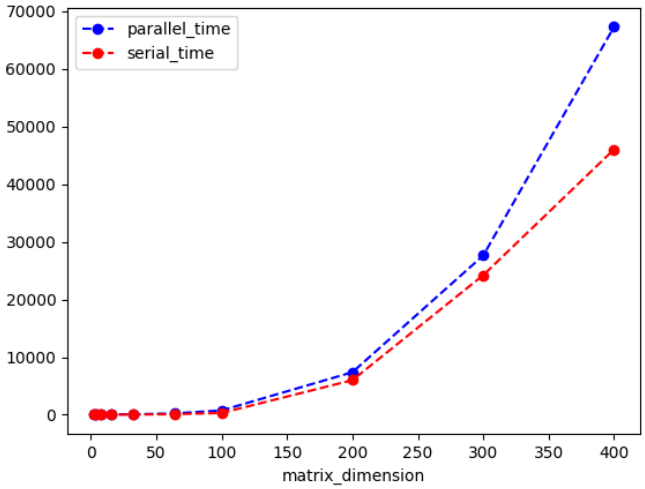

Figura 6: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	3.261057
1	4	3.307401
2	8	2.840481
3	16	3.442925
4	32	3.210622
5	64	3.111179
6	100	3.128236
7	200	2.654108
8	300	2.598620
9	400	2.781290

Figura 7: Tabla de valores del speed up

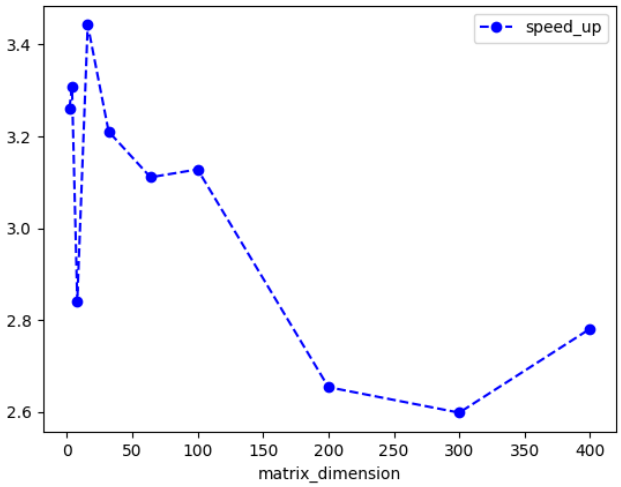

Figura 8: Grafico del speed up

4.2. Multiplicacion elemento por fila

4.2.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	47465.412378	5170.542240	300
1	2	29189.192533	5347.205639	300
2	3	24420.081377	4608.105659	300
3	4	24172.757149	5660.039663	300
4	8	22262.604475	5324.264050	300
5	16	23289.145708	4607.063293	300
6	32	23092.943907	5450.780869	300
7	64	16826.578140	5351.116180	300
8	128	34585.215330	5753.373861	300

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	10.179968
1	2	1.821109	1.731859	10.179968
2	3	2.507388	2.277054	10.179968
3	4	3.089527	2.549108	10.179968
4	8	4.740390	3.402808	10.179968
5	16	6.468614	4.601334	10.179968
6	32	7.910618	4.624386	10.179968
7	64	8.902955	3.950404	10.179968
8	128	9.498733	6.696790	10.179968

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

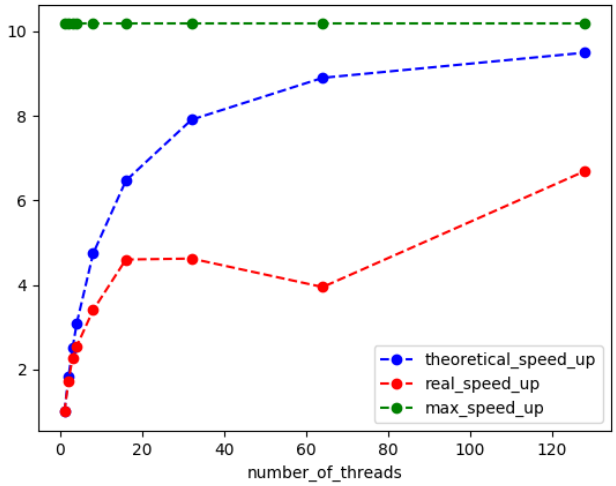

Figura 11: Grafico

Podemos observar que

4.2.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	40.018797	11.023521	2
1	4	31.272411	22.382975	4
2	4	32.216072	22.589684	8
3	4	59.314966	9.898901	16
4	4	53.644419	22.888184	32
5	4	214.006901	46.062469	64
6	4	744.369268	206.614494	100
7	4	6665.740728	1639.871597	200
8	4	23741.173267	5314.360380	300
9	4	159828.639984	13177.682877	400

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

Podemos ver que

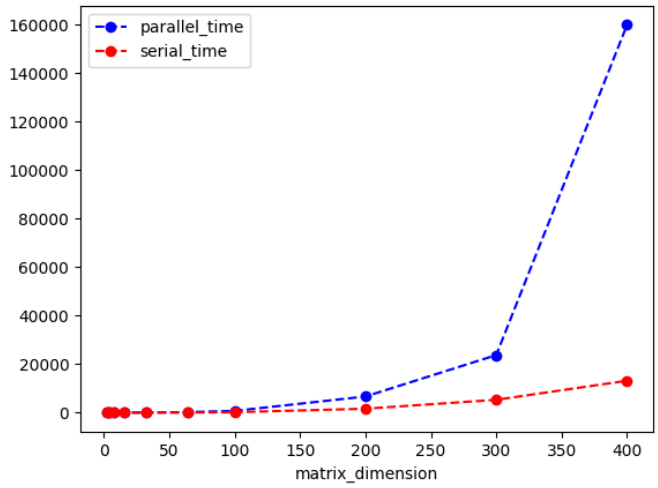

Figura 13: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	3.352095
1	4	2.748515
2	8	2.763468
3	16	3.570943
4	32	3.102807
5	64	3.468652
6	100	3.348208
7	200	3.407676
8	300	3.451289
9	400	3.771494

Figura 14: Tabla de valores del speed up

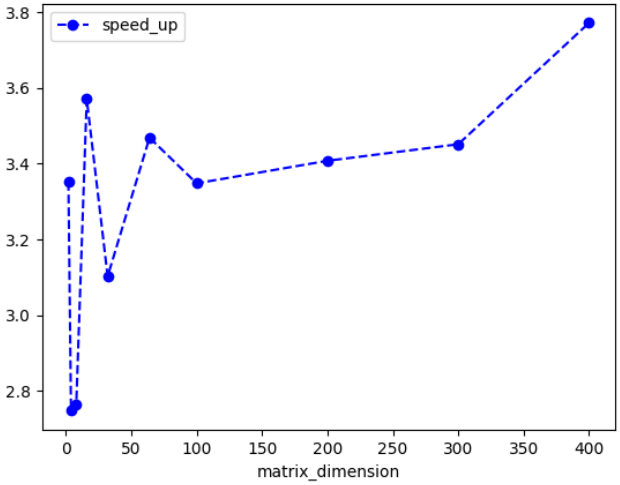

Figura 15: Grafico del speed up

4.3. Multiplicacion columna por fila

4.3.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	61218.903065	6065.199375	300
1	2	31848.276377	6086.799860	300
2	3	25023.083210	5590.039492	300
3	4	23893.268824	6024.660826	300
4	8	20918.287754	6246.512413	300
5	16	20267.532110	5517.182589	300
6	32	20621.957779	6125.909805	300
7	64	19379.518747	5536.347389	300
8	128	28277.473450	6766.212225	300

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	11.093469
1	2	1.834621	1.723465	11.093469
2	3	2.541756	2.197471	11.093469
3	4	3.148542	2.493581	11.093469
4	8	4.904961	3.065555	11.093469
5	16	6.802296	3.800867	11.093469
6	32	8.433399	3.950739	11.093469
7	64	9.582248	4.267035	11.093469
8	128	10.282630	5.015462	11.093469

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

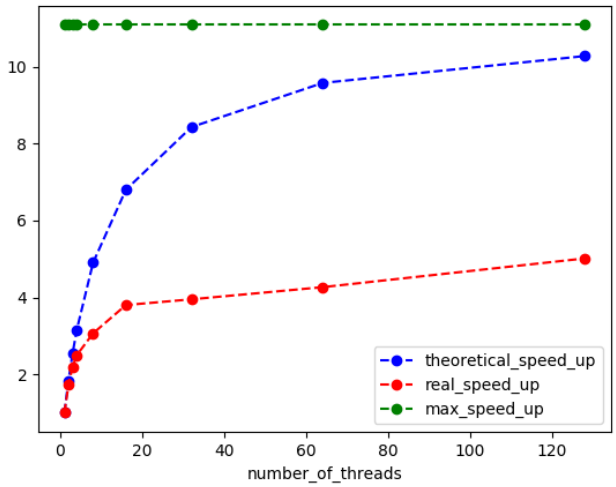

Figura 18: Grafico

Podemos observar que

4.3.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	38.901567	12.046814	2
1	4	62.201977	13.424397	4
2	4	65.268993	12.586832	8
3	4	66.662788	12.822390	16
4	4	61.312199	27.036190	32
5	4	221.490145	50.839663	64
6	4	724.222422	154.191971	100
7	4	6069.197178	1289.043188	200
8	4	24376.475811	5545.433998	300
9	4	220071.399450	17746.215582	400

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

Podemos ver que

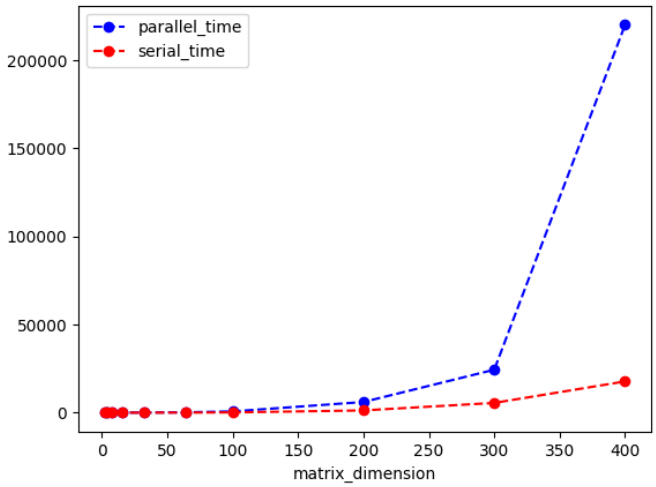

Figura 20: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	3.290646
1	4	3.467472
2	8	3.514995
3	16	3.516046
4	32	3.081946
5	64	3.439948
6	100	3.473397
7	200	3.474449
8	300	3.444009
9	400	3.776137

Figura 21: Tabla de valores del speed up

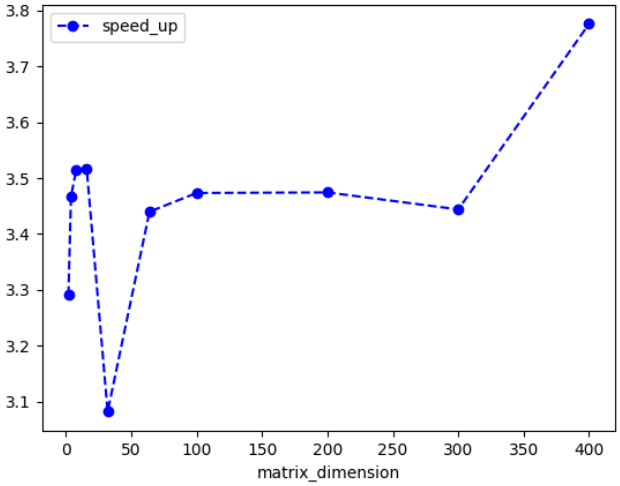

Figura 22: Grafico del speed up

5. Conclusiones

Podemos decir que nuestros resultados no son lo que deberia ocurrir en un ambiente ideal donde solo nuestro porceso corre. En la parte de amdahl, el speedup sigue aumentando aún cuando la cantidad de threads de procesamiento supera la cantidad de threads del sistema (4). Esto no es lo que se esperaba. Ocurre que al aumentar la cantidad de threads, la sección serie del problema no se mantuvo sino que tuvo una pequeña variacion equilibrando la caída de performance de la sección paralela. Esta variacion en el tiempo de la parte serie del problema se debe a la forma en que quedan organizados los datos una vez que son procesados por los map_workers .