

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

<b>1. Objetivo</b>	<b>1</b>
<b>2. Desarrollo teorico</b>	<b>2</b>
2.1. Speed up	2
2.2. Ley de Amdahl	2
2.3. Ley de Gustafson	3
2.4. Map-reduce	3
<b>3. Implementacion</b>	<b>4</b>
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
<b>4. Resultados</b>	<b>8</b>
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	14
4.3.0.1. Salida Amdahl	14
4.3.0.2. Salida Gustafson	15
<b>5. Conclusiones</b>	<b>18</b>

# 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  $N \times N$ ) 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:

$$speed\_up = \frac{Rendimiento\_con\_mejora}{Rendimiento\_sin\_mejora} \quad (1)$$

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

$$speed\_up = \frac{T\_ejecucion\_sin\_mejora}{T\_ejecucion\_con\_mejora} \quad (2)$$

### 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:

$$speed\_up = \frac{s + p}{s + \frac{p}{N}} \quad (3)$$

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:

$$speed\_up\_max = 1 + \frac{p}{s} \quad (4)$$

### 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 ejecucion 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:

$$speed\_up = \frac{s + p * N}{s + p} \quad (5)$$

### 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 posición  $r$ ,  $c$  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 índice en la lista de bloques  $b$  y como valor guardamos la multiplicación. Por cada multiplicación, agregamos una de estas tuplas al vector de salida para luego devolver este.

### 3.2.0.3 Reduccion

Recibimos la posición 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 de la multiplicación. Pero por cada valor se calcula la posición de salida del mismo en la matriz resultante y nos deshacemos de la posición 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 número 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 función map, obtenemos partes de esta lista de tuplas y devolvemos un par clave, valor donde la clave es la posición de salida de la matriz resultante  $(i, j)$  y el valor es la multiplicación del elemento  $a_{ij}$  contra cada elemento de la fila  $j$  de la matriz B

### 3.3.0.3 Reduccion

Obtenemos una posición de salida y una lista de valores que resultaron de la multiplicación que se hizo en el map. Entonces se suman las multiplicaciones parciales y se obtiene el valor en la posición 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	2834.711552	657.626390	100
1	2	1791.648865	593.303919	100
2	3	1691.109896	588.583708	100
3	4	1642.032862	592.027426	100
4	8	1597.375393	582.183361	100
5	16	1598.940134	591.347456	100
6	32	2126.093864	665.178299	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	5.31052
1	2	1.683069	1.601576	5.31052
2	3	2.179265	1.978408	5.31052
3	4	2.556047	2.228410	5.31052
4	8	3.451045	2.787675	5.31052
5	16	4.183463	3.168447	5.31052
6	32	4.680094	3.815201	5.31052

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

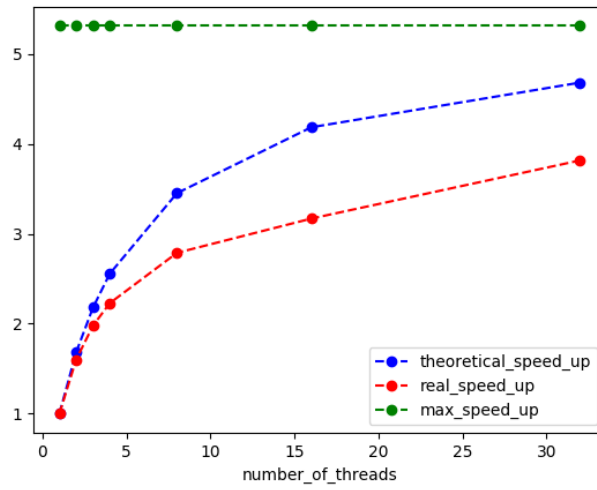


Figura 4: Grafico

Podemos observar que

#### 4.1.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	52.127361	16.464710	4
1	4	65.432072	17.514229	16
2	4	393.583298	160.211802	64
3	4	13709.108591	3810.233116	200
4	4	43599.804401	13375.952244	300

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

Podemos ver que

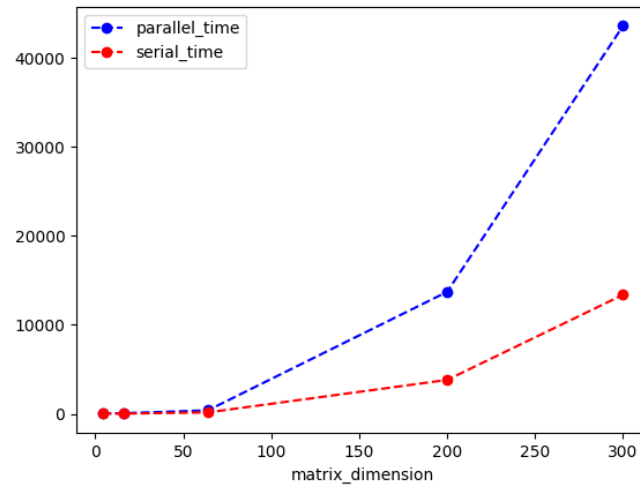


Figura 6: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	<b>matrix_dimension</b>	<b>speed_up</b>
<b>0</b>	4	3.279886
<b>1</b>	16	3.366546
<b>2</b>	64	3.132106
<b>3</b>	200	3.347538
<b>4</b>	300	3.295703

Figura 7: Tabla de valores del speed up

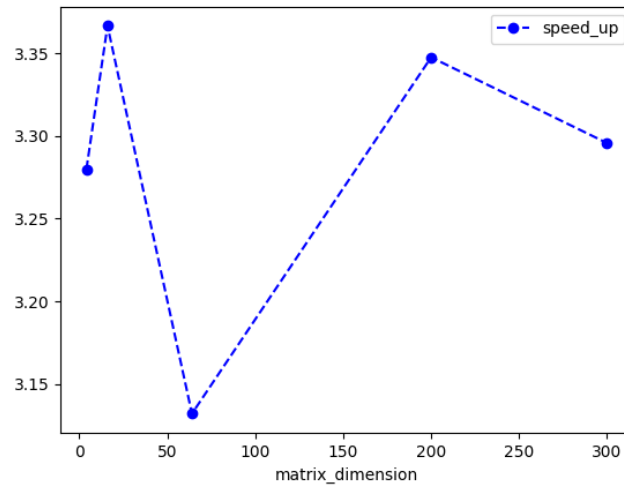


Figura 8: Grafico del speed up

## 4.2. Multiplicacion elemento por fila

### 4.2.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
<b>0</b>	1	2769.915104	545.826912	100
<b>1</b>	2	1876.152754	539.743662	100
<b>2</b>	3	1663.998604	555.776596	100
<b>3</b>	4	1650.401354	546.600342	100
<b>4</b>	8	1571.121216	536.647558	100
<b>5</b>	16	1542.981386	540.376663	100
<b>6</b>	32	1705.415249	539.793491	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	6.074713
1	2	1.717303	1.634770	6.074713
2	3	2.256940	1.999000	6.074713
3	4	2.677644	2.290451	6.074713
4	8	3.716923	2.875389	6.074713
5	16	4.611945	3.271538	6.074713
6	32	5.243219	3.785627	6.074713

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

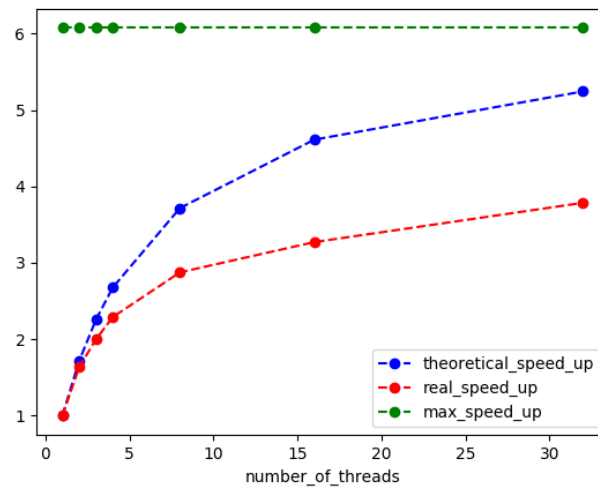


Figura 11: Grafico

Podemos observar que

#### 4.2.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	12.849808	7.454157	4
1	4	41.205883	9.935856	16
2	4	441.965580	192.625999	64
3	4	13986.226320	4880.016804	200
4	4	52713.987589	16649.103165	300

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

Podemos ver que

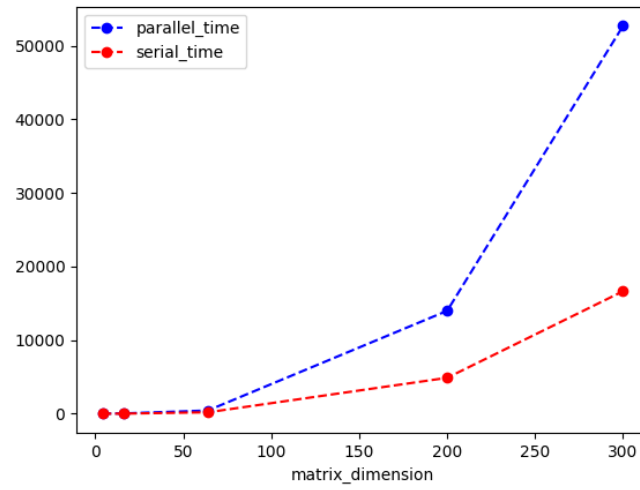


Figura 13: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	<b>matrix_dimension</b>	<b>speed_up</b>
<b>0</b>	4	2.898616
<b>1</b>	16	3.417158
<b>2</b>	64	3.089370
<b>3</b>	200	3.224008
<b>4</b>	300	3.279915

Figura 14: Tabla de valores del speed up

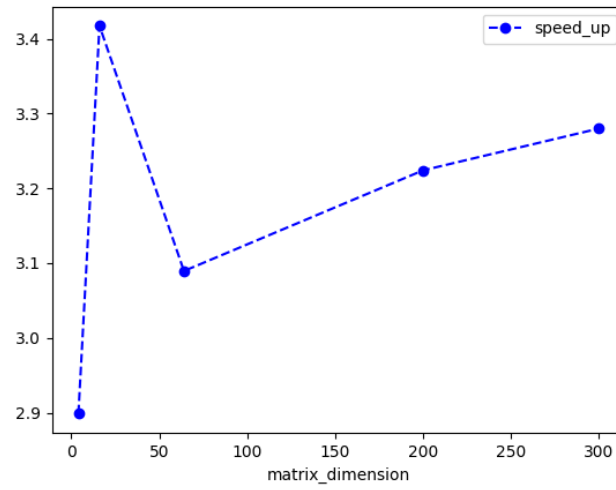


Figura 15: Grafico del speed up

### 4.3. Multiplicacion columna por fila

#### 4.3.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	2439.744473	559.961557	100
1	2	1729.050398	560.826540	100
2	3	1490.924597	550.949097	100
3	4	1528.548241	601.736784	100
4	8	1391.374111	613.751411	100
5	16	1676.132202	566.991091	100
6	32	1895.309448	562.489033	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	5.356986
1	2	1.685385	1.606535	5.356986
2	3	2.184449	1.948494	5.356986
3	4	2.564076	2.165201	5.356986
4	8	3.468150	2.545631	5.356986
5	16	4.210435	3.339227	5.356986
6	32	4.715010	3.953240	5.356986

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

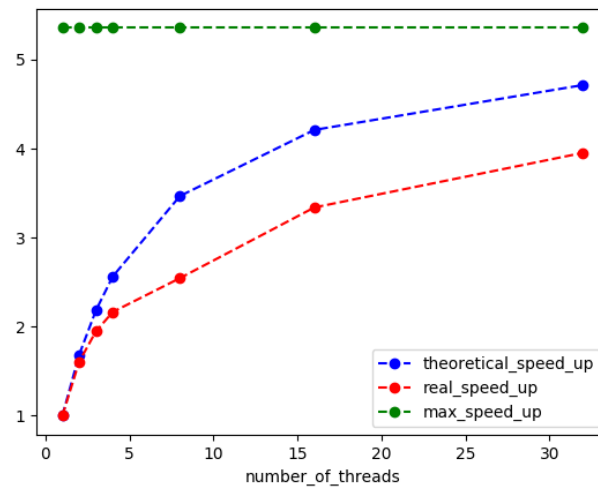


Figura 18: Grafico

Podemos observar que

#### 4.3.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	38.460016	13.428450	4
1	4	49.259663	31.679392	16
2	4	464.546919	194.267511	64
3	4	13617.492437	4878.481627	200
4	4	52970.852375	17476.922035	300

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

Podemos ver que

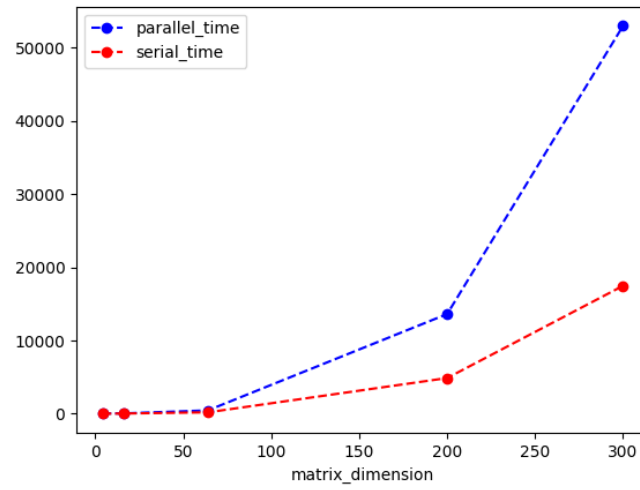


Figura 20: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	<b>matrix_dimension</b>	<b>speed_up</b>
<b>0</b>	4	3.223616
<b>1</b>	16	2.825806
<b>2</b>	64	3.115377
<b>3</b>	200	3.208723
<b>4</b>	300	3.255750

Figura 21: Tabla de valores del speed up

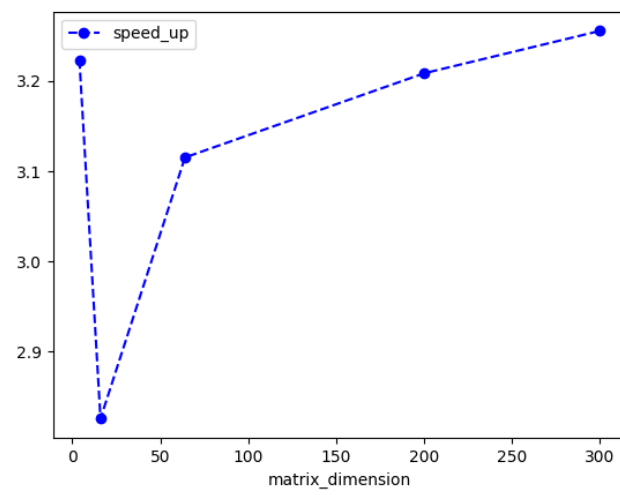


Figura 22: Grafico del speed up

## 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 programa (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 probó 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 viendo los de `python` pudimos ver el uso de cada CPU y graficos al respecto.