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:

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

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:

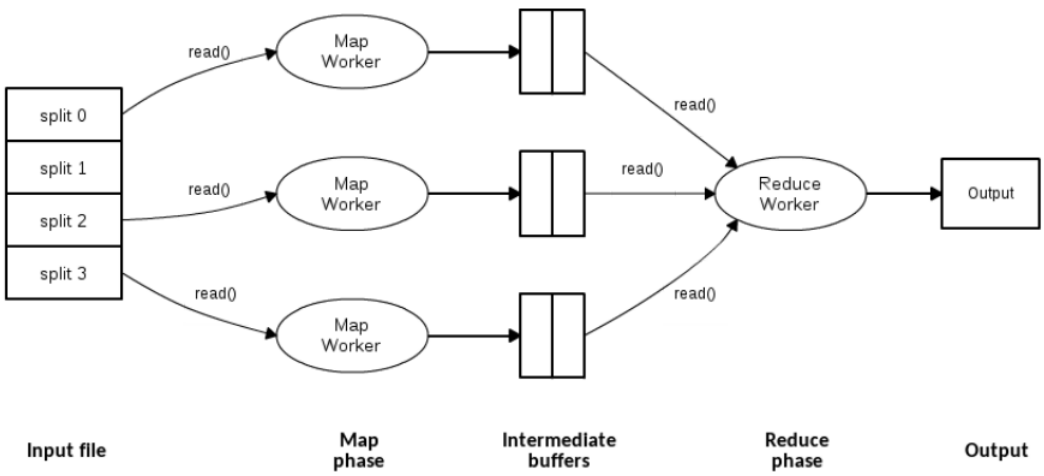

Figura 1: Esquema de un map reduce

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	14.011621	5.789518	10
1	2	8.067846	5.390406	10
2	4	10.849476	5.151749	10
3	8	19.106388	5.498171	10
4	16	55.459976	5.590916	10
5	32	72.217703	5.053759	10

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	3.420170
1	2	1.547529	1.428033	2.496705
2	4	2.130891	2.034713	3.105979
3	8	2.625808	3.119845	4.475044
4	16	2.970805	6.740619	10.919659
5	32	3.179690	10.569839	15.289900

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

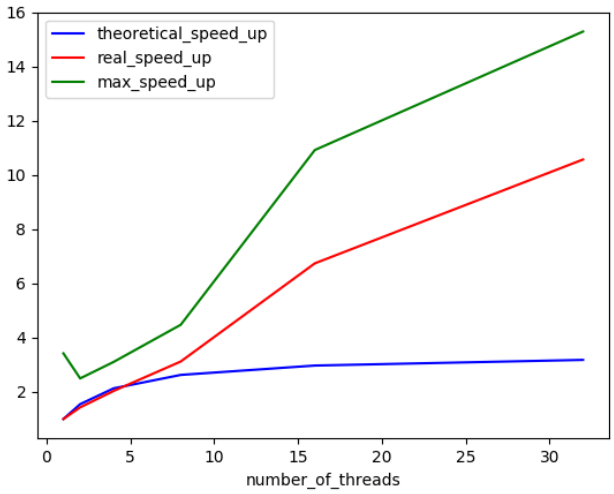

Figura 4: Grafico

Podemos observar que

4.1.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	8.489847	3.280163	2
1	4	9.699106	3.982067	4
2	4	11.635780	6.150961	8
3	4	15.550375	11.936665	16
4	4	73.670864	71.944714	32
5	4	267.300606	573.189497	64

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

Podemos ver que

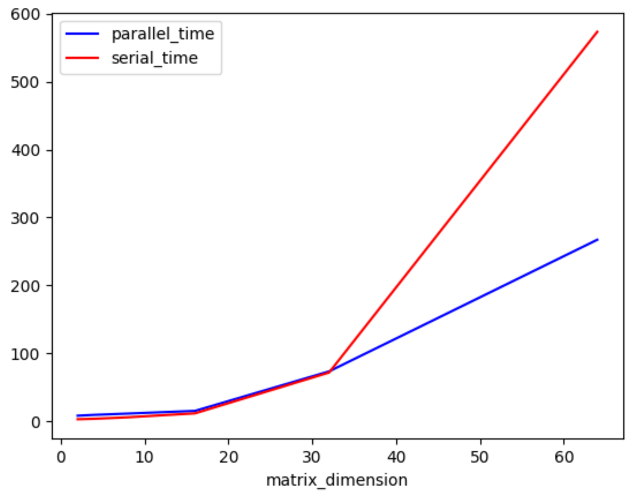

Figura 6: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	1.721312
1	4	3.126815
2	8	5.579280
3	16	9.486022
4	32	16.683739
5	64	21.035855

Figura 7: Tabla de valores del speed up

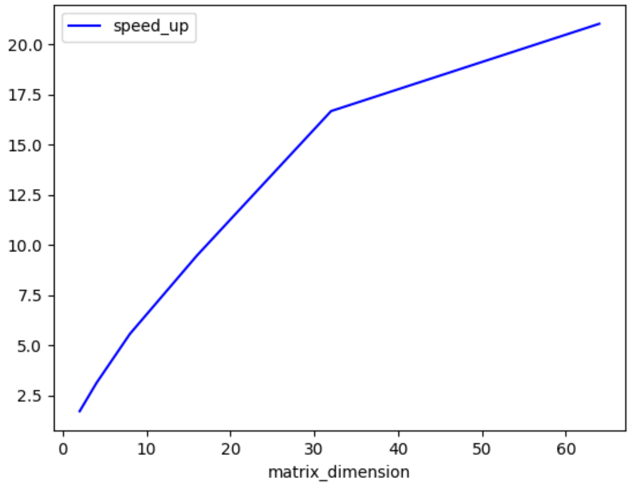

Figura 8: Grafico del speed up

4.2. Multiplicacion elemento por fila

4.2.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	9.133101	2.901793	10
1	2	9.196043	2.999783	10
2	4	11.723757	3.727198	10
3	8	18.582821	4.141808	10
4	16	33.499241	4.081249	10
5	32	78.140259	3.603935	10

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	4.147400
1	2	1.611454	1.605178	4.065570
2	4	2.321068	2.320612	4.145462
3	8	2.976407	3.515209	5.486645
4	16	3.465661	6.085957	9.208085
5	32	3.776006	13.520782	22.681926

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

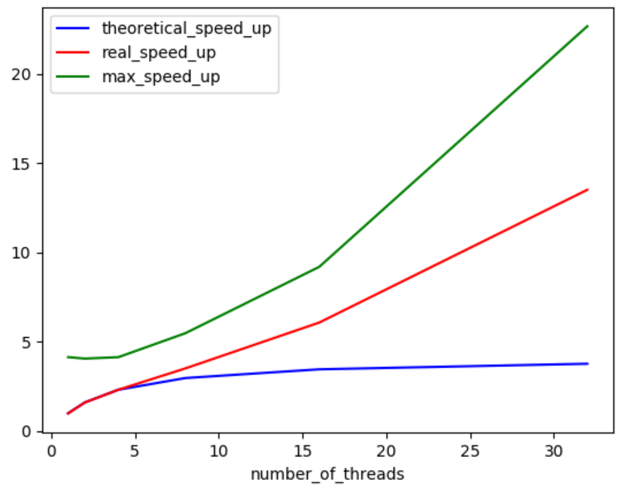

Figura 11: Grafico

Podemos observar que

4.2.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	11.436224	4.540682	2
1	4	9.222269	4.878044	4
2	4	12.021780	4.516602	8
3	4	23.777008	14.425755	16
4	4	90.613127	8.731127	32
5	4	376.791000	46.737671	64

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

Podemos ver que

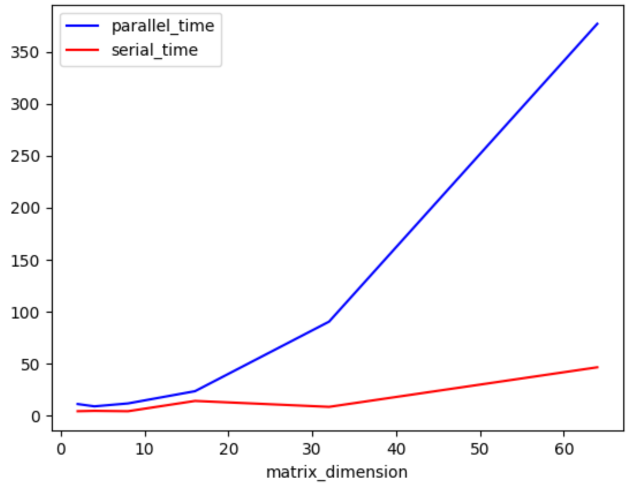

Figura 13: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	1.715797
1	4	2.962141
2	8	6.088313
3	16	10.335846
4	32	29.275485
5	64	57.047759

Figura 14: Tabla de valores del speed up

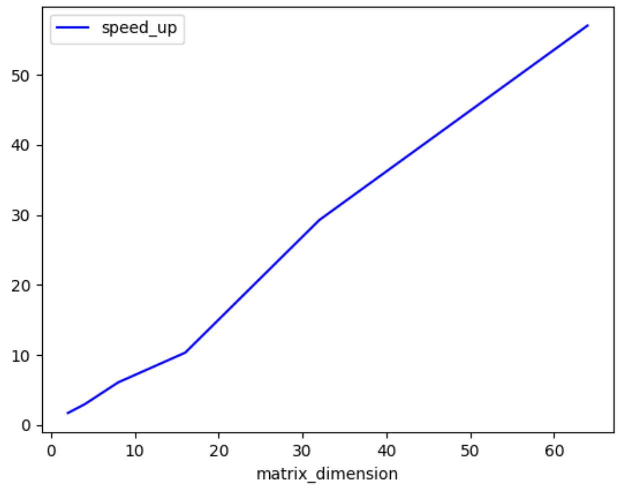

Figura 15: Grafico del speed up

4.3. Multiplicacion columna por fila

4.3.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	6.643772	3.440142	10
1	2	6.272554	4.094362	10
2	4	9.638786	4.036665	10
3	8	17.050743	5.125999	10
4	16	46.703815	4.005432	10
5	32	72.419882	4.395485	10

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	2.931250
1	2	1.491256	1.433748	2.531998
2	4	1.976818	2.121422	3.387809
3	8	2.361233	3.055766	4.326326
4	16	2.615545	7.323248	12.660119
5	32	2.764413	11.536253	17.475971

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

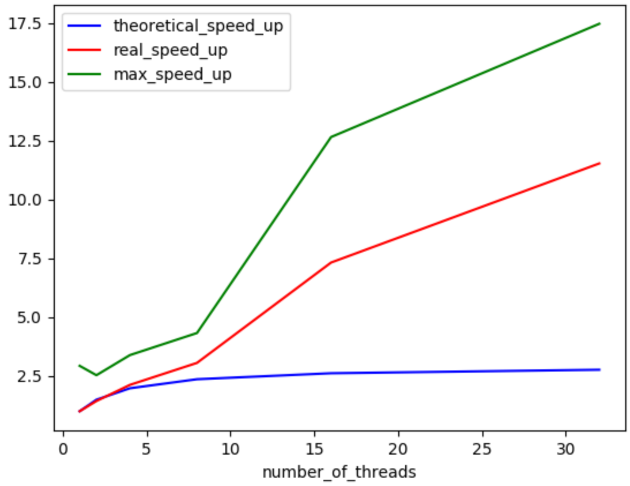

Figura 18: Grafico

Podemos observar que

4.3.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	7.745981	3.554344	2
1	4	8.606195	3.312349	4
2	4	10.372639	4.409552	8
3	4	12.062311	4.846096	16
4	4	26.208401	9.010553	32
5	4	154.640436	41.335821	64

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

Podemos ver que

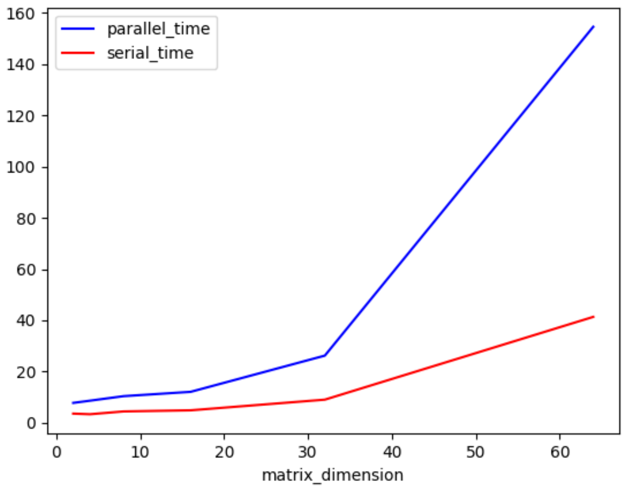

Figura 20: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	1.685465
1	4	3.166253
2	8	5.911889
3	16	11.700870
4	32	24.068840
5	64	50.711876

Figura 21: Tabla de valores del speed up

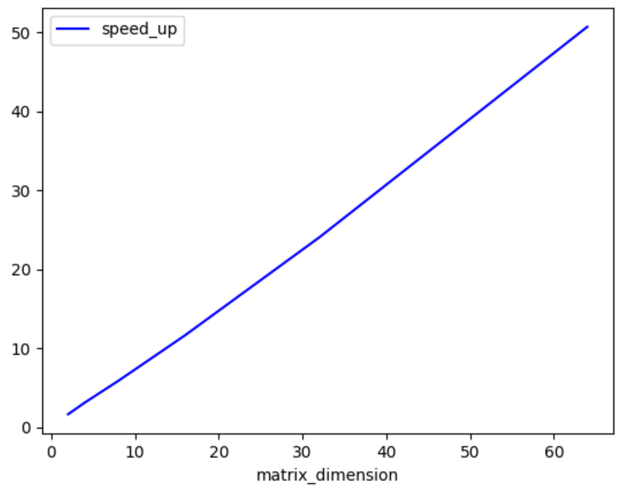

Figura 22: Grafico del speed up

5. Conclusiones

Podemos decir que en ambos casos (Amdahl y Gustafson) obtuvimos rsultados esperados. Es cierto que no fueron resultados perfectos ya que hubo puntos criticos donde el resultado no era del todo esperado como por ejemplo cuando en el caso de la ley de Gustafson, al multiplicar dos matrices de 64x64, la seccion serie aumento significativamente su tiempo de ejecucion. Esto se debe a que con solo conocer los datos de memoria, velocidad de reloj y cantidad de cores no es suficiente ya que la pc cuenta con un recolector de basura que optimiza el reordenamiento de datos al momento del reduce y mejora su tiempo de ejecucion.
Respecto a la ley de Amdahl obtuvimos resultados buenos donde reflejamos que mejora el procesamiento en paralelo cuanto mas threads tenemos para una misma cantidad de trabajo. Finalmente este trabajo muestra el poder de escalabilidad que tiene el map-reduce porque permite dividir el trabajo de manera eficiente y ademas se emplea generalmente en aquellos problemas de Computación concurrente entre los que se encuentran involucrados grandes datasets que deben ser procesandos por una gran cantidad de computadoras (nodos), a los que se refiere de forma colectiva como clusteres (si todos los nodos se encuentran en la misma red de área local y empleando el mismo hardware), o a grids (si los nodos se comparten de forma distribuida a lo largo de extensas zonas geográficas o administrativas, y que generalmente poseen un hardware más heterogéneo).
El procesamiento paralelo puede ocurrir con el empleo de datos almacenados tanto en filesystem (no estructurado) o en una database (estructurados). Es por esta razón por la que se emplea en aplicaciones que poseen datos a gran escala, tales como aplicaciones paralelas, indexación web, data mining, y simulación científica.