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.
Finalmente se hara una multiplicacion de dos matrices diferentes de `NxN` usando la libreria CBLAS y usando una intruccion de vectorizacion (MMX) para el compilador usando solo un procesador. De esta manera la idea es comparar el tiempo que tarda el map-reduce en serie frente a cblas y la vectorizacion.

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:

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

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

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

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:

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

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:

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

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:

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

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)`.

En el medio ocurre la fase de agrupamiento la cual consiste de agrupar los valores con misma clave en un vector para entregarle a la fase de reduce un conjunto de tuplas `(clave, valores)` donde en este caso el valor son todos los valores en una lista.

`Reduce` recibe un conjunto de tuplas `(clave, valores)` donde el valor es una lista de todos los valores que tenian la misma clave. Entonces reduce aplica una funcion a todos estos valores para retornar un unico valor y asi devolver un conjunto de tuplas `(clave, valor)`

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.

2.5. LAPACK

LAPACK está escrito en Fortran 90 y proporciona rutinas para resolver sistemas de ecuaciones lineales simultáneas, soluciones de mínimos cuadrados de sistemas de ecuaciones lineales, problemas de valores propios y problemas de valores singulares. También se proporcionan las factorizaciones matriciales asociadas (LU, Cholesky, QR, SVD, Schur, Schur generalizado), al igual que los cálculos relacionados, tales como la reordenación de las factorizaciones de Schur y la estimación de los números de condición. Se manejan matrices densas y con bandas, pero no matrices dispersas generales. En todas las áreas, se proporciona una funcionalidad similar para matrices reales y complejas, con precisión simple y doble.

2.6. ScaLAPACK

Es una liberia de rutinas de álgebra lineal de alto rendimiento para máquinas de memoria distribuida en paralelo. ScaLAPACK resuelve sistemas lineales densos y en bandas, problemas de mínimos cuadrados, problemas de valores propios y problemas de valores singulares. Las ideas clave incorporadas en ScaLAPACK incluyen el uso de:

- Una distribución de datos de bloques cíclicos para matrices densas y una distribución de datos de bloques para matrices en bandas, parametrizable en tiempo de ejecución.
- Algoritmos de partición de bloque para asegurar altos niveles de reutilización de datos.
- Componentes modulares de bajo nivel bien diseñados que simplifican la tarea de paralelizar las rutinas de alto nivel haciendo que su código fuente sea el mismo que en el caso secuencial.

2.7. CBLAS

BLAS (Subprogramas de Álgebra Lineal Básica) son rutinas que proporcionan bloques de construcción estándar para realizar operaciones básicas de vectores y matrices. Las BLAS de nivel 1 realizan operaciones escalares, vectoriales y vectoriales, las BLAS de nivel 2 realizan operaciones de vectores matriciales y las BLAS de nivel 3 realizan operaciones de matriz-matriz. Debido a que los BLAS son eficientes, portátiles y ampliamente disponibles, se usan comúnmente en el desarrollo de software de álgebra lineal de alta calidad, LAPACK, por ejemplo.
CBLAS es una interfaz de lenguaje C para BLAS.

3. Implementacion

3.1. Explicacion del modelo

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

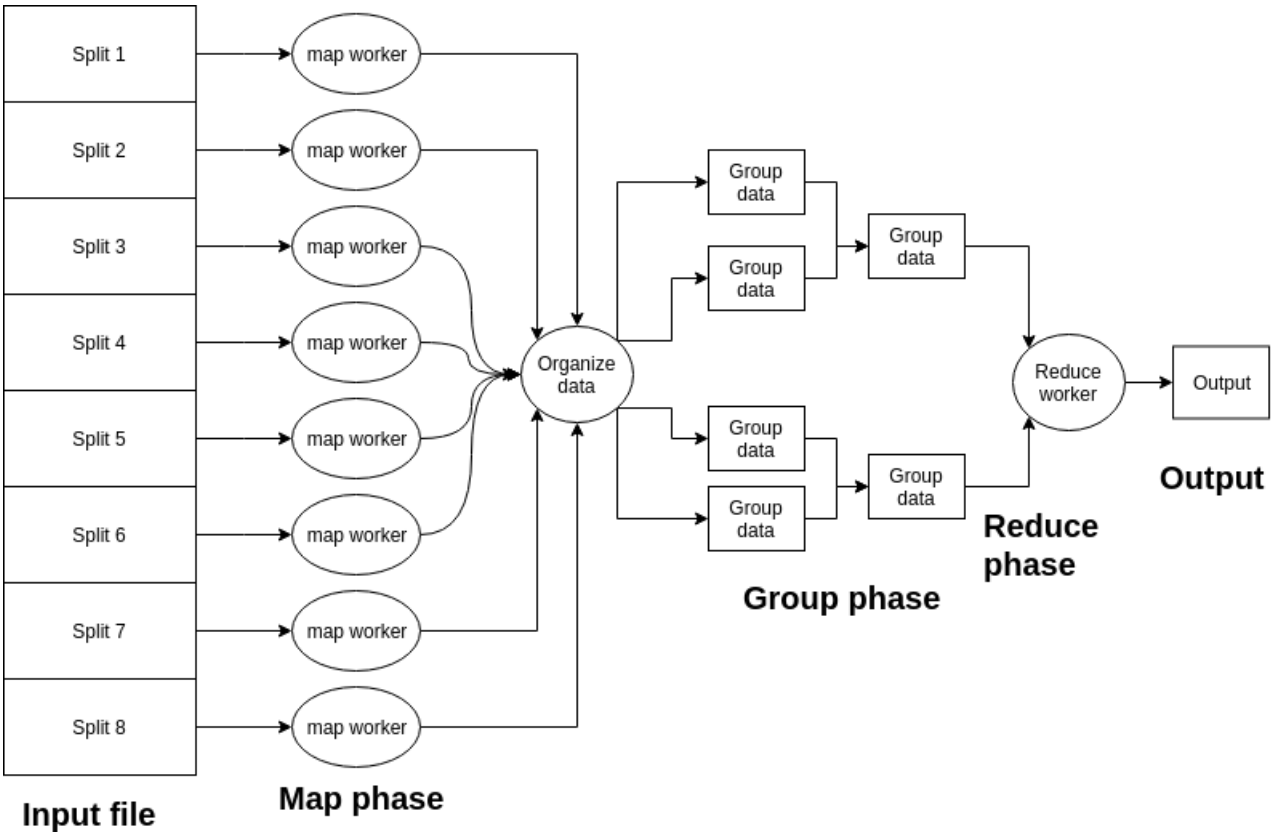

Figura 1: Esquema de un map reduce

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 sea la serie.

3.2. Multiplicacion de matrices por bloques

3.2.0.1 Preprocesamiento

Sean dos matrices `A` de `NxN` y `B` de `NxN` las dividimos en `(N/2)x(N/2)` bloques cada una. Luego 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

Sean dos matrices `A` de `NxN` y `B` de `NxN` generamos 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 `i` del elemento `a_ij`, el elemento `a_ij` y la fila `j` de la matriz `B`. Quedando cada tupla de la siguiente manera:
`(i, a_ij, B[j])`

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

Sean dos matrices `A` de `NxN` y `B` de `NxN` generamos 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`. Quedando cada tupla de la siguiente manera:
`(A[:][i], B[i])`

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 `100x100` y cada una de estas multiplicaciones la realizamos para `1`, `2`, `3`, `4`, `8`, `16`, `32`, `64` y `128` threads.

Para el caso de gustafson se usan siempre 4 threads multiplicando dos matrices de `2x2`, `4x4`, `16x16`, `64x64`, `100x100`, `200x200`, y `300x300`.

Luego para el caso de `cblas` y de instrucciones vectoriales (MMX) se usa un thread multiplicando dos matrices de `400x400`

Para compilar `cblas` y las instrucciones vectoriales que estan en lenguaje c se debe ejecutar:
`$ make.`

Para realizar el calculo de `cblas` y las instrucciones vectoriales que estan en lenguaje c se debe ejecutar:
`$ ./app.`

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`

Y finalmente para generar el informe debemos ejecutar:
`$ sh scripts/make_report.sh`

Tambien hay un script que corre estos tres comandos en un solo script:
`$ sh scripts/run_all.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	3178.862333	537.276983	100
1	2	2735.191107	537.839651	100
2	3	2217.736483	543.824673	100
3	4	1868.114948	537.915468	100
4	8	1906.499386	584.030390	100
5	16	1838.134766	536.170721	100
6	32	1919.524193	579.051971	100
7	64	1897.083282	517.779827	100
8	128	2035.792112	521.183252	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	6.916617
1	2	1.747367	1.717734	6.916617
2	3	2.327099	2.152307	6.916617
3	4	2.789910	2.394193	6.916617
4	8	3.976034	3.028579	6.916617
5	16	5.049405	3.646863	6.916617
6	32	5.837329	3.909908	6.916617
7	64	6.331306	4.411339	6.916617
8	128	6.611032	4.760814	6.916617

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

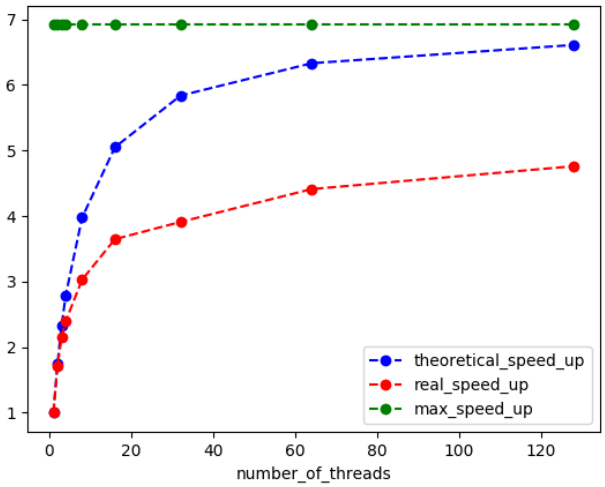

Figura 4: Grafico

Podemos observar que el speed up teorico tiende al maximo speed up mientras que el real encuentra un maximo luego de los 4 thread aproximadamente ya que la cantidad maxima de core de la computadora donde se corrio el programa es de 4

4.1.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	1.578808	2.407789	2
1	4	1.833916	1.959085	4
2	4	10.851145	11.962414	16
3	4	403.552532	135.166407	64
4	4	2299.946070	640.191793	100
5	4	16672.408342	4410.587788	200
6	4	56456.478119	14151.839733	300

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

Podemos ver que estos resultados demuestran que la sección serie del problema se mantiene casi constante respecto de la sección paralela que varía en forma ascendente con el tamaño de los datos de entrada. Así, la fracción secuencial representa menos al tiempo total en la medida que la carga de trabajo aumenta

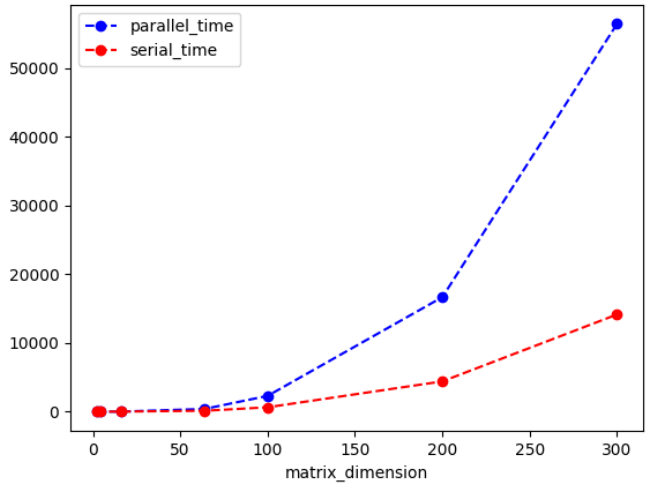

Figura 6: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	2.188087
1	4	2.450500
2	16	2.426934
3	64	3.247290
4	100	3.346774
5	200	3.372396
6	300	3.398718

Figura 7: Tabla de valores del speed up

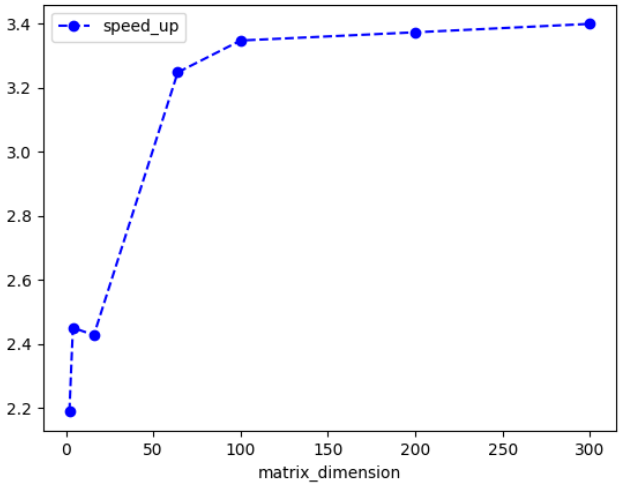

Figura 8: Grafico del speed up

4.2. Multiplicacion elemento por fila

4.2.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	3014.070272	565.473557	100
1	2	2426.795244	565.452337	100
2	3	1894.866705	563.212395	100
3	4	1854.820490	556.397915	100
4	8	1888.506889	557.536125	100
5	16	1852.516413	568.376541	100
6	32	1889.244318	566.127062	100
7	64	1901.825666	567.384481	100
8	128	1874.574900	557.483673	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	6.33017
1	2	1.727155	1.682125	6.33017
2	3	2.279727	2.057255	6.33017
3	4	2.713850	2.363701	6.33017
4	8	3.799003	3.082213	6.33017
5	16	4.748332	3.538496	6.33017
6	32	5.426320	3.927551	6.33017
7	64	5.843501	4.135334	6.33017
8	128	6.077108	4.250894	6.33017

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

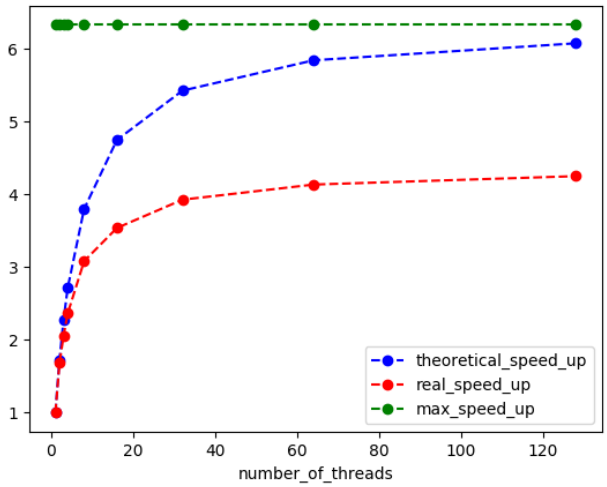

Figura 11: Grafico

Podemos observar que el speed up teorico tiende al maximo speed up mientras que el real encuentra un maximo luego de los 4 thread aproximadamente ya que la cantidad maxima de core de la computadora donde se corrio el programa es de 4

4.2.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	1.441479	2.287149	2
1	4	1.933575	1.022339	4
2	4	12.011766	4.426241	16
3	4	502.124786	178.322554	64
4	4	1892.707109	565.406799	100
5	4	16864.702940	5348.829508	200
6	4	66471.372128	18364.505768	300

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

Podemos ver que estos resultados demuestran que la sección serie del problema se mantiene casi constante respecto de la sección paralela que varía en forma ascendente con el tamaño de los datos de entrada. Así, la fracción secuencial representa menos al tiempo total en la medida que la carga de trabajo aumenta

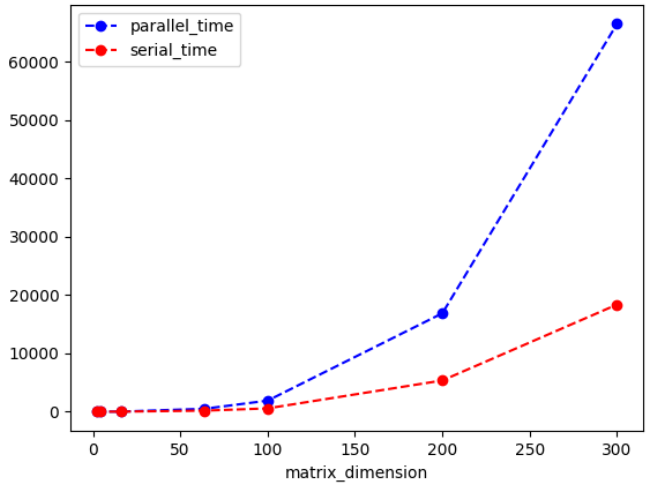

Figura 13: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	2.159793
1	4	2.962413
2	16	3.192194
3	64	3.213800
4	100	3.309950
5	200	3.277626
6	300	3.350587

Figura 14: Tabla de valores del speed up

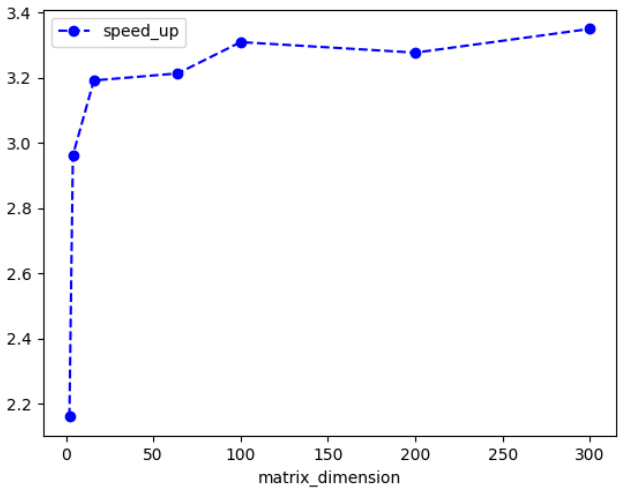

Figura 15: Grafico del speed up

4.3. Multiplicacion columna por fila

4.3.0.1 Salida Amdahl

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	1	2716.657877	613.165617	100
1	2	2333.233118	605.530024	100
2	3	1750.370502	608.537197	100
3	4	1775.277376	637.945175	100
4	8	1777.448416	643.797874	100
5	16	1838.029385	613.466740	100
6	32	2003.676891	618.367672	100
7	64	1804.571152	641.252518	100
8	128	2157.028198	621.368408	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	5.430545
1	2	1.688984	1.658307	5.430545
2	3	2.192522	1.978959	5.430545
3	4	2.576604	2.230821	5.430545
4	8	3.494968	2.795964	5.430545
5	16	4.252883	3.365851	5.430545
6	32	4.770103	3.850384	5.430545
7	64	5.078944	3.653488	5.430545
8	128	5.248863	4.353351	5.430545

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

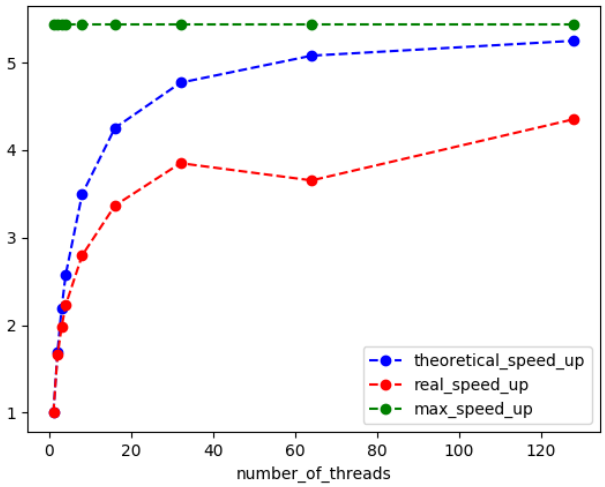

Figura 18: Grafico

Podemos observar que el speed up teorico tiende al maximo speed up mientras que el real empieza a tender a un maximo menor debido a que la cantidad maxima de core de la computadora donde se corrio el programa es de 4. Pero podemos ver tambien que hubo variaciones y que el speed up real tuvo anomalias y tendencias a crecer a pesar de el maximo de cores reales. Esto se debe a que al aumentar la cantidad de thread, la seccion serie del problema no se mantuvo constante si no que tuve ciertas variaciones equilibrando la caida de perfonmance de la seccion paralela. Esta variacion en el tiempo de la parte serie se debe a la forma en que quedan organizados los datos una vez procesados por los map workers

4.3.0.2 Salida Gustafson

	number_of_threads	parallel_time	serial_time	matrix_dimension
0	4	1.410007	2.422810	2
1	4	1.790047	3.273010	4
2	4	8.985043	7.999897	16
3	4	557.041168	226.171494	64
4	4	2036.477327	702.104807	100
5	4	17703.527451	5522.381067	200
6	4	67648.321867	19749.793291	300

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

Podemos ver que estos resultados demuestran que la sección serie del problema se mantiene casi constante respecto de la sección paralela que varía en forma ascendente con el tamaño de los datos de entrada. Así, la fracción secuencial representa menos al tiempo total en la medida que la carga de trabajo aumenta

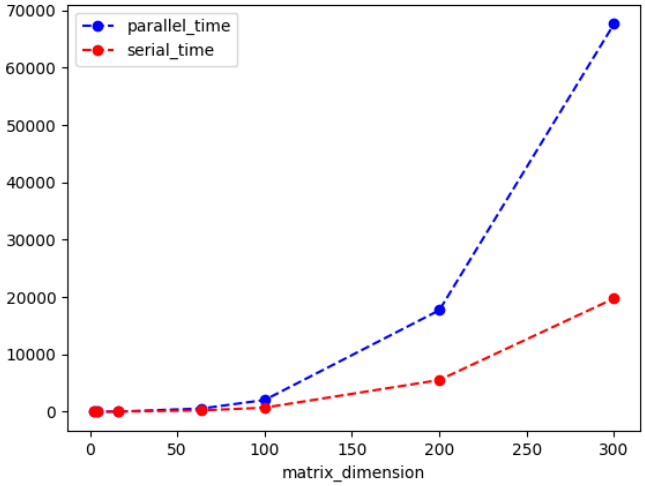

Figura 20: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	2.103633
1	4	2.060652
2	16	2.587002
3	64	3.133678
4	100	3.230874
5	200	3.286696
6	300	3.322075

Figura 21: Tabla de valores del speed up

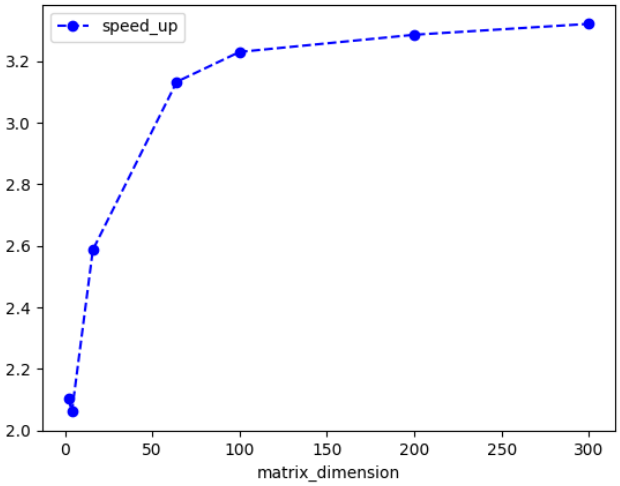

Figura 22: Grafico del speed up

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.