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 (aambas 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. Para realizar las mediciones se correrán N iteraciones del programa hasta lograr la convergencia de los resultados.

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.

En este trabajo se implementará una versión simplificada del problema clásico que consiste en multiplicar dos matrices de NxN. Lo que se hace es modificar la manera clasica en la que se multiplica las matrices de manera tal que podamos dividir la misma en sumas y cada una es la entrada de la función map. Tenemos que hacer un pre-procesamiento el cual consiste en generar una lista de tuplas a partir de las dos matrices. Se itera por cada elemento (`ai_ij`) de la matriz A y se guarda en cada tupla el numero de fila del elemento `ai_ij` , el elemento `ai_ij` y

la fila `j` de la matriz B.

De esta manera, en la funcion map, obtenemos 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 `ai_ij` contra el elemento de la fila j de la columna B en la columna `k` `b_jk` para k=0, ..., N.
Pseudocódigo:

3. Implementacion

3.1. Explicacion del modelo

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

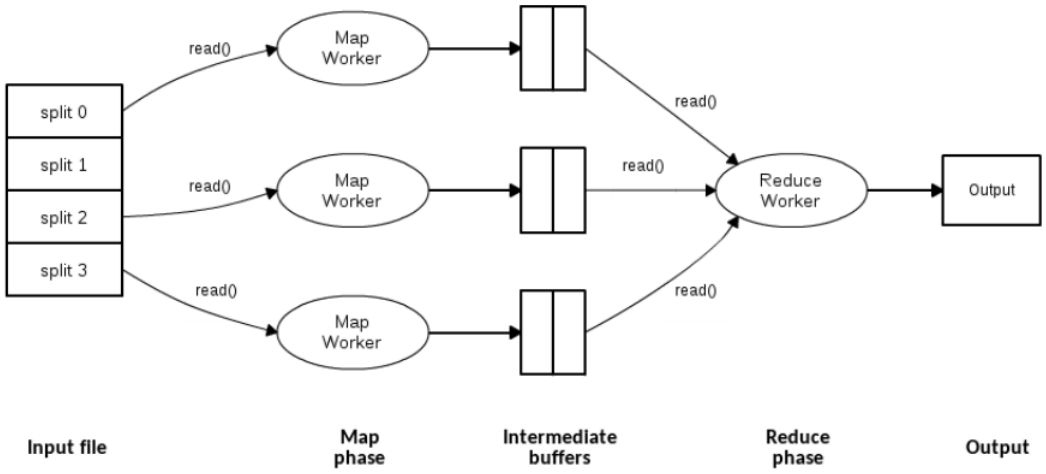

Figura 1: Esquema de un map reduce

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. Forma de ejecucion

Para ejecutar el calculo se debe ejecutar `$ sh scripts/run.sh`. Este comando hace el calculo de multiplicacion de dos matrices.
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

Luego para generar los graficos que vemos en el informe se debe ejecutar `$ sh scripts/generate_output_data.sh`

3.3. 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 amdahl

4.1. Salida

	number_of_threads	parallel_median_time	parallel_error	serial_median_time	serial_error	matrix_dimension
0	1	7.681568	0.264019	3.831466	0.169527	10
1	2	10.382141	0.474401	4.855445	0.174684	10
2	4	16.384542	0.339182	4.515926	0.184321	10
3	8	24.545074	0.479553	5.717891	0.430990	10
4	16	43.156774	0.499056	6.521398	0.292989	10
5	32	83.474787	0.490237	4.288983	0.063488	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	3.004864
1	2	1.500607	1.516704	3.138247
2	4	2.001620	2.426883	4.628169
3	8	2.402723	3.444443	5.292680
4	16	2.670269	5.388850	7.617718
5	32	2.827703	12.723867	20.462605

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

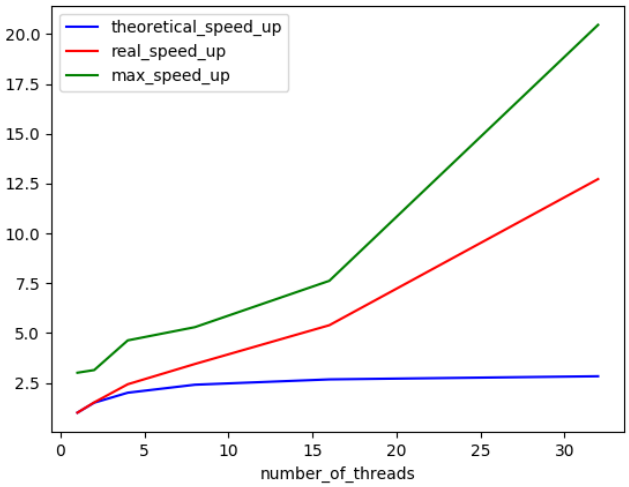

Figura 4: Grafico

Podemos observar que a medida que aumentan la cantidad de threads aumenta el speed up pero en el teorico, una vez que alcanzamos los cuatro threads (que es la canitdad maxima de nuestra pc) se mantiene constante. Sin embargo el speed up real sigue aumentando y en los 32 threads pega un salto considerable. Este resultado es bueno ya que aprovecha la paralelizacion para separar el trabajo en los distintos threads.'

5. Resultados gustafson

5.1. Salida

	number_of_threads	parallel_median_time	parallel_error	serial_median_time	serial_error	matrix_dimension
0	4	12.855482	0.490146	4.493618	0.132585	2
1	4	13.585548	0.443426	4.785220	0.152256	4
2	4	14.253338	0.356211	5.215327	0.283087	8
3	4	25.256325	0.497514	5.003732	0.094420	16
4	4	76.123207	0.619128	9.726147	0.100089	32
5	4	353.443490	1.111749	59.572598	0.785429	64

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

Podemos observar que la sección serie del problema se mantiene constante mientras que la sección paralela varía en forma lineal con la dimension de las matrices de entrada.

	matrix_dimension	parallel_median_time	serial_median_time
0	2	12.855482	4.493618
1	4	13.585548	4.785220
2	8	14.253338	5.215327
3	16	25.256325	5.003732
4	32	76.123207	9.726147
5	64	353.443490	59.572598

Figura 6: Tiempos en serie y paralelo

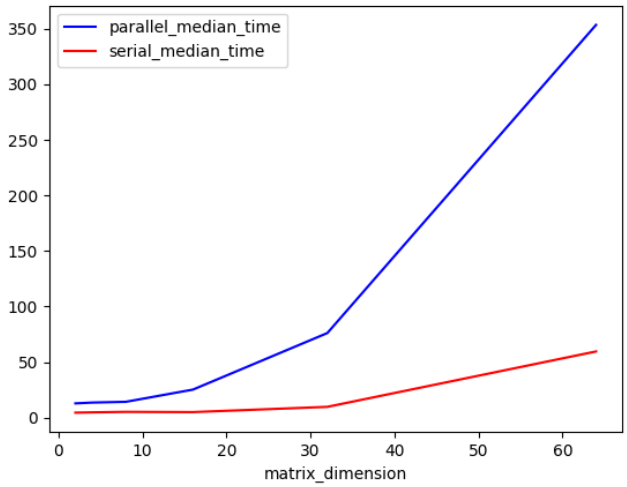

Figura 7: Tiempo paralelo y serie en funcion de la dimension de las matrices de entrada

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

	matrix_dimension	speed_up
0	2	1.740988
1	4	3.218560
2	8	6.124818
3	16	13.519635
4	32	28.487911
5	64	54.913009

Figura 8: Tabla de valores del speed up

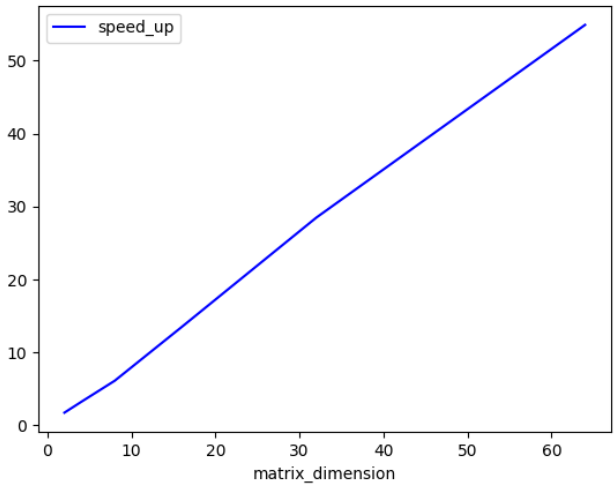

Figura 9: Grafico del speed up

6. 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.