Map - Reduce

Práctica 1 Sistemas distribuidos

Laboratorio Jueves 10-12h
BOSCÀ CANDEL, BERNAT
(bernat.bosca@estudiants.urv.cat)
PIZARRO LÓPEZ. ANNABEL
(annabel.pizarro@estudiants.urv.cat)

ÍNDICE

1. INTRODUCCIÓN

En este proyecto se nos ha pedido que implementemos un algoritmo distribuido simple de multiplicación de matrices. Esta versión distribuida se basa en definir un número de workers que trabajarán de forma concurrente en trozos (chunks) de las matrices a multiplicar.

Para la implementación de este programa, se nos ha pedido que utilicemos el IBM-PyWren. IBM-PyWren es un proyecto de código abierto que permite la ejecución de código Python a escala sobre las funciones de la nube de IBM, es decir, la plataforma de funciones como servicio (FaaS) de IBM, basada en Apache OpenWhisk. PyWren entrega el código del usuario en la plataforma sin servidores sin necesidad de conocer cómo se invocan y ejecutan las funciones. Soporta una API básica de MapReduce.

Para llevar el control de versiones del programa se nos ha pedido que usemos GitHub. El repositorio que contiene los ficheros de esta práctica es el siguiente: https://github.com/Annabelesca/DS_A1_Bosca_Pizarro

Este trabajo ha sido realizado de forma telemática por los dos miembros del grupo a partes iguales.

2. ESTRUCTURA DE LA PRÁCTICA

En este apartado, discutiremos los principales elementos estructurales de la práctica que básicamente recae en dos ficheros: Fichero MultMatriz.py, que se encarga de ejecutar el programa principal y el fichero Matriz.py que es un fichero que contiene la definición del objeto matriz. Después también cabe mencionar el fichero .pywren_config, este archivo permite configurar pywren y da acceso a las Cloud Function y al Cloud Storage de IBM. Por cuestiones de privacidad, no se sube el archivo con los datos de IBM que se han estado utilizando para realizar la práctica.

La práctica tiene varias secciones bastante diferenciadas, cuyas funciones las veremos a fondo en los apartados siguientes:

- Comprobación de los parámetros introducidos por el usuario (que no haya un número de filas/columnas inferior a 1, que el número de workers no exceda al número de filas de la matriz A y que el número de workers no exceda los 100).
- La creación de matrices (su creación, inicialización y posterior subida al COS de IBM).
- División de chunks en función del número de workers.
- Llamada a la función map, que se encarga de procesar cada uno de los chunks a partir de los archivos que contienen las matrices en el COS de manera concurrente y generar un archivo con los resultados temporales.
- Llamada a la función reduce, que se encarga de hacer un join de estos resultados temporales, creando la matriz resultante y subiéndola al COS.
- Llamada a la función de borrar archivos temporales creados durante el proceso de multiplicación.

Otro asunto que no está de más mencionar es que nuestro COS está compuesto por dos buckets, uno llamado “originalmatrix” que contiene las dos matrices iniciales (ya sean creadas

por nuestro programa o que hayan sido suministradas por el usuario) y otro que se llama "temporalresults" que contiene los ficheros con los resultados temporales de cada uno de los workers. Se ha decidido que el bucket llamado "temporalresults" sólo contendrá los ficheros temporales, por lo que el fichero creado como resultado de nuestro algoritmo se almacenará en el bucket "originalmatrix" bajo el nombre de "Matriz C.txt".

2.1 Matriz.py

Es la clase que define el objeto matriz en la cual se basa la práctica. Este objeto se encarga de crear matrices y de almacenarlas, así como de almacenar su número de filas y columnas. Además de crear las matrices, las inicializa con valores aleatorios en el rango [-10, 10]. También tiene un método para asignar valores en posiciones determinadas.

En caso de que sea necesario, las matrices creadas por el algoritmo se inicializarán en el constructor de este objeto. No obstante, lo harán en la nube a través de un map (se llama a un map en vez de un call_async por el hecho de crear las dos matrices de forma concurrente).

2.2 MultMatriz.py

Es la clase que se encarga de ejecutar el programa principal. Como nuestro algoritmo permite crear matrices y después dividir las matrices en diferentes chunks para paralelizar la multiplicación, la ejecución de este fichero requiere de algunos parámetros de entrada que se pasan al programa por la comanda para ejecutarlo:

>python MultMatriz.py m n l w

Parámetros de entrada:

m: Corresponderá a las filas de la matriz A a crear.

n: Corresponderá a las columnas de la matriz A y filas de B a crear.

l: Corresponderá a las columnas de la matriz B a crear.

w: Número de workers que usará nuestro programa.

Funciones:

crearFicheroMatriz: Recibe por parámetro el número de filas y columnas con las que se desea crear la matriz y, además, el nombre con el que se quiere guardar en el COS de IBM.

multMat: Nuestra función de map. Recibe por parámetro el número de línea de la matriz donde el worker debe empezar a trabajar y acabar. Cada worker que ejecuta esta función, baja los dos ficheros del COS y, a partir de una función auxiliar (multiplicacionMatrices, que veremos más adelante) se encarga de realizar la multiplicación de aquellas casillas que le corresponde. Cuando ha creado la matriz resultante, la sube al COS con el nombre "tempXXX" donde XXX corresponde a un número de 3 dígitos que indica la fila de inicio de la matriz calculada. Retorna este nombre.

multiplicacionMatrices: Función auxiliar que recibe una lista de listas, representando al conjunto de filas a trabajar de la matriz A y un diccionario que contiene las filas y columnas pertinentes para hacer la multiplicación de estas dos matrices. Retorna un diccionario que contiene la matriz con el resultado de la multiplicación realizada.

reduceFunction: Nuestra función de reduce. Se encarga de juntar todos los ficheros temporales para generar una matriz resultado. Recibe una lista ordenada de los nombres de los ficheros que se han generado en la función de map. Si esta lista tiene longitud 1 es que sólo había un worker y, por tanto, el fichero nombrado temp000.txt será el fichero que contenga toda la matriz resultante por tanto, se modifica el nombre de éste a MatrizC.txt y se sube al COS, al bucket "originalmatrix". En caso que esta longitud no sea 1 significará que hay más de un fichero temporal y la función hará el join de las submatrices que contiene cada uno de estos ficheros temporales.

resetBucket: Función que recibe como parámetro el nombre del bucket y el prefijo de los archivos a borrar. Se utiliza para borrar los ficheros temporales generados en la subdivisión de la matriz y aquellos resultantes de la serialización realizada por pickle.

3. JUEGO DE PRUEBAS

3.1 Pruebas básicas

Se han realizado un conjunto de pruebas para comprobar que las ejecuciones realizadas en diferentes circunstancias dieran las respuestas esperadas.

Prueba	Respuesta obtenida	¿Es correcto?
Se ejecuta el programa sin alguno de los parámetros necesarios.	Mensaje de error indicando que se deben pasar 4 parámetros (m, n, l y nº workers).	Sí
Se ejecuta el programa sin tener en el mismo directorio el fichero Matriz.py	Mensaje de error indicando que el fichero Matriz.py se debe encontrar en el mismo directorio que el script que se está ejecutando.	Sí
Se ejecuta el programa con el parámetro m, n, l o w con un valor negativo.	Mensaje de error indicando que las dimensiones de la matriz deben ser superiores a 1.	Sí
Se ejecuta el programa con el parámetro m, n, l o w con un valor no numérico.	Mensaje de error indicando que hay un error en el formato de los parámetros.	Sí
Se ejecuta el programa con el valor del parámetro m, n o l en 0.	Mensaje de error indicando que las dimensiones de la matriz deben ser superiores a 1.	Sí
Se ejecuta el programa con el parámetro w mayor a 100.	Mensaje indicando que el número de workers es máximo 100 y que el programa se ejecutará con este parámetro cambiado.	Sí
Se ejecuta el programa con el parámetro w inferior a 100 pero mayor que el número de filas de la matriz A.	Mensaje indicando que el número de workers como máximo debe ser el número de filas de la matriz A y que el programa se ejecutará con este parámetro cambiado.	Sí
Mientras se está ejecutando el	Mensaje de error indicando que el	Sí

programa se borran las matrices o los archivos temporales.	nombre del bucket al que se accede para hacer la multiplicación es incorrecto o que alguno de los ficheros de matriz no se encuentra.	
Se ejecuta el programa con nombres de buckets que no existen o no pueden ser alcanzados con las credenciales del pywren.	Mensaje de error indicando que hay un problema en la generación de matrices y que se revise el nombre de los buckets.	Sí
Se ejecuta el programa con los parámetros 100 100 100 1.	Se realiza la multiplicación de dos matrices de 100x100, generando una matriz resultante del 100x100 que se puede encontrar en el bucket.	Sí
Se ejecuta el programa con los parámetros 300 10 200 100.	Se realiza la multiplicación de dos matrices de 300x10 y 10x200 generando una matriz resultante del 300x200 que se puede encontrar en el bucket.	Sí

Tabla 1: Pruebas realizadas para comprobar una correcta ejecución del programa.

3.2 Pruebas de SPEED-UPS

Para realizar estas pruebas se ha comentado la parte de creación e inicialización de las matrices. Esto se ha hecho con la finalidad de poder comprobar el tiempo de ejecución de las diferentes configuraciones de workers de manera más fiable, ya que todas las ejecuciones se harían sobre unas mismas matrices. En este juego de pruebas no sólo mediremos el correcto funcionamiento del algoritmo sino que también comprobaremos como aumenta (o quizá disminuye en alguno de los casos) el tiempo de ejecución de la multiplicación de dos matrices.

NOTA: A pesar de que en las tablas mostradas a continuación sólo aparezca un valor, cada una de las configuraciones se ha ejecutado un mínimo de 3 veces y se ha hecho la media para poder normalizar el tiempo de ejecución de cada una de ellas. Se pueden encontrar las tablas con los tiempos propios para cada matriz en el anexo.

Tamaño matrices multiplicadas	**1 w**	**2 w**	**5 w**	**10 w**	**20w**	**50w**	**75 w**	**100w**	**100 w (solo map)**
100x100 * 100x100	3,25	3,63	3,38	3,51	5,25	7,17	8,90	10,04	3,88
200x200 * 200x200	5,95	4,29	3,67	4,32	5,28	6,88	10,36	12,15	5,20
300x300 * 300x300	10,86	6,84	5,31	5,83	6,94	10,51	11,62	13,15	5,71
400x400 * 400x400	19,52	12,40	8,28	8,76	9,93	10,78	15,40	17,20	6,11

500x500 * 500x500	36,42	19,82	13,07	13,75	15,04	15,01	21,49	22,44	8,85
600x600 * 600x600	36,35	23,38	13,07	15,86	15,86	17,52	21,34	24,62	9,27
700x700 * 700x700	94,80	49,37	30,84	31,16	34,53	30,08	35,57	41,18	16,45
800x800 * 800x800	139,39	73,42	44,44	44,99	49,46	39,95	46,65	46,42	23,57
900x900 * 900x900	199,03	106,33	60,62	63,28	69,08	54,05	62,46	61,38	35,73
1000x1000 * 1000x1000	276,59	141,83	83,13	83,30	92,17	74,59	83,37	77,68	43,94

Tabla 2: Muestra las medias de tiempos obtenidas para diferentes números de workers respecto a diferentes dimensiones de matrices.

No sólo hemos medido el tiempo para generar la matriz resultante (función de map y de reduce) sino que también hemos medido el tiempo de la multiplicación por separado (la función map). Con estos resultados hemos llegado a plantearnos una posible hipótesis del comportamiento de nuestro programa. Nuestro programa es competente a la hora de realizar las multiplicaciones de forma concurrente pero el efecto de cuello de botella de nuestro algoritmo se encuentra en el reduce.

Para analizar mejor estos datos, hemos decidido representarlos en forma de gráficos de dispersión con líneas.

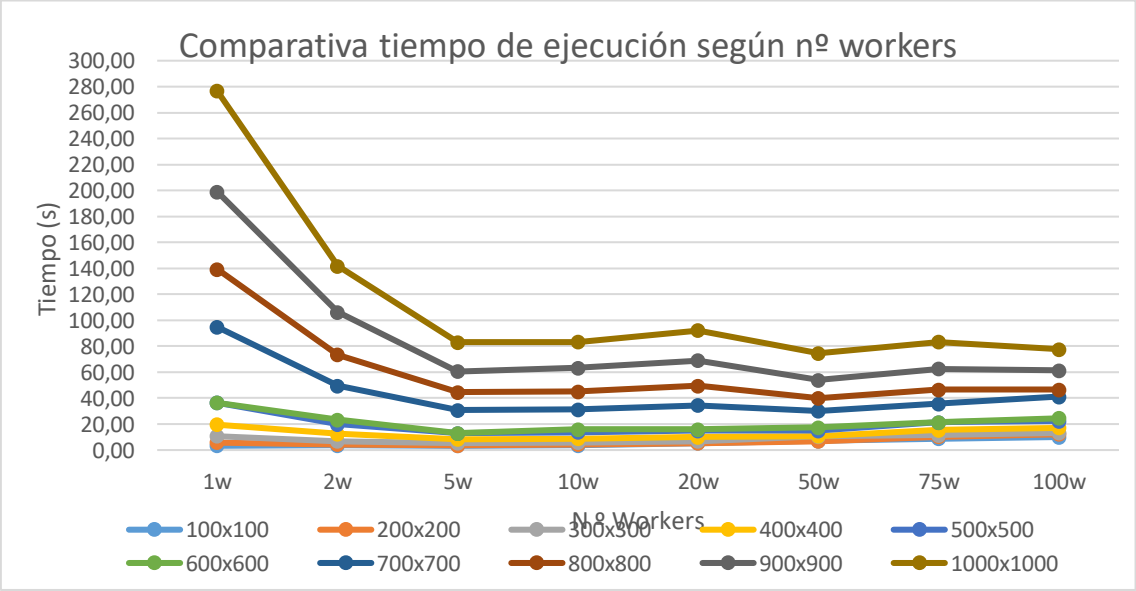

Ilustración 1. Comparativa tiempo de ejecución según el número de workers para matrices de 100x100 hasta 1000x1000

Si representamos todos los datos en el mismo gráfico, debido a la gran diferencia de tiempo entre la ejecución de las matrices más grandes respecto a aquellas más pequeñas, no podemos apreciar con detalle la curva de tiempo de ejecución de la multiplicación de matrices según el número de workers. Por ello, decidimos representar éstos datos por separado en dos grupos.

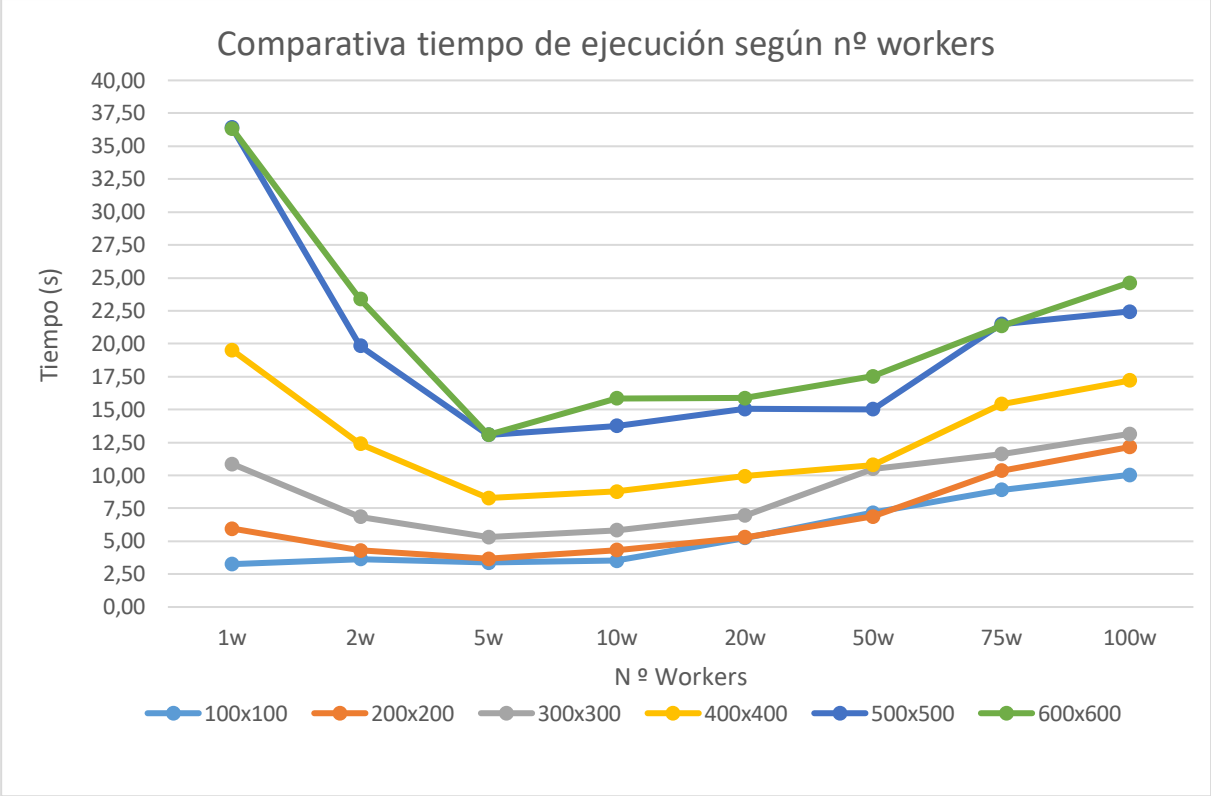

Ilustración 2. Comparativa tiempo de ejecución según el número de workers para matrices de 100x100 hasta 600x600

El primer grupo de datos representados corresponde a aquellos pertenecientes a la multiplicación de matrices de 100x100 hasta 600x600 ya que no presentan una gran diferencia temporal respecto al tiempo de ejecución.

Todos estos datos presentan una tendencia cóncava, es decir, que el tiempo de ejecución disminuye con un cierto número de workers y, pasado este punto, vuelve a incrementarse. Este resultado confirma nuestras hipótesis de que, conforme más workers haya, más rápido se lleva a cabo la multiplicación (la función de map), pero también se generan más ficheros temporales a los que luego hay que aplicarle un join (la función de reduce).

Como los workers trabajan de forma concurrente en la función de map, el tiempo de la multiplicación se reduce significativamente. No obstante, al llegar a la función de reduce ésta se ejecuta secuencialmente, por tanto, cuantos más ficheros temporales haya para procesar en la función de reduce, más lento será el programa ya que tiene que descargar cada uno de éstos.

A grandes rasgos, podríamos decir que el tiempo ahorrado en la multiplicación de forma concurrente no llega a compensar el tiempo usado en la función del reduce para hacer el join.

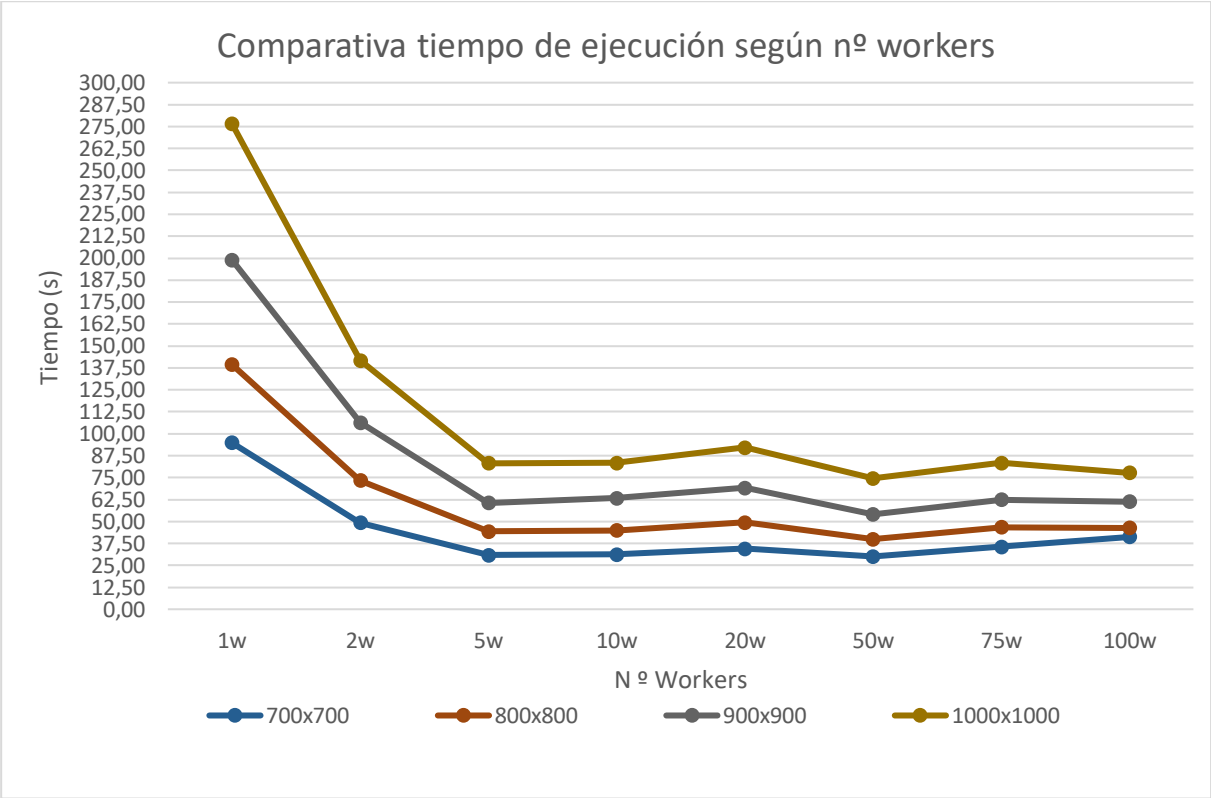

Ilustración 3. Comparativa tiempo de ejecución según el número de workers para matrices de 600x600 hasta 1000x1000

El segundo grupo de datos representados corresponde a aquellos pertenecientes a la multiplicación de matrices de 600x600 hasta 1000x1000. Éste grupo, al igual que el anterior, también presenta una tendencia cóncava con una disminución del tiempo de ejecución muy marcada y luego, en vez de aumentar, ésta se mantiene (e incluso en algunos casos, véase 50 workers) vuelve a bajar.

En primera instancia podemos observar una disminución bastante considerable en el tiempo de ejecución. Probablemente esto sea debido a que hay un ahorro importante de tiempo al realizar las multiplicaciones de manera concurrente y que el reduce no tiene que procesar demasiados archivos temporales, lo que significaría menos tiempo descargando archivos.

Por otro lado tenemos un punto (20 workers) donde este tiempo aumenta y genera un máximo local. Esto probablemente sea debido a que el tiempo ahorrado haciendo multiplicaciones de forma concurrente no llega a compensar el tiempo que se tarda en realizar el join de los archivos temporales.

Si además nos fijamos en los 50 workers podemos visualizar un mínimo local. Probablemente el tiempo que se gana haciendo las multiplicaciones de forma concurrente llegue a suplir esa pérdida de tiempo que se tiene al tener que procesar más ficheros temporales en la función de reduce.

4. CONCLUSIÓN

A partir de los datos obtenidos y las reflexiones presentadas anteriormente, hemos llegado a la conclusión de que se produce el efecto cuello de botella en nuestra función de reduce. Una forma de solventar esto sería realizando una función de shuffle que complementara a la función reduce. Esta función podría realizar de forma concurrente joins parciales de los resultados temporales, de tal manera que se ahorraría tiempo ya que el join no recaería todo sobre una función secuencial como es nuestro reduce.

Además, también hemos concluido que trabajar con más workers no significa seguir bajando el tiempo de forma exponencial, ya que llega un punto que esta bajada deja de ser significativa e incluso empieza un despunte hacia más tiempo ejecución a causa de tener más ficheros temporales que procesar.

ANEXO

Matriz 100x100

Workers	T1	T2	T3
1	3,34	3,67	2,75
2	3,85	3,45	3,59
5	3,21	3,56	3,37
10	3,42	3,87	3,23
20	5,73	5,29	4,74
50	6,39	8,00	7,12
75	7,88	8,87	9,95
100	8,98	10,66	10,47
100 solo map	3,53	4,09	4,02

Tabla 3: Muestra las de tiempos obtenidos para la matriz de 100x100.

Matriz 200x200

Workers	T1	T2	T3
1	5,87	6,48	5,50
2	4,59	4,71	3,57
5	3,91	3,73	3,37
10	4,35	4,48	4,12
20	5,23	5,46	5,16
50	6,35	7,89	6,39
75	10,10	10,33	10,64
100	13,37	10,54	12,53
100 solo map	5,13	5,39	5,08

Tabla 4: Muestra las de tiempos obtenidos para la matriz de 200x200.

Matriz 300x300

Workers	T1	T2	T3
1	10,68	10,78	11,12
2	6,61	6,87	7,04
5	5,09	5,86	4,99
10	5,76	5,88	5,84
20	6,88	6,99	6,95
50	10,16	10,84	10,52
75	11,94	11,12	11,80
100	12,29	14,01	13,14
100 solo map	6,12	5,57	5,43

Tabla 4: Muestra las de tiempos obtenidos para la matriz de 300x300.

Matriz 400x400

Workers	T1	T2	T3
1	20,04	19,58	18,95
2	12,29	12,34	12,56
5	8,03	8,49	8,31
10	8,96	8,92	8,40
20	9,75	10,29	9,74
50	10,58	11,08	10,68
75	15,79	15,83	14,59
100	17,42	18,17	16,01
100 solo map	5,48	6,15	6,71

Tabla 5: Muestra las de tiempos obtenidos para la matriz de 400x400.

Matriz 500x500

Workers	T1	T2	T3
1	36,28	36,20	36,78
2	19,73	19,72	20,02
5	12,80	12,48	13,92
10	13,22	13,97	14,07
20	15,39	15,12	14,62
50	14,81	15,64	14,58
75	22,33	21,15	20,99
100	21,48	23,01	22,84
100 solo map	9,38	7,98	9,20

Tabla 6: Muestra las de tiempos obtenidos para la matriz de 500x500.

Matriz 600x600

Workers	T1	T2	T3
1	35,29	36,24	37,52
2	23,16	23,71	23,26
5	13,11	13,23	12,88
10	16,08	15,47	16,02
20	16,24	15,29	16,06
50	17,23	18,15	17,18
75	21,38	21,25	21,39
100	24,40	24,04	25,41
100 solo map	8,89	8,67	10,24

Tabla 7: Muestra las de tiempos obtenidos para la matriz de 600x600.

Matriz 700x700

Workers	T1	T2	T3
1	93,65	95,83	94,93
2	49,08	49,63	49,40
5	31,81	30,42	30,30
10	33,07	30,75	29,66
20	34,52	33,79	35,29
50	29,64	30,29	30,30
75	35,67	36,09	34,95
100	42,20	42,86	38,49
100 solo map	17,06	15,89	16,39

Tabla 8: Muestra las de tiempos obtenidos para la matriz de 700x700.

Matriz 800x800

Workers	T1	T2	T3
1	139,93	138,67	139,56
2	73,38	73,43	73,44
5	44,15	47,25	41,92
10	45,74	44,10	45,14
20	52,46	48,23	47,68
50	39,77	39,86	40,22
75	46,42	45,68	47,85
100	44,56	47,37	47,32
100 solo map	22,66	24,51	23,55

Tabla 9: Muestra las de tiempos obtenidos para la matriz de 800x800.

Matriz 900x900

Workers	T1	T2	T3
1	198,27	199,14	199,67
2	107,01	105,58	106,41
5	60,20	61,20	60,45
10	64,41	63,65	61,78
20	71,44	67,41	68,40
50	55,45	53,77	52,94
75	64,06	63,59	59,73
100	63,89	57,98	62,28
100 solo map	36,74	32,94	37,51

Tabla 10: Muestra las de tiempos obtenidos para la matriz de 900x900.

Matriz 1000x1000

Workers	T1	T2	T3
1	279,70	275,65	274,43
2	141,74	141,89	141,85
5	87,01	82,61	79,76
10	83,49	82,97	83,44
20	93,00	91,94	91,57
50	76,85	72,24	74,69
75	85,75	83,92	80,45
100	80,25	75,29	77,49
100 solo map	43,70	44,35	43,77

Tabla 11: Muestra las de tiempos obtenidos para la matriz de 1000x1000.