Comparativa de rendimiento haciendo uso de Pthreads entre nodos de cómputo a través de aplicaciones de juguete (Benchmarks)

Oscar Julián Reyes Torres
Universidad Sergio Arboleda.
Bogotá, Colombia
oscar.reyes0101@correo.usa.edu.co

***Abstract*—In this laboratory, the formal definitions of elements necessary for the execution of the practice are initially presented, such as fundamentals in matrices, memory in a computing node, static memory reservation, dynamic memory reservation, threads present in a computing node, principle of spatial and temporal locality, parallel computing, OpenMP interface and Amdahl's Law, the above, in order to perform a comparison of performance in four computing nodes through toy applications programmed in c, which were designed for the calculation of matrix multiplication, adding interfaces and references to achieve a faster parallel execution in each computing node, saving the samples taken and analyzing through bar diagrams, the most efficient computing node and as its internal architecture is the main responsible for optimizing runtime on square arrays from 100 x 100 to 3200 x 3200.**

***Index Terms*—benchmark, nodo de computo, Pthreads, puntero, c, c++, computación parela, Ley de Amdahl, Localidad temporal, Reserva de memoria y dirección de memoria**

I. Objetivo

Comparar de forma exhaustiva el rendimiento de diferentes nodos de cómputo aprovechando la capacidad máxima de los cores disponibles en cada arquitectura por medio de tres pruebas de rendimiento realizadas en lenguaje c y c++

II. Introducción

En este documento se compara el rendimiento de cuatro nodos de cómputo, por medio de baterías experimentales de multiplicación de matrices cuadradas con el fin de emplear al máximo todos los recursos disponibles en cada nodo de cómputo y visualizar la optimización en el cálculo matemático.

Lo anterior, presentando algunos aspectos fundamentales como la programación paralela en general y elementos teóricos sobre la memoria como los términos de localidad espacial, localidad temporal y la eficiencia de la caché, al igual que la ley de Amdahl y como esta busca mejorar el rendimiento de un sistema.

III. Fundamentos

Matriz Esta sección presenta algunos aspectos teóricos sobre cómo se manejaron las matrices en los programas para que el lector tenga una comprensión absoluta sobre la discusión que se lleva a cabo durante todo el trabajo.

- Representación matriz-vector
  A nivel de maquina es importante precisar que un computador "no ve" los datos de forma bidimensional, por el contrario, los tratan de manera dimensional, lo que conlleva a definir que el tratamiento que se tiene de las matrices es de una estructura vector-unidimensional, de esta manera:

Fig. 1. Ilustración de matriz - vector

- Comportamiento del bus en la multiplicación con matriz clásica
  Teniendo en cuenta lo anterior, es importante precisar que a nivel de máquina la multiplicación de matrices se realiza haciendo uso de un bus, en el cual se cargan los datos a causa de la localidad espacial, esto representa el peor de los casos, debido a que solo se usa un dato del bus de una matriz-vector para realizar la multiplicación, lo que implica penalizaciones y fallos de memoria haciendo que haya más viajes para buscar los datos siguientes incrementando el tiempo de ejecución.

Reserva de memoria
En la creación de cualquier software mediante el uso de un lenguaje de programación, para este caso c y c++ se busca que este se ejecute de forma óptima. Para el lenguaje c, utilizado en esta comparativa, obtiene espacio de memoria teniendo dos opciones, la reserva de memoria de forma dinámica y la reserva de memoria de forma estática.

- Reserva de memoria estática
  Esta viene definida como una memoria de duración fija, la cual se reserva al momento de compilar y libera de forma automática. Esto se traduce en que esta no se puede modificar cuando se está ejecutando y que previamente conoce el tamaño de la estructura a ser ejecutada.

  Como programador, este tipo de asignación de memoria no se basa en funciones utilizadas en la asignación de memoria dinámica, dado que para este caso el espacio de memoria asignado dependerá de la declaración de variables de cualquier tipo de dato ya sean primitivos *(int, char,float, double, entre otros)* o derivados *(struc, matrices, punteros, etc)*.

- Reserva de memoria dinámica
  A diferencia de la memoria estática, esta de reserva en tiempo de ejecución, por lo que su tamaño puede variar mientras el programa se ejecuta. Como programador, la razón principal para hacer uso de la reserva de memoria dinámica es cuando al momento de crear el programa, no se sabe con exactitud el número de datos o elementos a utilizarse.

  Su funcionamiento se basa en las operaciones de petición y liberación de memoria, por lo que cuando se ejecuta el programa se solicita tanta memoria en bytes, según los datos que se vayan a almacenar y cuando ese dato o datos no sean necesarios, se libera la memoria para que se pueda reutilizar después. Este tipo de ciclo es conocido como "gestión explícita de memoria", ya que se tiene una operación para pedir memoria y otra para liberarla. Las operaciones principales para la gestión de memoria son:

  - $void * malloc(size_t size)$: Función para reservar bytes consecutivos de memoria según indique su parámetro y retorna la dirección de memoria de la parte que se reservó.

  - $void * calloc(size_t nmemb, size_t size)$: Función para reservar espacio para la cantidad de elementos según indique su parámetro "nmemb", y cada elemento tendrá una cantidad de bytes reservados según indique el segundo parámetro "size", y retornará la dirección de memoria al comienzo del bloque reservado.

  - $void\ free(void * ptr)$: unción que recibe un puntero como parámetro y libera el espacio que se reservó para este.

  - $void * realloc(void * ptr, size_t size)$: Función para redimensionar una porción de memoria que se haya reservado al primer parámetro,según el tamaño que indique el segundo parámetro. Esta función retornará la dirección de memoria de la nueva porción redimensionada. Es importante aclarar que los datos no se modificarán en tantos bytes como el mínimo entre el tamaño antiguo y el nuevo.

Pthreads

Un pthread o hilo, se refiere a un proceso ligero o subproceso que realiza una secuencia de tareas encadenas que pueden ser ejecutadas por un sistema operativo. En la actualidad, la mayoría de procesos simples ejecutados en un ordenador suelen ser representados como un único hilo de control, ejecutándose un proceso a la vez, lo que significa que no se están aprovechando todos los recursos que provee el sistema.

Por lo anterior, se entiende que si se ejecutan múltiples hilos paralelos a una tarea específica esto traduce en optimizar un proceso dado que se crea una distribución de trabajo, ya que estos efectúan eventos independientes pero de forma sincrónica. [1]

Principio de localidad espacial

Este principio hace referencia a que los datos contiguos a un dato solicitado tienen una alta probabilidad de ser solicitados. Existe la localidad espacial entre las posiciones de memoria que son referenciadas en momentos cercanos. En este caso es común estimar las posiciones cercanas para que estas tengan un acceso más rápido.
En resumen, cuando un programa accede a una instrucción o a un dato, existe una elevada probabilidad de que instrucciones o datos cercanos sean accedidos pronto. [3]

Principio de localidad temporal

El cual define que, si un dato fue ejecutado en o referenciado, entonces es muy probable que ese mismo dato vuelva a ser utilizado en un futuro cercano. A nivel de máquina, en estos casos es común que se almacene una copia del dato referenciado en cache para lograr un acceso más rápido a este. [3]

Computación Paralela

La computación paralela es una forma de cómputo en la que se hace uso de 2 o más procesadores para resolver una tarea. La técnica se basa en el principio según el cual, algunas tareas se pueden dividir en partes más pequeñas que puedes ser resueltas simultáneamente como en la siguiente figura:

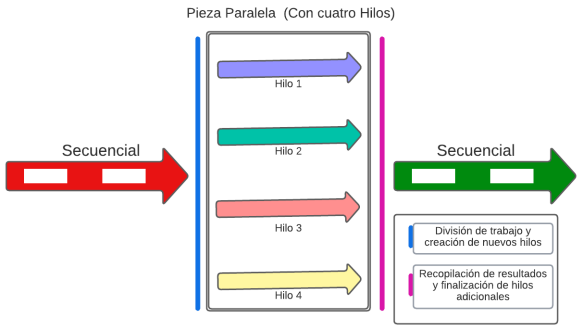

Fig. 2. Ilustración de un programa Paralelo

Como se puede observar en la Figura 2 se divide la tarea, en este caso un proceso en varios subprocesos para de esta forma resolverlo de forma más rápida. En algunos casos la computación paralela también da acceso a más memoria, lo que da la posibilidad de resolver problemas mayores.

Pero esto puede presentar algunos desafíos. Además de resolver el problema en cuestión, se agregan las tareas adicionales de dividir y recolectar el trabajo. Es más, puede haber un costo adicional en la creación de nuevos hilos. Los diferentes procesadores también necesitan comunicarse, para que estén sincronizados y los problemas se resuelvan correctamente. Además de esto, la ejecución de un programa paralelo puede ser no determinista, por los hace difíciles de depurar [4]

En la computación paralela existe una memoria compartida (SMPP) la cual es visible para todos los procesadores y es la encargada de la comunicación entre los subprocesos, sin embargo si los datos se comparten de forma no intencionada pueden producirse cuellos de botella, por ejemplo, dos o más subprocesos escriben en la misma ranura de memoria simultáneamente, acarreando en sobre escritura entre sí.

Pero la programación paralela también se puede ejecutar sobre una memoria distribuida (DMPP), donde cada hilo se le asigna su propio almacenamiento privado, lo que significa que cada hilo trabaja sin tener en cuenta otros hilos y de esta forma se corrige cuestiones como los conflictos de escritura y los de intercambio. Sin embargo, al momento de comunicarse entre subprocesos se debe enviar y recibir fragmentos de datos entre todos, suponiendo un coste adicional.

OpenMP

OpenMP u (Open Multi-Processing) es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. También podría definirse como un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas para las plataformas que van desde las computadoras de escritorio hasta los supercomputadores.

Esta API funciona con c, c++ y Fortran. Las funciones de OpenMP se crean mediante construcciones. La mayoría de las construcciones son directivas del compilador, y normalmente se aplican a un bloque de código estructurado. En C / C ++, las construcciones están escritas utilizando la sentencia *pragma omp ¡constructo¿ [¡cláusula¿].* Los programas que usan OpenMP operan en una estructura de unión, donde se genera un hilo maestro y nuevos hilos para crear secciones paralelas, el cual los une una vez que la sección paralela ha terminado. [2]

El número de subprocesos que crea se decide mediante una función en tiempo de ejecución o una cláusula. Una sección paralela en OpenMP generalmente funciona con cualquier cantidad de subprocesos. Por tanto, un programa que utiliza OpenMP puede funcionar tanto secuencialmente como en paralelo. OpenMP también tiene una construcción de bucle for paralelo que se puede utilizar en paralelo secciones o para crear una nueva sección paralela. Un bucle paralelo divide la iteración del bucle entre los diferentes hilos. Al ejecutar un bucle en paralelo, es importante que las iteraciones del bucle se pueden ejecutar independientemente unas de otras. Esto se debe a que el bucle paralelo puede ejecutar las iteraciones en un orden no determinista, lo que puede tener una gran influencia en el rendimiento del programa. Además, los hilos pueden comunicarse entre sí a través de las variables compartidas, OpenMP ofrece varias construcciones para la sincronización.

Ley de Amdahl

Esta se define como un modelo matemático que 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 utiliza dicho componente”, su ecuación viene dada por:

$$T_m = T_a \cdot ((1 - F_m) + \frac{F_m}{A_m}) \tag{1}$$

Donde:
- T_m : Tiempo de ejecución meorado.
- T_a : Tiempo de ejecución inmediatamente anterior.
- F_m : Fracción de tiempo que el sistema utiliza el subsistema mejorado.
- A_m : Factor de mejora que se ha introducido en el sistema.

Visto de otra forma, también puede representar la ganancia de velocidad *(A)* al dividir 1 entre el tiempo de ejecución mejorado:

$$A = \frac{1}{T_a \cdot ((1 - F_m) + \frac{F_m}{A_m})} \quad (2)$$

Lo anterior, tiene como finalidad en la informática, el análisis de factibilidad de introducir una mejora en el sistema, dado un software y un hardware. *Cita*

IV. Metodología

Para realizar la comparativa del rendimiento de los nodos de cómputo, se desarrollaron tres benchmarks que realizan la multiplicación matricial, haciendo uso de los Cores disponibles en cada nodo de cómputo, para así tener un mejor rendimiento y optimización del cálculo matemático

Lo anterior se realiza en cuatro nodos de cómputo los cuales efectúan la multiplicación matricial dividiendo y repartiendo en subprocesos para así optimizar el rendimiento y tiempo de ejecución, repitiendo esta acción 30 veces para matrices cuadradas de $100 * 100$, $400 * 400$, $800 * 800$, $1600 * 1600$, $2400 * 2400$, y $3200 * 3200$, a continuación se presenta la configuración de cada nodo de cómputo utilizado:

A. Configuraciones de los Nodos

TABLE I
CONFIGURACIONES DE NODOS DE COMPUTO

Nodo	Nucleos	Ram	Procesador	Velocidad
A	8	4 GB	Intel(R) Core(TM) i5-8250	1.6 GHz
B	8	8 GB	AMD Ryzen 7 3700U	2.3 GHz
C	8	16 GB	Intel(R) Core(TM) i7-1185G7	3 GHz
D	8	8 GB	Intel(R) Core(TM) i7-8550U	1.8 GHz

Vale la pena mencionar, que, para la parte experimental, los nodos presentados en la tabla I trabajan con el sistema operativo Linux en algunas de sus diferentes distribuciones.

Por otra parte , se tiene un script de extensión perl, este código cumple el objetivo de facilitar la ejecución de las baterías de prueba para los benchmarks, ya que define la cantidad de ejecuciones, tamaño de las matrices y los programas de juguete que se deben utilizar, por otra parte, en base a estos datos ejecuta los benchmarks y guarda los resultados de tiempo de ejecución en segundos en un archivo plano, los cuales ya se utilizan para su respectivo análisis.
Los códigos empleados en esta comparativa, están disponibles en el siguiente repositorio: click aqui

V. Procedimiento

Parta obtener los resultados de las baterías de ejecución de cada benchmark, se definieron el número de ejecuciones como 31 y las cargas de las matrices a utilizar:

- $100 * 100$
- $400 * 400$
- $800 * 800$
- $1600 * 1600$
- $2400 * 2400$
- $3200 * 3200$

Estas cargas y el número de ejecuciones se registraron el el script perl y se procedio a ejecutar por cada nodo. Luego estos resultados se procesaron mediante graficas para su respectivo analisis.

VI. Resltados

A continuación se presentan los resultados obtenidos. Primero se comparala el tiempo promedio que le tomo a cada nodo de cómputo ejecutar cada benchmark con una carga específica, los resultados se muestran a continuación:

TABLE II
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 100 X 100

Carga 100 x 100 con 1 Hilo			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,004116s	0,003582s	0,003545133333s
B	0,003092s	0,002671066667s	0,002538166667s
C	0,002703s	0,002554266667s	0,0027448s
D	0,002559s	0,002389366667s	0,004902566667s

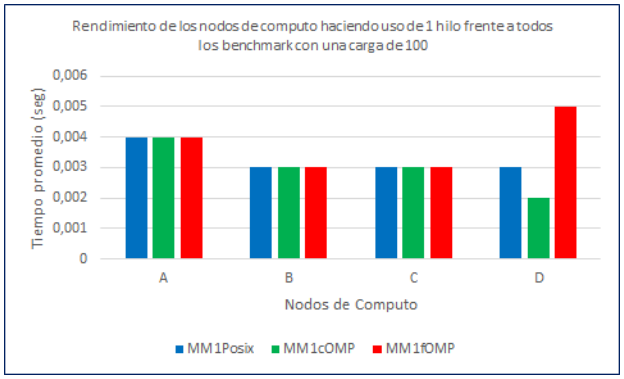

Fig. 3. Grafica del rendimiento de los nodos con un hilo y una carga de $100 * 100$

TABLE III
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 400 X 400

Carga 400 x 400 con 1 Hilo			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,27118s	0,2438164667s	0,2196675333s
B	0,20383s	0,1651745667s	0,1498214333s
C	0,19024s	0,1704039667s	0,1736329333s
D	0,13483s	0,1492808667s	0,191695s

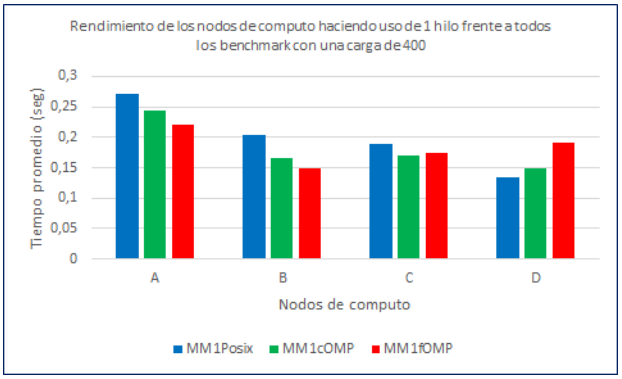

Fig. 4. Grafica del rendimiento de los nodos con un hilo y una carga de $400 * 400$

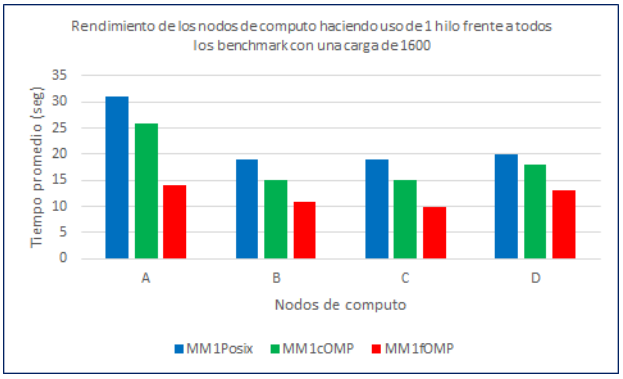

Fig. 6. Grafica del rendimiento de los nodos con un hilo y una carga de $1600 * 1600$

TABLE IV
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 800 X 800

Carga 800 x 800 con 1 Hilo			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	3,35170s	2,991565467s	1,742672367s
B	1,94934s	1,3182805s	2,148153667s
C	1,69513s	1,317928033s	1,210607667s
D	1,66344s	1,717643533s	1,5509672s

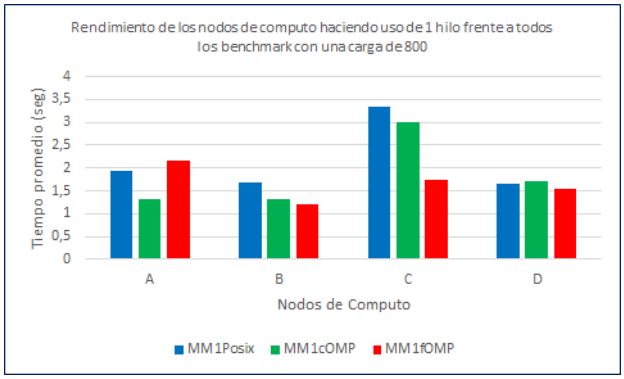

Fig. 5. Grafica del rendimiento de los nodos con un hilo y una carga de $800 * 800$

TABLE V
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 1600 X 1600

Carga 1600 x 1600 con 1 Hilo			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	30,999s	26,231s	13,90121697s
B	19,394s	15,236s	10,67020057s
C	19,169s	15,091s	10,24532023s
D	20,120s	17,696s	12,7797624s

TABLE VI
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 2400 X 2400

Carga 2400 x 2400 con 1 Hilo			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	110,063s	94,53920833s	46,93259563s
B	65,191s	51,39316753s	36,30259593s
C	62,441s	49,76746603s	34,98868093s
D	75,361s	68,70275927s	43,8481657s

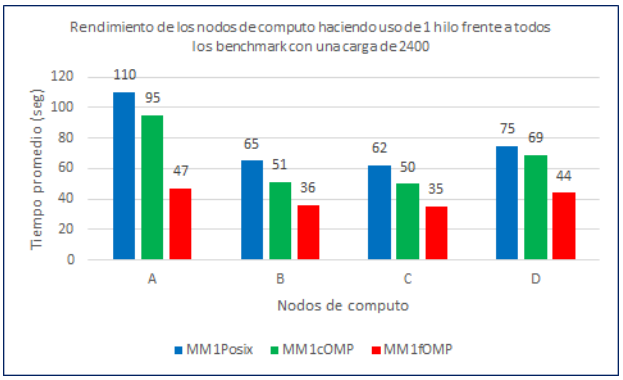

Fig. 7. Grafica del rendimiento de los nodos con un hilo y una carga de $2400 * 2400$

TABLE VII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 3200 X 3200

Carga 3200 x 3200 con 1 Hilo			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	573,522s	509,9898584s	111,2497072s
B	189,478s	255,2566809s	87,1057993s
C	171,276s	137,5772753s	82,61284823s
D	239,660s	224,6522969s	104,7710712s

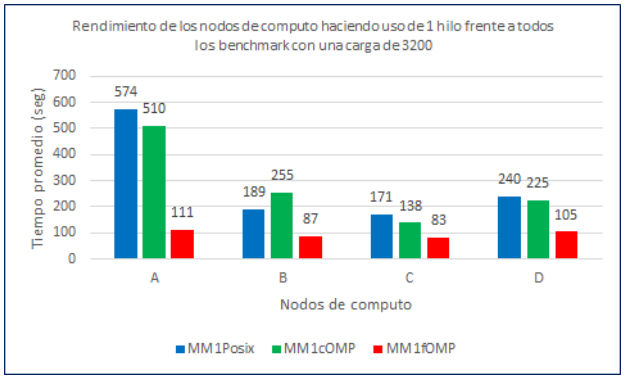

Fig. 8. Grafica del rendimiento de los nodos con un hilo y una carga de $3200 * 3200$

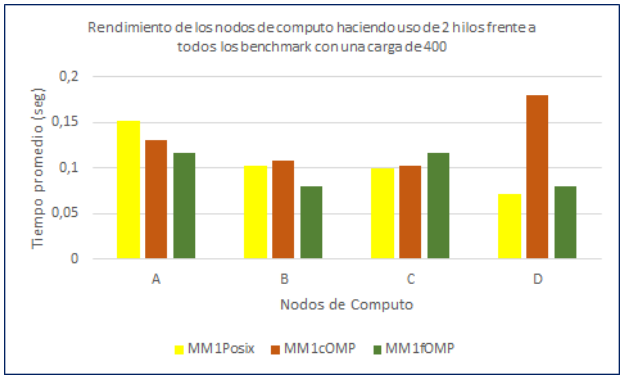

Fig. 10. Grafica del rendimiento de los nodos con dos hilos y una carga de $400 * 400$

TABLE VIII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 100 X 100

Carga 100 x 100 con 2 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,002201s	0,0020407s	0,002017066667s
B	0,001638s	0,0014787s	0,001399166667s
C	0,001467s	0,001358233333s	0,001432133333s
D	0,001341s	0,001349833333s	0,002536s

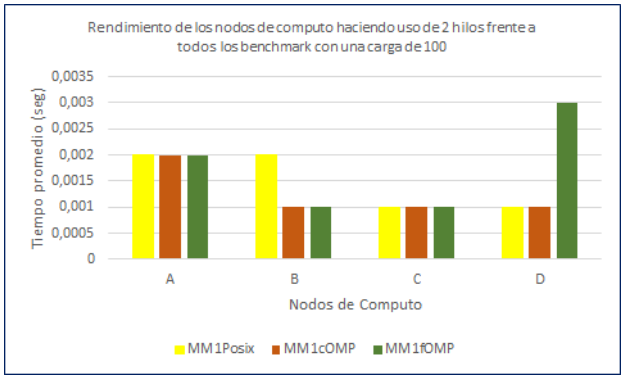

Fig. 9. Grafica del rendimiento de los nodos con dos hilos y una carga de $100 * 100$

TABLE IX
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 400 X 400

Carga 400 x 400 con 2 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,15089s	0,1306855333s	0,1172469333s
B	0,10313s	0,1082560333s	0,07954346667s
C	0,10044s	0,1021418333s	0,1155296s
D	0,07242s	0,1800771333s	0,07962696667s

TABLE X
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 800 X 800

Carga 800 x 800 con 2 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	1,68610s	1,455471267s	0,9426377s
B	1,18620s	0,8768333333s	0,7928061s
C	0,95296s	0,8144303667s	0,7382525667s
D	1,00230s	1,017765633s	0,9066185333s

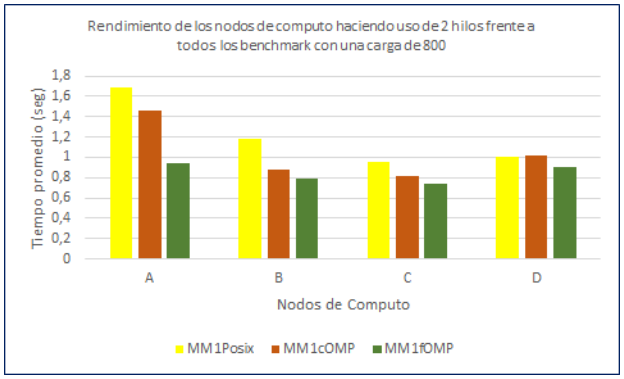

Fig. 11. Grafica del rendimiento de los nodos con dos hilos y una carga de $800 * 800$

TABLE XI
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 1600 X 1600

Carga 1600 x 1600 con 2 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	15,467s	12,692s	7,5279604s
B	11,982s	9,386s	6,693904767s
C	11,493s	8,908s	6,3515487s
D	10,397s	9,410s	7,2774482s

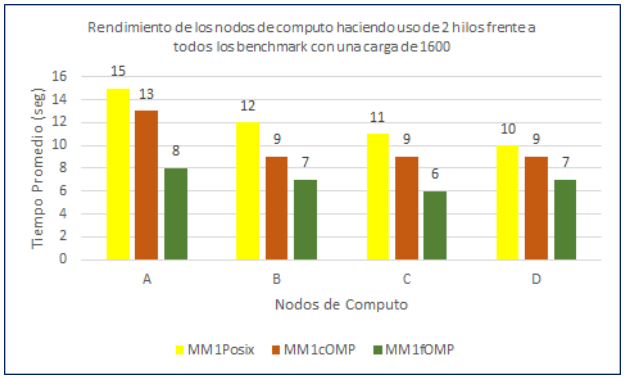

Fig. 12. Grafica del rendimiento de los nodos con dos hilos y una carga de $1600 * 1600$

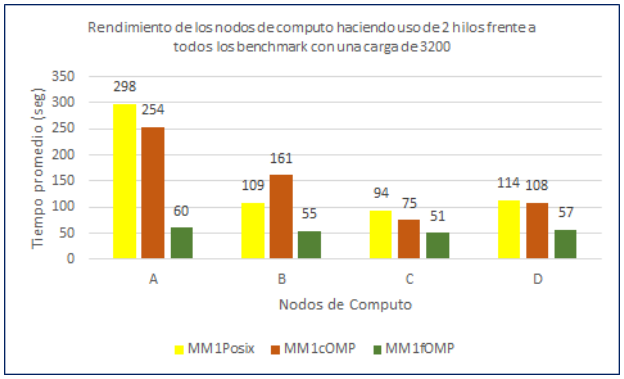

Fig. 14. Grafica del rendimiento de los nodos con dos hilos y una carga de $3200 * 3200$

TABLE XII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 2400 X 2400

Carga 2400 x 2400 con 2 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	56,450s	47,47530087s	25,40307903s
B	41,228s	33,06020403s	23,0735719s
C	37,935s	30,4606522s	21,5410012s
D	38,442s	35,70143307s	24,81750117s

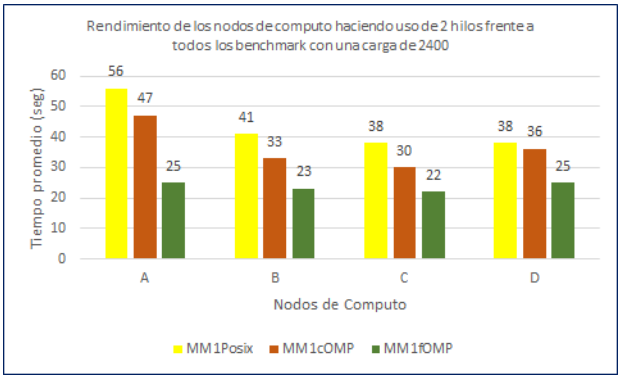

Fig. 13. Grafica del rendimiento de los nodos con dos hilos y una carga de $2400 * 2400$

TABLE XIII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 3200 X 3200

Carga 3200 x 3200 con 2 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	298,076s	254,0539448s	60,2181854s
B	109,356s	161,1410397s	54,89304363s
C	94,331s	75,36346707s	51,27373433s
D	113,608s	107,8668562s	57,31622907s

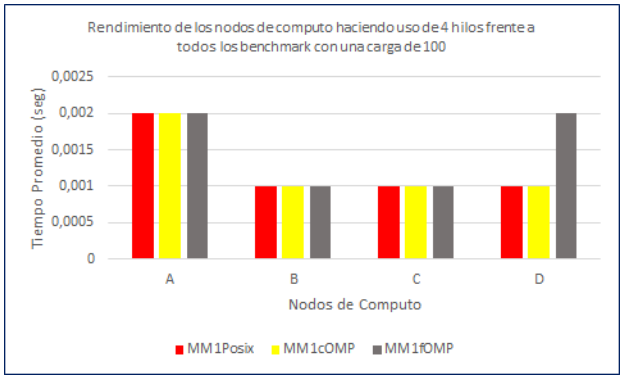

Fig. 15. Grafica del rendimiento de los nodos con cuatro hilos y una carga de $100 * 100$

TABLE XIV
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 100 X 100

Carga 100 x 100 con 4 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,002216s	0,002015s	0,001935966667s
B	0,001485s	0,001117666667s	0,0008356666667s
C	0,001028s	0,001065633333s	0,000925s
D	0,001093s	0,0011321s	0,0019658s

TABLE XV
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 400 X 400

Carga 400 x 400 con 4 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,15198s	0,1325735667s	0,1084815s
B	0,06557s	0,07698526667s	0,05298373333s
C	0,06582s	0,07703673333s	0,08304026667s
D	0,04910s	0,1172562s	0,0799335s

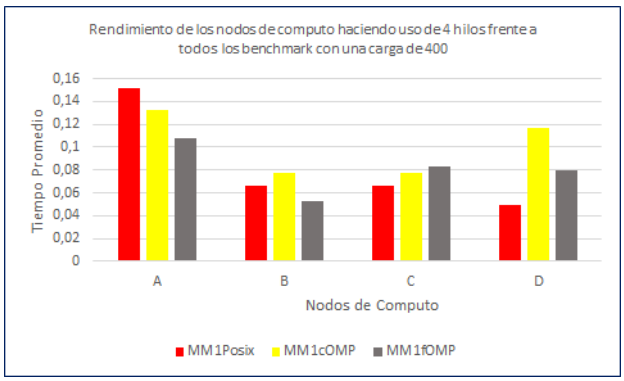

Fig. 16. Grafica del rendimiento de los nodos con cuatro hilos y una carga de $400 * 400$

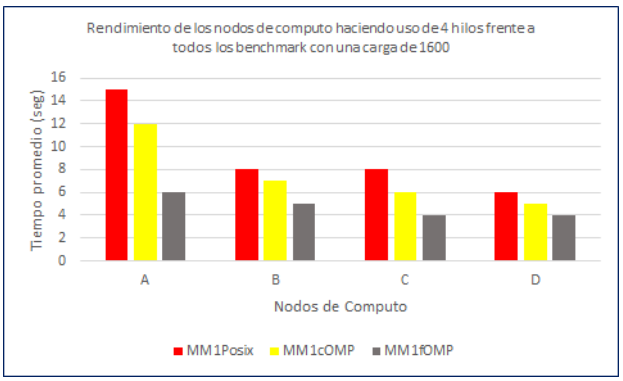

Fig. 18. Grafica del rendimiento de los nodos con cuatro hilos y una carga de $1600 * 1600$

TABLE XVI
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 800 X 800

Carga 800 x 800 con 4 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	1,57188s	1,2491997s	0,8162168s
B	0,81678s	0,6124258s	0,5408898333s
C	0,75545s	0,6119431333s	0,5323437333s
D	0,68442s	0,6283824667s	0,4778734667s

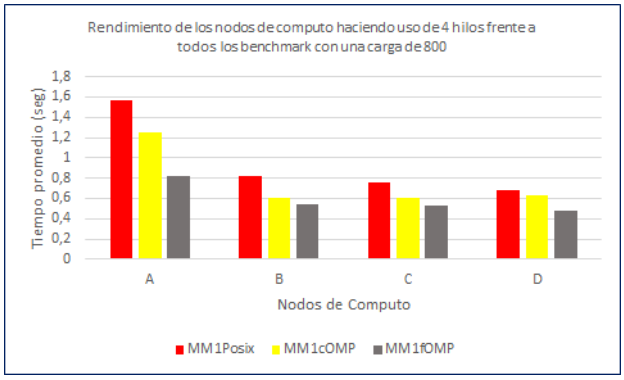

Fig. 17. Grafica del rendimiento de los nodos con cuatro hilos y una carga de $800 * 800$

TABLE XVIII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 2400 X 2400

Carga 2400 x 2400 con 4 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	53,140s	44,92245653s	21,49270283s
B	29,111s	23,5467583s	15,4862854s
C	27,369s	22,07352073s	14,64649657s
D	21,521s	20,53402303s	14,18101007s

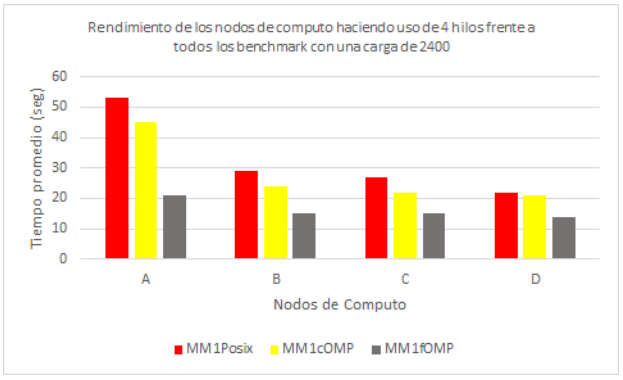

Fig. 19. Grafica del rendimiento de los nodos con cuatro hilos y una carga de $2400 * 2400$

TABLE XVII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 1600 X 1600

Carga 1600 x 1600 con 4 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	15,447s	12,021s	6,383315267s
B	8,362s	6,686s	4,5442389s
C	8,090s	6,418s	4,394886467s
D	5,672s	5,406s	4,4637792s

TABLE XIX
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 3200 X 3200

Carga 3200 x 3200 con 4 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	264,052s	209,1437599s	50,92272753s
B	83,128s	122,320107s	36,68858723s
C	76,223s	61,404923s	34,86545533s
D	70,357s	66,94946457s	32,7034265s

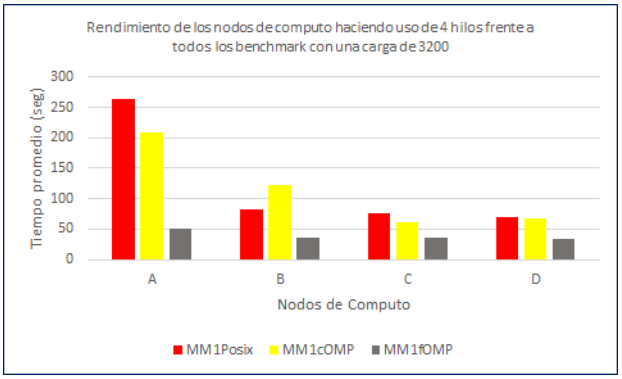

Fig. 20. Grafica del rendimiento de los nodos con cuatro hilos y una carga de $3200 * 3200$

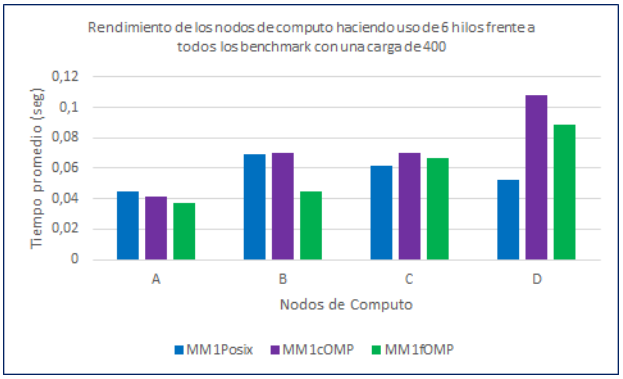

Fig. 22. Grafica del rendimiento de los nodos con seis hilos y una carga de $400 * 400$

TABLE XX
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 100 X 100

Carga 100 x 100 con 6 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,000686s	0,00059699s	0,0005908533333s
B	0,001019s	0,0012001s	0,0009479666667s
C	0,000990s	0,0011756s	0,001083633333s
D	0,000838s	0,0009352333333s	0,001560833333s

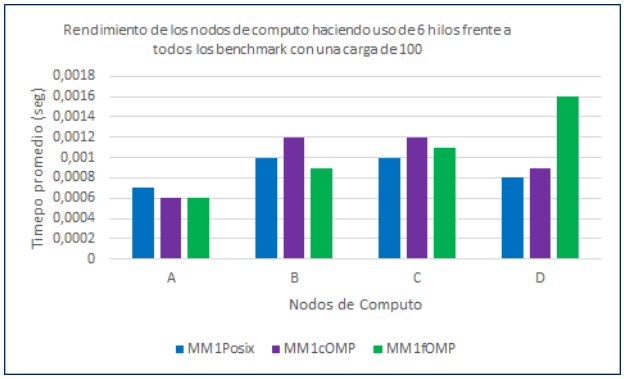

Fig. 21. Grafica del rendimiento de los nodos con seis hilos y una carga de $100 * 100$

TABLE XXII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 800 X 800

Carga 800 x 800 con 6 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,55862s	0,4985942433s	0,2904454s
B	0,71978s	0,5335586s	0,4602528667s
C	0,64465s	0,5270297667s	0,4529510667s
D	0,63718s	0,7677553667s	0,5291805333s

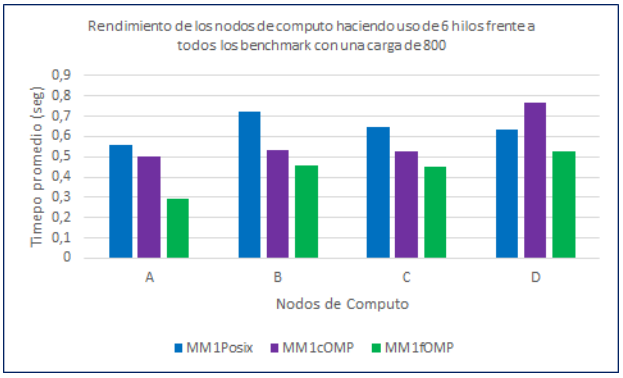

Fig. 23. Grafica del rendimiento de los nodos con seis hilos y una carga de $800 * 800$

TABLE XXI
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 400 X 400

Carga 400 x 400 con 6 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,04520s	0,04063608s	0,03661126s
B	0,06870s	0,0700123s	0,0449306s
C	0,06232s	0,06998336667s	0,06694783333s
D	0,05164s	0,1084215667s	0,08913546667s

TABLE XXIII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 1600 X 1600

Carga 1600 x 1600 con 6 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	5,167s	4,372s	2,3168695s
B	7,194s	5,802s	3,829860133s
C	6,884s	5,523s	3,646469633s
D	5,979s	5,773s	4,1528633s

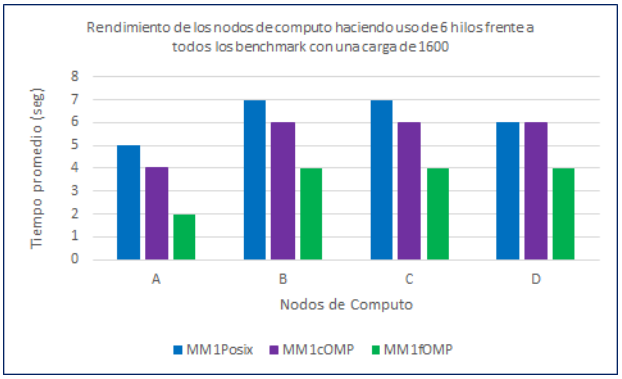

Fig. 24. Grafica del rendimiento de los nodos con seis hilos y una carga de $1600 * 1600$

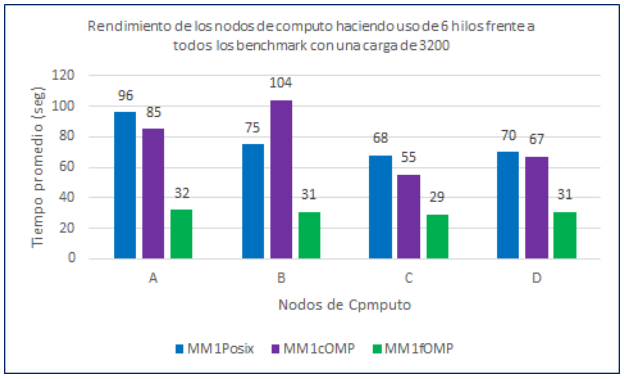

Fig. 26. Grafica del rendimiento de los nodos con seis hilos y una carga de $3200 * 3200$

TABLE XXIV
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 2400 X 2400

Carga 2400 x 2400 con 6 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	25,344s	21,75653483s	13,822099273s
B	25,068s	20,68764427s	12,98824633s
C	23,619s	19,24823777s	12,2720631s
D	22,651s	21,95089377s	13,4031741s

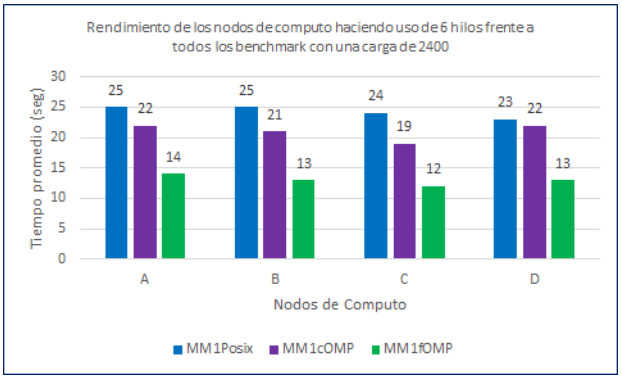

Fig. 25. Grafica del rendimiento de los nodos con seis hilos y una carga de $2400 * 2400$

TABLE XXV
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 3200 X 3200

Carga 3200 x 3200 con 6 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	95,587s	80,99830987s	32,54161803s
B	75,334s	103,9275648s	30,7094659s
C	67,726s	54,9438721s	29,20911707s
D	69,922s	66,9732975s	30,8200497s

TABLE XXVI
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 100 X 100

Carga 100 x 100 con 8 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,00514s	0,004477666667s	0,000044316s
B	0,001236s	0,001045266667s	0,0008093666667s
C	0,000951s	0,001266366667s	0,001044366667s
D	0,000998s	0,0191239s	0,02889046667s

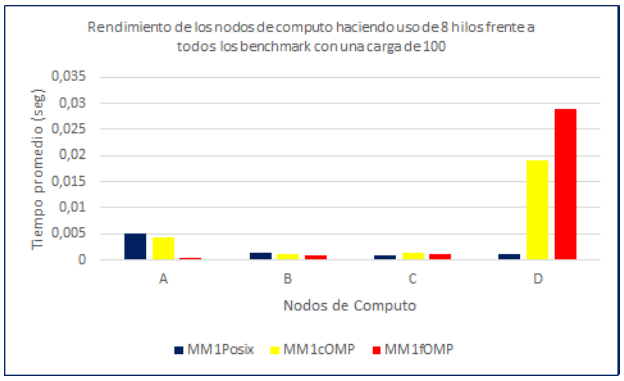

Fig. 27. Grafica del rendimiento de los nodos con ocho hilos y una carga de $100 * 100$

TABLE XXVII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 400 X 400

Carga 400 x 400 con 8 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,10390s	0,0927706667s	0,12745845s
B	0,06171s	0,06395526667s	0,03882063333s
C	0,05300s	0,06193533333s	0,06100956667s
D	0,05312s	0,0985988s	0,08615483333s

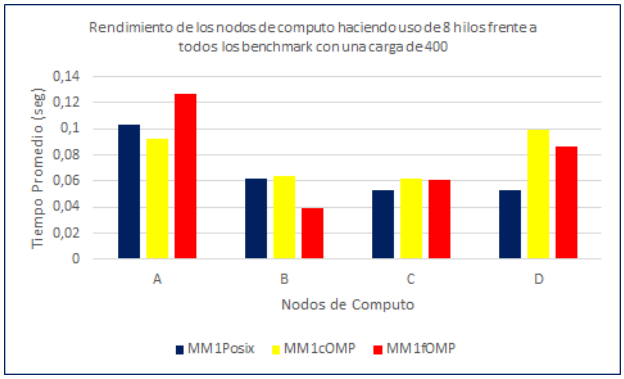

Fig. 28. Grafica del rendimiento de los nodos con ocho hilos y una carga de $400 * 400$

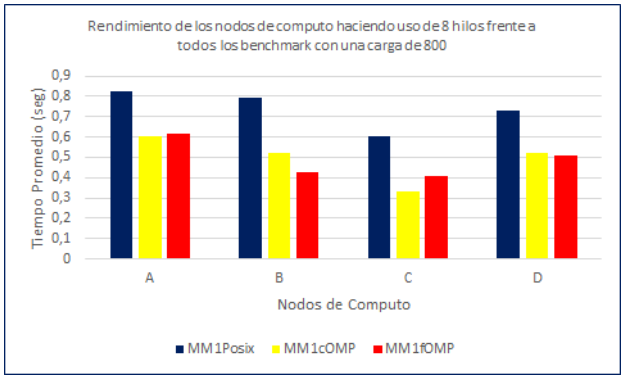

Fig. 29. Grafica del rendimiento de los nodos con ocho hilos y una carga de $800 * 800$

TABLE XXVIII
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 800 X 800

Carga 800 x 800 con 8 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	0,82896s	0,60394569s	0,61783406s
B	0,79377s	0,5204860667s	0,4236948667s
C	0,60396s	0,3290361s	0,4068723333s
D	0,73247s	0,5236866s	0,5061841333s

TABLE XXIX
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 1600 X 1600

Carga 1600 x 1600 con 8 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	7,875s	7,279s	8,13765213s
B	6,407s	5,211s	3,544689467s
C	6,039s	4,923s	3,3483502s
D	7,553s	7,017s	3,7693822s

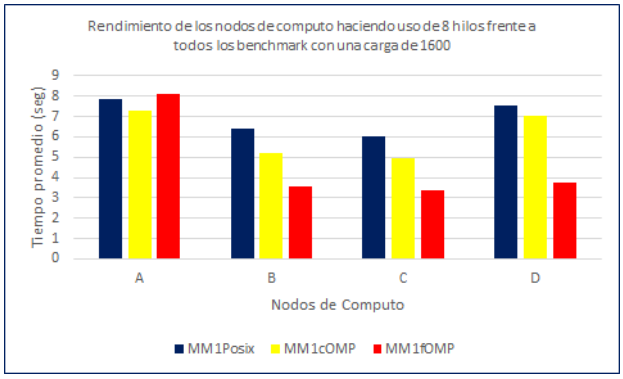

Fig. 30. Grafica del rendimiento de los nodos con ocho hilos y una carga de $1600 * 1600$

TABLE XXX
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 2400 X 2400

Carga 2400 x 2400 con 8 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	28,758s	26,817401s	13,866574467s
B	21,861s	18,0816576s	12,053077s
C	20,334s	17,08928837s	11,31127227s
D	26,551s	25,84215233s	12,21616307s

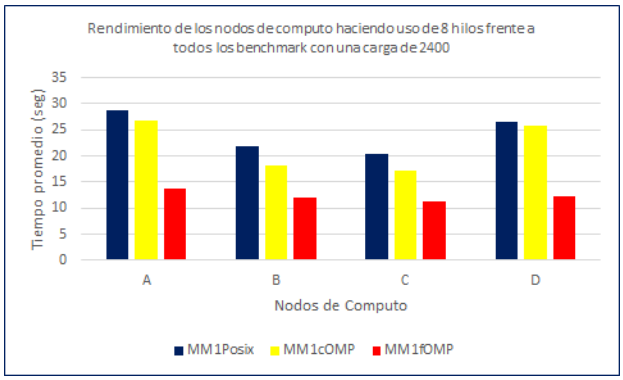

Fig. 31. Grafica del rendimiento de los nodos con ocho hilos y una carga de $2400 * 2400$

TABLE XXXI
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 3200 X 3200

Carga 3200 x 3200 con 8 Hilos			
Nodo	Bench MM1Posix	Bench MM1cOMP	Bench MM1fOMP
A	75,690s	90,74873237s	30,90621337s
B	57,392s	92,38677687s	28,6014987s
C	52,416s	44,1600961s	26,86650763s
D	73,950s	70,42309123s	28,7554583s

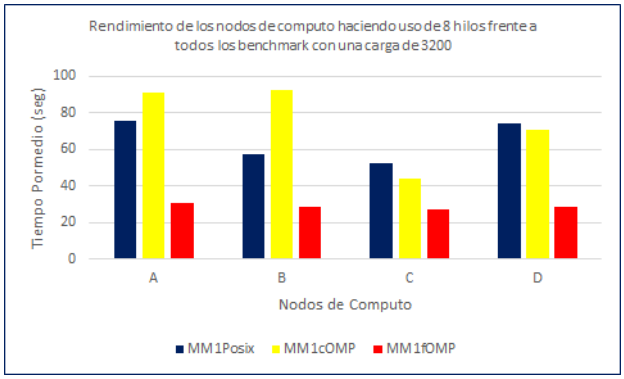

Fig. 32. Grafica del rendimiento de los nodos con ocho hilos y una carga de $3200 * 3200$

Dado lo anterior y según especificaciones de la tabla II el nodo de cómputo **C** presenta el mejor rendimiento en el cálculo matricial, lo anterior, dada su arquitectura de cómputo que presenta una superioridad ajustada tomando como referencia los demás nodos de cómputo utilizados.

Como último, se presenta el rendimiento de cada benchmark en cada nodo de cómputo de acuerdo a la estructura de programación para cada uno y el número de hilos presentes en cada ejecución:

TABLE XXXII
RENDIMIENTO DE LOS NODOS PARA EL BENCHMARK MM1POSIX

Promedio tiempo de ejecución (seg) para Benchmark en MM1Posix					
Nodos	1 Hilo	2 Hilos	4 Hilos	6 Hilos	8 Hilos
A	574	298	264	96	76
B	189	109	83	75	57
C	171	94	76	68	52
D	240	114	70	70	74

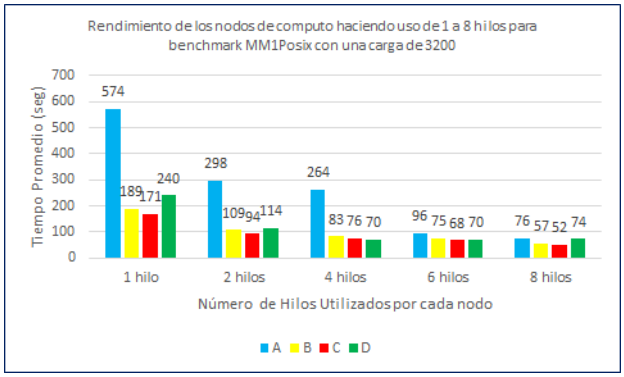

Fig. 33. Gráfica del Rendimiento de los nodos haciendo uso de 1 a 8 hilos para el Benchmark MM1Posix con una carga de 3200

TABLE XXXIII
RENDIMIENTO DE LOS NODOS PARA EL BENCHMARK MM1FOMP

Promedio tiempo de ejecución (seg) para Benchmark en MM1fOMP					
Nodos	1 Hilo	2 Hilos	4 Hilos	6 Hilos	8 Hilos
A	111	60	51	32	31
B	87	55	37	31	29
C	83	51	35	29	27
D	105	57	33	31	29

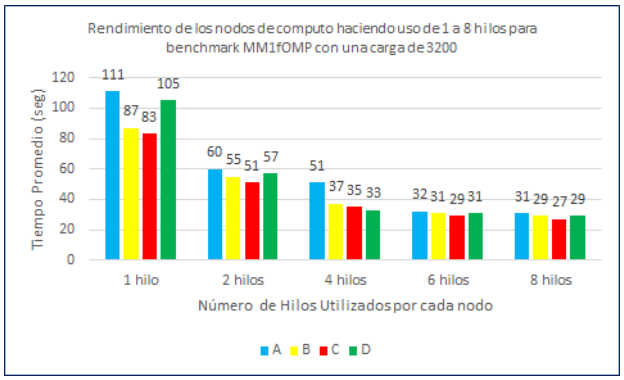

Fig. 34. Gráfica del Rendimiento de los nodos haciendo uso de 1 a 8 hilos para el Benchmark MM1fOMP con una carga de 3200

TABLE XXXIV
RENDIMIENTO DE LOS NODOS PARA EL BENCHMARK MMCOMP

Promedio tiempo de ejecución (seg) para Benchmark en MMcOMP					
Nodos	1 Hilo	2 Hilos	4 Hilos	6 Hilos	8 Hilos
A	510	254	209	85	91
B	255	161	122	104	92
C	138	75	61	55	44
D	225	108	67	67	70

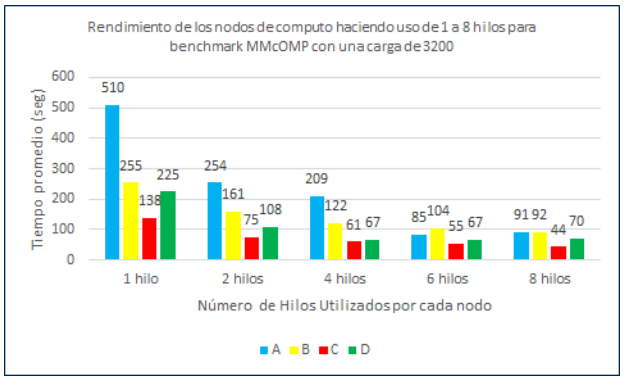

Fig. 35. Gráfica del Rendimiento de los nodos haciendo uso de 1 a 8 hilos para el Benchmark MMcOMP con una carga de 3200

VII. DISCUSIÓN

En los resultados de la primera comparativa del rendimiento de los nodos frente a diferentes cargas, *el nodo de cómputo* ***C*** obtuvo el mejor resultado, ya que sus tiempos promedio son menores en todas las cargas y benchamarks en comparación de los otros tres. A modo de ranking y observando los tiempos

de desempeño, los nodos quedan organizados de la siguiente forma:

- Nodo C.
- Nodo D.
- Nodo B.
- Nodo A.

Por otra parte, de acuerdo a las comparativas realizadas usando la carga de 3200 por Benchmark, se evidenció que el codificado con lenguaje c por referencia o paso por referencia y usando la interfaz de programación OpenMP , es el más eficiente, dados los tiempos de ejecución en el cálculo matricial.

Como también se puede observar en las gráficas desempeño, el poder ejecutar la programación paralela a través de subprocesos que dividen la función, optimiza en gran medida el tiempo de ejecución y permite un mayor rendimiento en la tarea.

Por último, se evidencia que para los cuatro nodos de cómputos involucrados, el tiempo de ejecución del producto de matrices cuadradas de dimensiones comprendidas de 100 a 400 no son significativos, puesto que la divergencia promedio de cada carga es de aproximadamente 0.2 segundos

VIII. Conclusiones

- Al terminar la comparativa, el componente que le permite al nodo C tener un mejor rendimiento en todas las pruebas de rendimiento, es su procesador Intel core i7 de onceava generación, ya que cuenta con una velocidad de procesamiento de 3 GHz, la más alta de los entre los 4 nodos, y un número mayor de hilos físicos y lógicos.

- Los lenguajes de programación utilizados, juegan un papel fundamental a la hora de poner a prueba el rendimiento de un nodo de cómputo específico, pero, evidentemente, la lógica utilizada en la programación de los benchmark provoco que incluso en un mismo lenguaje que compila y ejecuta la misma operación aritmética sea más eficiente que otra como en el caso de utilizar la interfaz Open Mp en el benchamrk programado con c++ por referencia.

- Al momento de hablar del rendimiento de un nodo de cómputo hay que fijarse en la cantidad de hilos lógicos que poseen los dispositivos, pero la velocidad de procesamiento gana más importancia, ya que si esta es baja evita el aprovechamiento del máximo rendimiento del procesador. Finalmente, cabe destacar que, aunque la capacidad de memoria con la que cuenta un nodo de cómputo es importante, para tareas similares a las que ejecutaron los benchmarks, debe primar la velocidad de procesamiento que posee el procesador sobre la capacidad de la memoria que posee el nodo de computo.

IX. BIOGRAFÍA

Óscar Julián Reyes Torres nació el 29 de junio de 1999 Duitama, Boyacá, Colombia. Es estudiante de noveno semestre de Ingeniería de Sistemas y Telecomunicaciones en la universidad Sergio Arboleda. Ha trabajado en la Rama Judicial brindando soporte y capacitación en el nuevo software de reparto en la Oficina Judicial de Tunja.
Porcentaje de Contribución:100%

References

[1] Catellanos J. 2020 ”ESTUDIO COMPARATIVO DEL SOPORTE MULTIHILO” Departamento de Computacion Universidad de Carabobo. PDF

[2] Sanchez Luis 2012. ”Diseño y evaluación de un complemento para refactorización paralela de código C usando OpenMP” Universidad Carlos III de Madrid. Departamento de Informática. PDF

[3] García Bertila 2021 ”Diseño de software para el uso de estructuras dinámicas con lenguaje de programación C++”. Universidas Nacional del Callao. PDF

[4] Hoeger H. 2020 ”Introducción a la Computación Paralela”. Centro Nacional de Cálculo Científico Universidad de Los Andes PDF