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

1st Andrés Camilo López Ramírez
Universidad Sergio Arboleda.
Bogotá, Colombia
andres.lopez01@correo.usa.edu.co

2nd Oscar Julian Reyes Torres
Universidad Sergio Arboleda.
Bogotá, Colombia
oscar.reyes01@correo.usa.edu.co

3rd Luis Felipe Velasquez Puentes
Universidad Sergio Arboleda.
Bogotá, Colombia
luis.velasquez01@correo.usa.edu.co

***Abstract*—Due to the variety of hardware configurations that a compute node can have, we want to know which configuration performs the best against high-throughput tasks. For this, it is proposed to make a comparison between four computing nodes that have different configurations, through three performance tests, one developed on C and the other two on C++. To achieve an ideal test environment, each compute node is evaluated under isolated conditions it means that the node only executed the benchmarks and the priority tasks, in addition, each benchmark is executed in a battery of 31 executions per load, these loads being square arrays of size 100, 300, 600, 1000 and 2000. The results, that is, the time it took for each node to finish each execution, are stored in text files for the respective calculation of the average time. The result of averaging the execution times and the comparation against the different loads used, shows us that the *computing node C* presents the best performance against all the loads in the performance tests, also, that the benchmark programmed in C is the most optimal than those developed in C++ , finally, with the purpose of having a more objective evaluation, the loads of the tests must be greater than 1000, because these loads require a greater number of processing that takes the processor to high stress level.**

***Index Terms*—benchmark, nodo de computo, puntero, c, c++, variable por referencia, matriz,arquitectura de Von Nuemann, procesador de propocito general y dirección de memoria**

I. Objetivo

Comparar de forma exhaustiva diferentes nodos de cómputo, por medio de tres pruebas de rendimiento, realizadas en lenguaje C y las otras dos programadas en 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 evaluar la sobrecarga en la jerarquía de memoria y del procesador.

En los nodos de cómputo, el procesador cumple la función de realizar cálculos aritméticos para el cumplimiento de instrucciones con el objetivo de ejecutar tareas específicas, estas tareas pueden ser iniciar y mantener el sistema operativo, abrir el explorador de archivos o en nuestro caso, realizar una multiplicación de matrices cuadradas. Para efectuar estos cálculos, el proceso requiere de unos datos suministrados por unidades de almacenamiento conocidas cómo memoria, que por cuestiones de velocidad y capacidad implementa el concepto de jerarquía de memoria, tema que se desarrollará más adelante. En pocas palabras, el procesador actúa como una calculadora que opera sobre unos datos suministrados por una memoria con el fin de cumplir tareas específicas, rol que lo convierte en el componente más importante de los nodos de cómputo, ya que ejecuta alrededor de un 90% de las operaciones del dispositivo.

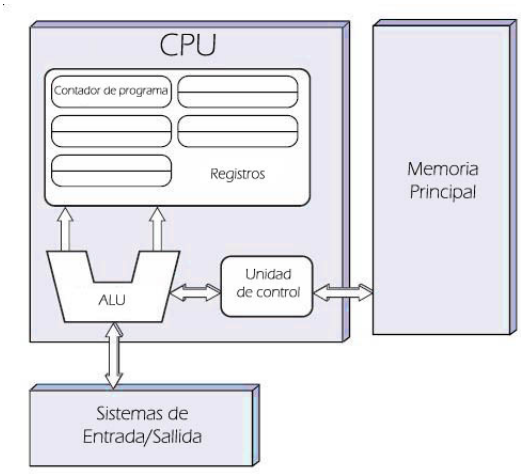

Fig. 1. Representación gráfica de la arquitectura de Von Neumann

En la figura 1 se observan los componentes principales de la unidad central de procesamiento (CPU) y sus dispositivos de entrada y salida. Siguiendo una jerarquía de importancia, la unidad central aritmética lógica (ALU) se encarga de efectuar cálculos aritméticos y lógicos, la unidad de control se encarga de activar o desactivar los diversos componentes, los registros actúa cómo memoria de almacenamiento de instrucciones o datos muy usados por el procesador. Externo al procesador se encuentran los sistemas de entrada y salida que permiten el ingreso y despliegue de los datos y la memoria principal, lugar donde se almacenan los datos e instrucciones que puede

solicitar la CPU. Cabe aclarar, que la comunicación entre los dispositivos de entrada y salida, la memoria y la CPU se da mediante buses de datos, que se pueden entender cómo vías en donde los datos viajan de forma bidireccional.
La desventaja principal de esta propuesta es el bus de datos bidireccional entre la memoria y la CPU porque en este modelo las instrucciones y los datos se deben recuperar de forma secuencial limitando el ancho de banda de los buses de datos, esta congestión de datos se denomina Cuello de botella de Von Neumann.

Para corregir el cuello de botella, se empezó a iterar entre diferentes arquitecturas de procesadores para llegar a un modelo universal eficiente, terminando en un procesador de propósito general compuesto por dos módulos, el primero módulo se denomina "Datapath", se encarga de recibir los datos e instrucciones, posteriormente realiza los respectivos cálculos aritméticos y almacena los datos e instrucciones recurrentes. Simultáneamente, el "Datapath" envía datos de prueba para ser administrado por un controlador mediante vectores de control para que posteriormente el "Datapath" proporcione la información necesaria a los sistemas de salida, todo esto es sincronizado mediante las señales del reloj del sistema que recibe el controlador. En la figura 2 se ve gráficamente el funcionamiento de esta arquitectura de procesador de propósito general.

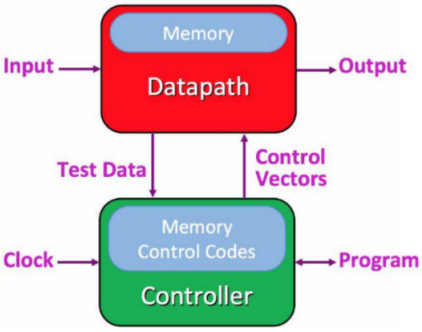

Fig. 2. Representación gráfica del procesador universal de propósito general

Un aspecto importante para hablar de un rendimiento eficaz, es la jerarquía de memoria, la cual tiene el objetivo de diseñar un modelo de tal manera que, en el mismo instante de tiempo, sea capaz de almacenar grandes cantidades de datos, procesamiento y velocidades altas y que sea accesible en cuanto a costos y beneficio. Para que esto se cumpla, la memoria se divide en múltiples niveles, pasando por el disco duro, memoria RAM, memoria Cache (L1,L2,L3) y los registros del procesador. El rendimiento de este modelo se mide en términos de capacidad y velocidad, por ejemplo si el nivel es alto, representa una velocidad alta y una capacidad baja, en contraparte si esta en un nivel bajo, significaría una velocidad baja y una capacidad alta, esto se debe a la cercanía de los componentes con la CPU.

Actualmente, es común que los controladores de memoria se alojen en la memoria RAM, pero esto no fue así siempre. Hubo un tiempo en el que se implementaron en el **North Bridge**, solía ser el que mayor banda ancha tenía, evitando el tan famoso cuello de botella entre el controlador y el procesador. La integración de estos controladores surge por primera vez en los ordenadores de escritorio, un poco después del año 2000, con la llegada de los procesadores AMD ATHLON 64. Sin embargo, Intel siguió por más de 5 años implantando los controladores de memoria por fuera del procesador hasta la llegada del Intel Nehalem, en el año 2008.

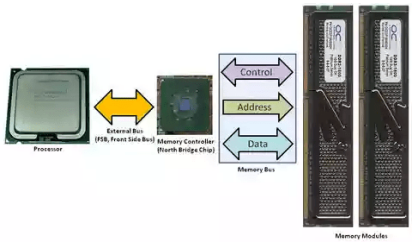

Fig. 3. Funcionamiento de controlador de memoria Intel

Como todo proceso de intervención, existen las ventajas y desventajas acerca de la instalación de los controladores de memoria en el procesador. Para empezar, la latencia de acceso de memoria RAM se reduce de manera considerable; lo cual define que ni el procesador ni el North Bridge tienen que esperar al otro la información que se envíen, obtenida por la memoria.

Por otra parte, la latencia se define como el retraso entre el procesador que emite una petición de memoria y el elemento que llega realmente, medida en nanosegundos o periodos de reloj. Donde encontramos diferentes latencias en las relaciones de transferencia de memoria a la caché, la caché al registro o cualquier latencia presente entre cualquier memoria y el procesador.

Para esta comparativa de rendimiento, se hizo uso de benchmark o aplicación de juguete, los cuales son programas, para este caso, aplicaciones programadas en lenguaje c y c++ que ponen a prueba el rendimiento resultado de la compilación y ejecución de estos en cada nodo de cómputo. Los benchmark elaborados, adicional a la solución de multiplicación de matrices cuadradas clásica, devuelven el tiempo que tardó cada nodo de cómputo en realizar la operación aritmética, siendo esta la métrica de desempeño, lo anterior, con el objetivo de comprobar y evaluar el rendimiento de cada nodo de cómputo involucrado.

Por último, es importante definir la multiplicación de matrices, entendida como un área de las matemáticas en la que el producto de matrices viene dado por una operación efectuada entre dos matrices o entre una matriz y un escalar, generando una unificación de matrices a una sola, lo anterior, se logra mediante la multiplicación y suma de los elementos de las

filas y las columnas de las matrices origen teniendo en cuenta el orden de los factores.
La dimensión de la matriz resultado es la combinación de la dimensión de las matrices. En otras palabras, la dimensión de la matriz resultado serán las columnas de la primera matriz y las filas de la segunda matriz.

Para este caso se van a tener dos matrices cuadradas **n.n**, las cuales, siguiendo la multiplicación clásica de matrices se empezará multiplicando la primera fila de una matriz **A** con la primera columna de una matriz **B**. Despues se procede de igual forma con la columna y la fila siguiente de esta forma:

$$A_{3x3} = \begin{pmatrix} 1 & 6 & 0 \\ -1 & 3 & 1 \\ 4 & 6 & 2 \end{pmatrix} \cdot B_{3x3} = \begin{pmatrix} 0 & 4 & 0 \\ 2 & 3 & 1 \\ 1 & -2 & 1 \end{pmatrix} \quad (1)$$

Empezamos multiplicando la primera fila de la matriz A con la primera columna de la matriz B repitiendo el procedimiento hasta completar la matriz resultado:

$$AB_{3x3} = \begin{pmatrix} 12 & 22 & 6 \\ 7 & 3 & 4 \\ 14 & 30 & 8 \end{pmatrix} \quad (2)$$

III. Metodología

Para realizar la comparativa del rendimiento de los nodos de cómputo, se desarrollaron tres benchmarks que realizan la multiplicación una matrices aplicando el algoritmo clásico, filas por columnas:

- **Benchmark con C**: Esta escrito en el lenguage C y busca evaluar el desempeño de un aspecto específico de la computadora usando la multiplicación de matrices con el algoritmo clasico (filas x columnas) mediante la apuntamiento de vectores aun cierto espacio de memoria reservado con dimensión NxN
- **Benchmark con C++**:Esta escrito en el lenguage C++ y busca evaluar el rendimiento , mediante la multiplicación de matrices con el algoritmo clasico (filas x columnas)
- **Benchmark reference con C++**: Esta escrito en el lenguage C++ y busca evaluar el desempeño de un aspecto específico de la computadora usando la multiplicación de matrices mediante variables de referencia(&)

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 codigos empleado en esta comparativa, estàn disponibles en el siguiente repositorio click aqui

A. Configuraciones de los Nodos

Para la parte experimental, se utilizaron 4 nodos (A,B,C,D) con el sistema operativo basado en Linux y con diferentes configuraciones, tal como se aprecia en la siguiente tabla:

TABLE I
CONFIGURACIONES DE LOS NODOS A EXPERIMENTAR

Nodo	Nucleos	Ram	Procesador	Velocidad
A	4	8 GB	Intel(R) Core(TM) i5-5200	2.2 GHz
B	4	8 GB	AMD Ryzen 7 3700U	2.3 GHz
C	8	16 GB	Intel(R) Core(TM) i7-1185G7	3 GHz
D	8	12 GB	Intel(R) Core(TM) i7-8550U	1.8 GHz

B. 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 x 100
- 300 x 300
- 600 x 600
- 1000 x 1000
- 2000 x 2000

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.

IV. Resltados

A continuación se presentan los resultados obtenidos, estos se presentan de dos modos. El primero es una comparativa que muestra 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 2000 X 2000

Carga 2000 x 2000			
Nodo	Bench en C	Bench en C++	Bench en C++ por referencia
A	36.91s ± 2.25	252.37s ± 11.44	115.86s ± 5.38
B	28.14s ± 2.33	168.76s ± 1.16	85.14s ± 1.38
C	27.51s ± 0.85	166.76s ± 5.85	70.10s ± 2.25
D	34.96s ± 0.28	129.68s ± 0.65	234.19s ± 1.54

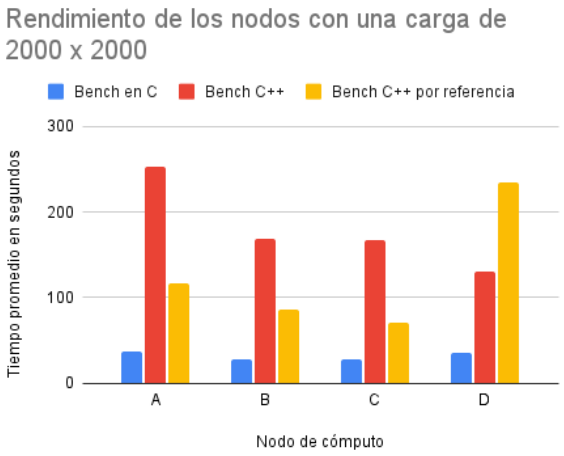

Fig. 4. Grafica del rendimiento de los nodos con una carga de 2000 x 2000

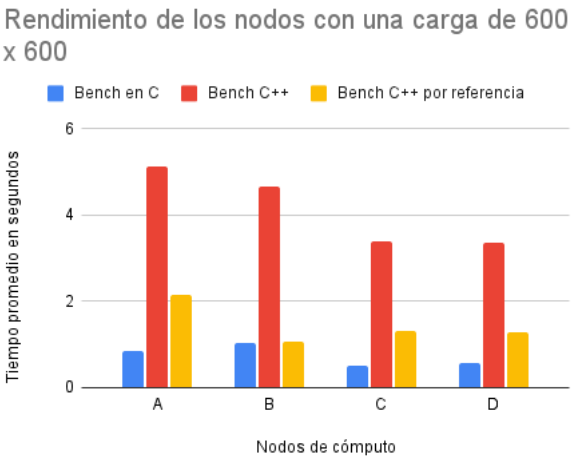

Fig. 6. Grafica del rendimiento de los nodos con una carga de 600 x 600

TABLE III
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 1000 X 1000

Carga 1000 x 1000			
Nodo	Bench en C	Bench en C++	Bench en C++ por referencia
A	3.77s ± 0.26	26s ± 0.32	12.35s ± 0.43
B	4.48s ± 0.62	20.97s ± 2.27	8.47s ± 0.093
C	3.19s ± 0.85	15.59s ± 0.65	6.38s ± 0.75
D	3.08s ± 0.27	7.86s ± 0.5	17.54s ± 0.71

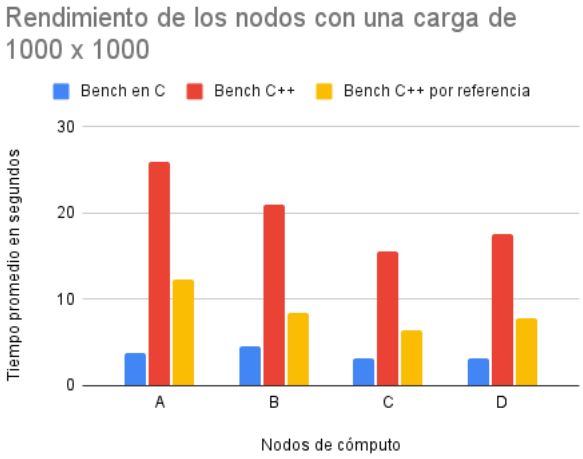

Fig. 5. Grafica del rendimiento de los nodos con una carga de 1000 x 1000

TABLE V
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 300 X 300

Carga 300 x 300			
Nodo	Bench en C	Bench en C++	Bench en C++ por referencia
A	0.11s ± 0.02	0.55s ± 0.005	0.21s ± 0.02
B	0.13s ± 0.002	0.66s ± 0.004	0.129s ± 0.02
C	0.055s ± 0.02	0.36s ± 0.15	0.17s ± 0.042
D	0.073s ± 0.002	0.39s ± 0.003	0.15s ± 0.004

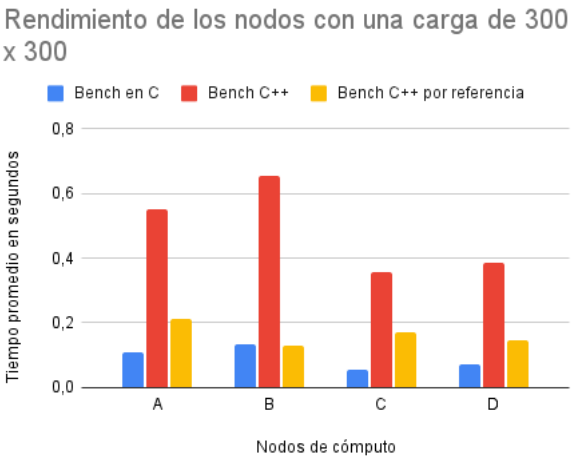

Fig. 7. Grafica del rendimiento de los nodos con una carga de 300 x 300

TABLE IV
RENDIMIENTO DE LOS NODOS CON UNA CARGA DE 600 X 600

Carga 600 x 600			
Nodo	Bench en C	Bench en C++	Bench en C++ por referencia
A	0.85s ± 0.072	5.15 ± 0.072	2.15s ± 0.08
B	1.03s ± 0.002	4.65s ± 0.85	1.05s ± 0.03
C	0.51s ± 0.05	3.4s ± 0.88	1.31s ± 0.42
D	0.57s ± 0.018	3.35s ± 0.13	1.29s ± 0.09

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

Carga 100 x 100			
Nodo	Bench en C	Bench en C++	Bench en C++ por referencia
A	0.004s ± 0.0007	0.021s ± 0.001	0.007s ± 0.001
B	0.007s ± 0.002	0.026s ± 0.003	0.008s ± 0.003
C	0.06s ± 0.003	0.36s ± 0.15	0.17s ± 0.042
D	0.004s ± 0.0004	0.02s ± 0.001	0.005s ± 0.001

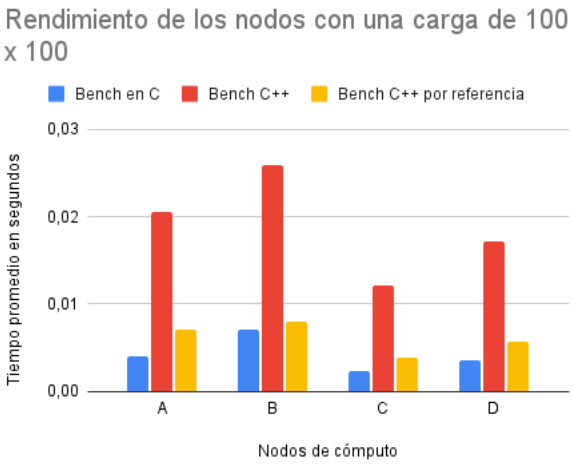

Fig. 8. Grafica del rendimiento de los nodos con una carga de 100 x 100

El segundo modo de comparar los resultados de las pruebas de rendimiento, se enfoca en ver el rendimiento de cada nodo de cómputo sobre un benchmark específico, los resultados de esta, se presentan a continuación:

TABLE VII
RENDIMIENTO DE LOS NODOS PARA EL BENCHMARK EN C

Benchmark en C				
Carga	Nodo A	Nodo B	Nodo C	Nodo D
600	0.85s ± 0.071	1.03s ± 0.002	0.51s ± 0.052	0.57s ± 0.017
1000	3.77s ± 0.258	4.48s ± 0.621	3.19s ± 0.854	3.08s ± 0.269
2000	36.91s ± 2.247	28.14s ± 2.325	27.51s ± 0.851	34.96s ± 0.279

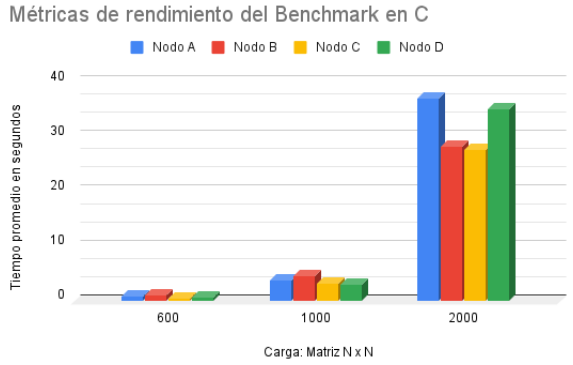

Fig. 9. Gráfica del Rendimiento de los nodos para el Benchmark en C

TABLE VIII
RENDIMIENTO DE LOS NODOS PARA EL BENCHMARK EN C++

Benchmark en C++				
Carga	Nodo A	Nodo B	Nodo C	Nodo D
600	5,14 ± 0,072	4,67 ± 0,846	3,40 ± 0,880	3,35 ± 0,131
1000	26,00 ± 0,319	20,97 ± 2,274	15,59 ± 0,654	17,54 ± 0,711
2000	252,37 ± 11,43	168,76 ± 1,160	166,76 ± 5,846	234,19 ± 1,538

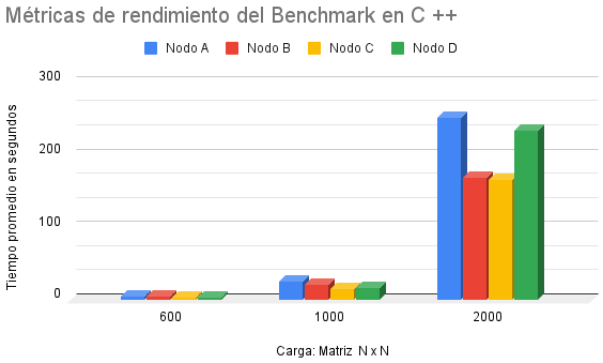

Fig. 10. Grafica del Rendimiento de los nodos para el Benchmark en C++

TABLE IX
RENDIMIENTO DE LOS NODOS PARA EL BENCHMARK EN C++ POR REFERENCIA

Benchmark en C++ por referencia				
Carga	Nodo A	Nodo B	Nodo C	Nodo D
600	5,14 ± 0,072	4,67 ± 0,846	3,40 ± 0,880	3,35 ± 0,131
1000	26,00 ± 0,319	20,97 ± 2,274	15,59 ± 0,654	17,54 ± 0,711
2000	252,37 ± 11,43	168,76 ± 1,160	166,76 ± 5,846	234,19 ± 1,538

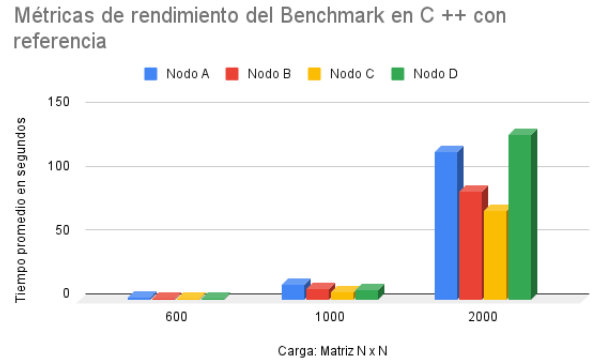

Fig. 11. Gráfica del Rendimiento de los nodos para el Benchmark en C++ por referecia

V. 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 B.
- Nodo D.
- Nodo A.

Por otra parte, de acuerdo a las comparativas realizadas por Benchmark, se evidencio que el codificado con C, es el más eficiente , representando la tercera parte de tiempo que ocupa

el desarrollado con C++ mediante referencias y una décima parte del tiempo de ejecución con respecto a el Benchmark con C++.

Como también se puede observar en las gráficas desempeño, el uso de c++ por referencia o también llamado paso por referencia, fue de más rápida ejecución dado que al utilizar punteros a un elemento específico dentro del vector este demora menos en solucionar el producto, puesto que no va dando saltos de uno en uno, sino de un tamaño específico

A pesar del intervalo significativo de los tiempos obtenidos por cada benchmark, se logra percibir que entre los diferentes códigos se presenta una proporcionalidad dependiente a la carga y de acuerdo a las configuraciones del nodo, por ejemplo, para el nodo D y carga de 1000, tienen una escala similar en cuanto a los datos obtenidos en los benchmark de C y C++

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 600 no son significativos, puesto que la divergencia promedio de cada carga es de aproximadamente 0.3 segundos

VI. 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.
- El tiempo de ejecución de un programa puede depender de la forma en que esté codificado y optimizado, sin embargo un aspecto como lo es el lenguaje con el que se desarrolla puede llegar a afectar este rendimiento, para este caso, se demostró que C fue el lenguaje que obtuvo resultados más eficaces a pesar de tener 2 contradictores del mismo lenguaje
- 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 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, es decir, tareas donde el número de cálculos aritméticos es colosal, debe primar la velocidad de procesamiento que posee el procesador sobre la capacidad de la memoria.

References

[1] Santana C. F. “Arquitectura de Von Neuman“. Chile, Universidad de Concepción. https://www2.udec.cl/ crisantana/von

[2] Corredor J. 2022. “Computación Paralela y Distribuida“ [Diapositivas]. Escuela de Ciencias Exactas e Ingeniería, Universidad Sergio Arboleda. https://drive.google.com/file/d/1ASx-JXqS6Od1RlYA5494WMHlyVfV5PBh/view

[3] Santos C. A. 2019 “Synaptic axons and dendrits of HPC: memory controllers“. Universidas Industrial de Santander. http://wiki.sc3.uis.edu.co/images/2/23/TF9.pdf

VII. BIOGRAFÍAS

Andrés Camilo López Ramírez nació el 10 de enero de 2001, Bogotá D.C, Colombia. Es estudiante de noveno semestre de Ingeniería de Sistemas y Telecomunicaciones en la universidad Sergio Arboleda, cuenta con la certificación del Ministerio de las TIC en habilidades de programación con una intensidad horaria total de 800 horas.
Contribución: Realizó la explicación de la arquitectura de Von Neumann y los procesadores de uso general, también realizo las baterias del nodo de cómputo C y D, además de la presentación y análisis de los mismos. Porcentaje de Contribución:33.33%

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 los nuevos softwares de reparto en la Oficina Judicial de Tunja.
Contribución: Realizó la explicación de conceptos como latencia, North Bridge y multiplicación de matrices clásica, al igual que la realización de las baterías del nodo de cómputo B. Porcentaje de Contribución:33.33%

Luis Felipe Velasquez Puentes nació el 29 de junio de 2001 en Villa de Leyva, Boyacá, Colombia. Es estudiante de noveno semestre de Ingeniería de Sistemas y Telecomunicaciones en la universidad Sergio Arboleda.Ha trabajado en el área de las telecomunicaciones en una empresa ISP (Internet Services Provider) del municipio de Villa de Leyva.
Contribución:Realizó la explicación del funcionamiento de la jerarquía de memoria y la metodología, por otra parte, desarrolló las baterías experimentales del nodo A y D y luego contribuyó en el análisis de los resultados obtenidos. Porcentaje de Contribución:33.33%