UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Explorando Clústeres de Cómputo de Alto Rendimiento (HPC) con Raspberry Pi para Investigación y Desarrollo Asequible

Máximo Andrés Pinta Pacheco

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 07 de diciembre de 2023

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Explorando Clústeres de Cómputo de Alto Rendimiento (HPC) con Raspberry Pi para Investigación y Desarrollo Asequible

Máximo Andrés Pinta Pacheco

Nombre del profesor, Título académico **Ricardo Flores Moyano, Ph.D**

Quito, 07 de diciembre de 2023

© DERECHOS DE AUTOR

Nombres y apellidos: Máximo Andrés Pinta Pacheco

Código: 00211312

Cédula de identidad: 0703985150

Lugar y fecha: Quito, 07 de diciembre de 2023

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

Este trabajo explora un enfoque básico y actualizado para la construcción y aplicación de clústeres de cómputo de alto rendimiento (HPC) utilizando Raspberry Pi. Ante el elevado costo y complejidad de la implementación de un sistema HPC, esta investigación propone una alternativa accesible, con énfasis en la implementación, configuración, y evaluación de un clúster de cuatro Raspberry Pi 4B.

El núcleo de este estudio se centra en la versatilidad provista por los Raspberry Pi en el campo académico e investigativo, brindando la capacidad de emular un sistema HPC a pequeña escala. Se discute la capacidad de realizar tareas con un alto costo computacional de manera más eficiente al distribuirlas en diferentes nodos de cómputo. Se destacan experimentos significativos como la estimación de Pi mediante el método Monte Carlo, multiplicación matricial, y el uso de HPL, un programa con el cual se mide la capacidad de cómputo de las mejores supercomputadoras en el mundo.

Los resultados más destacados muestran la efectividad del clúster, demostrando que el añadir recursos computacionales tiende a disminuir el tiempo de trabajo del clúster. Además, se identificaron áreas en las que este comportamiento puede variar. También se sugieren áreas de mejora y líneas de investigación futura, que incluyen, pero no se limitan a la optimización de algoritmos y cambios en configuraciones de hardware y software.

Palabras clave: Raspberry Pi, Clúster de Cómputo, Alto Rendimiento, Supercomputadores, Computación Paralela, MPI, Python, HPL

ABSTRACT

This work explores a basic and updated approach to the construction and application of high-performance computing (HPC) clusters using Raspberry Pi. Given the high cost and complexity of implementing an HPC system, this research proposes an accessible alternative, with an emphasis on the implementation, configuration, and evaluation of a cluster of four Raspberry Pi 4B.

The core of this study focuses on the versatility provided by the Raspberry Pi in the academic and research fields, offering the ability to emulate an HPC system on a small scale. The capacity to perform tasks with a high computational cost more efficiently by distributing them across different computing nodes is discussed. Significant experiments are highlighted, such as the estimation of Pi using the Monte Carlo method, matrix multiplication, and the use of HPL, a program that measures the computing capacity of the world's best supercomputers.

The most notable results show the effectiveness of the cluster, demonstrating that adding computational resources tends to reduce the cluster's working time. Additionally, contexts were identified where this behavior may vary. Areas for improvement and future research lines are also suggested, which include, but are not limited to, algorithm optimization and changes in hardware and software configurations.

Keywords: Raspberry Pi, Computing Cluster, High Performance, Supercomputers, Parallel Computing, MPI, Python, HPL.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El significado del término cómputo de alto rendimiento o HPC por sus siglas en inglés (High Performance Computing), cambia conforme el desarrollo tecnológico avanza. Solo en términos de procesamiento, una computadora considerada básica para oficina poseerá mucha más capacidad que una que se vendía como de alto desempeño hace una década. De esta forma, se puede decir que los ordenadores que poseemos en la actualidad tienen un grado de cómputo de alto rendimiento (Dowd & Severance, 2010).

Es importante comprender el contexto histórico en el cual se han desarrollado los HPC. Durante la década de los 60 es cuando empezaron a emerger los primeros supercomputadores. Estos sistemas eran grandes máquinas extremadamente costosas y complejas diseñadas exclusivamente para aplicaciones gubernamentales y académicas de alto nivel (Severance, 2020). Durante este punto del desarrollo tecnológico, la capacidad de procesamiento de uno de estos sistemas era impresionante en comparación a las computadoras que existían para el consumidor. Con la llegada de los años 70, aparecieron las primeras supercomputadoras disponibles de manera comercial. A pesar de esto, todavía el acceso a estos sistemas estaba restringido a grandes compañías o entidades debido al exorbitante costo que representaba adquirir una supercomputadora (Sterling et al., 2018).

Para finales de los años 80, sin embargo, la industria necesitaría innovar. Aquellos microprocesadores que habían sido inventados en los 70s llegarían a poseer la capacidad necesaria para poder ser utilizados en sistemas de cómputo de alto rendimiento (Severance, 2020). Esto representaba un posible problema para las grandes compañías de supercomputadoras y se vio reflejado en un artículo escrito por John Markoff para el New York Times. En este artículo con el título de “The Attack of the ‘Killer Micros’”, Markoff habla de cómo los avances en microprocesadores, brindándoles un bajo costo y un creciente

rendimiento, los hacen una opción más atractiva a adquirir un supercomputador (Markoff, 1991).

Según Chuck Severance de la Universidad de Michigan, para la época, una supercomputadora y 400 computadoras personales conectadas entre sí, tendrían el mismo rendimiento por menos de la mitad del costo (Severance, 2020). Es en este punto cuando tratar sobre HPC o supercomputadoras se relaciona inequívocamente con clústeres de cómputo. Al hablar de estos equipos, se entiende a estos sistemas como varias computadoras de bajo costo, o microprocesadores, trabajando en paralelo para lograr la misma tarea (Dowd & Severance, 2010). Este es el concepto del multiprocesamiento simétrico (SMP), sin embargo, siempre se ve alcanzado por la innovación en la arquitectura y creación de microprocesadores. Donde en un HPC cada nodo de procesamiento contaba con un solo CPU, eventualmente cada nodo se convirtió en una unidad multiprocesador con mucho más poder que antes (Sterling et al., 2018).

En búsqueda de más poder de procesamiento, los nodos dejaron de ser computadoras conectadas entre sí y se convirtieron en sistemas embebidos donde los CPU poseen una comunicación más directa y por ende el procesamiento se vuelve más eficiente y rápido. Sin embargo, esta especialización implica costos más elevados de inversión (Sterling et al., 2018).

La gran mayoría de sistemas HPC están basados en computadores con conjunto reducido de instrucciones o procesadores RISC por sus siglas en inglés. Estos procesadores están diseñados con la capacidad para poder ser insertados en un sistema multiprocesador con el propósito de incrementar la eficiencia del sistema (Severance, 2020). La implementación de dichos sistemas es compleja y necesita un entendimiento especializado; es por esto por lo que hoy en día existen muchas soluciones HPC disponibles para el usuario en el mercado.

Desde grandes empresas como Amazon y Microsoft hasta universidades y otras compañías que han incursionado en este campo.

Este proyecto busca emular el funcionamiento de un HPC utilizando Raspberry Pi. Estos dispositivos son microcomputadores que tienen una gran capacidad de procesamiento a bajo costo. La versatilidad de estos dispositivos es amplia, a pesar de su reducido tamaño y recursos (Upton & Halfacree, 2017). Desde su lanzamiento en 2012, la Raspberry Pi se ha convertido en una herramienta muy útil en educación, ciencia e ingeniería. Con aplicaciones en prototipado de dispositivos, análisis de datos, robótica, electrónica e informática, este dispositivo se ha vuelto extremadamente popular y por ende ha sido muy bien estudiado. Para llevar a cabo la implementación de un HPC con Raspberry Pi, se va a utilizar Python como lenguaje de programación principal. Para permitir la comunicación entre los microprocesadores, estos se conectan a través de una red de alta velocidad. Estas redes pueden ir desde una red de área local (LAN) hasta estructuras de red más complejas como redes de interconexión de baja latencia. Mientras más especializada es la red, mejora la escalabilidad del sistema HPC (Severance, 2020).

Con la infraestructura base, existen dos formas principales de implementar la comunicación entre cada nodo de cómputo. El uso de software que se encargue de transmitir las instrucciones a cada microprocesador se conoce como pasar mensajes y se puede implementar mediante una máquina virtual paralela (PVM) o una interfaz de paso de mensajes (MPI) (Sterling et al., 2018). También se puede implementar un sistema de acceso a memoria escalable no uniforme (NUMA) (Severance, 2020).

ESTADO DEL ARTE

Raspberry Pi ha sido una plataforma muy versátil para trabajar con prototipado e implementaciones reales de HPC. En su artículo "Supercomputing with Raspberry Pi", publicado en la edición 41 de la revista *HackSpace*, Andrew Gregory describe algunos de los proyectos más significativos que han utilizado clústeres de Raspberry Pi para crear sistemas HPC. Las implementaciones han variado en número de nodos utilizados, tipo de enfriamiento, y diferentes aplicaciones más allá del desarrollo como experimentación con HPC (Gregory. 2021). Este artículo sirvió como punto de partida al momento de analizar los requerimientos de un clúster de Raspberry Pi. Lo primero que se hace evidente es que muchas de estos proyectos se han llevado a cabo sobre Raspberry Pi 3, que, si bien en principio solo implica un menor desempeño, también representa un cambio en la implementación de MPI utilizada.

MPI es un estándar de comunicación que se emplea en sistemas de cómputo paralelo y distribuido, especialmente en el desarrollo de HPC, con el fin de facilitar la comunicación entre múltiples procesadores. MPI brinda herramientas que le permiten a los procesos la capacidad de comunicarse entre sí, esta comunicación puede ser para compartir información o sincronizarse entre sí, ya sea entre dos procesos o más. Este estándar resulta de mucha utilidad ya que evita que el programador tenga que hacer su propia implementación de sincronización y control de flujo de procesos, lo cual es una tarea muy compleja y puede devenir en condiciones de carrera si no se implementan de forma adecuada (Dennis, 2013).

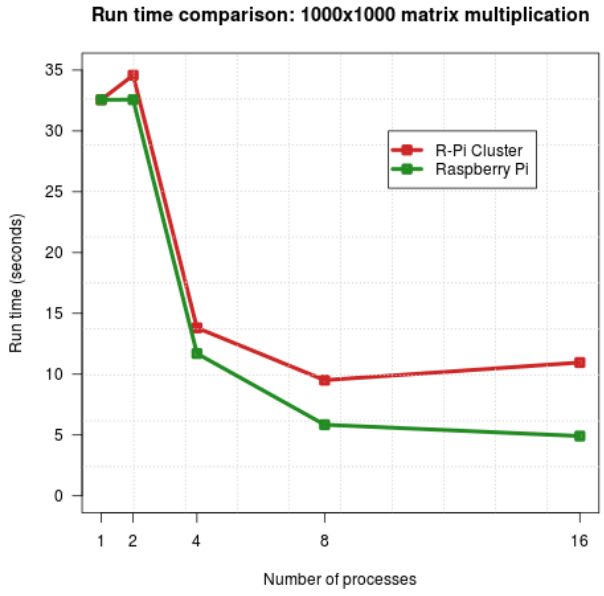
Al ser una interfaz, MPI puede tener muchas implementaciones, para este proyecto se estudiaron dos de las más populares, OpenMPI y MPICH. MPICH es una de las implementaciones más antiguas y establecidas, con un enfoque más orientado hacia la investigación y desarrollo. A pesar de tener una complejidad más alta al momento de

configurar e instalar, se considera una base sólida para investigación y desarrollo de nuevas características para MPI. Debido a esto se considera la mejor opción en entornos académicos con usuarios con experiencia en el manejo de MPI. Por otra parte, OpenMPI busca ser la opción más flexible y fácil de utilizar al proporcionar un proceso de instalación y configuración más simplificado. Se lo conoce por su alta compatibilidad con varios sistemas y arquitecturas ya que, al ser de fuente abierta, está en constante revisión y actualización (Dennis, 2013).

Para el alcance de este proyecto, se implementará la infraestructura de un sistema HPC sobre OpenMPI, con el objetivo de ahorrar en tiempo de desarrollo. Para esto, se utilizará la librería externa MPI para Python (MPI4PY) (Pajankar, 2017), herramienta que brinda las funciones necesarias a Python para poder trabajar sobre múltiples procesadores ya sea en un sistema con un procesador con múltiples núcleos, clústeres de microprocesadores o supercomputadoras (Dalcin, 2023).

En una publicación realizada por Russel Barnes en *The MagPi,* a través de una entrevista, Barnes muestra los resultados de una prueba de rendimiento realizada a un clúster construido por Rigoberto Moreno de la Universidad de Sonoma (Barnes, 2017).

Figura 1. Tiempos de ejecución por Moreno en clúster de Raspberry Pi 3

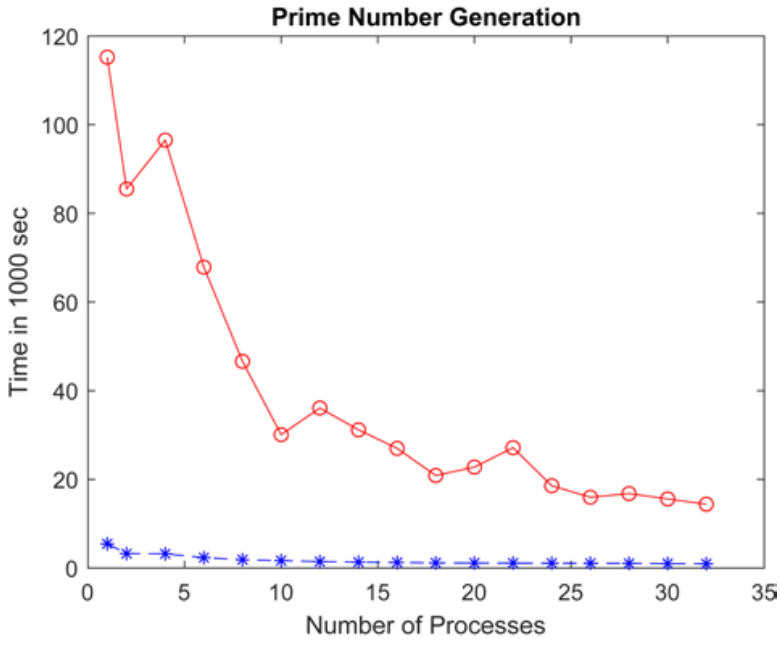


Nota. El gráfico muestra los resultados obtenidos por Rigoberto Moreno en un clúster de cuatro Raspberry Pi 3. Tomado de *Run-time comparison*, por Russel Barnes, 2017, The MagPi (https://magpi.raspberrypi.com/articles/benchmarking-raspberry-pi-cluster)

Como prueba de rendimiento Moreno implementó la distribución del cálculo de la multiplicación de dos matrices cuadradas de 1000 x 1000. Sin embargo, como se puede observar en la figura 1, solo se realizó la prueba a números de procesadores en potencias de dos. Otro hecho que llama la atención es el incremento en tiempo de ejecución al pasar de uno a dos procesadores.

De forma similar, en un estudio realizado en 2019 por Wazir, et al., al realizar un análisis comparativo de eficiencia entre MPICH, una implementación de MPI en C++, y MPI4PY los resultados de sus pruebas utilizando generación de números primos muestran una curva con puntos cúspides visibles en la figura 2.

Figura 2. Tiempos de ejecución por Wazir, et al., en clúster de Raspberry Pi 3



Nota. El gráfico muestra los resultados obtenidos por Wazir, et al., en un clúster de ocho Raspberry Pi 3. En rojo se encuentra el rendimiento obtenido al ejecutar un programa escrito en Python, mientras que en azul se observa el tiempo de ejecución de un programa

equivalente desarrollado en C++. Tomado de *Prime No with MPI4py*, por Wazir, et al., 2019, ResearchGate (https://www.researchgate.net/publication/337184208_Performance_Comparison_of_MPICH_and_MPI4py_on_Raspberry_Pi-3B_Beowulf_Cluster). CC BY 4.0

En la figura 2 se puede observar la extrema diferencia en rendimiento entre un programa realizado en C++, utilizando MPICH (en azul), y Python con MPI4PY (en rojo). Esta comparación expone claramente la influencia que tienen en el rendimiento el lenguaje de programación y la implementación de MPI utilizados.

De manera similar a los resultados vistos en la figura 1, los resultados en la figura 2 muestran un patrón inesperado en el rendimiento. Se espera que con el hecho de agregar más recursos de cómputo los tiempos de ejecución se encuentren en constante decrecimiento, sin embargo, lo observado es que este no siempre es el caso. Ya que este no era el enfoque en ninguna de las anteriores publicaciones, no se discute la razón detrás de este tipo de resultado.

La última prueba para realizarse será la High-Performance LINKPACK, o HPL por sus siglas en inglés. Este es un set de pruebas de rendimiento utilizado por Top 500 List, organización que categoriza las mejores supercomputadoras en el mundo, basándose en cuantas operaciones de punto flotante pueden realizar por segundo (FLOPS). Esta prueba consiste en la resolución de un sistema de ecuaciones lineales densas, y al ser autocontenida, luego de un breve proceso de instalación y configuración, está lista para ser ejecutada (Barnes, 2017).

Para finalizar, se optó por una LAN para la comunicación entre procesadores. Ya que la disponibilidad de switches con capacidad Gigabit Ethernet en la actualidad, provee una alternativa atractiva y de bajo costo que incrementa la eficiencia del clúster en comparación a las redes Fast Ethernet implementadas en las publicaciones antes mencionadas.

PROPUESTA

Este proyecto tiene como objetivo principal la implementación de un clúster de cómputo de alto rendimiento (HPC) utilizando Raspberry Pi como nodos de procesamiento. La motivación radica en generar una solución asequible y eficiente frente a la necesidad de recursos de cómputo de alto rendimiento para investigación y desarrollo. El proyecto se dividirá en fases que incluyen la configuración de hardware, software, pruebas de rendimiento, análisis comparativo de costos y documentación de mejores prácticas.

Se espera que los resultados demuestren la viabilidad de esta alternativa y contribuyan al conocimiento de las ventajas y desafíos que presenta el utilizar Raspberry Pi en aplicaciones de HPC. De esta forma también se busca beneficiar a la parte académica, investigadores, y pequeñas empresas que necesiten soluciones de cómputo de alto rendimiento a baja escala y de manera asequible.

PROTOTIPADO

La fase de prototipado se encuentra dividida en cinco partes: configuración inicial de Raspberry Pi, pruebas de configuración, generación de imagen y respaldo, ensamblaje del clúster, y pruebas de comunicación.

1. Configuración inicial

Se selecciona uno de los Raspberry Pi para realizar la instalación inicial del sistema operativo, este dispositivo será el punto de partida y modelo para las siguientes etapas. El procedimiento fue el siguiente:

1. Se descargó el software necesario para el proyecto, **Win32DiskImager** y **Raspberry Pi Imager** en una laptop corriendo Windows 10.
2. Utilizando **Raspberry Pi Imager** se procedió a realizar la instalación del sistema operativo Raspberry Pi OS (Legacy), basado en la distribución **Debian Bullseye** de Linux. El sistema operativo se instaló en una memoria microSD que servirá como disco duro del Raspberry Pi.
3. Se conectaron un monitor, teclado, y ratón al Raspberry Pi, además de insertar la memoria microSD en su ranura correspondiente, y se encendió el dispositivo.
4. Una vez terminada la configuración automática del sistema, se procedió a configurar fecha y hora, y autologin, seguido de un reinicio del sistema.
5. Se actualizó el sistema operativo y sus paquetes usando los comandos *apt-get update* y *apt-get dist-update -y.*
6. Se activó SHH para permitir conexiones remotas.
7. Se instaló la implementación openMPI para Python3 en Raspbian usando el comando *apt-get install python3-mpi4py -y*

8. Se realizó la instalación de *nmap* usando el comando *apt-get install nmap* para facilitar el descubrimiento de los demás nodos en la red.

2. Pruebas de configuración

Con el primer nodo configurado se probó el software más importante, Python y la instalación de MPI4PY:

1. Para Python se corrió un muy simple programa *Hello World* desde la terminal utilizando el comando *print ("Hello World")*
2. Para MPI4PY se utilizó el comando *mpirun hostname* lo que devolvió satisfactoriamente el Hostname utilizado por el dispositivo, raspberrypi, 4 veces, una por cada unidad de procesamiento del dispositivo, es decir una vez por cada CPU.
3. Luego se ejecutó el comando *mpirun -np 3 hostname* lo que devolvió el Hostname 3 veces. Este es el comportamiento esperado, ya que el comando indica que queremos ejecutar solo 3 procesos.

3. Generación de imagen y respaldo

Habiendo confirmado la correcta configuración del sistema se procede a generar una imagen con el objetivo de crear un respaldo que puede ser utilizado a manera de disco de instalación. Al instalar el sistema de esta manera no será necesario hacer la configuración inicial de manera manual en cada uno de los dispositivos.

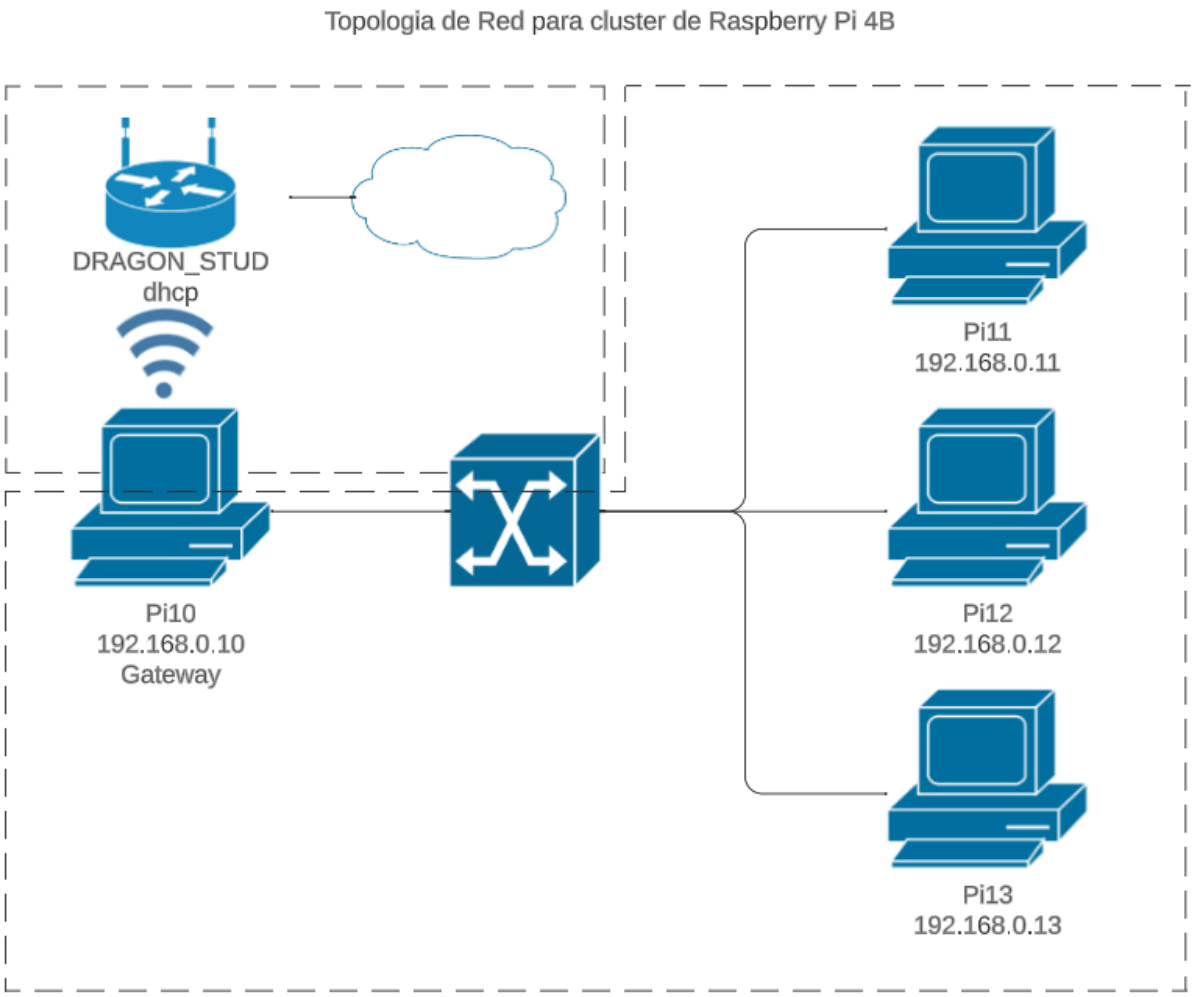
1. Se apagó el nodo 1 y se retiró la memoria microSD para conectarla a la laptop de desarrollo.
2. Se utilizó el programa **Win32DiskImager** para crear una imagen a partir de la memoria microSD.

3. Con dicha imagen, se procedió a utilizar **Raspberry Pi Imager,** utilizando su opción *Use custom* para instalarla en las otras 3 memorias microSD.
4. Se dejó el respaldo almacenado en el disco duro de la laptop y se procedió a insertar las memorias en sus respectivos Raspberry Pi.
5. Se repitieron las pruebas de configuración para cada Raspberry Pi.

4. Ensamblaje del clúster

Con todos los dispositivos listos se procedió a ensamblar el clúster de Raspberry Pi de acuerdo con la figura 3.

Figura 3. Topología de Red para Clúster de Raspberry Pi 4B

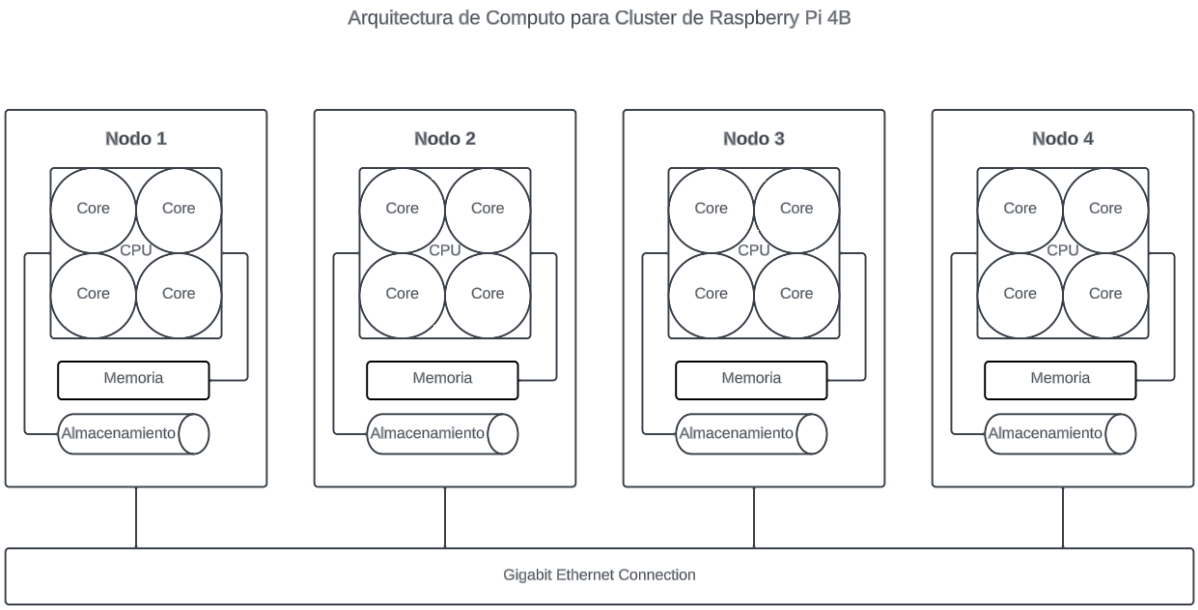


1. Se conectaron los dispositivos a un switch no administrable mediante ethernet. Y se confirmó de manera visual la conexión.
2. Se procedió con la configuración de red para cada uno de los dispositivos dentro de la red *192.168.0.1/24*, asignando *192.168.0.10-192.168.0.13* a los dispositivos.
3. Adicionalmente se configuro el nodo en 192.168.0.10 como Gateway de acceso a internet a través de la red WiFi DRAGON_STUD para estudiantes de la universidad.

5. Pruebas de comunicación

Una vez ensamblado el clúster se realizaron pruebas para confirmar que los dispositivos puedan comunicarse entre sí a través del switch. Estas pruebas consistieron en utilizar ping y posteriormente realizar distribución de carga de trabajo. Sobre el clúster con la arquitectura definida en la figura 4.

Figura 4. Arquitectura de cómputo para el clúster de Raspberry Pi



Nota. Cada nodo representa un Raspberry Pi y cada uno de sus núcleos de procesamiento se denominan *ranks* en terminología MPI. A cada rank se le asigna un numero que depende del orden en el cual se listen los nodos al momento de ejecutar un comando sobre el clúster. De forma similar, se utilizan todos los núcleos de un procesador antes de pasar a usar los del siguiente. La cantidad total de núcleos de procesamiento a usarse por una tarea y la disponibilidad de estos en cada nodo depende de las especificaciones del programador.

1. En cada uno de los Raspberry Pi se utilizó el comando *sudo ping -c 4 [IP address]* para confirmar la adecuada comunicación con cada uno de los demás nodos en el clúster.
2. Se actualizó el hostname de cada uno de los nodos a *Pi10-Pi13* basado en su dirección IP.
3. Se verificó las direcciones IP de todos los nodos utilizando el comando *nmap -sn 192.168.0.1/24*.
4. En el nodo maestro se creó el archivo *myhostfile* que alberga las direcciones IP de los nodos del clúster.
5. Se procedió a la generación de pares de llaves RSA para permitir el fácil acceso remoto y comunicación entre el nodo maestro y los nodos esclavos.
6. Se ejecutó el comando *mpirun -hostfile myhostfile -np 16 hostname*, el cual retornó 16 respuestas, 4 por cada nodo, indicando que el hostname del nodo que las ejecutó.
7. Se realizaron 8 diferentes pruebas para confirmar que el MPI4PY funcione adecuadamente. Una descripción más detallada de estas pruebas puede encontrarse en el Anexo A.
    i. Impresión de *Hello World* en cada nodo juntamente con su número de proceso.

ii. Condicionales que solo realizan ciertas tareas dependientes del número de proceso.
iii. Enviar y recibir datos hacia y desde nodos específicos de manera estática.
iv. Enviar y recibir datos hacia y desde nodos específicos de manera dinámica.
v. Etiquetado de información.
vi. Difusión de información.
vii. Dispersión de información.
viii. Unificación de información.

EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

Para cada uno de los experimentos realizados, se llevaron a cabo 21 ejecuciones, utilizando un de procesadores diferentes por cada ejecución. Para el cálculo de pi y la multiplicación matricial se tomó el tiempo de ejecución del nodo 0, el coordinador, luego de la distribución de la carga y una vez recopilados los datos procesados. El objetivo fue tener una representación más estadísticamente aceptable de los resultados. A continuación, se detallan cada uno de los experimentos que formaron parte de la suite de pruebas de rendimiento.

1. Cálculo de Pi

Este experimento consistió en la implementación paralela del cálculo de Pi utilizando un método Monte Carlo. En este caso se decidió utilizar un billón de puntos aleatorios para la estimación de Pi. Cada proceso recibe una semilla diferente basada en el número del proceso, de ahora en adelante definido como rango del proceso, cantidad de procesos, representativo del número total de procesadores que se utilizarán, y número total de puntos a usar durante la estimación (ver Anexo A).

Tabla 1. Tiempos Promedio de Cálculo de Pi utilizando un método Monte Carlo

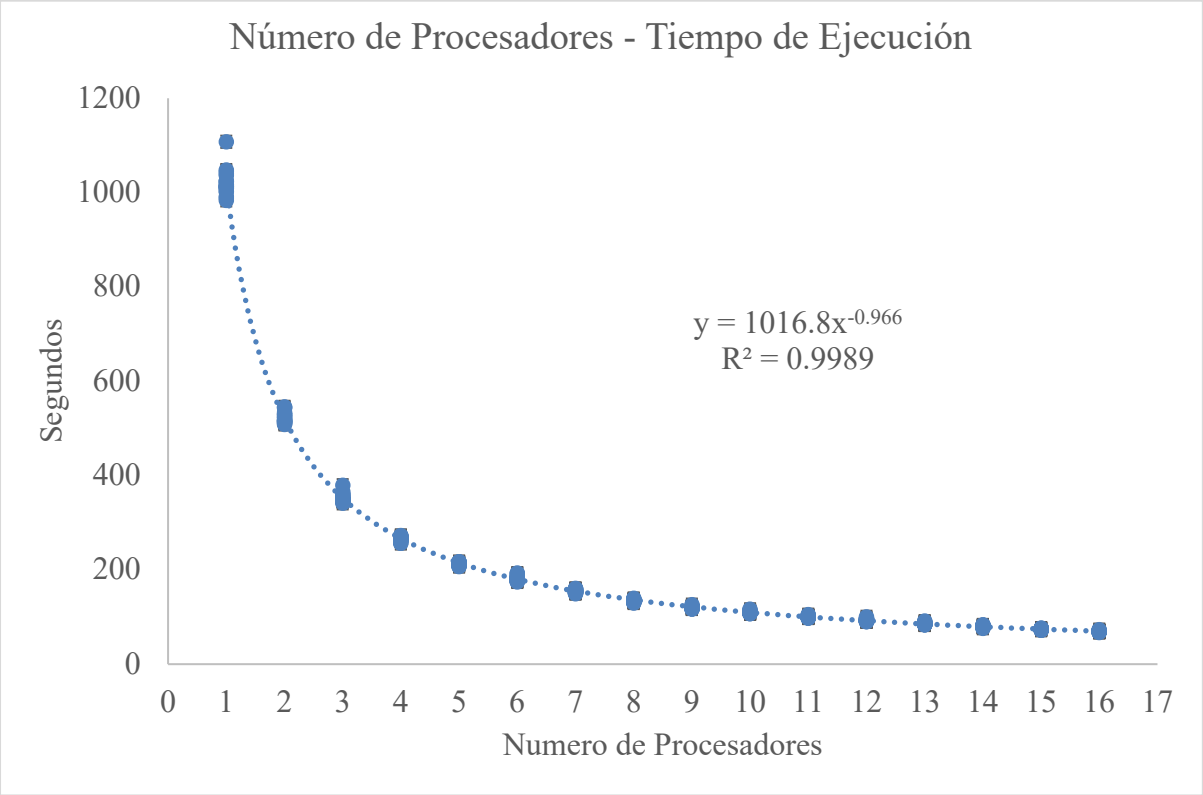
Número de Procesos	Pi Estimado	Precisión	Tiempo de Ejecución Promedio (s)
1	3.141665592	99.9927062%	1019.813395
2	3.1415976	99.9995054%	522.9094009
3	3.141597744	99.9999949%	351.7485868
4	3.141595456	99.9997198%	265.7937777
5	3.141616384	99.9976270%	212.6611283
6	3.141553368	99.9960714%	181.7053082

Número de Procesos	Pi Estimado	Precisión	Tiempo de Ejecución Promedio (s)
7	3.141573612	99.9980958%	154.972597
8	3.141514668	99.9999220%	134.3940782
9	3.141615836	99.9976818%	120.4348482
10	3.141491868	99.9899214%	110.1056324
11	3.141635116	99.9999575%	100.5693861
12	3.141640736	99.9951918%	93.24695949
13	3.14162366	99.9968994%	86.00928689
14	3.14154122	99.9948566%	79.76831866
15	3.141577072	99.9984418%	74.16599896
16	3.141614732	99.7792159%	69.58675356

La tabla 1 muestra un resumen de los datos obtenidos de cada ejecución para un número de procesadores. La variación en el Pi estimado y, por ende, su precisión se debe a que la semilla utilizada en la generación aleatoria de puntos para el método Monte Carlo se ve directamente afectada por el número total de procesos y el número de proceso que corresponde al nodo donde se ejecuta el experimento. Inicialmente se optó por este paso en la prueba para comprobar si existía un cambio en la estimación de Pi.

También se puede observar la disminución esperada en el tiempo de ejecución a medida que se aumenta el número de procesadores utilizados para llevar a cabo las pruebas.

Figura 5. Tiempos Promedio de Cálculo de Pi utilizando un método Monte Carlo



La figura 5 muestra la relación entre el número de procesadores y el tiempo de ejecución para este tipo de trabajo computacional. Se observa como el tiempo para realizar la estimación de Pi disminuye de manera drástica hasta usar 4 procesadores, punto en el cual se empieza a estabilizar la curva. Esto es consistente con el comportamiento esperado de un entorno de procesamiento paralelo. Se alcanzó un punto de saturación, pasado el cual, los beneficios de adicionar más procesadores se vuelven marginales. Este patrón puede darse debido a varios factores como sobrecarga de coordinación entre procesadores, limitaciones de ancho de banda, o velocidad de acceso a memoria.

Se generó una línea de tendencia en azul con el fin de estimar el tiempo en base al número de procesadores de manera teórica. Con un coeficiente de

determinación R^2 de 0.9989, se puede indicar que la curva exponencial obtenida se ajusta muy bien a los resultados obtenidos de las pruebas.

Tabla 2. Cambio en Rendimiento para el Cálculo de Pi

Número de Procesadores	Cambio en Rendimiento
2	95%
3	49%
4	32%
5	25%
6	17%
7	17%
8	15%
9	12%
10	9%
11	9%
12	8%
13	8%
14	8%
15	8%
16	7%

Nota. Se refleja el cambio con respecto al número de procesadores anterior. Es decir, usar 2 procesos con respecto a 1 aumenta el rendimiento un 95%, mientras que usar 15 procesadores con respecto a 14 solo aumenta el rendimiento un 8%.

La tabla 2 lista de forma más clara como la adición de un procesador influencia el rendimiento del clúster. Se puede observar como a partir del décimo procesador el incremento en rendimiento empieza a estabilizarse.

2. Multiplicación Matricial

La siguiente prueba en la suite creada para este proyecto fue la resolución del producto punto de dos matrices cuadradas. Con este fin se escribió un programa capaz de separar las operaciones correspondientes en tareas separadas que podían ser divididas entre todos los procesadores disponibles y luego unir los resultados de

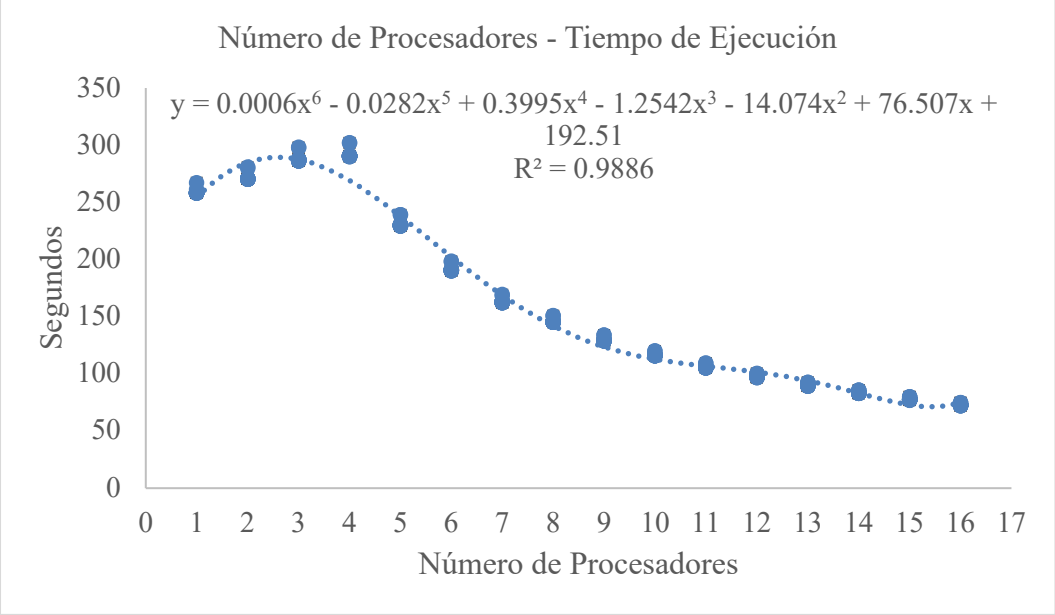
dichos procesos para formar la matriz resultante final. Ambas matrices se generaron de manera aleatoria, con un tamaño de 5000 x 5000 y con elementos entre 0.0 y 1.0 (ver Anexo A).

Tabla 3. Resultados de Multiplicación Matricial

Número de Procesos	Tiempo de Ejecución Promedio (s)	Incremento en Rendimiento Promedio
1	258.8725	
2	271.40884	-5%
3	288.062	-6%
4	291.59046	-1%
5	230.79702	26%
6	191.41319	21%
7	163.38496	17%
8	146.0621	12%
9	129.69026	13%
10	116.56171	11%
11	105.98972	10%
12	97.74023	8%
13	90.093368	8%
14	83.615205	8%
15	77.935912	7%
16	72.88013	7%

Nota. Se refleja el cambio con respecto al número de procesadores anterior.

Figura 6. Tiempos Promedio en Multiplicación Matricial



Este experimento provee resultados que son notables en la tabla 3, y mucho más evidentes en la figura 6. En lugar de existir un descenso inmediato como en el caso de la estimación de pi, existe un incremento en el tiempo de ejecución al añadir los tres primeros procesadores. Esto es un resultado inusual ya que de acuerdo con los resultados del experimento anterior se esperaba que luego de añadir un procesador siempre disminuya el tiempo de ejecución.

Para la implementación de este experimento se utilizó la biblioteca de Python NumPy para realizar las multiplicaciones entre matrices. Se concluyó que la implementación de producto punto entre matrices en NumPy podría no ser compatible con las operaciones de cómputo en paralelo y se escribió un algoritmo personalizado para realizar esta tarea.

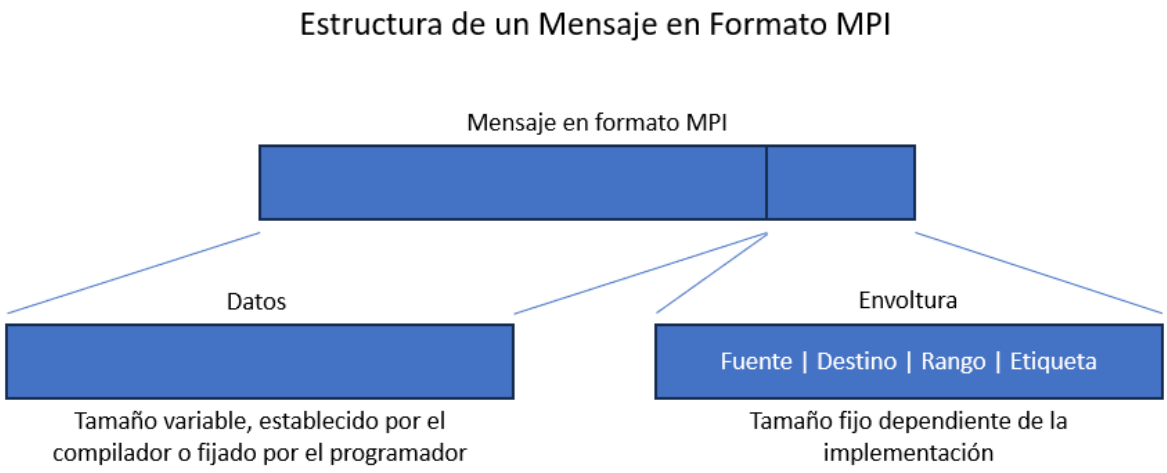
Las pruebas con el algoritmo personalizado presentaron la misma forma de la curva que aquellas que utilizaron NumPy para realizar el producto punto entre las matrices, lo que descartó un problema radicado en el algoritmo utilizado. La causa de estos resultados inusuales podría hallarse en un manejo ineficiente de las funciones provistas por MPI4PY. Sin embargo, al observar resultados similares a los mostrados por Barnes y Wazir, et al., se puede inferir que la causa puede radicar en la naturaleza de los experimentos llevados a cabo.

A pesar de que las tareas en las publicaciones antes mencionadas inspiraron a las pruebas que se realizaron en este proyecto, la implementación de dichos algoritmos fue a discreción del autor, sin consultar otros proyectos. La literatura encontrada no describe o menciona el motivo de estos resultados que, si bien no son intuitivos, pueden producirse al momento de realizar estas pruebas. Por lo tanto, se descartó un manejo ineficiente del MPI y el enfoque cambió a un análisis del

algoritmo; este fenómeno se puede atribuir a una sobrecarga debido al pase de mensajes o indexación local.

Para concluir se realizaron pruebas de velocidad de red y se encontró que la velocidad de transferencia promedio entre los nodos es de 935 Mbps. Esta evidencia sirvió para descartar la posibilidad de problemas de comunicación durante las pruebas. Además, se realizó una investigación de la estructura del paquete mediante el cual MPI envía mensajes. El mensaje contiene un encabezado conocido como envoltura, donde se almacena la fuente, destino, rango, y la etiqueta, y la carga que se distribuye entre los procesos. El tamaño del encabezado es dependiente de la implementación (Dalcin, 2023). La figura 7 ilustra de mejor manera dicha estructura.

Figura 7. Estructura de un Mensaje en Formato MPI



Nota. El gráfico fue construido con el objetivo de proveer una mejor visualización de la estructura de datos que se utiliza para enviar los datos de un procesador a otro. Este esquema le permite al nodo maestro saber hacia que nodo enviar la información que lleva en el mensaje y otros datos importantes como su rango, o posición asignada de entre todos los procesos, y la etiqueta de llevar alguna (Dalcin, 2023).

3. HPL

Se ejecutó la suite HPL en el clúster para diferentes números de procesos.

Tabla 4. Resultados de Ejecución de HPL

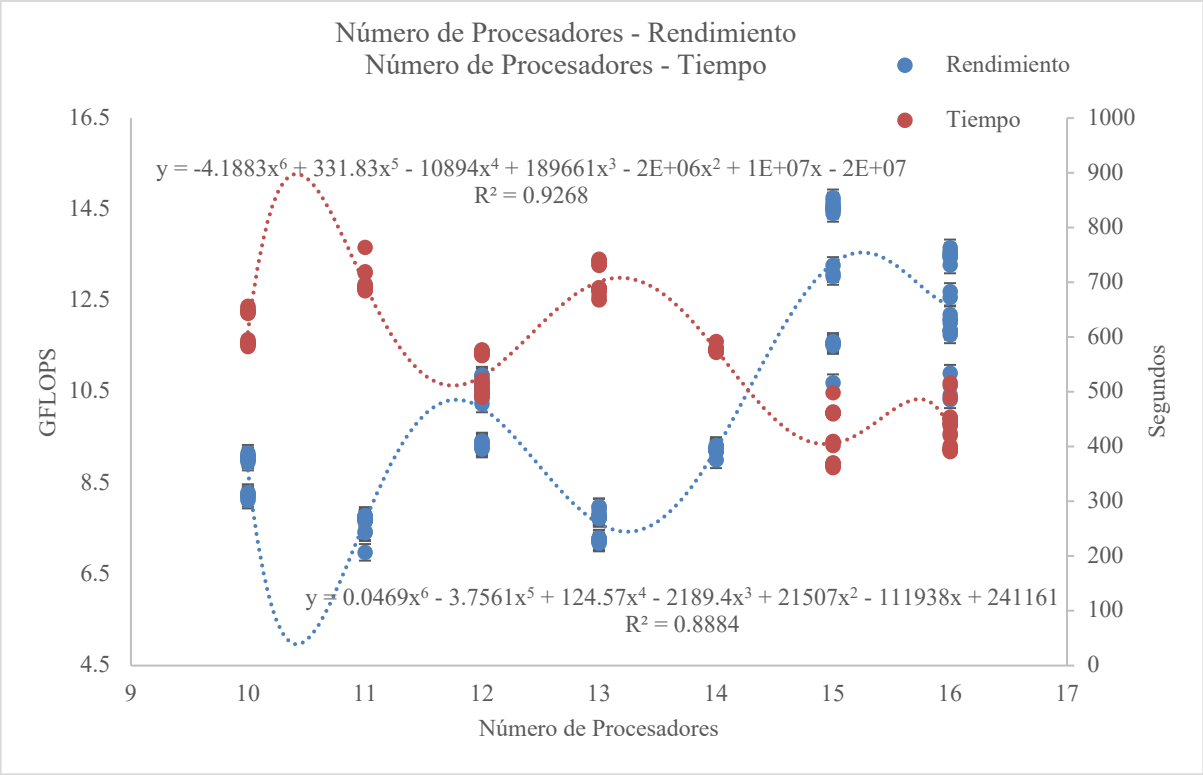
Número de Procesos	Tiempo de Ejecución Promedio (s)	Rendimiento Promedio (GFLOPS)
16	609.0695238	12.40652381
15	696.7285714	13.31252381
14	528.0466667	9.238457143
13	701.2690476	7.616052381
12	577.3861905	10.14432857
11	405.5657143	7.660080952
10	433.0114286	8.775690476

Nota. Para el cambio en el número de procesos se realizó un ajuste en los parámetros de ejecución de HPL. Estos cambios impactan que tan eficientemente se distribuye el poder de cómputo durante las pruebas de rendimiento.

En la tabla 4 se muestran los promedios de los resultados de las ejecuciones del HPL. Los puntos naranjas en la figura 8 representan el cambio de tiempo de ejecución con respecto al número de procesadores, mientras que los puntos azules denotan el rendimiento. En este gráfico se aprecia de mejor manera la relación que existe entre el rendimiento en forma de GFLOPS y el tiempo.

Como se mencionó en la nota de la tabla 4, existe una notable variación en los resultados de estas pruebas de rendimiento. De acuerdo con la documentación del programa, esto se debe a que la proporción de la cuadricula de procesos P x Q que representan los procesos distribuidos a lo largo de filas y columnas respectivamente se ve alterada. La documentación también indica que en una red de malla o con un switch es recomendable usar configuraciones donde Q es ligeramente mayor a P, mientras que en conexiones ethernet simples donde se usa solo un cable, es recomendable que P sea mucho menor a Q (Dongarra, 2007).

Figura 8. Resultados de Ejecución de HPL



Nota. El rendimiento se encuentra marcado en naranja mientras que el tiempo de ejecución se muestra en azul.

CONCLUSIONES Y TRABAJO FUTURO

Este proyecto exploró la eficacia de la computación paralela mediante el uso de operaciones intensivas de cálculo, especialmente algebra lineal. Los resultados demostraron que el uso de más recursos computacionales significará un aumento en la eficiencia del sistema.

La implementación de MPI utilizada, MPI4PY, resultó ser una herramienta muy poderosa, con distintos grados de granularidad que permite utilizar cada procesador de un sistema de manera eficiente. No obstante, se encontró que el manejo de muchas funciones importantes queda en manos del programador. Si bien la interfaz provee mucha funcionalidad, existen variables como, el lenguaje de programación, la arquitectura del sistema, en términos de sistema operativo, hardware, y red, y la aplicación final que va a tener el clúster, que dependen del programador para ser optimizados y alcanzar el mejor rendimiento.

Asimismo, se demostró la importancia de conocer cuál es el límite de la escalabilidad del sistema. Dependiendo de la tarea a realizar y los recursos disponibles, si bien el rendimiento seguirá incrementando en pequeñas cantidades con cada procesador añadido, es necesario considerar cuando se llega a un punto de saturación. Al llegar a este cuello de botella, añadir más procesadores representa un desperdicio de recursos económicos; el conocimiento de este punto solo se puede lograr con investigación previa y experimentación.

A continuación, se sugiere una lista de recomendaciones para futuras investigaciones.

- Optimización de algoritmos. Se puede hacer una revisión de los algoritmos implementados con el objetivo de analizar posibles áreas en las cuales se pueda mejorar el rendimiento de la suite de pruebas.

- Análisis de la configuración de hardware. Lamentablemente no se logró adquirir el rack para el clúster a tiempo, sería interesante observar si la instalación en un rack vertical cambia los resultados de rendimiento. De la misma manera, se utilizaron diferentes cables ethernet al momento de construir el clúster, valdría la pena el homogenizar esa configuración e investigar el impacto que tendría en el rendimiento.
- Overclocking del cluster. Todos los nodos del clúster corrieron a su frecuencia normal durante las pruebas, en las condiciones adecuadas hacer overclock de los nodos de cómputo puede incrementar significativamente el rendimiento.
- Implementación de las pruebas de rendimiento en otros lenguajes. FORTRAN y C++ son los lenguajes más utilizados al momento de programar para clústeres de cómputo, sería beneficioso el implementar las pruebas en esos lenguajes y comparar los resultados entre rendimiento y tiempo de desarrollo para conseguir una buena eficiencia.
- Investigación de resultados atípicos. Los resultados de la multiplicación matricial no tienen una respuesta concreta. Sería importante poder realizar una investigación más a fondo de porque se produce ese fenómeno. Asimismo, sería importante recrear el experimento de generación de números primos mencionado en la investigación de Wazir, et al., para observar si se pueden replicar los resultados inesperados que obtuvieron en su experimento.

REFERENCIAS BIBLIOGRÁFICAS

Barnes, R. (2017, agosto). Benchmarking a Raspberry Pi cluster. *The MagPi*, (60). Recuperado el 25 de noviembre de 2023 https://magpi.raspberrypi.com/articles/benchmarking-raspberry-pi-cluster

Dalcin, L. (2023, octubre 4). *MPI for python*. MPI for Python. https://mpi4py.readthedocs.io/

Dennis, A. K. (2013). *Raspberry Pi Super Cluster*. Packt Publishing.

Dongarra, J. (2007, mayo 8). Frequently Asked Questions on the Linpack Benchmark and Top500. *Netlib*. Recuperado de https://www.netlib.org/utk/people/JackDongarra/faq-linpack.html#_For_HPL_what_process%20grid%20ratio%20P%20x

Dowd, K., & Severance, C. (2010). *High performance computing.* OpenStax-CNX.

Gregory, A. (18 de marzo de 2021). *Supercomputing with Raspberry Pi - HackSpace 41*. Raspberry Pi. Recuperado el 29 de noviembre de 2023, de https://www.raspberrypi.com/news/supercomputing-with-raspberry-pi-hackspace-41/

Markoff, J. (1991, mayo 6). *The attack of the "Killer Micros."* The New York Times. https://www.nytimes.com/1991/05/06/business/the-attack-of-the-killer-micros.html

Pajankar, A. (2017). *Raspberry pi supercomputing and scientific programming MPI4PY, NumPy, and scipy for enthusiasts.* Apress.

Severance, C. (2020, mayo 18). *Introduction to High Performance Computing.* Engineering LibreTexts. https://eng.libretexts.org/@go/page/14745

Sterling, T. L., Anderson, M., & Brodowicz, M. (2018). *High performance computing: Modern systems and practices.* Morgan Kaufmann.

Upton, E., & Halfacree, G. (2017). *Raspberry pi user guide.* John Wiley & Sons.

Wazir, S., Ikram, A., Imran, H., Khan, H., Ikram, A., & Ehsan, M. (2019). Performance Comparison of MPICH and MPI4py on Raspberry Pi-3B Beowulf Cluster. *Journal of Advanced Research in Dynamical & Control Systems, 11*(03-Special Issue). Recuperado el 24 de noviembre de 2023, de https://www.researchgate.net/publication/337184208_Performance_Comparison_of_MPICH_and_MPI4py_on_Raspberry_Pi-3B_Beowulf_Cluster

ANEXO A: REPOSITORIO EN GITHUB

Se presenta una breve explicación de cada uno de los scripts de Python utilizados durante este estudio. El repositorio puede ser encontrado en https://github.com/Max-AP/RasPi_HPC_Cluster.

- Configuration Tests

  Estas pruebas de configuración tienen como objetivo ser una introducción a algunas de las funciones más importantes que forman parte de MPI4PY y fueron tomadas de Raspberry pi supercomputing and scientific programming MPI4PY, NumPy, and scipy for enthusiasts, por Pajankar, A.

  1. mpi_test. Una simple prueba de comunicación entre los procesos. Solo puede ser ejecutado en 16 procesos. Se envía una string del proceso 0 (maestro) a los otros 15 y estos completan la información faltante con su correspondiente rango.
  2. mpi_test_1.py. Una versión del clásico programa de prueba “Hello World” que imprime este mensaje en la consola, especificando el nombre de host del Raspberry Pi donde se ejecuta y el número de proceso asignado.
  3. mpi_test_2.py. Revisión de condicionales. El objetivo con este programa era observar de qué manera se puede ejecutar código en un nodo de cómputo designado utilizando su número asignado como referencia.
  4. mpi_test_3.py. De manera programática se comprueba el número de procesos total que pueden ser utilizados en el clúster.
  5. mpi_test_4.py. Prueba de transferencia de datos entre procesos. En el nodo maestro se crea una variable, la cual es enviada a los demás nodos. Para comprobarlo se imprime el paquete de datos recibido.

6. mpi_test_5.py. De manera similar a mpi_test_4.py se crea una variable, pero en esta ocasión, su valor es asignado de forma dinámica. Los destinos de dichos datos también son asignados de forma dinámica por el script.
7. mpi_test_6.py. Prueba de etiquetado de datos. Con el fin de identificar el origen de los datos que cada proceso recibe, es posible añadir una etiqueta a los mismos. Esta prueba realiza esta tarea y luego verifica la información imprimiendo en pantalla el contenido de los datos dependiendo de su etiqueta.
8. mpi_test_7.py. Prueba de transmisión. Hasta este punto cada envió de datos ha consistido en especificar el destino de los datos. En este caso la tarea consiste en transmitir datos de un nodo en particular a todos los demás. Para comprobarlo los datos que se crearon en el nodo maestro se imprimen desde los demás nodos.
9. mpi_test_8.py. Prueba de difusión. De manera similar a la prueba de transmisión, esta tarea envía datos generados en el nodo maestro a los demás. Sin embargo, en lugar de enviar una copia de todos los datos, los distribuye en partes equitativas antes de enviarlos.
10. mpi_test_9.py. Prueba de recopilación. El nodo maestro recopila los resultados de las operaciones de cómputo en los demás nodos para usarlos.

- Performance Tests

  Test2.py y test4.py fueron desarrolladas por el autor con el objetivo de medir el rendimiento del clúster a través de la medida del tiempo de ejecución de operaciones matemáticas con un alto costo computacional. HPL fue desarrollado por J. Dongarra, J. Bunch, C. Moler and G. W. Stewart y se encuentra disponible como software gratuito en https://www.netlib.org/benchmark/hpl/

1. test2.py. Prueba de rendimiento utilizando el método Monte Carlo para la estimación de pi. Este método consiste separar los puntos a procesar de manera equitativa para cada nodo. De esta forma se reduce el tiempo de conteo y se agiliza el proceso de estimación de pi.
2. test3.py, test4.py, test5.py. Prueba de rendimiento utilizando multiplicación matricial. Estos tres scripts contienen el algoritmo para separar la multiplicación matricial en partes que pueden ser procesadas de manera independiente por cada nodo. Al presentarse los valores atípicos se crearon diferentes versiones para comprobar que no se tratase de un error en el algoritmo. Luego de comprobar esto se estableció a test4.py como el método preferido al usar un método de producto punto optimizado por la biblioteca NumPy.