USANDO UN CLUSTER

Monte Carlo Needles y Dartboard

Presentado por Leidy Johanna Martinez Ramirez y Valentín Valencia Valencia

Índice

1. Resumen

El presente informe aborda la implementación de la optimización en un sistema distribuido mediante un cluster con tres working nodes. Se usó una configuración NFS para que todas las máquinas tuvieran acceso a una carpeta compartida.

2. Introducción

En el ámbito de la computación distribuida, la eficiencia y el rendimiento son aspectos críticos para garantizar un procesamiento rápido y efectivo de tareas complejas. Uno de los desafíos recurrentes en este dominio es la optimización de algoritmos para operaciones intensivas, como la multiplicación de matrices. Este informe se centra en el análisis y la mejora del rendimiento de un programa específico diseñado para calcular el tiempo requerido para multiplicar matrices de tamaño n x n en un entorno distribuido con un cluster basado en sistemas operativos Linux.

El programa en cuestión representa una aplicación crítica en numerosos campos, desde la simulación científica hasta el aprendizaje automático distribuido. El objetivo principal es maximizar la eficiencia delos algoritmos Darthboard y Monte Carlo Needles, teniendo en cuenta las complejidades inherentes a la distribución de tareas en un entorno de cluster.

3. Marco conceptual

3.1. Teoría de los algoritmos

3.1.1. Monte Carlo Needles

Durante la Guerra Civil Americana, el Capitán C.O. Fox se encontraba recuperándose de una herida en un hospital militar. Para pasar el tiempo, arrojó una serie de agujas idénticas de manera aleatoria sobre un tablero en el que previamente había dibujado una serie de líneas paralelas, separadas por la longitud de una aguja. Contó el número de lanzamientos y el número de aciertos, es decir, las instancias en las que una aguja tocó o intersectó una línea. Después de 1100 lanzamientos, el Capitán había determinado con dos decimales. ¿Cómo es posible? En primer lugar, parece que fue el Conde de Buffon (1707-1788) quien examinó este tipo de experimento y en cuyo honor ahora se conoce como el problema de la aguja de Buffon. En 1777, Buffon demostró que la relación entre los aciertos y los lanzamientos era de 2:pi, o dicho de otra manera, que la probabilidad de que una aguja lanzada al azar sobre el área terminara descansando sobre una de las líneas era de aproximadamente 2/pi ≈ 63.7%. Con este conocimiento, Fox pudo calcular pi al duplicar el número de lanzamientos y dividirlo por el número de aciertos.

Lo interesante del problema de la aguja es que establece una conexión entre el "pi geométrico" y el área muy diferente de las probabilidades. Existes otras relaciones similares entre π y la probabilidad, a partir de las cuales se derivan otros métodos para calcular pi. Estos métodos se conocen de manera informativa como métodos de Monte Carlo.

3.1.2. Dartboard

Consideremos el área del tablero de dardos circular. Tiene un radio de uno, por lo que su área es π. El área del trozo cuadrado de madera es 4 (2 x 2). La proporción del área del círculo al área del cuadrado es π/4. Si lanzamos un montón de dardos y los dejamos caer al azar sobre

el trozo cuadrado de madera, algunos también caerán en el tablero de dardos. La cantidad de dardos que caen en el tablero de dardos, dividida por la cantidad total que lanzamos, estará en la proporción descrita anteriormente ($\pi/4$). Multiplicamos por 4 y obtenemos π.

3.1.3. Cluster en Linux

El cluster está compuesto por un conjunto de **tres working nodes** que trabajan de manera colaborativa para ejecutar tareas concurrentes, y el sistema operativo Linux proporciona el entorno necesario para coordinar estas operaciones distribuidas de manera eficiente.
Se usó una configuración NFS para que todas las máquinas tuvieran acceso a una carpeta compartida.

3.1.4 Open MP

OpenMP (Open Multi-Processing) es una interfaz de programación de aplicaciones (API) para la programación paralela en sistemas con memoria compartida. Su objetivo principal es facilitar la escritura de programas que puedan aprovechar eficazmente los múltiples núcleos de CPU en máquinas multiprocesadoras y en clústeres de cómputo.

- Programación paralela: OpenMP se utiliza para escribir programas paralelos que pueden descomponer tareas en subprocesos o hilos que se ejecutan simultáneamente en múltiples núcleos de CPU. Esto mejora el rendimiento y reduce el tiempo de ejecución de aplicaciones intensivas en cómputo.
- Memoria compartida: OpenMP se adapta mejor a sistemas con memoria compartida, donde múltiples hilos pueden acceder a la misma memoria. Esto facilita la comunicación y la sincronización entre los hilos, lo que es esencial para la programación paralela.
- Directivas y pragmas: OpenMP utiliza directivas y pragmas en el código fuente para identificar secciones que pueden ejecutarse en paralelo. Estas directivas proporcionan instrucciones al compilador sobre cómo dividir y asignar tareas a los hilos.

- Modelo de programación fácil de usar: OpenMP está diseñado para ser relativamente fácil de aprender y usar en comparación con otras técnicas de programación paralela, como la programación con hilos nativos o la programación de paso de mensajes. Esto hace que sea más accesible para programadores que desean paralelizar sus aplicaciones.
- Portabilidad: OpenMP es compatible con una variedad de lenguajes de programación, como C, C++, Fortran, y más. Esto permite a los desarrolladores escribir programas paralelos independientemente del lenguaje de programación y ejecutarlos en diferentes plataformas.
- Escalabilidad: OpenMP facilita la escalabilidad de las aplicaciones, lo que significa que los programas pueden ejecutarse de manera eficiente en sistemas con diferentes cantidades de núcleos de CPU, desde máquinas de un solo núcleo hasta clústeres de alto rendimiento.

En resumen, OpenMP es una API que simplifica la programación paralela en sistemas con memoria compartida, permitiendo a los desarrolladores escribir código que aproveche eficazmente el potencial de cómputo de múltiples núcleos de CPU. Facilita la creación de aplicaciones paralelas portátiles y escalables, lo que es especialmente útil en entornos de cómputo intensivo y científico.

3.2. Características de las máquinas

3.2.1 Máquina de Valentín

Característica	Especificación
Procesador	Procesador Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz, 1190 Mhz,
Sistema Operativo	ubuntu-22.04.3-live-server-amd64
Número de	4 procesadores principales,

núcleos	8 procesadores lógicos

3.3. Desarrollo

3.3.1. Implementación del cluster

- Configuración del Hardware:

  Se verificó la disponibilidad de hardware compatible para la construcción del cluster, asegurando la presencia de al menos dos nodos interconectados.

- Sistema Operativo:

  Se instaló una distribución de Linux en cada nodo, en este caso se hizo copiando una imagen de esta proporcionada por el docente.

- Conexión de Red:

  Se configuró la red para permitir la comunicación entre nodos, utilizando conexiones Ethernet o estableciendo una red privada según las necesidades.

- Configuración de SSH:

  Se implementó la autenticación SSH entre nodos para facilitar la administración remota y la ejecución de comandos.

- Instalación de Software de Cluster:

  Se seleccionó e instaló el software de gestión de clúster según los requisitos específicos de la aplicación, en este caso se usó OpenMP.

- Configuración del Software de Cluster:

  Se configuró el software de clúster para definir roles de nodos, establecer configuraciones de almacenamiento compartido y adaptar la infraestructura a las necesidades particulares del proyecto.

- Prueba de Comunicación:

Se realizó una prueba exhaustiva para garantizar una comunicación fluida entre los nodos, empleando tanto comandos remotos como herramientas específicas del software de clúster utilizado.

- Desarrollo y Ejecución de Aplicaciones:

  Se desarrollaron aplicaciones específicas para aprovechar la capacidad de procesamiento distribuido del cluster, asegurando que estuvieran configuradas para ejecutarse eficientemente en este entorno. En este caso, se evaluó el rendimiento del programa para la multiplicación de matrices N x N.

4. Pruebas

4.1. Pruebas Valentin

4.1.1. Monte Carlo Needles

Tamaño (N)	1000	20000	3000000	50000000	80000000
1	0,001922	0,007801	0,089482	1,343951	2,160184
2	0,004086	0,003853	0,100141	1,388064	2,058244
3	0,005673	0,005311	0,080023	1,408692	1,999899
4	0,007701	0,005346	0,085092	1,562344	2,584082
5	0,002681	0,007331	0,079983	1,520744	2,333771
6	0,004203	0,003793	0,088279	1,492176	2,545864
7	0,004616	0,004476	0,090261	1,502068	2,435926
8	0,246882	0,259431	0,082139	1,680719	2,384129
9	0,004206	0,004357	0,114160	1,829241	2,766778
10	0,004189	0,004197	0,077618	1,538916	2,767176
Promedio	**0,0286159**	**0,0305896**	**0,0887178**	**1,5266915**	**2,4036053**

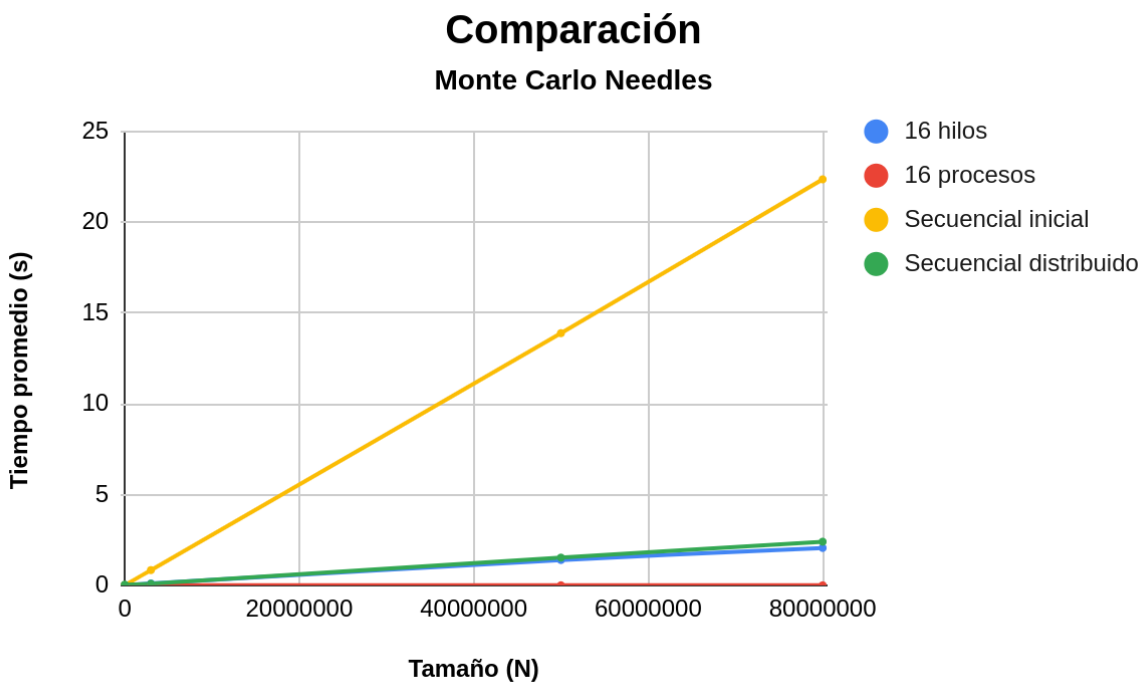
4.1.2. Dartboard

Tamaño (N)	1000	20000	3000000	50000000	80000000
1	0,002465	0,003432	0,052515	0,640975	1,008678
2	0,001013	0,007870	0,046403	0,620835	0,996781
3	0,003079	0,004207	0,051207	0,619719	1,063733
4	0,002760	0,217749	0,043165	0,661305	1,052818
5	0,197340	0,003375	0,046056	0,625421	1,021863
6	0,207970	0,004521	0,048580	0,620325	1,084312
7	0,002137	0,001012	0,325103	0,697548	1,051295
8	0,002614	0,001518	0,056894	0,617027	1,081927
9	0,002061	0,005446	0,210163	0,683121	1,058804
10	0,003721	0,002677	0,053427	0,674783	1,036814
Promedio	**0,0425160**	**0,0251807**	**0,0933513**	**0,6461059**	**1,0457025**

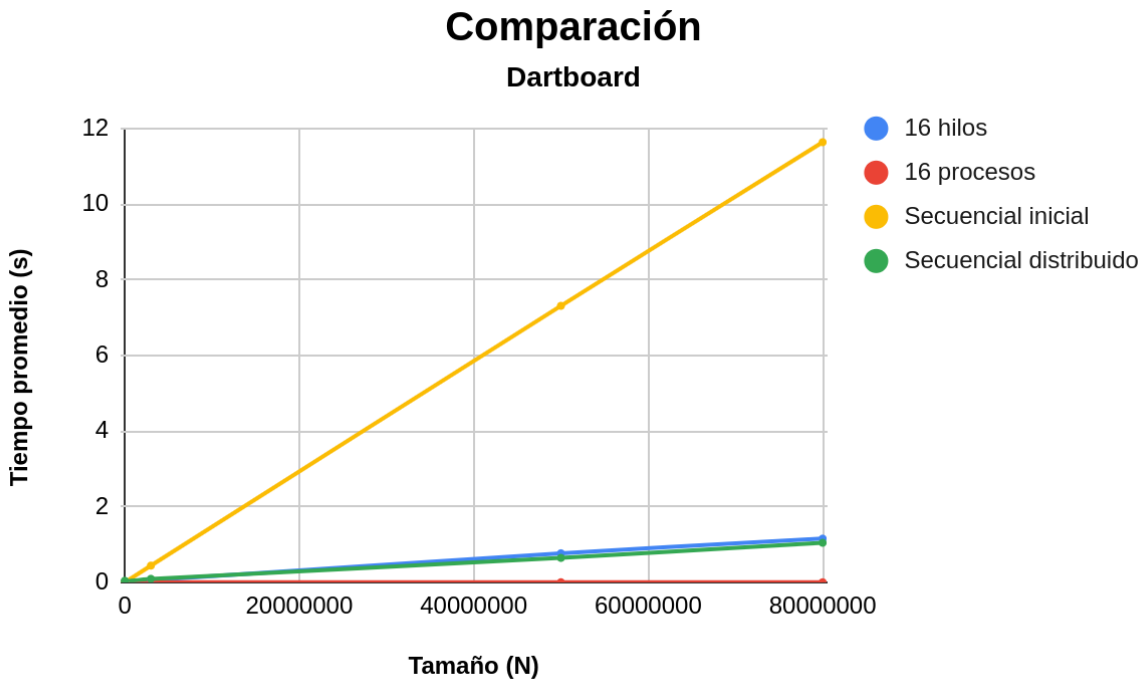
5. Resultados (Comparación entre Hilos)

5.1. Máquina Valentin

5.1.1. Monte Carlo Needles



5.1.2. Dartboard



5. Conclusiones

- El peor tiempo de ejecución, para ambos algoritmos, es el de la implementación inicial de manera secuencial sin ningún uso de herramientas de optimización.
- La ejecución mediante un sistema distribuido es una de las que mejores tiempos da, sin embargo, no es la mejor.
- El mejor tiempo de ejecución lo realizan los 16 procesos, en ambos algoritmos.

7. Bibliografía

[1] Cortéz, A. (2004). TEORÍA DE LA COMPLEJIDAD COMPUTACIONAL Y TEORÍA DE.

Revista De Investigación De Sistemas E Informática, 1(1), 102-105. Recuperado de: https://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/view/3216

[2] Montero, L. H., & Antunez, R. R. (2011, 07). Parallel programming: definitions, mechanisms

and trouble. From

https://www.researchgate.net/publication/274960405_Parallel_programming_definitions_mechanisms_and_trouble

[3] Qaz Wiki. (2020). Qaz Wiki. From https://es.qaz.wiki/wiki/Matrix_multiplication_algorithm

[4] Arm. (2023). Arm Compiler for embedded User Guide. *developer.arm.com*. https://developer.arm.com/documentation/100748/0620/Using-Common-Compiler-Options/Selecting-optimization-options

[5] Wikipedia contributors. (2023). Inline expansion. *Wikipedia*. https://en.wikipedia.org/wiki/Inline_expansion

[6] *PI - unleashed*. (n.d.). Google Books. https://books.google.com.co/books?id=JlG5rFH7Ge0C&pg=PA39&lpg=PA39&dq=Dartboard+Method+algorithm&source=bl&ots=t76R30Q342&sig=NjguOYMc0ILqZs8Bcz6uIpfejdc&hl=en&ei=-YzTSuutFMefkQXj9_H7Aw&sa=X&oi=book_result&ct=result&redir_esc=y#v=onepage&q&f=false