Reto Final - Message-Passing Programming and High Massive Parallelization Cellular Automaton Exercise

Presentado por:

Santiago Sosa Herrera

Presentado a:

Ramiro Andrés Barrios Valencia

HPC: High Performance Computing

Ingeniería de Sistemas y Computación

Universidad Tecnológica de Pereira

2023

**Tabla de contenido**

**Resumen4**

**Introducción5**

**Marco Conceptual6**

High Performance Computing6

Complejidad Computacional6

Programación paralela6

Speedup6

MPI6

**Marco Contextual7**

Características de la máquina7

Desarrollo y Pruebas7

Grafica comparativa 9

Conclusiones10

**Bibliografía11**

**Resumen**

Este informe describe la implementación paralela de un modelo de simulación de tráfico en un clúster basado en Linux Mint. El modelo utiliza un autómata móvil unidireccional para simular el movimiento de los automóviles a lo largo de la carretera de circunvalación. El objetivo es analizar el comportamiento del tráfico y calcular la velocidad promedio de los autos durante cada iteración.

El modelo se basa en reglas de actualización simples en las que cada automóvil avanza una posición si la celda frente a él está vacía. Se utilizan condiciones de contorno periódicas para modelar la derivación. Las implementaciones en el lenguaje de programación C utilizan el modelo de programación MPI (interfaz de transferencia de mensajes) para la comunicación entre procesos. El informe analiza cómo se distribuye la computación entre los procesadores del clúster, identificando las partes que pueden ejecutarse de forma independiente y las que requieren interacción y sincronización.

Se muestra el pseudocódigo para la implementación paralela y cómo MPI maneja las condiciones de frontera y comunicación. Se presentan dos enfoques de comunicación: uno de forma asíncrona, similar al mailing, y otro síncrono, similar a una conversación telefónica. Se discuten las ventajas y desventajas de cada método y se dan recomendaciones para evitar interbloqueos durante la implementación síncrona. El informe también menciona que la implementación se realizará en un grupo de computadoras basadas en Linux Mint. Se llama la atención sobre las ventajas del entorno Linux Mint, como la estabilidad del sistema operativo, la facilidad de configuración y la disponibilidad de herramientas de desarrollo de código abierto. En resumen, la implementación paralela del modelo de simulación de tráfico en un clúster de Linux Mint permite una computación más rápida y eficiente utilizando potencia de computación distribuida. El informe destaca la importancia del paralelismo a la hora de modelar modelos complejos y destaca la flexibilidad y eficiencia del entorno Linux Mint para este tipo de aplicaciones.

**Introducción**

En la simulación de sistemas, el uso de la tecnología de computación paralela se ha convertido en una necesidad para resolver problemas complejos y lograr resultados más rápidos y precisos. Este informe presenta una implementación paralela de un modelo de simulación de tráfico en un grupo de computadoras basadas en Linux Mint. El modelo de simulación de tráfico se basa en un autómata celular unidimensional, donde cada celda representa una sección de la carretera. Cada celda puede tener dos valores: 1 si hay automóviles en el segmento de carretera y 0 si no hay automóviles. El objetivo es estudiar el comportamiento del tráfico y calcular la velocidad media del coche en cada iteración.

La implementación del modelo se realiza en el lenguaje de programación C utilizando el modelo de programación de paso de mensajes MPI. Le permite distribuir cálculos entre múltiples procesadores en un clúster y coordinar la comunicación entre ellos.

Esta revisión examina la regla de actualización del modelo y proporciona pseudocódigo para una implementación paralela. Se analizan las partes de la computación que se pueden realizar de forma independiente, así como aquellas que requieren la cooperación y sincronización del procesador. Además, se analizan dos métodos de comunicación: asincrónico y sincrónico, y se proporcionan pautas para evitar problemas de punto muerto. La elección del clúster de Linux Mint como entorno de ejecución se destacó por sus ventajas en términos de estabilidad del sistema operativo, facilidad de configuración y disponibilidad de herramientas de desarrollo de código abierto.

En resumen, la implementación paralela de modelos de simulación de tráfico en un clúster de Linux Mint permite operaciones más rápidas y eficientes desde el punto de vista computacional, lo que permite un estudio más detallado del comportamiento del tráfico. Este informe examina los detalles de esta implementación y presenta los resultados alcanzados.

**Marco conceptual**

**High Performance Computing:** High Performance Computing es un área de la informática que se dedica a crear sistemas informáticos y software que puedan manejar tareas de alta complejidad y procesamiento de grandes volúmenes de datos en un período de tiempo muy corto.

**Complejidad Computacional:** La complejidad computacional es una forma de medir los recursos que se necesitan para resolver un problema computacional. Sirve para determinar la dificultad de un problema y para comparar la eficiencia de distintos algoritmos.

**Programación paralela:** La programación paralela es una técnica de programación que se basa en dividir un problema en tareas más pequeñas y ejecutarlas en múltiples procesadores al mismo tiempo. Esta técnica se utiliza para mejorar la eficiencia de los sistemas informáticos y acelerar la resolución de problemas.

**Speedup:** El speedup se utiliza para medir la mejora en la velocidad de ejecución de un programa cuando se ejecuta en un sistema de cómputo más rápido o en paralelo. Se mide en términos de una relación entre el tiempo que tarda un programa en ejecutarse en un sistema de cómputo determinado y el tiempo que tarda en ejecutarse en un sistema más rápido. Cuanto mayor sea el speedup, mayor será la mejora en el rendimiento.

**MPI**: La interfaz de paso de mensajes o MPI es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores, escritas generalmente en C, C++, Fortran y Ada.

**Marco Contextual**

**Características de la maquina**

Las características del pc donde se realizaron las pruebas son los siguientes:

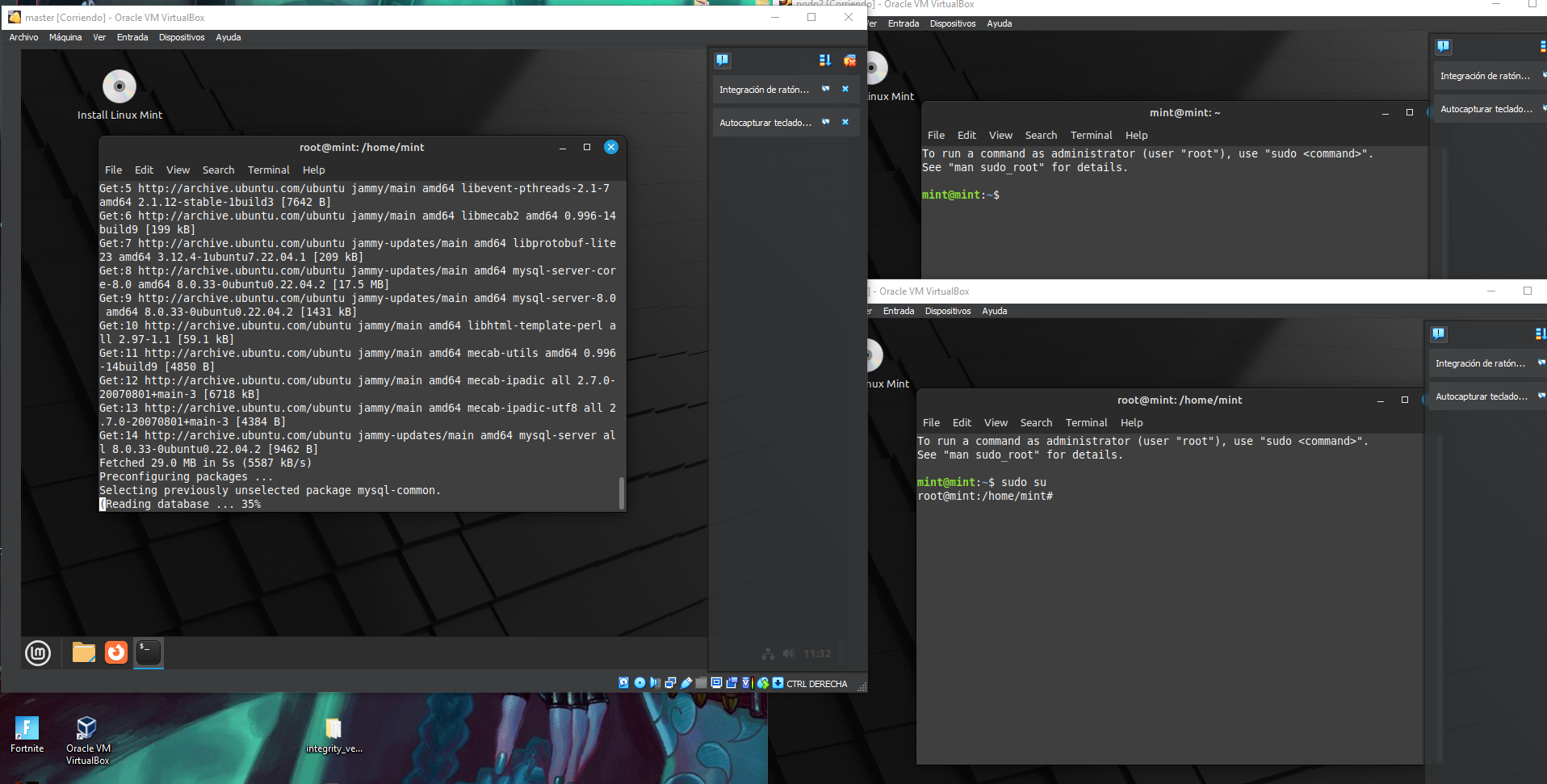
* Procesador AMD A8-7410 apu with amd Radeon r5 graphics x 4
* Memoria 6,7 GiB
* SO 64 bits
* Disco de 295 GB
* Sistema operativo Linux Mint 19
  + Gnome 3.28.2

**Características de maquinas virtuales x3**

* Procesadores 1
* Memoria 1024 MB
* SO 64 bits
* Disco de 8 GB
* Sistema operativo Linux

**Desarrollo Y Pruebas**

Creación de Maquinas virtuales:  
Para la solución de este trabajo se tuvo que crear un cluster virtual con la herramienta de virtualbox, en el que se emularon 3 máquinas Linux las cuales tuvieron que ser configuradas desde 0 para poder llevar esto a cabo.



Para este caso se utilizaron configuraciones de mysql para configurar el cluster.

También está la posibilidad de hacer la configuración mediante aws, pero se escogió este modelo como prueba, pero los resultados son similares, por comodidad creo que aws funciona mejor.

Ahora, para comenzar con la solucion del modelo matemático se tuvo en cuenta lo que se pedía; el modelo consistía en una línea de N celdas numeradas del 1 al N, donde cada celda puede tener uno de dos valores: 1 si hay un automóvil presente en esa sección de la carretera, o 0 si está vacía. El valor de la celda i en el tiempo t se denota como Rt(i).

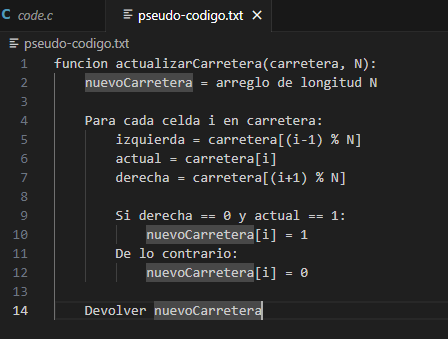
**Las reglas** que nos daban eran las siguientes:

reglas de actualización para cada paso de tiempo

* Si el espacio frente a un automóvil está vacío, se mueve una celda hacia adelante.
* De lo contrario, permanece en su posición actual.

Y Después de cada iteración, se calcula la velocidad promedio de los automóviles. Esto se calcula como el número de automóviles que se mueven en una iteración dividido por el número total de automóviles, y es un valor entre 0 y 1.

Eso y la demás información que nos daba el problema, llegamos a que para implementar estas reglas en forma de pseudocódigo para una carretera de longitud N, se podían utilizar dos arreglos, uno correspondiente a los valores antiguos y otro a los nuevos valores; ¿cómo? Se debe iterar a través de los elementos y actualizar el nuevo arreglo basado en el antiguo. Además, se debe considerar cómo implementar las condiciones de frontera periódicas.  
  
El pseudocódigo que implementa las reglas de actualización del autómata celular para una carretera de longitud N debería ser algo así:



En este pseudocódigo, carretera representa el arreglo de valores de las celdas en el tiempo t, y nuevoCarretera es el arreglo que almacenará los nuevos valores en el tiempo t+1. La expresión (i-1) % N se utiliza para implementar las condiciones de frontera periódicas.

Este código iterará sobre todas las celdas en la carretera, aplicando las reglas de actualización descritas anteriormente. Al finalizar, devolverá el nuevo arreglo de valores de las celdas en el siguiente paso de tiempo.

La solución secuencial del modelo en C seria de esta forma…

**Texto

Descripción generada automáticamente**

Pero como la meta es utilizar otros métodos para mejorar el rendimiento de la solucion.

La función actualizarCarretera implementa las reglas de actualización descritas en el problema, y la función imprimirCarretera se utiliza para imprimir el estado actual de la carretera en cada iteración.

El programa principal (main) crea una carretera inicial con una configuración específica y realiza un número máximo de iteraciones (definido por MAX\_ITERATIONS), aplicando las reglas de actualización en cada iteración y mostrando el estado de la carretera después de cada actualización.

**Para la solución de la paralelización** había que tener en cuenta varios puntos para el calculo entre varios procesadores, como el cálculo Independiente que es la actualización de cada celda en la carretera se puede hacer de forma independiente por diferentes procesadores, la cooperación que es el cálculo del número total de autos que se mueven para calcular la velocidad requiere cooperación entre los procesadores y otros.

Ahora, se considero cómo implementar el esquema paralelo utilizando un modelo de programación de paso de mensajes, asumiendo que tenemos acceso a rutinas simples para enviar y recibir datos entre procesos. Se consideraron dos casos: la comunicación asíncrona (como enviar una carta) y la comunicación sincrónica (como hacer una llamada telefónica).

La mayor diferencia entre ambos códigos esta en esta parte:

**Asincrono**

Texto

Descripción generada automáticamente

Este código divide la carretera en porciones iguales entre los procesos. Cada proceso actualiza de forma independiente su porción local de la carretera y calcula el número local de autos movidos. Luego, se recopila el número local de autos movidos de todos los procesos utilizando MPI\_Allreduce para calcular el número total de autos movidos. El proceso 0 imprime la velocidad promedio basada en el número total de autos movidos. El MPI\_Barrier garantiza la sincronización entre las iteraciones.

**Sincrono**

**Texto

Descripción generada automáticamente**

Este código utiliza MPI\_Send y MPI\_Recv para enviar y recibir el número local de autos movidos entre los procesos. Cada proceso envía su número local a todos los demás procesos y espera a recibir los números locales de los demás procesos antes de continuar. El proceso 0 imprime la velocidad promedio basada en el número total de autos movidos.

Es importante tener en cuenta que, en ambos casos, se debe evitar el bloqueo mutuo (deadlock) asegurándose de que los procesos se sincronicen adecuadamente y se realicen las comunicaciones necesarias.

Los resultados para la ejecución asíncrona y sincrónica fueron los siguientes:

|  |  |
| --- | --- |
| Asincrono | Sincrono |
| 0,5 | 0,599 |
| 0,5 | 0,646 |
| 0,375 | 0,514 |
| 0,6 | 0,49 |
| 0,5 | 0,53 |

|  |  |
| --- | --- |
|  | Promedio |
| Asincrono | 0,495 |
| Sincrono | 0,5558 |

**Conclusiones**

1. Los resultados muestran la velocidad media del coche en cada iteración de la simulación. Se observó que la velocidad promedio cambia con el tiempo, lo que indica cambios en el flujo de tráfico y la congestión en la carretera. Estos resultados se pueden utilizar para comprender el comportamiento del tráfico y tomar decisiones relacionadas con la planificación y gestión de carreteras..
2. Las implementaciones paralelas exploran dos métodos de comunicación: asíncrona y síncrona. Ambos enfoques permiten la distribución y sincronización de cálculos entre procesadores, pero manejan la comunicación de manera diferente.

Comunicación asincrónica: en este método, la comunicación se produce de forma independiente, similar a enviar una carta. Cada proceso envía su información y continúa su ejecución sin esperar una respuesta inmediata. Esto mejora el rendimiento al reducir los tiempos de espera, pero también introduce cierta asincronía en el resultado.

Comunicación Sincrónica: Con este método, la comunicación es sincrónica, similar a una llamada telefónica. Cada proceso envía y recibe información de manera transparente, asegurando que toda la comunicación esté completa antes de continuar. Esto proporciona una mejor sincronización entre procesos, pero aumenta el tiempo de espera general.

La elección entre estos métodos depende de las especificaciones del problema y de los requisitos de tiempo y rendimiento.

1. Diferencia con la implementación secuencial: si se implementa secuencialmente en lugar de usar MPI en paralelo, el código se ejecutará en un solo proceso y no se beneficiará de la potencia de procesamiento distribuida del clúster. Esto dará como resultado un mayor tiempo de ejecución para completar la simulación, especialmente a medida que aumenta la cantidad de elementos y la complejidad del modelo.

**Bibliografía**

* HPC for Research (Sitio web de recursos)
* CUDA Programming (Sitio web de recursos)
* Understanding Speedup in Parallel Computing (Artículo)
* Introduction to Scientific Computing in C++