Implementación y análisis del algoritmo cellular automaton en lenguaje C/C++ usando MPI.

Presentado por:
Angel David Santa Giraldo
Luis Miguel Marulanda Torres
Samir Mateo Soto Buitrago

Presentado a:
Ramiro Andrés Barrios Valencia

HPC: High Performance Computing
Ingeniería de Sistemas y Computación
Universidad Tecnológica de Pereira

Tabla de Contenido.

Resumen

A lo largo de este documento se expone el proceso realizado para llevar al lenguaje de programación C el algoritmo Cellular Automaton enfocado en simular el desplazamiento vehicular hacia la derecha cuando se tienen múltiples “vías”, debido a que en función de la cantidad de vías o de la cantidad de desplazamientos a realizar en varios instantes de tiempo pueden llegar a ser necesarios bastantes recursos computacionales, se ha optado por realizar intentos para optimizar este proceso. Con el fin de analizar hasta qué punto es posible optimizar, se toman mediciones de tiempo de ejecución del algoritmo variando los recursos computacionales en los que se hacen las pruebas, es decir, se tratan de realizar las pruebas utilizando un proceso y un procesador y múltiples procesos en múltiples procesadores de máquinas de cómputo conectadas en forma de cluster.

Introducción

Cellular Automaton representa una solución para diversos problemas de simulación que pueden aparecer en diferentes ciencias, para este caso el algoritmo se utilizará para resolver a través de computo el problema asociado al desplazamiento del tráfico, esto simulando en forma de arreglos las vías y la ocupación de estas para realizar desplazamientos con condiciones específicas de velocidad, posición o de ocupación de celdas predecesoras y sucesoras a la posición del carro en cuestión.

Marco Conceptual

Con el fin de sumar entendimiento al proceso de análisis realizado para llevar al lenguaje C la implementación del algoritmo Cellular Automaton se definen los siguientes términos:

Cellular Automaton

Si bien cellular automaton puede definirse de varias maneras por ser un algoritmo para múltiples fines, para asumir este reto una buena definición es: “A CA is a collection of cells on a grid that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells” (Nagel & Schereckenberg, 1992). Considerando que el movimiento asociado a los vehículos en una simulación de tráfico depende en algunos casos de la celda siguiente y en otros de las celdas anteriores.

Complejidad Computacional

De manera general “La Teoría de la Complejidad estudia la eficiencia de los algoritmos en función de los recursos que utiliza en un sistema computacional (usualmente espacio y tiempo)” (Cortés, 2004). Debido a la complejidad computacional creciente en función del aumento de “vías” e iteraciones en la ejecución de este algoritmo surge la necesidad de identificar técnicas de **optimización** para minimizar el consumo de espacio y tiempo en las máquinas en las que se ejecutará la implementación.

Paralelizar

Ya que ejecutar como programa el algoritmo **Cellular Automaton** implementado en lenguaje C supone un grado de **complejidad computacional** creciente en función de la cantidad de vías, aparece la necesidad de minimizar este tiempo para cuando se hacen bastantes iteraciones para una cantidad grande de vías utilizadas, esta **optimización** es posible gracias a la posibilidad de ejecutar múltiples **procesos** de cómputo en múltiples recursos computacionales interconectados como **cluster** y comunicados a través de la herramienta **MPI**.

Speedup

Speedup traducido al español sin ponerlo en determinado contexto se refiere a un aumento de velocidad, en el caso de la computación el speedup representa la ganancia que se obtiene en la versión paralela de un programa con respecto a su versión secuencial. Para este estudio se ha tomado el speedup como:

$$Speedup = \frac{T*(n)}{T_p(N)}$$

Donde T*(n) hace referencia al tiempo de ejecución secuencial y

$T_P(N)$ hace referencia al tiempo de ejecución paralelo.

Optimización

La optimización es una técnica de transformación de programas, que intenta mejorar el rendimiento del código haciéndolo consumir menos recursos (por ejemplo, CPU, Memoria) y ofrecer mayor velocidad.
Un proceso de optimización de código debe seguir tres reglas:

- La salida del código no debe, en ningún caso, cambiar.
- La optimización debería aumentar la velocidad del programa y de ser posible, el programa debería demandar una menor cantidad de recursos.
- La optimización en si misma deberá ser rápida.

Procesos

Dentro de las operaciones más básicas y la vez más complejas de nuestra PC encontramos los procesos, los cuales están también en las computadoras distribuidas en el clúster utilizado. Siempre que le pidamos a las computadoras del cluster hacer un trabajo computacional, los procesos asumirán el trabajo y de esta manera el microprocesador de cada nodo del cluster dará ejecución al plan que realice el sistema operativo a través de los procesos, “un proceso se trata básicamente de un programa que entra en ejecución. Los procesos son una sucesión de instrucciones que pretenden llegar a un estado final o que persiguen realizar una tarea concreta.” (Castillo, 2019) Lo más importante de este concepto, es de dónde sale un proceso o qué es realmente un programa y un sistema operativo.

Cluster

También conocido como cúmulo o granja, un cluster computacional “es un sistema de procesamiento paralelo o distribuido en el que un conjunto de computadoras independientes e interconectadas entre sí funcionan como un único recurso computacional” (Revista Unam MX, n.d.).
Sí bien cada máquina computacional del clúster puede tener características de hardware diferente para este reto se ha optado por utilizar máquinas iguales, las cuales comparten recursos (como archivos) a través de la red y de un **network file system** implementado en el clúster.

MPI

MPI es una especificación para programación de paso de mensajes, que proporciona una librería de funciones para C, C + + o Fortran que son empleadas en los programas para comunicar datos entre procesos.
MPI es la primera librería de paso de mensajes estándar y portable, especificada por consenso por el MPI Forum, con unas 40 organizaciones participantes, como modelo que permite desarrollar programas que puedan ser migrados a diferentes computadores paralelos.

Network File System (NFS)

“The Network File System (NFS) is a client/server application that lets a computer user view and optionally store and update files on a remote computer as though they were on the user's own computer. The NFS protocol is one of several distributed file system standards for network-attached storage (NAS).” (Rouse et al., 2019)

Ya que para la ejecución del algoritmo en el cluster es necesario que las computadoras compartan algunos archivos, el NFS representa una alternativa ideal para facilitar el acceso a archivos requeridos por las máquinas y a los recursos dispuestos para los cálculos (las matrices a multiplicar).

Marco Contextual

Características del equipo de computo

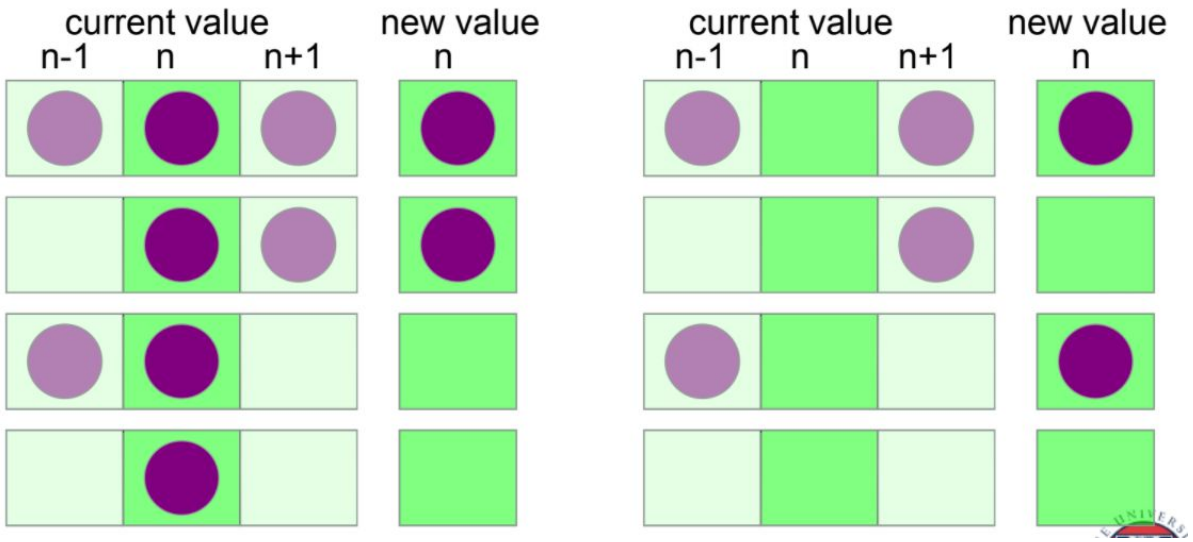
Se testeó en un cluster de instancias de AWS el código correspondiente al algoritmo Cellular Automaton. Dichas pruebas se realizaron con diversas cantidades de nodos, en este caso, de manera secuencial con 1 nodo, y de manera paralela con 2 y 4 nodos.

Características de las instancias donde se realizaron las pruebas:

- Instancia: EC2 t2.micro - Free Tier
- CPU: 1 vCPU Intel Xeon, 2.5 GHz
- Ram: 1 GiB
- SSD: 8 GB
- Sistema operativo: Ubuntu 20.04
- Arquitectura: 64-bit (x86)

Análisis

Antes de intentar paralelizar el algoritmo para optimizarlo es necesario realizar su implementación en forma secuencial, para esto nos basamos en la siguiente información:

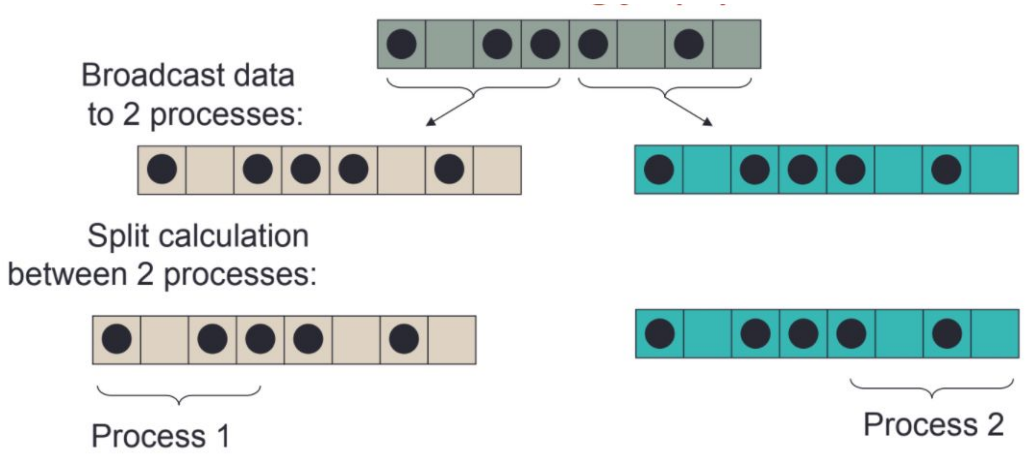


La cual nos permite identificar todos los posibles escenarios, para cuando una vía está ocupada o desocupada y sus vecinos más cercanos se encuentran ambos ocupados, uno de los dos ocupados o ambos desocupados.

Si bien esta implementación de forma secuencial resulta sencilla gracias a que consiste en una simple verificación de las casillas más cercanas a la casilla enésima, este algoritmo puede comenzar a consumir bastantes recursos computacionales en la medida de que aumenten la cantidad de vías a evaluar y la cantidad de desplazamientos a realizar, por lo que comienzan a abrirse las posibilidades a la paralelización con el fin de minimizar los tiempos de espera asociados a la ejecución del código en casos específicos con número de vías muy grande y bastantes intentos.

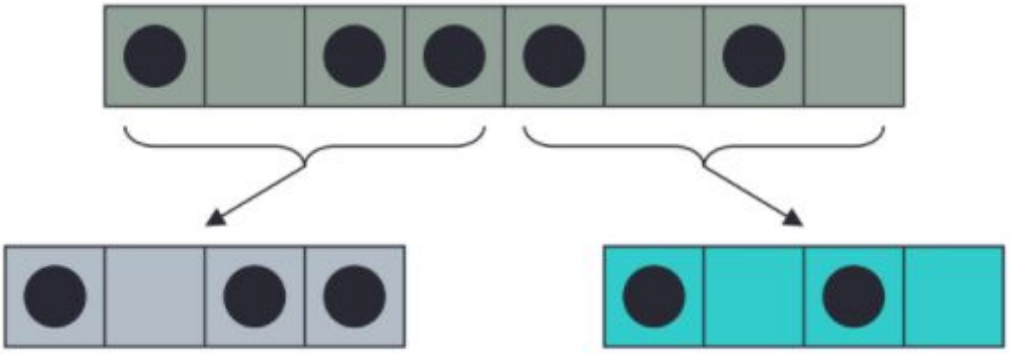
Para la paralelización se ha optado por la propuesta realizada por el docente, la cual consiste en utilizar MPI para utilizar una red de computadoras en cluster con el fin de subdividir las tareas, sin embargo, para esto existen dos posibilidades:

1. Comunicación Colectiva

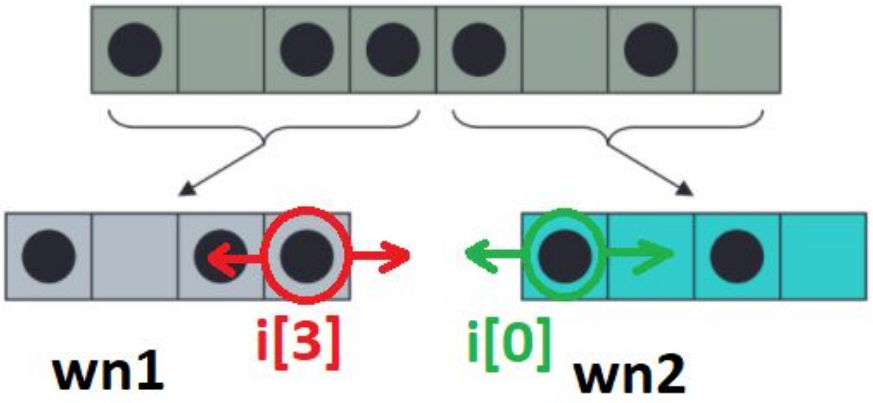


Esta alternativa se presentó como ideal para tratar de hacer varias iteraciones del algoritmo, además es relativamente sencilla ya que los datos se difunden a través de un mensaje en broadcast para todos los procesos alojados en las diferentes máquinas del clúster, sin embargo, esta alternativa representa un alto consumo de datos a nivel de red y tiempos de intercambio de mensajes muy grandes en comparación a la implementación secuencial, lo que implica una desaceleración de la ejecución.

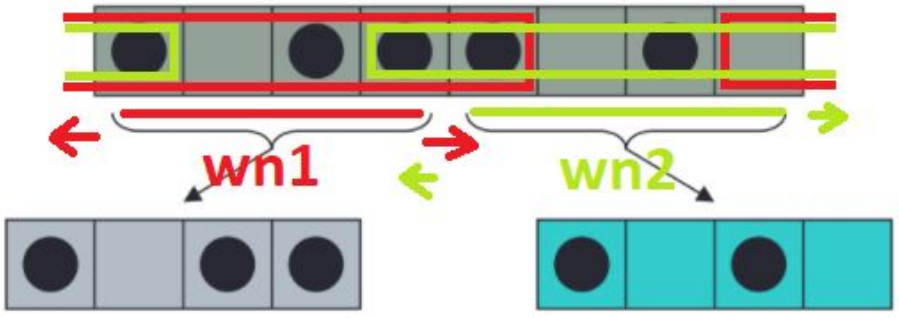
2. Comunicación 1 a 1, point-to-point (P2P) o intercambio de mensajes:



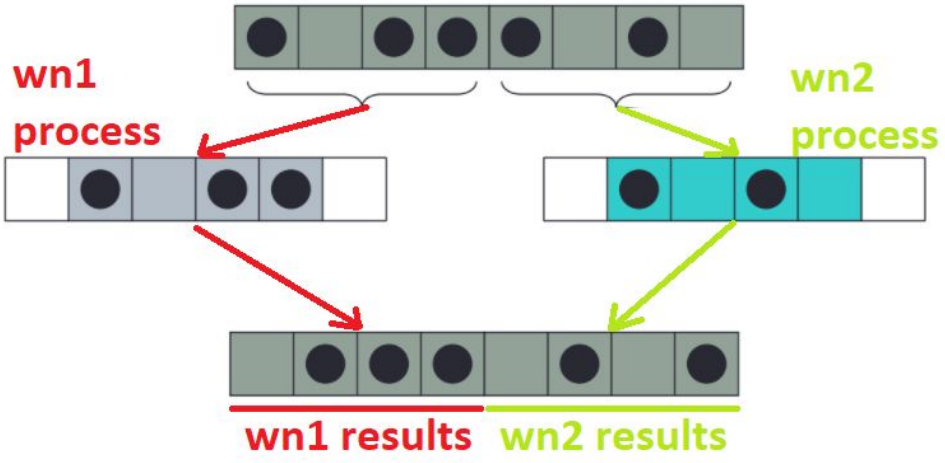
Esta técnica consiste en dividir la cantidad de vías entre los diferentes procesos, por ejemplo, en esta ilustración podemos observar como un conjunto de ocho vías es dividido en dos arreglos de cuatro vías cada uno y cada uno de estos arreglos es enviado a un proceso, cada proceso alojado en máquinas diferentes pero intercambiando mensajes a través de MPI. Si bien esta solución resuelve un gran problema de intercambio de mensajes, debido a que cada nodo trabajador del cluster recibirá menor cantidad de datos y además tendrá menos trabajo por realizar, el principal inconveniente radica en que para los nodos no es suficiente la información que tiene, como por ejemplo en el siguiente caso:



Para un correcto funcionamiento del algoritmo, el elemento i[3] del wn1 requiere conocer el elemento i[0] del wn2, sin embargo, en una comunicación uno a uno esto no es posible, y lo mismo pasa con el elemento i[0] de wn2 el cual requiere conocer el valor de i[3] de wn1, pero para solucionar este inconveniente se ha optado por realizar lo siguiente:



Es decir, que wn1 conociera las vías predecesoras y sucesoras a su conjunto de datos y que así mismo wn2 conociera las vías predecesoras y sucesoras a su conjunto de datos. Así las cosas se podrían aprovechar los beneficios de la comunicación 1-1 y del paralelismo para que cada uno de los nodos haga el trabajo y finalmente reúna los resultados que entrega cada proceso en cada una de las máquinas de la red como se muestra en la siguiente figura:



Además, en el caso de que deban realizarse varias iteraciones, el algoritmo debe ser capaz al final de cada iteración de comunicar su primer y última celda a los nodos adyacentes para continuar con el procesamiento de la siguiente iteración .

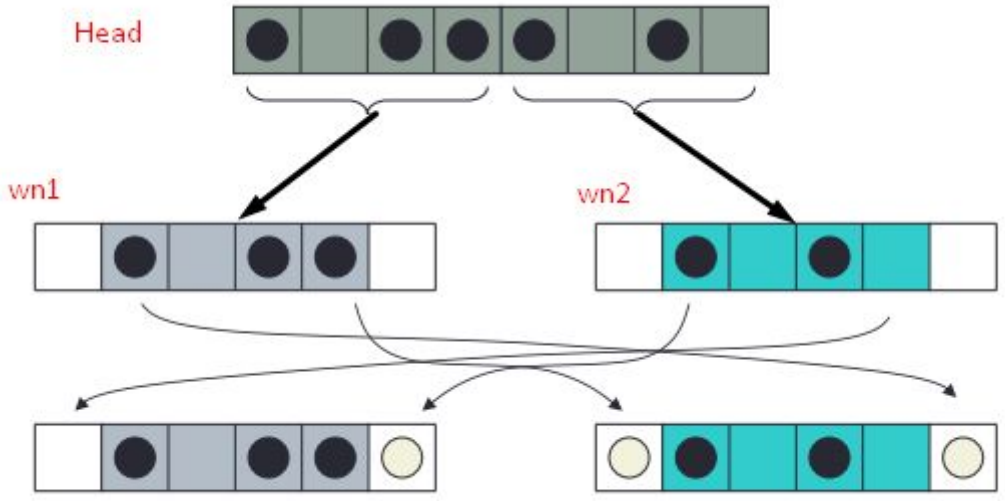
Ya habiendo comprendido el funcionamiento del algoritmo y de haber tomado la decisión de utilizar la comunicación 1-1 (P2P) fue posible proceder al desarrollo, el cual se describe a continuación.

Desarrollo

En primer lugar se desarrolló un algoritmo secuencial con el fin de identificar el algoritmo base a ser paralelizado. Teniendo ya una base funcional (código) y una base teórica en cuanto al funcionamiento de MPI tanto en comunicación colectiva como en comunicación punto a punto, es posible desarrollar la implementación de la siguiente manera:

Inicialmente se crea un vector de tamaño “n” en el nodo head el cual almacenará de manera aleatoria valores de 1 o 0, y será sobre el cual trabajará el algoritmo. Ya teniendo un caso inicial es entonces momento de iniciar la región MPI, ya dentro de la región paralela en primer lugar se reserva espacio para el vector de salida de tamaño “n” y para los vectores que utilizara cada working node, estos vectores serán de tamaño $(n/numranks + 2)$, distribuyendo así, a través de comunicación colectiva (función Scatter) la misma cantidad de celdas para cada nodo y guardando espacio para la celda predecesora y sucesora, Posterior a ello se comunica el valor de estas últimas dos celdas a través de comunicacion 1-1 entre nodos adyacentes.
Gráficamente se vería así:

Ilustración 1: Distribución de trabajo

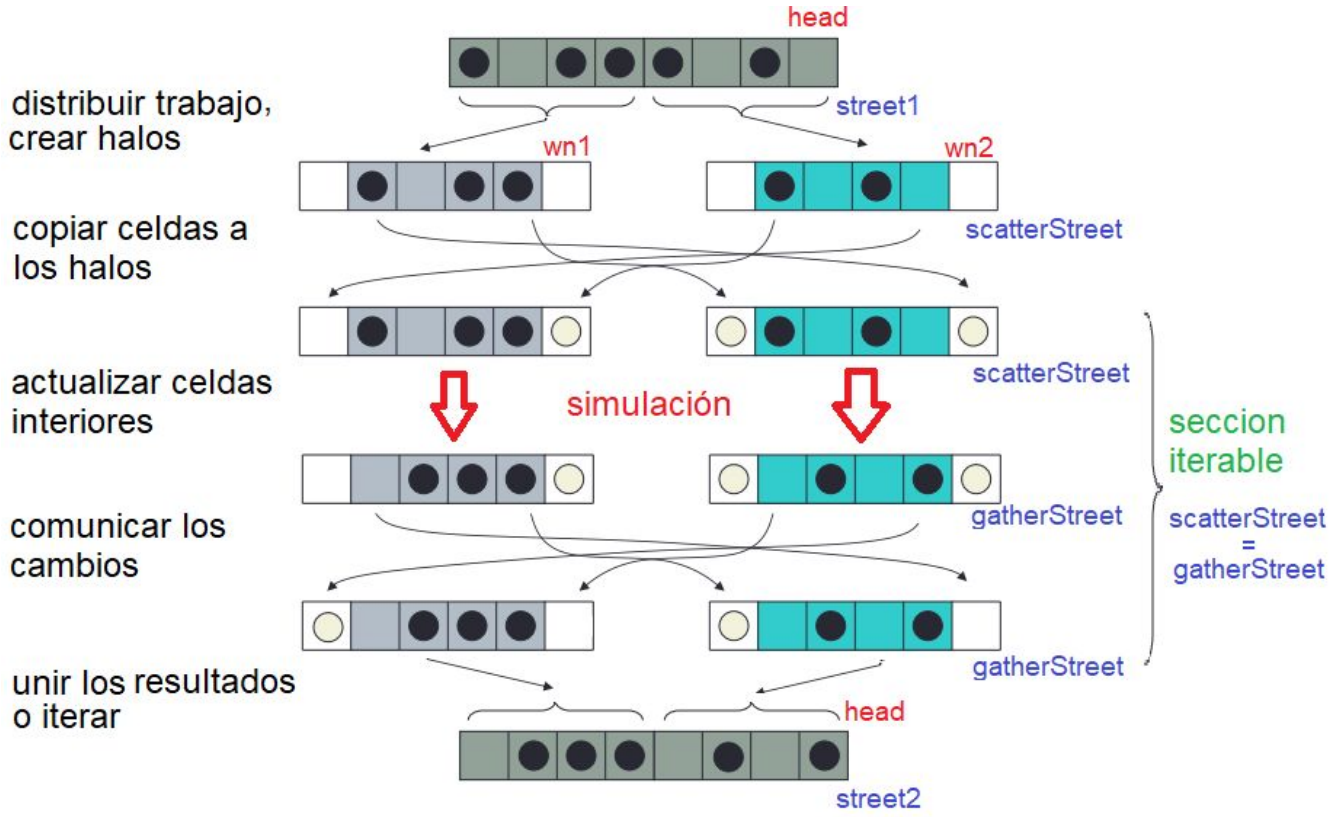


Fuente: Elaboración propia a partir de “Message-Passing Programming - Cellular Automaton Exercise”.

Habiendo distribuido el trabajo entre cada uno de los working node, se procede a ejecutar el algoritmo de simulación de tráfico con base a la teoría estudiada en la sección de Análisis, el resultado de esta ejecución se almacenará en nuevos vectores y se comunicará los cambios relevante entre nodos adyacentes a través de comunicación P2P, estos vectores serán el caso inicial de la siguiente iteración o se reunirán en el vector de salida del nodo head (función Gather).

Gráficamente el algoritmo MPI completo se veria asi:

Ilustración 2: Funcionamiento del algoritmo MPI



Fuente: Elaboración propia a partir de “Message-Passing Programming - Cellular Automaton Exercise”.

En cuanto a problemas encontrados en el desarrollo del algoritmo, este funcionaba en gran medida como se esperaba, sin embargo se tuvo un problema por el cual se obtenían resultados erróneos, este problema se debía a un uso incorrecto en los índices de la función MPI_Scatter, para solucionar dicho problema se tuvo que acomodar la forma en cómo se distribuía el trabajo y luego de esto copiar la información a los Halos a través de comunicación P2P.

Pruebas

Para la realización de las pruebas se utilizó el código Reto 3: Cellular Automaton MPI, el cual proporciona el tiempo de ejecución del algoritmo de simulación de tráfico con implementación MPI. Las pruebas se realizaron en un cluster AWS con un máximo de 8 nodos trabajadores u 8 procesos en términos de ejecución. En el algoritmo se parametrizarón tanto el tamaño del vector Street inicial, como la cantidad de iteraciones que pueden realizarse a un solo vector.

Para organizar los resultados se han dispuesto de unas tablas anexadas al documento con la información plasmada tal que cada tabla representa un número diferente de working nodes usados, las filas representan las combinaciones posibles entre tamaño de las vías e iteraciones realizadas y las columnas representan los 5 intentos realizados para cada combinación de las filas, al final de cada fila se encuentra el promedio obtenido a partir de los 5 intentos realizados. Todos los valores numéricos internos y la fila de promedios se encuentran en milisegundos (ms).

El orden utilizado para realizar las pruebas fue utilizando variando primero la cantidad de working nodes a usar (1, 4 y 8), luego variando el tamaño del vector (10, 100, … , 10’000.000) y finalmente variando la cantidad de iteraciones (10, 100, … , 10.000). Este proceso se realizó 5 veces, cada una representando 1 intento.

Las pruebas se realizadas son:

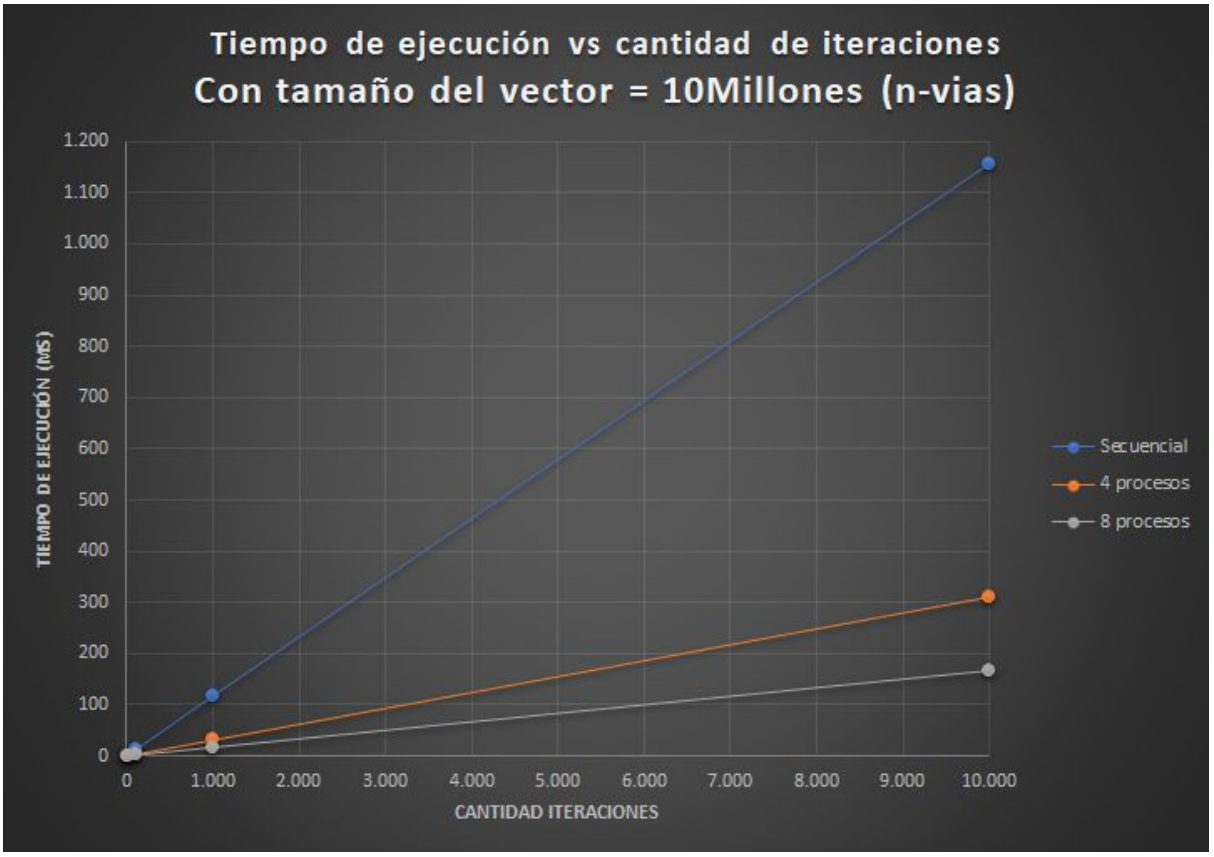
- Código Cellular Automaton con implementación MPI (1, 4 y 8 procesos) y comunicación point-to-point (correspondientes al archivo “cellautoP2P-MPI.c”).

Las tablas con los resultados obtenidos se encuentran en los anexos, al final del documento.

Adicionalmente, realizando un análisis a las tablas es posible generar las siguientes gráficas con el fin de facilitar la comprensión de los resultados obtenidos:

1. Comparativa de los tiempos de ejecución en función de la cantidad de iteraciones, con vectores tamaño 10 millones y variando la cantidad de working nodes:

Ilustración 3. Tiempo de ejecución en función de la cantidad de iteraciones.



Fuente: Elaboración propia.

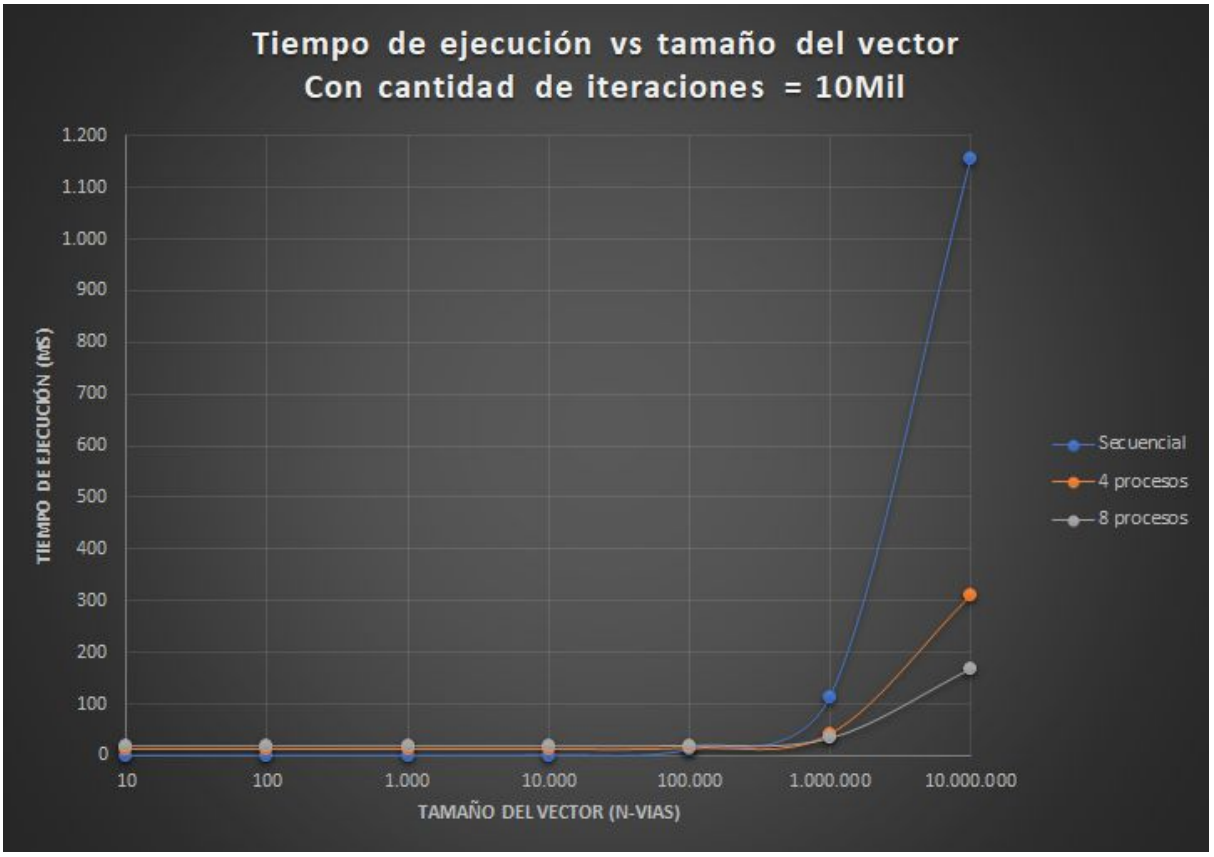
Analizando los resultados obtenidos en las tablas y graficados en la Ilustración 3, observamos un comportamiento lineal muy similar en las 3 tablas en cuanto a los tiempos obtenidos en función de la cantidad de iteraciones realizadas, en el cual para más de 100 iteraciones el tiempo obtenido aumenta en escala de acuerdo a la cantidad de iteraciones, teniendo así que si 1.000 iteraciones tardan 2 segundos, 10.000 tardarían aproximadamente 20 segundos y así sucesivamente.

Por lo tanto podemos decir que la cantidad de iteraciones es un factor consistente a partir de 100 iteraciones, y fácilmente podríamos hacer una estimación del tiempo que podría tardar en ejecutarse el algoritmo con x cantidad de iteraciones.

Adicionalmente se puede apreciar desde ya la optimización que brinda la ejecución con 8 procesos respecto a la ejecución secuencial.

2. Comparativa de los tiempos de ejecución en función del tamaño del vector, tiempos medidos tras completar 10 mil iteraciones y variando la cantidad de working nodes:

Ilustración 4. Tiempo de ejecución en función del tamaño del vector.



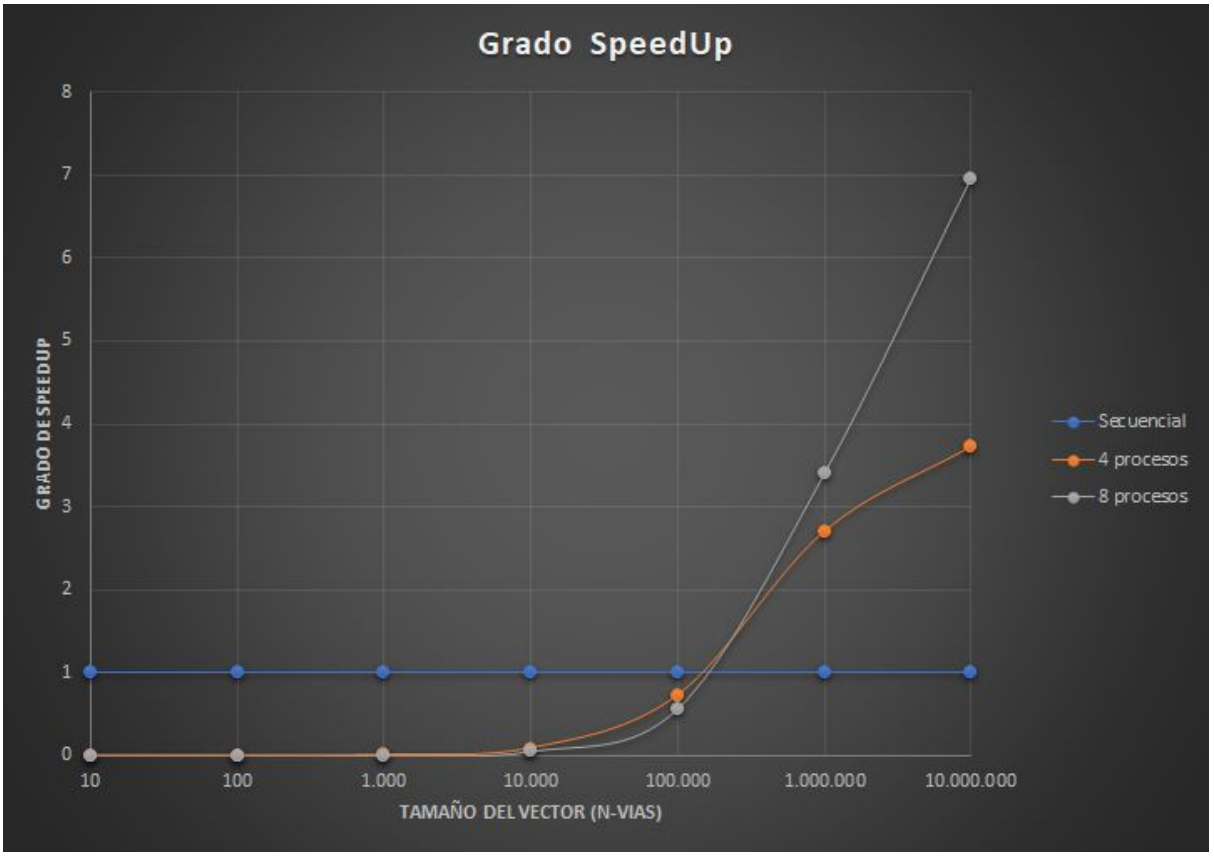
Fuente: Elaboración propia.

Por otro lado, si realizamos una comparativa entre el tiempo obtenido y el tamaño del vector usado en la prueba podremos observar como inicialmente los valores son realmente pequeños y cercanos, pero con tamaños superiores a 1 millón se puede empezar a evidenciar una clara mejora en los tiempos de ejecución.

Y al igual que la ilustración 3, podemos apreciar la mejoría en optimización que puede brindar el uso de una mayor cantidad de nodos/procesos en la ejecución del algoritmo.

3. Grado de SpeedUp respecto a la ejecución secuencial, variando la cantidad de procesos/nodos, tamaño del vector y midiendo el tiempo tras 10 mil iteraciones:

Ilustración 5. Grado de SpeedUp.



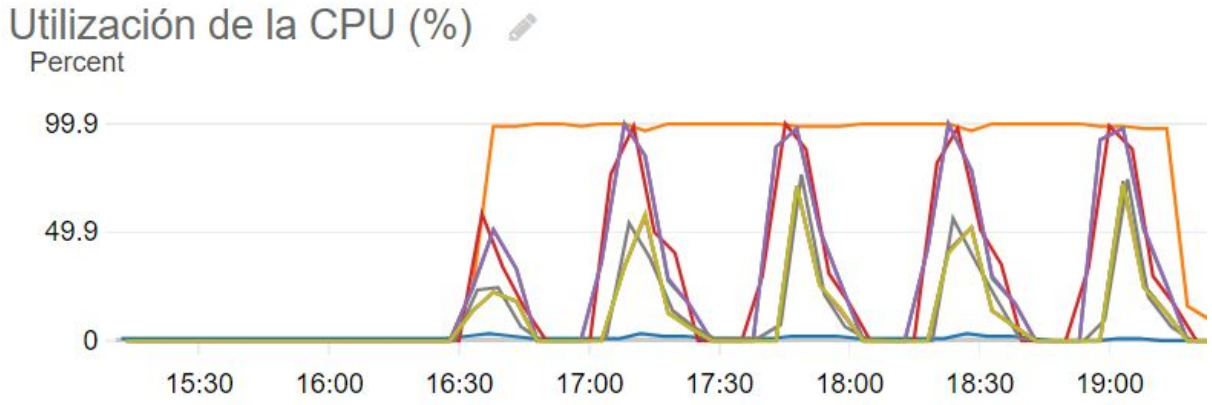
Fuente: Elaboración propia.

Para la elaboración del gráfico del SpeedUp consideramos utilizar el tamaño del vector como un factor determinante puesto que como se observó en la Ilustración 4, el tiempo llega a variar bastante de acuerdo al tamaño del vector, por otro lado la cantidad de iteraciones es un valor de escala constante para el tiempo, entonces no se tuvo en cuenta.

A partir del gráfico de SpeedUp es más claro apreciar la mejora en tiempos de ejecución, donde para vectores pequeños es más óptima una ejecución secuencial, pero para vectores de tamaño mayor a 100 mil empieza a notarse la mejoría, teniendo una mejoría de casi x4 con 4 working nodes y x7 con 8 nodos para vectores tamaño 10 millones.

Finalmente, haciendo uso de la plataforma de AWS nos es posible monitorear el uso de CPU y red de cada nodo a lo largo de la ejecución del algoritmo en dichas máquinas.

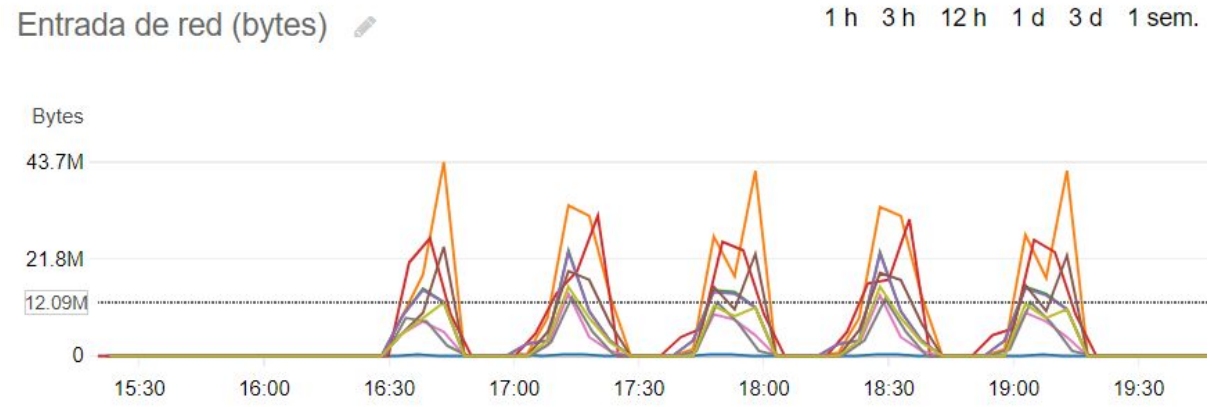
Ilustración 6. Monitor de utilización de CPU en los nodos.



Fuente: Extraído del monitor de recursos de AWS.

En la Ilustración 6 se puede evidenciar varios de los nodos en ejecución (no todos, puesto algunos están superpuestos), donde cada uno de los picos representa cada uno de los 5 intentos realizados en las pruebas. También se puede apreciar como el wn1 (línea naranja) estuvo en constante ejecución, ya que este se encargaba de la ejecución secuencial y paralela en todos los casos, por otro lado está el nodo head (linea azul) el cual tuvo poco uso debido a que solo se encargaba de comunicar los demás nodos, más no de procesar el algoritmo de tráfico vehicular.

Ilustración 6. Monitor de utilización de de red en los nodos.



Fuente: Extraído del monitor de recursos de AWS.

En cuanto al uso de red podemos apreciar que en los casos más extremos un nodo podría llegar a consumir aproximadamente 44 MB de ancho de banda, el cual es un valor relativamente bajo en comparación a la comunicación colectiva, donde se podían llegar a sobrepasar varias GB de transferencia por iteración para estos mismos casos.

Conclusiones

Una de las grandes ventajas de MPI para el caso particular de este algoritmo es la posibilidad de poder crecer en cantidad de worker nodes, acelerando cada vez más la ejecución del algoritmo en comparación con una ejecución secuencial, esto lo podemos evidenciar claramente en la gráfica de speedup comparativa de la ejecución utilizando 1 proceso, 4 procesos u 8 procesos.

Por otro lado, es posible afirmar que hasta el momento una de las optimizaciones más complejas es la que se hace a través de MPI, ya que a nivel de código esta representa un grado de dificultad mayor en comparación a una optimización con OpenMP, threads o procesos.

Para aprovechar al máximo el poder de procesamiento que pueden ofrecer los clusters debemos asegurar en la medida de lo posible usar equipos homogéneos y conexiones de alta velocidad.

Bibliography

Castillo, J. A. (2019, Septiembre 23). *Qué es un proceso informático y qué función tiene*. ProfesionalReview. https://www.profesionalreview.com/2019/09/23/proceso-informatico/

Cortés, A. (2004). Teoría de la complejidad computacional y teoría de la computabilidad . *Revista de Investigación de Sistemas e Informática.*, 102.

Nagel, K., & Schereckenberg, M. (1992). *A cellular automaton model for freeway traffic*. Journal de Physique I.

Revista Unam MX. (n.d.). ¿Qué es un clúster? *Revista Unam*. http://www.revista.unam.mx/vol.4/num2/art3/cluster.htm

Rouse, M., Brunskill, V.-I., Kun, L., & Williams, A. (2019, Mayo). *Network File System (NFS)*. Search Enterprise Desktop. Retrieved Noviembre, 2020, from https://searchenterprisedesktop.techtarget.com/definition/Network-File-System#:~:text=The%20Network%20File%20System%20(NFS,%2Dattached%20storage%20(NAS)

Anexos

Tabla A1. Resultados ejecución secuencial (1 nodo).

Secuencial							
iteraciones	n-vias	Ejecución 1	Ejecución 2	Ejecución 3	Ejecución 4	Ejecución 5	TpromedioEjecución
10	10	0,000018	0,000021	0,000017	0,000019	0,000017	0,000018
	100	0,000030	0,000033	0,000034	0,000029	0,000033	0,000032
	1000	0,000164	0,000160	0,000158	0,000159	0,000162	0,000161
	10000	0,001487	0,001507	0,001500	0,001494	0,001538	0,001505
	100000	0,015070	0,014943	0,014937	0,014916	0,014961	0,014965
	1000000	0,150910	0,149588	0,149238	0,151716	0,149371	0,150165
	10000000	1,518490	1,511834	1,522282	1,503851	1,515863	1,514464
100	10	0,000066	0,000031	0,000035	0,000073	0,000032	0,000047
	100	0,000140	0,000144	0,000144	0,000144	0,000140	0,000142
	1000	0,001248	0,001232	0,001207	0,001212	0,001202	0,001220
	10000	0,012123	0,012241	0,012248	0,012191	0,012116	0,012184
	100000	0,124109	0,123988	0,123678	0,125197	0,123811	0,124157
	1000000	1,251442	1,248086	1,244513	1,248254	1,244635	1,247386
	10000000	12,653052	12,634715	12,639163	12,673835	12,659471	12,652047
1000	10	0,000180	0,000167	0,000171	0,000169	0,000172	0,000172
	100	0,001251	0,001229	0,001245	0,001214	0,001227	0,001233
	1000	0,011481	0,011447	0,011641	0,011410	0,011480	0,011492
	10000	0,115643	0,114313	0,114509	0,114386	0,114527	0,114676
	100000	1,160001	1,153801	1,158124	1,163665	1,154505	1,158019
	1000000	11,688052	11,699997	11,720973	11,686321	11,682080	11,695485
	10000000	118,421023	118,315581	120,793972	118,315251	118,379138	118,844993
10000	10	0,001558	0,001544	0,001542	0,001534	0,001577	0,001551
	100	0,011980	0,011961	0,011936	0,011927	0,011982	0,011957
	1000	0,112809	0,113866	0,112999	0,112867	0,114394	0,113387
	10000	1,136011	1,124182	1,130790	1,125614	1,132794	1,129878
	100000	11,381334	11,335485	11,330450	11,333514	11,362823	11,348721
	1000000	114,339694	114,359339	114,323826	114,395323	114,458866	114,375410
	10000000	1157,210483	1156,575851	1156,330642	1156,125176	1155,148676	1156,278166

Fuente: Elaboración propia.

Tabla A2. Resultados ejecución paralela (4 nodos).

Paralelo (4 wn 4 procesos)								
iteraciones	n-vias	Ejecución 1	Ejecución 2	Ejecución 3	Ejecución 4	Ejecución 5	TpromedioEjecución	GradoSpeedUp
10	10	0,019521	0,020764	0,019396	0,019300	0,019147	0,019626	0,00
	100	0,019608	0,019701	0,023797	0,019043	0,019283	0,020286	0,00
	1000	0,019020	0,023942	0,019991	0,019518	0,020388	0,020572	0,01
	10000	0,020901	0,019220	0,020666	0,019336	0,021032	0,020231	0,07
	100000	0,032208	0,029092	0,029669	0,030522	0,029217	0,030142	0,50
	1000000	0,112570	0,103518	0,105552	0,757085	0,103404	0,236426	0,64
	10000000	0,864599	0,888697	0,864930	0,880219	0,883934	0,876476	1,73
100	10	0,132828	0,148074	0,124699	0,134749	0,126152	0,133300	0,00
	100	0,127800	0,131084	0,148257	0,130640	0,129456	0,133447	0,00
	1000	0,133016	0,146450	0,139443	0,133023	0,137599	0,137906	0,01
	10000	0,131128	0,133180	0,127676	0,127995	0,145632	0,133122	0,09
	100000	0,167502	0,200226	0,170143	0,166515	0,156503	0,172178	0,72
	1000000	1,979704	0,504917	0,493745	1,159782	0,514940	0,930618	1,34
	10000000	3,801751	3,769286	3,773883	4,282429	3,797194	3,884909	3,26
1000	10	1,273225	1,360429	1,240223	1,282598	1,173073	1,265910	0,00
	100	1,189844	1,225024	1,355946	1,259769	1,283711	1,262859	0,00
	1000	1,279113	1,295387	1,330146	1,337447	1,320289	1,312476	0,01
	10000	1,292171	1,255033	1,223065	1,209002	1,302140	1,256282	0,09
	100000	1,436754	1,688086	1,548995	1,487729	1,612594	1,554832	0,74
	1000000	4,282040	4,535958	4,492551	4,300168	4,519806	4,426105	2,64
	10000000	31,596286	31,521488	31,701388	31,695243	31,833225	31,669526	3,75
10000	10	12,686682	12,944920	12,728532	12,896758	12,627762	12,776931	0,00
	100	13,570928	12,328366	12,697297	11,864838	13,953280	12,882942	0,00
	1000	13,603213	13,160836	12,943650	14,123707	12,904307	13,347143	0,01
	10000	13,315027	13,385626	12,400775	12,639883	13,228101	12,993882	0,09
	100000	15,443313	16,605583	15,418249	15,212705	15,361160	15,608202	0,73
	1000000	42,686636	42,217064	41,605466	42,370610	42,243215	42,224598	2,71
	10000000	306,948466	305,262297	305,727300	315,496990	314,578190	309,602649	3,73

Fuente: Elaboración propia.

Tabla A3. Resultados ejecución paralela (8 nodos).

Paralelo (8 wn 8 procesos)								
iteraciones	n-vias	Ejecución 1	Ejecución 2	Ejecución 3	Ejecución 4	Ejecución 5	TpromedioEjecución	GradoSpeedUp
10	10	0,026725	0,035477	0,026214	0,027226	0,026337	0,028396	0,00
	100	0,026578	0,026454	0,031504	0,028897	0,027042	0,028095	0,00
	1000	0,025910	0,027905	0,027201	0,026705	0,026264	0,026797	0,01
	10000	0,026879	0,026403	0,028672	0,026917	0,027444	0,027263	0,06
	100000	0,053195	0,033921	0,035625	0,036502	0,038646	0,039578	0,38
	1000000	0,114973	0,107835	0,105455	0,338527	0,103098	0,153978	0,98
	10000000	0,781162	0,803155	0,780002	0,781359	0,776719	0,784479	1,93
100	10	0,183730	0,207056	0,177432	0,184178	0,179035	0,186286	0,00
	100	0,194173	0,177992	0,226847	0,180270	0,209801	0,197817	0,00
	1000	0,177234	0,251304	0,188304	0,203211	0,207465	0,205504	0,01
	10000	0,188748	0,182681	0,177810	0,178930	0,190444	0,183723	0,07
	100000	0,219545	0,197160	0,219018	0,213789	0,201187	0,210140	0,59
	1000000	0,398815	0,418744	0,426683	0,650169	0,413364	0,461555	2,70
	10000000	2,393898	2,379122	2,342929	4,042821	2,395777	2,710909	4,67
1000	10	1,659572	1,882658	1,705753	1,820642	1,754169	1,764559	0,00
	100	1,699624	1,762992	2,067129	1,710903	1,897619	1,827653	0,00
	1000	2,015074	1,740981	1,792734	3,278883	1,834395	2,132413	0,01
	10000	1,845630	1,992483	2,447950	1,749123	1,849320	1,976901	0,06
	100000	1,908615	1,887771	1,958955	2,042010	1,887060	1,936882	0,60
	1000000	3,368658	3,320511	4,052161	3,340735	4,135457	3,643504	3,21
	10000000	17,281282	17,508442	17,398225	17,305284	18,255924	17,549831	6,77
10000	10	18,558920	17,604393	18,413318	18,527536	18,079744	18,236782	0,00
	100	17,597595	20,186366	17,842497	17,866971	17,623796	18,223445	0,00
	1000	17,954195	17,483808	19,221400	18,071762	18,920349	18,330303	0,01
	10000	19,560871	18,026393	18,489083	19,888774	18,479697	18,888964	0,06
	100000	20,518760	19,604143	19,316829	20,438062	19,586633	19,892885	0,57
	1000000	32,333774	33,594446	33,815685	34,813621	33,651921	33,641889	3,40
	10000000	166,513199	166,553939	165,979434	166,686546	165,807243	166,308072	6,95

Fuente: Elaboración propia.