idetec

Libro de Actas

Estudiantes de Grado y Posgrado

Secretaría de Ciencia, Tecnología y Posgrado

UTN VILLA MARIA

Compilación:
Ing. Marcelo Cejas, Ing. Javier Gonella, Ing. Fabián Sensini

Congreso de Investigaciones y Desarrollos en Tecnología y Ciencia, IDETEC 2020 : Libro de Actas - Estudiantes de Grado y Posgrado / Micaela Mariel Achetoni ... [et al.] ; compilado por Marcelo Oscar Cejas ; Javier Nicolás Gonella ; Fabián Marcelo Sensini. - 1a ed. - Ciudad Autónoma de Buenos Aires : edUTecNe, 2021.
268 p. ; 240 x 150 cm.

ISBN 978-987-4998-69-9

1. Ingeniería. 2. Tecnologías. 3. Medio Ambiente. I. Achetoni, Micaela Mariel. II. Cejas, Marcelo Oscar, comp. III. Gonella, Javier Nicolás, comp. IV. Sensini, Fabián Marcelo, comp.

CDD 607.3

Edición y Diseño:

Universidad Tecnológica Nacional – República Argentina
Rector: Ing. Héctor Eduardo **Aiassa**
Vicerrector: Ing. Haroldo **Avetta**
Secretaria Académica: Ing. Liliana Raquel **Cuenca Pletsch**

Universidad Tecnológica Nacional – Facultad Regional Villa María
Decano: Ing. Pablo Andrés **Rosso**
Vicedecano: Ing. Franco **Salvático**

edUTecNe – Editorial de la Universidad Tecnológica Nacional
Coordinador General a cargo: Fernando H. **Cejas**
Director Colección Energías Renovables, Uso Racional de Energía, Ambiente: Dr. Jaime **Moragues.**

Queda hecho el depósito que marca la Ley Nº 11.723

Sarmiento 440, Piso 6 (C1041AAJ)
Buenos Aires, República Argentina
Publicado Argentina – Published in Argentina

ISBN 978-987-4998-69-9

CÓMPUTO PARALELO PARA LA RESOLUCIÓN DE LA MULTIPLICACIÓN DE MATRICES DE GRAN TAMAÑO

Karvin O. Díaz-Acevedo [1] , Alejo G. Ponce de León [1] , Paola G. Caymes-Scutari [1,2] , Germán Bianchini [1]
1 Laboratorio de Investigación en Cómputo Paralelo/Distribuido (LICPaD)
Facultad Regional Mendoza/Universidad Tecnológica Nacional
Rodríguez 273 (M5502AJE) Mendoza, +54 261 5244579
2 Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
karvin.diaz@alumnos.frm.utn.edu.ar, alejo.ponce@alumnos.frm.utn.edu.ar,
gbianchini@frm.utn.edu.ar, pcaymesscutari@frm.utn.edu.ar

Resumen

Nuestro objetivo con este trabajo de investigación es poder determinar la forma más eficiente para realizar multiplicaciones de matrices de gran volumen, cálculos que son de gran utilidad en diversas áreas con aplicaciones tales como resolución de sistemas de ecuaciones de muchas variables, cálculo numérico, siendo también utilizado con frecuencia en el cálculo de *microarrays*, en el área de la bioinformática, entre otros. Esta investigación se puede llevar a cabo por los integrantes gracias a la formación en cómputo paralelo y distribuido en el marco del Laboratorio de Investigación en Cómputo Paralelo/Distribuido (LICPaD) de la Universidad Tecnológica Nacional.

La multiplicación se define dadas dos matrices A y B, se dicen multiplicables si el número de columnas de A coincide con el número de filas de B. En la nueva matriz C, los elementos C_{ij} parten del producto que se obtiene multiplicando cada elemento de la fila i de la matriz A por cada elemento de la columna j de la matriz B y sumándolos.[1] Cabe destacar que a medida que aumenta el tamaño de las matrices el volumen de cómputo también aumenta considerablemente.

Para este estudio comparativo, se propone desarrollar dos algoritmos: uno que aproveche la capacidad de cómputo de un *cluster* de computadoras parte del Laboratorio de Investigación en Cómputo Paralelo/Distribuido (LICPaD) ubicado en la UTN-FRM, y otro algoritmo cuya ejecución sea secuencial. Una vez desarrollados, procederemos a realizar una serie de pruebas con matrices de distintos tamaños y con distinta cantidad de nodos.

Para concluir, luego de obtenidos los resultados de tiempos de cálculo para cada experimento podremos realizar un análisis de métricas como el *Speedup*, la Eficiencia, el Balanceo de Carga, etc., [2] y en base a ellos determinar en qué medida o en qué casos el cómputo paralelo resulta la forma más conveniente y/o eficiente para este cálculo, y también implementar mejoras y optimizaciones en los algoritmos utilizados.

Palabras clave: Multiplicación; Matrices; Eficiente; Algoritmos

1. Introducción

Para comenzar con la presentación del tema, se establece el marco teórico del problema. En el cálculo de matrices hay una condición fundamental. Sean A y B matrices de tamaño n*m (siendo m y n números enteros) se puede realizar la multiplicación, si y sólo si, la cantidad de columnas de A es igual a la cantidad de filas de B. Si esto se cumple se obtiene una matriz C, la cual siempre tendrá el número de filas igual al número de filas de la matriz A y el número de columnas igual al número de columnas de B. Una vez presentada la condición, el cálculo se resuelve al multiplicar los elementos de la fila de la matriz A con la columna de la matriz B como lo indica la Fig. 1.

$$\boldsymbol{Z}_{nxm} \cdot \boldsymbol{Y}_{nxm} = \begin{pmatrix} z_{11} & \cdots & z_{1m} \\ \vdots & \ddots & \vdots \\ z_{n1} & \cdots & z_{nm} \end{pmatrix} \cdot \begin{pmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nm} \end{pmatrix} = \begin{pmatrix} z_{11} \cdot y_{11} + \cdots + z_{1m} \cdot y_{n1} & \cdots & z_{1m} \cdot y_{1m} + \cdots + z_{1m} \cdot y_{nm} \\ \vdots & \ddots & \vdots \\ z_{n1} \cdot y_{11} + \cdots + z_{nm} \cdot y_{n1} & \cdots & z_{n1} \cdot y_{1m} + \cdots + z_{nm} \cdot y_{nm} \end{pmatrix}$$

Fig. 1.- Proceso de multiplicación de matrices.

2. Proceso de resolución secuencial

El hecho de la palabra secuencial significa que los pasos del programa se ejecutan seguidamente uno después del otro, este concepto conlleva a sí mismo que estamos trabajando sobre un único procesador que recorrerá dichos pasos. Una vez entendida la manera de procesar pasamos a la programación del cálculo.

Primero se solicita el tamaño de las matrices a calcular. Con esa información, por medio de una función, se llenan las matrices y finalmente por medio de iteraciones anidadas se recorren tanto las filas como las columnas de ambas matrices con el fin de realizar en cálculo y guardar los resultados en la nueva matriz, en las celdas correspondientes.

3. Proceso de resolución paralelo

Siguiendo el algoritmo de multiplicación de matrices utilizamos cómputo paralelo para intentar obtener mayor eficiencia en la resolución de los cálculos, paralelizando la multiplicación de las columnas. Esto resulta gracias a la librería MPI, la interfaz de paso de mensajes que define la sintaxis y la semántica de las funciones diseñadas para ser usada en programas que exploten la existencia de múltiples procesadores.[3]

3.1. Modelo *Master-Worker*

Para resolver esto utilizamos un modelo *Master-Worker* en el cual el proceso 0 actuará de *Master*, indicando qué debe ser calculado por cada proceso *Worker*, y enviando la información necesaria para que puedan realizar estos cálculos. La Fig. 2 ilustra el modelo a seguir.

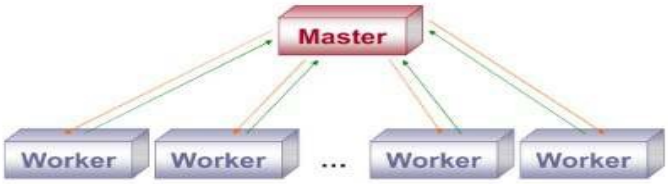

Fig. 2.- Modelo *Master-Worker*

3.2. Algoritmo 1 - Balanceo de carga dinámico

Dado que nuestro objetivo era paralelizar el cálculo de la matriz, se mantuvo como porción secuencial la asignación dinámica y el llenado de matrices. Una vez realizado esto el proceso Máster procede a comenzar la porción paralela del programa enviando a los procesos *Worker* el tamaño de la matriz B y tamaño de las filas de A para que puedan realizar la asignación de espacio para estas en su propio espacio de memoria. Posteriormente se le envía a cada proceso *Worke*r la matriz B que es la que utilizaremos para que cada proceso pueda calcular columnas de esta. Ambos envíos se realizan mediante la función ***MPI_Bcast.*** Luego se comienza un bucle que envía filas de la matriz A entre los *Worker*. Cada *Worker* que haya terminado de resolver su multiplicación, envía el resultado al Máster con ***MPI_Send*** y la petición de una nueva fila si la hubiese. Finalmente el Máster es el encargado de recopilar la información de los *Worker* y ubicarlos en la matriz resultado.

3.2.1 Balanceo de cargas dinámico

En este primer algoritmo se utiliza un balanceo de cargas dinámico, porque se considera que no todos los nodos son homogéneos por lo tanto el cálculo en cada nodo tomará distintos tiempos, y para tratar de evitar los tiempos ociosos de los nodos, el algoritmo se implementa de manera que los *Workers* le pidan más trabajo al máster una vez que terminan el trabajo previo. La granularidad se mantuvo al nivel de 1 fila de la matriz A y 1 columna matriz B a cada nodo para realizar el cálculo, es considerada una granularidad intermedia, dado que una granularidad gruesa sería pasar 1 fila de la matriz A y todas las columnas de la matriz B a cada nodo. Y una granularidad demasiado fina sería pasar 1 posición de la fila de la matriz A y 1 posición de la columna de la matriz B a cada nodo.[4]

3.3. Algoritmo 2 - Balanceo de carga estático

Al igual que el Algoritmo 1, se mantuvo como porción secuencial la asignación dinámica y el llenado de matrices.

A diferencia del anterior, el *Master* envía tanto la matriz A como la B y la dimensión de las matrices por ***MPI_Bcast***. Cuando hablamos de dimensión nos referimos tanto al número de filas como de columnas, es decir, solo acepta matrices cuadradas. Inmediatamente después del envío el *Máster* comenzará el proceso de recepción de datos junto con ***MPI_recv***, proveniente de cualquier fuente mediante la etiqueta ***MPI_ANY_SOURCE*** y ordenándolos en la matriz Resultante.

Primeramente el *Worker* recibe los datos ya mencionados del Máster. A partir de la cantidad de columnas de la Matriz y el número de proceso del *Worker*, él mismo calcula la columna desde donde debe comenzar el producto y la última para terminar el cálculo. Luego se entra en el bucle que resuelve las multiplicaciones. Cada resultado se envía instantáneamente al *Master* por ***MPI_Send***, con los resultados **y además con la posición en la que debe ubicarse dicho resultado en la matriz Resultado**. Finalmente, al terminar el proceso, el *Worker* libera el espacio que ocuparon los datos recibidos.

3.3.1. Balanceo de carga estático

Ahora nos encontramos con la hipótesis de que todos los nodos son iguales, el único tiempo ocioso a considerar es en la división de las columnas, debido a que si la división de columnas no es exacta respecto al número de nodos, el último nodo se encargará de tomar las columnas faltantes, lo que implica más procesamiento para ese nodo. La granularidad es considerada gruesa, ya que debo enviar las dos matrices de tamaños bastante considerables.[4]

4. Experimentos

El objetivo de la realización de los experimentos es poder observar el rendimiento del procesamiento secuencial comparado al procesamiento paralelo y luego ambos códigos paralelos entre sí. Para ello contamos con un cluster compuesto por 10 PCs con Procesador AMD FX-6300, 16GB de memoria RAM cada uno, que se encuentran conectados mediante Red Gigabit Ethernet 1000 Mbps.

A continuación se plantean una serie de tres experimentos en los que mediremos el tiempo de los programas y a partir de ello se obtendrán el *Speedup* y la respectiva eficiencia. Ver y comparar las tablas Tabla 1-1.1, Tabla 2-2.2 y Tabla 3-3.3

Experimento 1: matriz 100x100
Tiempo de cálculo secuencial: 0.036 segundos

Tabla 1.- Resultados Algoritmo 1 matriz 100x100

Algoritmo 1	2 nodos	4 nodos	6 nodos	8 nodos
Tiempo (en segundos)	0.799	0.456	0.656	0.55
Speedup	0.045	0.08	0.055	0.072
Eficiencia	2.26%	2%	0.91%	0.9%

Tabla 1.1.- Resultados Algoritmo 2 matriz 100x100

Algoritmo 2	2 nodos	4 nodos	6 nodos	8 nodos
Tiempo (en segundos)	0.036	0.029	0.029	0,040
Speedup	0.996	1.235	1.247	0.886
Eficiencia	49.81%	30.88%	20.79%	11.08%

Experimento 2: matriz 1000x1000
Tiempo de cálculo secuencial: 46.517 segundos

Tabla 2. - Resultados Algoritmo 1 matriz 1000x1000

Algoritmo 1	2 nodos	4 nodos	6 nodos	8 nodos
Tiempo (en segundos)	102.529	50.694	44.806	44.513
Speedup	0.453	0.915	1.035	1.042
Eficiencia	22%	22.88%	17.26%	13.03%

Tabla 2.2. - Resultados Algoritmo 2 matriz 1000x1000

Algoritmo 2	2 nodos	4 nodos	6 nodos	8 nodos
Tiempo (en segundos)	42.140	14.368	9.876	7.765
Speedup	1.101	3.237	4.698	5.975
Eficiencia	55.06%	80.93%	78.31%	74.69%

Experimento 3: matriz 2000x2000
Tiempo de cálculo secuencial: 397.323 segundos

Tabla 3. Resultados Algoritmo 1 matriz 2000x2000

Algoritmo 1	2 nodos	4 nodos	6 nodos	8 nodos
Tiempo (en segundos)	543.129	223.768	192.202	183.204
Speedup	0.723	1.787	2.058	2.135
Eficiencia	36.15%	44.67%	34.31%	26.68%

Tabla 3.3. Resultados Algoritmo 2 matriz 2000x2000

Algoritmo 2	2 nodos	4 nodos	6 nodos	8 nodos
Tiempo (en segundos)	390.406	131.454	84.308	62.663
Speedup	1.013	3.010	4.693	6.314
Eficiencia	50.67%	75.25%	78.22%	78.93%

5. Análisis de resultados y conclusiones

Podemos observar que a medida que las matrices crecen de tamaño, tiende a mejorar el *Speedup* y la Eficiencia del Algoritmo 1, es decir que aproximadamente a partir de una matriz de 4000x4000 se notará una eficiencia mayor a 60%. Para llegar a estas conclusiones observar Fig. 3 y Fig. 4. Luego, observando el Algoritmo 2 notamos que tanto el *Speedup* y la Eficiencia son bastante superiores, ver Fig. 5 y Fig. 6, pero hay que recordar que este algoritmo tiene un gran uso de memoria, por lo que al momento de utilizar matrices muy grandes en un almacenamiento limitado, puede llegar a no funcionar, no así como el Algoritmo 1, que si bien puede sufrir de la misma falla, está se dará mucho más adelante. Estas observaciones vienen acompañadas al concepto de escalabilidad, el Algoritmo 1 es mucho más escalable que el Algoritmo 2, es decir mediante un supuesto crecimiento de datos, el Algoritmo 1 tendrá mayor capacidad para manejarlos, lo que también está fuertemente acompañado al uso de balanceo dinámico. Concluyendo, el algoritmo que utilizaremos se verá muy influenciado por la cantidad de nodos y el tamaño de las matrices, además de tener un buen tamaño de memoria disponible.

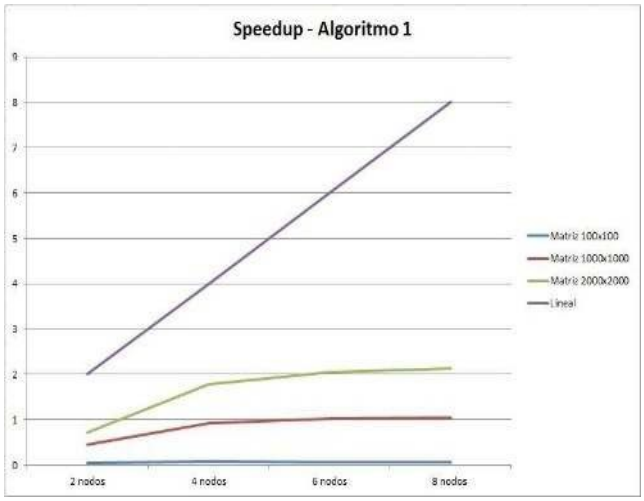

Fig. 3.- Análisis *Speedup* Algoritmo 1

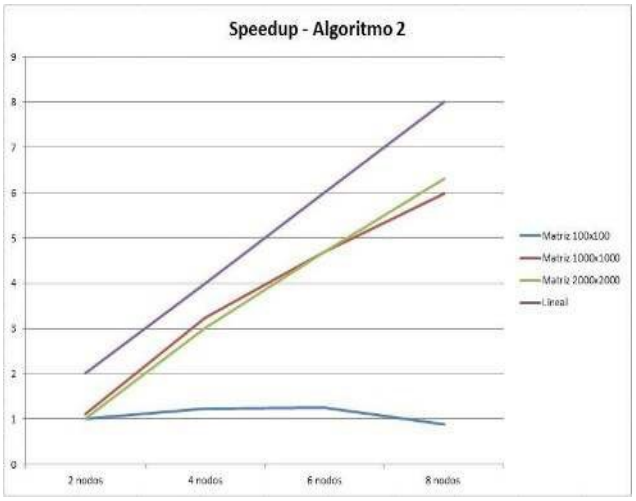

Fig. 4.- Análisis *Speedup* Algoritmo 2

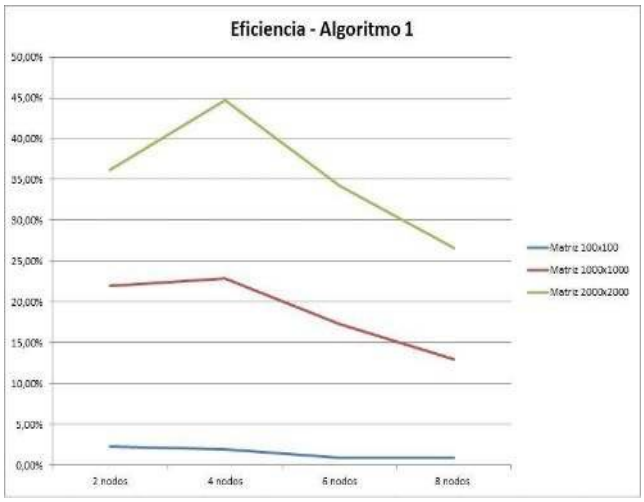

Fig. 5.- Análisis *Speedup* Algoritmo 1

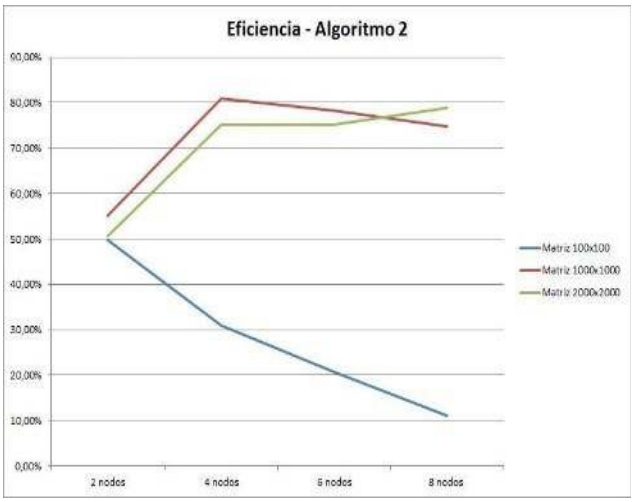

Fig. 6.- Análisis *Speedup* Algoritmo 2

6. Referencias bibliográficas

[1] Nykamp, Duane. "Multiplying matrices and vectors". Math Insight. https://mathinsight.org/matrix_vector_multiplication Accedida en febrero de 2021.

[2] Wilkinson, B., Allen, M. (2005) Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Pearson Prentice Hall.

[3] MPI Forum – MPI Documents. https://www.mpi-forum.org/docs/. Accedida en febrero de 2021.

[4] Morrison , Cluster Computing Theory