Revista Innovación y Software
Vol. 1, No. 2, Mes Septiembre-Febrero, 2020
ISSN: 2708-0935
Pág. 27-39
https://revistas.ulasalle.edu.pe/innosoft

Tipo de artículo: Artículos originales
Temática: Programación paralela y distribuida
Recibido: 04/09/2020 | Aceptado: 10/09/2020 | Publicado: 30/09/2020

Resolución paralela de sistemas triangulares

Parallel resolution of triangular systems

Ihosvany Rodríguez González[1*][0000-0003-0212-9556], **Anié Bermudez Peña**[2][0000-0002-1387-7472]
[1] Centro Nacional de Biopreparados, Bejucal, Mayabeque, Cuba. ihosvany@biocen.cu
[2] Universidad de las Ciencias Informáticas, La Habana, Cuba. abp@uci.cu

[*] Autor para correspondencia: ihosvany@biocen.cu

Resumen

La resolución de sistemas triangulares es un núcleo computacional ampliamente utilizado en diversas aplicaciones científicas. Esta investigación realiza la implementación y comparación de varios algoritmos paralelos frente a un algoritmo secuencial eficiente para la resolución de sistemas triangulares. Los algoritmos se distinguen por la forma de particionado de la matriz y la asignación a los procesadores. Se realiza el análisis del comportamiento de los algoritmos en la solución de sistemas de ecuaciones lineales triangulares superiores en un clúster de computadoras. Para ello se tienen en cuenta las métricas de tiempo aritmético, tiempo de comunicaciones, aceleración y eficiencia máxima. Se realizaron experimentos para cada algoritmo con distintos tamaños de matrices sobre varios procesadores. El algoritmo con mejores resultados fue el que divide por bloques las filas de la matriz y aplica una distribución cíclica en el clúster.

Palabras clave: particionado de matrices, programación paralela, sistema triangular.

Abstract

The resolution of triangular systems is a computational nucleus widely used in various scientific applications. This research performs the implementation and comparison of several parallel algorithms against an efficient sequential algorithm for solving triangular systems. The algorithms are distinguished by the way of partitioning the matrix and the allocation to the processors. The analysis of the behavior of the algorithms is performed in the solution of systems of linear superior triangular equations in a cluster of computers. For this, the arithmetic time, communication time, speed-up, and maximum efficiency metrics are taken into account. Experiments were

performed for each algorithm with different matrix sizes on various processors. The algorithm with the best results was the one that blocks the rows of the matrix and applies a cyclical distribution in the cluster.

***Keywords**: matrix partitioning, parallel programming, triangular system.*

1. Introducción

La resolución de sistemas triangulares es un núcleo computacional ampliamente utilizado en diversas aplicaciones científicas[1]. Esto se debe a la relación existente entre este tipo de sistemas y el desarrollo de sistemas de ecuaciones genéricos. Generalmente, la resolución de un sistema de ecuaciones genérico se traduce en la resolución de dos sistemas triangulares una vez realizada la triangularización de la matriz [2].

La computación paralela se ha convertido en uno de los pilares más potentes de las computadoras. Para muchos problemas que requieren computación intensiva basta con disponer de un conjunto de computadoras conectadas mediante una red. De esta forma se han difundido los clústeres de procesadores como herramienta típica para la programación paralela [3].

El problema que se aborda en esta investigación trata de la resolución de sistemas triangulares. Para ello se realiza una comparación detallada de varios algoritmos paralelos frente a un determinado algoritmo secuencial eficiente. Los algoritmos paralelos son: Escalar, TRB (*Triangular Row Block*), TRS (*Triangular Row Scatter*) y TRBS (*Triangular Row Block Scatter*). Se realiza el análisis del comportamiento del algoritmo secuencial y de dichos algoritmos paralelos para la solución de sistemas de ecuaciones lineales triangulares superiores en el clúster de computadoras con las siguientes características: 4 HP Proliant DL180 G6, con dos procesadores Quad Core Intel Xeon serie E5520, 2.26 GHz., memoria RAM 24 GB, DDR3 1066 MHz., 2 puertos Gigabit Ethernet, almacenamiento local 60 GB SATA2.

2. Marco teórico

En un primer momento es necesario mostrar el algoritmo secuencial eficiente que soluciona los sistemas triangulares. Dada la matriz $U \in \Re^{nxn}$, $u_{ij}=0$ si $i>j$, y $b \in \Re^{n}$, el algoritmo siguiente resuelve el sistema de ecuaciones $Ux=b$.

Para j=n-1, n-2,.....,0

$x_j = b_j/u_{jj}$ *//Resolver*

Para i=0,1,.....,j-1

$b_i = b_i - u_{ij}x_j$ *//Actualizar*

Fin

Fin

Revista Innovación y Software
Vol. 1, No. 2, Mes Septiembre-Febrero, 2020
ISSN: 2708-0935
Pág. 27-39
https://revistas.ulasalle.edu.pe/innosoft

Los diferentes algoritmos paralelos para la resolución de sistemas triangulares superiores se pueden clasificar en función de las distribuciones de datos realizadas [4]. El primer factor a tener en cuenta para realizar la distribución de una matriz (en este caso triangular superior, U) es si esta se realiza por filas o por columnas. Es decir, a los diferentes procesadores se les asignará un conjunto de filas de la matriz U, o bien un conjunto de columnas de esta. En el desarrollo de este caso de estudio se aplican solo distribuciones por filas, de esta forma se reduce el número de las posibles divisiones de la matriz U.

2.1. Distribuciones de datos

Existen varias formas en que se puede dividir la matriz U por filas, así como asignaciones de dichas particiones a los diferentes procesadores. De la adecuada elección del particionado y la asignación (distribución de datos) de la matriz U dependerá la eficiencia del algoritmo paralelo resultante [5]. Dicha elección debe hacerse atendiendo a dos factores: equilibrado de la carga y reducción del coste de comunicación.

A continuación, se introducen las formas de particionado de la matriz U y posteriormente se enumeran las formas en las que se puede realizar la asignación sobre los diferentes procesadores de la aplicación paralela.

Particionado de la matriz [6]

El particionado por filas de la matriz U puede ser Escalar o por Bloques. En el particionado por bloques se divide la matriz de *n* filas en bloques de filas de tamaño *s* (con *s>=1*), mientras que en el particionado escalar se consideran tantas divisiones como filas posee la matriz. Por tanto, el particionado escalar es el particionado por bloques cuando *s=1*. En la Figura 1 se muestran los dos tipos de particionado de la matriz U[7].

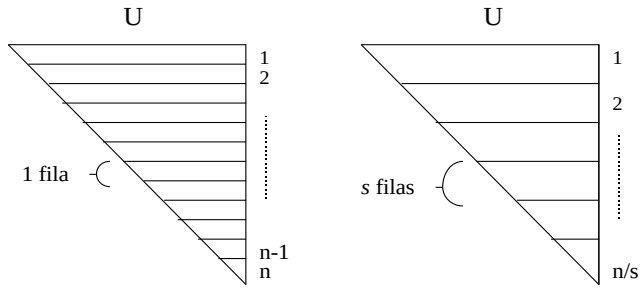


Figura 1. Particionado Escalar y por Bloques de la matriz triangular superior U.

Asignación a los diferentes procesadores

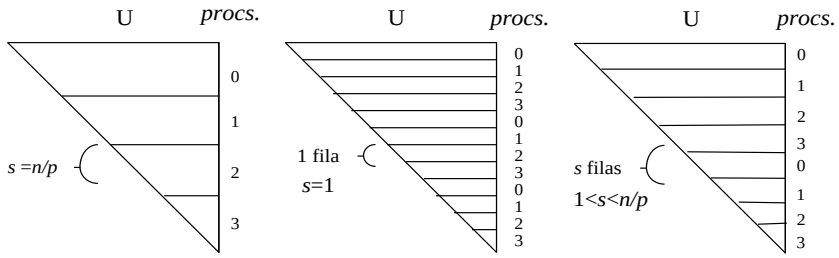
La asignación de las particiones a los procesadores *p*, se denomina por Bloques si el número de particiones coincide con el número de procesadores. Mientras que, si el número de particiones es mayor que *p*, la asignación se realiza de forma Cíclica.

Revista Innovación y Software
Vol. 1, No. 2, Mes Septiembre-Febrero, 2020
ISSN: 2708-0935
Pág. 27-39
https://revistas.ulasalle.edu.pe/innosoft

Cuando $p<<n$ el número de distribuciones de datos posibles se reduce a tres. Las características de estas tres distribuciones se muestran en la Tabla 1 y la Figura 2.

Tabla 1. Diferentes distribuciones de la matriz triangular.

Algoritmo	**Distribución**	**Particionado**	**Asignación**	**Notas**
TRB y Escalar	Bloques de filas	Escalar	Bloques	s=n/p
TRS	Cíclica por filas	Escalar	Cíclica	s=1
TRBS	Cíclica por bloques de filas	Bloques	Cíclica	1<s<n/p



a) Distribución por bloques b) Distribución Cíclica c) Distribución Cíclica por bloques

Figura 2. Diferentes distribuciones de la matriz triangular *U* sobre cuatro procesadores.

En la Figura 3 se brinda una visión global de un ejemplo de distribución por bloques de los datos en 4 procesadores.

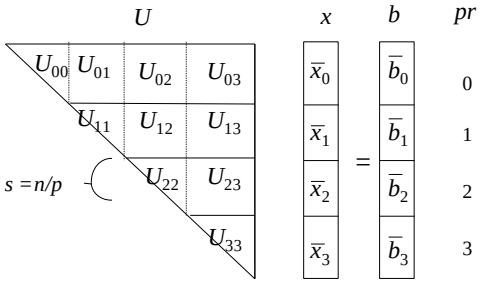


Figura 3. Distribución por bloques de filas del sistema lineal $Ux=b$ entre $p=4$ procesadores.

Para completar la distribución de datos del problema de resolver el sistema Ux=b es necesario comentar que los vectores b y x quedan distribuidos de igual forma que U.

2.2. Algoritmos paralelos implementados

En esta sección se realiza una descripción de los cuatro algoritmos paralelos para la resolución de sistemas triangulares. Al algoritmo paralelo que utiliza la distribución por bloques se le denomina TRB (*Triangular Row Block*), y si la

implementación se hace orientada a elementos sin tener en cuenta los bloques se le denomina TRB Escalar. Al algoritmo paralelo que utiliza la distribución Cíclica se le denomina TRS (*Triangular Row Scatter*) y al que utiliza la distribución Cíclica por Bloques se le denomina TRBS (*Triangular Block Scatter*).

A continuación, se describen los algoritmos atendiendo a una visión global de las variables del problema en los procesadores P_{pr}.

Algoritmo Escalar

Para j=n-1, n-2, ..., 0
Si la fila j∈ P_{pr} (pr=jDIVs)
Entonces
$x_j = b_j/u_{jj}$
Difunde x_j a P_0 P_1 ....P_{pr-1}
Sino
Si pr < (jDIVs) Entonces Recibe x_j
Para i=0, 1, ..., j-1
Si la fila i ∈ P_{pr} (pr=iDIVs) Entonces $b_i = b_i - u_{ij} * x_j$
Fin
Fin

Algoritmo TRB

Para j=p-1, p-2, ..., 0
Si el bloque j∈ P_{pr} (pr=j)
Entonces
RESOLVER $U_{jj}\bar{x}_j = \bar{b}_j$
Difunde $\bar{x}_j$ *a P_0 P_1 ...P_{pr-1}*
Sino
Si pr < j Entonces Recibe $\bar{x}_j$
Para i=0, 1, ..., j-1
Si el bloque i ∈ P_{pr} (pr=i) Entonces Actualizar $\bar{b}_i = \bar{b}_i - U_{ij}\bar{x}_j$
Fin
Fin

Algoritmo TRS

Para j=n-1, n-2, ..., 0

Si la fila j∈ P_{pr} (pr=jMODp)
Entonces
$x_j = b_j/u_{jj}$
Difunde x_j a P_k con k≠ pr
Sino
Recibe x_j
Para i=0, 1, ..., j-1
Si la fila i ∈ P_{pr} (pr=iMODp) Entonces $b_i = b_i - u_{ij} * x_j$
Fin
Fin

Algoritmo TRBS

Para j=n/s-1, n/s-2, ..., 0
Si el bloque j∈ P_{pr} (pr=jMODp)
Entonces
Resolver $U_{jj}\bar{x}_j = \bar{b}_j$
Difunde $\bar{x}_j$ *a P_k con k≠ pr*
Sino
Recibe $\bar{x}_j$
Para i=0, 1, ..., j-1
Si el bloque i∈ P_{pr} (pr=iMODp) Entonces actualizar $\bar{b}_i = \bar{b}_i - U_{ij}\bar{x}_j$
Fin
Fin

La distribución de datos escogida caracteriza el algoritmo paralelo implementado y, por tanto, las prestaciones del mismo. Todas las distribuciones tienen sus ventajas e inconvenientes; la distribución por Bloques supone un mal equilibrio de carga, mientras que la distribución Cíclica supone un aumento en el tiempo de comunicación [8]. Utilizando la distribución Cíclica por Bloques se persigue equilibrar estos dos factores.

A continuación, en la Tabla 2, se muestra un resumen de las prestaciones de dichos algoritmos, atendiendo al tiempo aritmético (t_A), tiempo de comunicaciones (t_C), *speed-up* (S_p o aceleración) y eficiencia máxima (E_p).

Revista Innovación y Software
Vol. 1, No. 2, Mes Septiembre-Febrero, 2020
ISSN: 2708-0935
Pág. 27-39
https://revistas.ulasalle.edu.pe/innosoft

Tabla 2. Prestaciones comparativas de los diferentes algoritmos paralelos.

Algoritmo	$\mathbf{t_A}$	$\mathbf{t_C}$	$\mathbf{S_p}$	$\mathbf{E_p}$
Escalar	$\frac{2n^2F}{p}$	$n\tau + n\beta$	$\frac{p}{2}$	$\frac{1}{2}$
TRB	$\frac{3n^2F}{p}$	$n\tau + p\beta$	$\frac{p}{3}$	$\frac{1}{3}$
TRS	$\frac{n^2F}{p}$	$n\tau + n\beta$	p	1
TRBS	$\frac{n^2F}{p} + nsF$	$n\tau + \frac{n}{s}\beta$	$\frac{p}{1+\frac{sp}{n}}$	$\frac{1}{1+\frac{sp}{n}}$

Se resalta el desequilibrio de carga presente en el TRB y el alto costo de las comunicaciones en el TRS y Escalar.
Esto viene dado porque como cada procesador almacena un bloque de tamaño *n/p* el de más alto índice calcula su bloque envía su información y no hace nada más y así va pasando con el resto. No sucede lo mismo en el TRS al estar las filas repartidas cíclicamente. Aquí el equilibrio de carga es bueno, pero el tiempo de comunicación es en extremo elevado, ya que es proporcional al tamaño del problema.

3. Evaluación y resultados

Se utilizó el lenguaje C para programar los algoritmos. Se implementaron las rutinas para la carga de las matrices, las cuales son guardadas en un archivo con sus respectivos vectores filas separados por tabulaciones. Para calcular el tiempo estimado consumido por cada algoritmo en el cálculo del sistema de ecuaciones se utilizó la función *clock* del lenguaje C definida en la librería *time.h*. Esta función devuelve la cantidad de tiempo de procesamiento que un programa ha utilizado. Es bueno aclarar que en este tiempo no se incluye la carga de los datos, ni el resultado que se muestra por pantalla.
Se utiliza MPI (*Message Passing Interface*) como API que define la sintaxis y semántica de una librería que provee rutinas básicas para construir programas paralelos siguiendo el paradigma de paso de mensaje y comunicación por memoria distribuida. Al ser una API, MPI puede usarse desde cualquier lenguaje de programación (Fortran, C, C++, Java, Python, etc.). Existen bindings de MPI para varios lenguajes. Su diseño se realizó teniendo en cuenta: eficiencia, potencia de comunicación, portabilidad y uso sencillo de topologías. MPI permite estandarizar: rutinas globales de comunicación (ej. *broadcasting*) y rutinas para realizar cálculos globales (ej. *reduction*).

3.1. Evaluación de la red de interconexión del clúster de computadoras

Se trabajó con un algoritmo que calcula la latencia de la red [9], basado en la técnica *ping-pong* que consiste en la trasmisión de información desde el *front-end* a los nodos y una vez en estos, se trasmite la información desde los nodos al *front-end*.

Para obtener la latencia de red, es decir, el retardo provocado por el medio físico se transmitieron tramas sin información, de forma que al realizar un control de tiempos (tiempo de inicio - tiempo de llegada); se obtenga el retardo provocado por el medio de transmisión utilizado[10].

Después se utilizó un algoritmo que calcula el ancho de banda de la red, basado en la técnica *ping-pong*, pero con tramas de tamaño de información muy grande de forma que se obtenga el ancho de banda absoluto y no relativo. El objetivo en enviar grandes tramas de información sin llegar a sobrepasar la tasa máxima de transferencia es obtener el ancho de banda real del medio de transmisión.

Se realizaron pruebas utilizando un algoritmo que calcula el producto vector-matriz. La finalidad de este algoritmo es observar la ganancia de tiempo en la resolución de dicho algoritmo según se vayan añadiendo nuevos nodos al clúster. Este algoritmo consiste en realizar la multiplicación de un vector por una matriz, de forma que cada nodo del clúster se encargue de realizar la multiplicación del vector por una parte de la matriz. El método que se ha llevado a cabo para partir la matriz ha sido dividir el número de filas entre el número de procesos que se van a ejecutar.

Se demostró que a medida que se utilizan matrices y vectores grandes, el proceso paralelo resulta más conveniente que la resolución de dicho algoritmo por el método secuencial.

La Ecuación 1 muestra el resultado por mínimos cuadrados de la curva obtenida.

$$Y= 8.10^{-8} + 0.0002 \quad (1)$$

De donde se obtienen los valores de latencia (*ß*) y costo por paquetes (*t*) para la red del clúster utilizado: $ß = 2.10^{-4}\ sg$, $t = 8.10^{-8}\ sg$.

Se realizaron pruebas con 8 procesadores para tener una mejor respuesta de la paralelización, se trabajó con *cblas* y el *clapack* [11], [12].

3.2. Evaluación de los algoritmos y análisis de las prestaciones

En las pruebas se utilizaron 24 sistemas de ecuaciones lineales con matrices triangulares superiores de la forma $Ux=b$, estas matrices cuadradas y los vectores incógnitas "*x"* fueron generados con números reales y aleatorios entre 1 y 5000 con dimensiones entre 64 y 2048. Además, se ejecutó un programa para calcular los vectores independientes *b*. Estas matrices fueron de orden 64, 128, 256, 512, 1024 y 2048. Además, para cada dimensión se realizaron 4 sistemas de ecuaciones.

Revista Innovación y Software
Vol. 1, No. 2, Mes Septiembre-Febrero, 2020
ISSN: 2708-0935
Pág. 27-39
https://revistas.ulasalle.edu.pe/innosoft

Se probaron los algoritmos en todas las matrices de pruebas y los tiempos medidos con cada grupo de matrices en cada una de las dimensiones fueron promediados y graficados respectivamente. Se tuvo en cuenta hacer las corridas con 1 procesador para el algoritmo secuencial, mientras que para los algoritmos paralelos se utilizaron 2, 4 y 8 procesadores.

3.3. Análisis del Speed-Up

El análisis teórico del *Speed-Up* o aceleración del algoritmo Escalar muestra que este debe comportarse entre 1 y *p/2*, mientras que para el algoritmo TRS es entre 1 y p. En la Figura 4 se observa que cuando se mantiene constante el número de procesadores y se hace crecer el tamaño de la matriz hay un intervalo en el cual el *Speed-Up* es prácticamente 1, lo que indica que los algoritmos paralelos no tienen ventajas sobre el algoritmo secuencial para matrices de dimensión menor que 256.

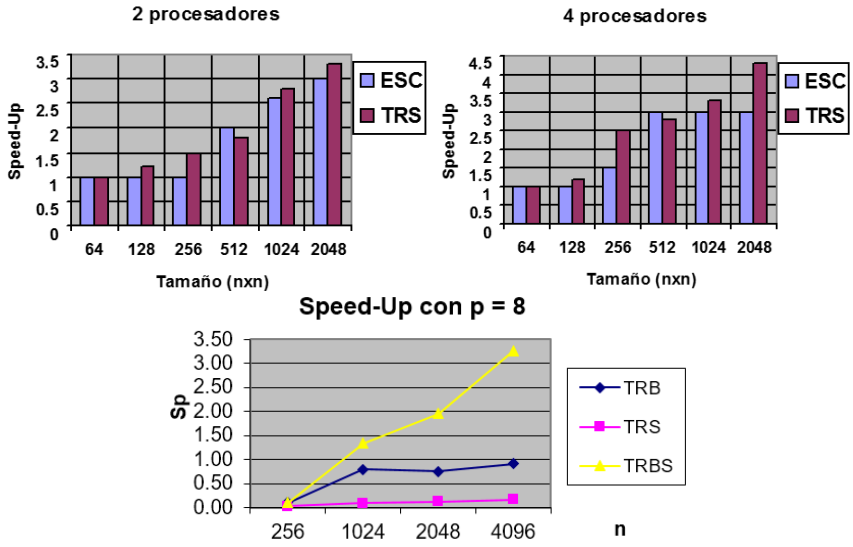


Figura 4. Comportamiento del parámetro Speed-Up para los algoritmos con distinta cantidad de procesadores.

Para problemas de tamaños entre 256 y 2048, el comportamiento de los algoritmos difiere en los casos de 2 y 4 procesadores. En el caso de 2 procesadores, los algoritmos se comportan de la forma esperada, mostrando un incremento del *Speed-Up* al crecer el tamaño del problema de forma tal que los problemas de mayor tamaño son resueltos en menos tiempo por cualquiera de los algoritmos paralelos que por el secuencial. Es bueno señalar que el algoritmo TRS se comporta ligeramente mejor con valores de *Speed-Up* algo superiores al algoritmo Escalar.

Para problemas de tamaño 512 y mayores, el algoritmo Escalar ha alcanzado su mejor valor para el caso de 4 procesadores y supera al caso de 2 procesadores, pero para problemas de 1024 y 2048 las diferencias de *Speed-Up* de este algoritmo son poco significativas cuando se comparan los casos de 2 y 4 procesadores.

Para 4 procesadores, el *Speed-Up* del algoritmo TRS crece a medida que crece el problema, mientras que el algoritmo Escalar a partir de un tamaño se mantiene constante. En la comparación de los algoritmos, para tamaños de problemas creciente con el número de procesadores constantes, el *Speed-Up* se mantiene en la velocidad 1 para tamaños de problema con dimensiones de 64 y 128, tanto con 2 y 4 procesadores. Para tamaño 256, el *Speed-Up* es marcadamente mejor en el algoritmo TRS con 4 procesadores que para el mismo algoritmo con 2 procesadores y que el algoritmo Escalar con 2 y 4 procesadores. El algoritmo TRS para problemas de tamaño 512 y mayores se comporta mejor para el caso de 4 procesadores que para 2.

Se aprecia que al aumentar el orden de las matrices hasta 4096 el valor del Sp aumenta significativamente hasta alcanzar su máximo valor de 3.40 con n = 4096, s = 64 y p = 8, correspondiendo al algoritmo TRBS. Este algoritmo es el que mejores prestaciones logra, con la ganancia de velocidad con valores de s = 64. Es con este valor donde mejor balance se logra entre el t_a y el t_c. Un aumento de *s* trae por consecuencia una disminución de las comunicaciones, pero un aumento del t_a, desequilibrando la carga. Para problemas grandes la incorporación de más procesadores puede disminuir el t_a, de ahí que se logró el máximo Sp cuando se utilizaron los 8 nodos del clúster.

3.4. Análisis de la eficiencia

Otro aspecto de interés a la hora de evaluar un algoritmo paralelo es su eficiencia [13]. Esta nos aporta el grado de utilización del sistema. El análisis teórico de los algoritmos muestra que la eficiencia del algoritmo Escalar debe estar comprendida entre 1/p y ½ y para el algoritmo TRS entre 1/p y 1. En la Figura 5 se observa el comportamiento de la eficiencia de los algoritmos, al variar el tamaño del problema con 2 y 4 procesadores. Se parece mucho al análisis del comportamiento del *Speed-Up*[14]. Es notable que los valores de eficiencia calculados para el caso de 4 procesadores se ajustan más a los valores teóricos que los obtenidos con 2 procesadores, porque estos últimos son superiores a los valores esperados.

Revista Innovación y Software
Vol. 1, No. 2, Mes Septiembre-Febrero, 2020
ISSN: 2708-0935
Pág. 27-39
https://revistas.ulasalle.edu.pe/innosoft

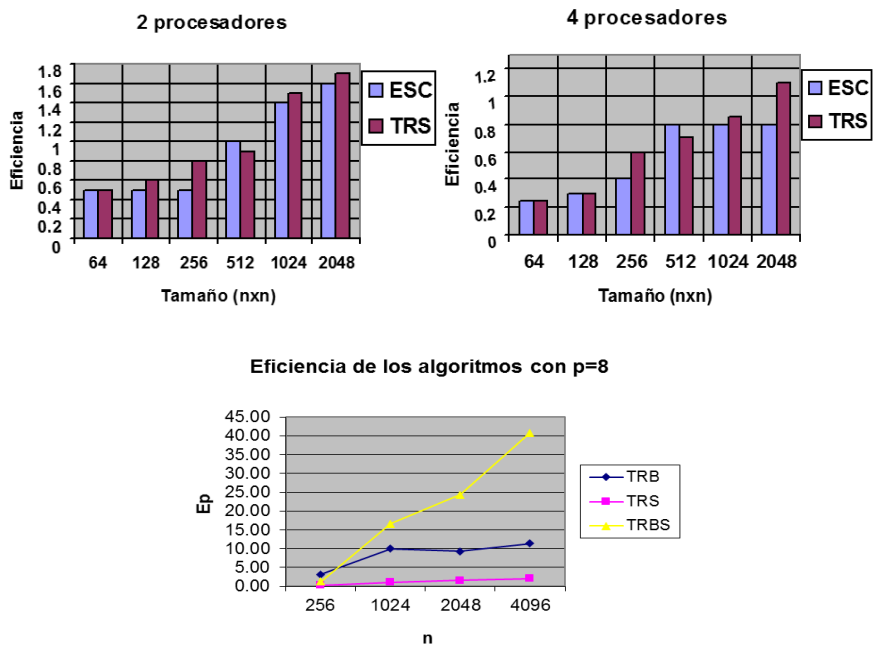


Figura 5. Comportamiento de la eficiencia de los algoritmos.

Al igual que con el *Sp*, es el algoritmo TRBS [15] el que alcanza una mayor eficiencia. Análogamente como sucedía al analizar el desempeño de los algoritmos respecto al Sp, para valores pequeños de n la eficiencia de estos es baja. Al aumentar el tamaño del problema, el TRBS logra aprovechar mejor los recursos del clúster, no así los otros que se ven lastrados por el alto costo de las comunicaciones.

Finalmente, resalta el desequilibrio de carga presente en el TRB y el alto costo de las comunicaciones en el TRS. Esto viene dado porque como cada procesador almacena un bloque de tamaño *n/p* el de más alto índice calcula su bloque, envía su información. No sucede lo mismo con TRS, al estar las filas repartidas cíclicamente. El equilibrio de carga es bueno pero el tiempo de comunicación es en extremo elevado, ya que es proporcional al tamaño del problema.

El TRBS trata de mejorar tanto el desequilibrio como el tiempo de comunicación. En este, el parámetro β está multiplicado por *n/s* en lugar de *n* como en el TRS y por *p* en el TRB. Es de esperar entonces que para valores altos de *n* el Sp tienda a *p* sin un aumento tan marcado del tiempo de comunicación, el cual también va a depender del tamaño del bloque. TRBS toma lo mejor del TRS y el TRB, logrando alcanzar un mejor desempeño al mantener la carga balanceada y no muy alto el costo de las comunicaciones.

4. Conclusiones

El resolver de forma eficiente un sistema de ecuaciones, utilizando algoritmos paralelos sobre múltiples procesadores, puede aportar grandes ventajas al reducir drásticamente el tiempo necesario para computar la solución.

Como se ha visto con la distribución cíclica por filas y el algoritmo TRS asociado a esta distribución los resultados no son nada halagüeños, mucho menos en un clúster con una red de interconexión como la empleada para realizar los experimentos con un elevado valor de β.

El TRB logra disminuir el costo de las comunicaciones ya que *n* no multiplica a β, esto está dado porque cada procesador almacena un bloque de la matriz, pero no es bueno el equilibrio de la carga.

Se aprecia que el algoritmo TRBS, con el aumento de las dimensiones del problema, las comunicaciones no se convierten en un obstáculo y la carga entre los distintos procesadores del sistema se mantenga equilibrada.

Con este trabajo se evidencia la relación existente entre la distribución de los datos y la influencia de esta, tanto en la aceleración como en la eficiencia que logran los algoritmos, así como el equilibrio de carga y reducción de las comunicaciones máximas para lograr el mejor rendimiento.

Referencias

[1] R. Marichal, E. Dufrechou, and P. Ezzatti, “Assessing the solution of one sparse triangular linear system on multi-many core platforms,” 2019.

[2] V. Sonzogni, P. Sanchez, and M. Storti, “Resolución de grandes sistemas de ecuaciones en un cluster de computadoras,” *Mecánica Computacional*, vol. 23, pp. 3211–3227, 2004.

[3] L. Chuquiguanca, E. Malla, F. Ajila, and R. Guamán, “Arquitectura Clúster de Alto Rendimiento Utilizando Herramientas de Software Libre High Performance Cluster Architecture Using Free Software Tools,” vol. 2, no. 1, 2015.

[4] J. L. Bolaño Herazo, “Estudio de rendimiento para la solución de ecuaciones lineales usando computación en paralelo,” 2015.

[5] J. D. Jaramillo, A. M. V. Maciá, and F. J. C. Zabala, “Métodos directos para la solución de sistemas de ecuaciones lineales simétricos, indefinidos, dispersos y de gran dimensión,” *Universidad Eafit*, 2006.

[6] C. Baeza Sanz, “Explotación de una política de partición de datos para aplicaciones paralelas,” 2015.

[7] S. Piña, “El impacto de shocks contractivos de política monetaria en un modelo DSGE estimado con métodos bayesianos para Chile,” 2016.

[8] H. L. Bodlaender and T. Hagerup, “Parallel algorithms with optimal speedup for bounded treewidth,” *SIAM Journal on Computing*, vol. 27, no. 6, pp. 1725–1746, 1998.

[9] C. Gómez Crespo, “Diseño y evaluación de un cluster HPC: aplicaciones,” thesis, Universitat Politècnica de Catalunya, 2014.

[10] P. E. Leibovich, F. Issouribehere, and J. C. Barbero, “Ensayo y comparación de métodos de transmisión de sincrofasores sobre redes Ethernet,” in *XVIII Encontro Regional Ibero-Americano do CIGRE (ERIAC 2019)(Foz do Iguaçu, Brasil, 19 a 23 de maio de 2019)*, 2019.

[11] G. H. A. Salinas and E. M. A. Salinas, “Utilización de CLAPACK para resolver sistemas de ecuaciones lineales mediante paralelismo y optimización de memoria,” *UNACIENCIA*, vol. 2, no. 3, pp. 7–7, 2009.

[12] P. Ezzatti, E. S. Quintana-Ortí, and A. Remón, “Resolución de sistemas triangulares en tarjetas Gráficas (gpu),” *Mecánica Computacional*, vol. 29, no. 30, pp. 3053–3061, 2010.

[13] M. Hernández, A. A. Del Barrio, and G. Botella, “Clúster de Computación Científica de Bajo Coste y Consumo,” 2018.

[14] J. B. B. Darmawan and S. Mungkasi, “Parallel computations using a cluster of workstations to simulate elasticity problems,” in *Journal of Physics: Conference Series*, 2016, vol. 776, p. 012081.

[15] I. S. Silva, L. O. Luz, R. Nepomuceno, and J. C. dos Santos, “Programaçao de processadores multi-core: Uma experiência educacional utilizando plataformas didáticas embarcadas em fpga,” *International Journal in Computer Architecture Education (IJCAE)*, vol. 3, no. 1, pp. 9–12, 2014.