Cómputo Intensivo en *Clusters* de Nodos *Multicore:* Análisis de *Speed Up* y Eficiencia

Gustavo Wolfmann
Laboratorio de Computación
Fac.Cs. Exac. Fis. y Naturales
Universidad Nacional de Córdoba
gwolfmann@gmail.com

Fernando G. Tinetti[1]
Instituto de Investigación en Informática-LIDI
Facultad de Informática
Universidad Nacional de La Plata
fernando@info.unlp.edu.ar

Resumen

En este artículo se presentan algunas conclusiones derivadas del trabajo de paralelización de problemas de álgebra lineal en *clusters* (o redes locales de computadoras dedicadas a cómputo paralelo/distribuido). Específicamente, se presentan las ideas básicas de paralelización de la multiplicación de matrices relacionándolas con la ley de Amdahl y su revisión hecha por J. L. Gustafson (también conocida como *ley de Gustafson*), analizando el rendimiento con el índice o factor de *Speed Up*. Dado que actualmente los *cluster*s instalados para cómputo intensivo contienen nodos con multiprocesamiento simétrico (SMP: Symmetric Multiprocessing), incluyendo posiblemente nodos con múltiples núcleos, se presentan también las posibilidades de paralelización en el contexto de memoria compartida y memoria distribuida simultáneas. Se muestra que específicamente en el área de aplicaciones de álgebra lineal, el problema de paralelización se puede circunscribir a la paralelización en el contexto de memoria distribuida y, además, que el rendimiento obtenido en términos de eficiencia es siempre mayor al 85%.

Palabras Clave: Cómputo Paralelo en Clusters, Cómputo Paralelo en Multiprocesadores, Cómputo de Alto Rendimiento, Evaluación de Rendimiento.

1.- Introducción

Si bien los *cluster*s se han establecido desde hace varios años como una de las plataformas de cómputo intensivo más utilizadas [3], aún sigue siendo interesante el problema de la paralelización de aplicaciones sobre este tipo de arquitectura de cómputo distribuido. El principal problema de las propuestas de algoritmos paralelos es el rendimiento o, desde otra perspectiva, la adaptación del

1 Investigador Asistente, Comisión de Investigaciones Científicas de la Provincia de Buenos Aires.

procesamiento paralelo a la arquitectura de procesamiento. Los *clusters* (o redes locales de computadoras dedicadas a cómputo paralelo/distribuido) no escapan a esta realidad: si el algoritmo paralelo es apropiado para su implementación y ejecución en un *cluster* se obtendrá buen rendimiento, sino, las mejoras de rendimiento del procesamiento paralelo se perderán o, en el caso extremo, será mejor no utilizar más de una computadora para procesar y llegar al resultado que se busca.

Las aplicaciones de álgebra lineal han estado entre las primeras a ser paralelizadas en la búsqueda de mejorar/reducir el tiempo de ejecución [8, 9]. En el contexto específico de los *cluster*s, se podría decir que la adopción de algoritmos definidos para máquinas paralelas tradicionales en bibliotecas como ScaLAPACK (Scalable LAPACK) [5] han impuesto, de alguna manera, una fuerte presión sobre las redes de interconexión internas de los *clusters* utilizados para cómputo intensivo. La combinación de este factor con las posibilidades de comercialización han dado paso a la propuesta y establecimiento de redes de interconexión de alto rendimiento, como SCI (*Scalable Coherent Interface*) [12], Myrinet [16], Infiniband [6], Quadrics [4] y otras. Sin embargo, es importante recordar o recuperar la idea original de los algoritmos paralelos: la adaptación del procesamiento a la arquitectura de cómputo disponible. En este sentido, la adopción de los algoritmos tradicionales no ha hecho más que ocultar, en cierto modo, las características propias de los *clusters* para cómputo paralelo y esto, a su vez, puede conducir a una complejidad innecesaria en los algoritmos (porque no fueron propuestos para clusters) y/o a penalización de rendimiento.

En este artículo se utilizará el problema de la multiplicación de matrices inicialmente por sencillez, pero la propuesta algorítmica no carece de generalidad. Por un lado, la multiplicación de matrices está siempre en la base de los problemas de rendimiento en aplicaciones de álgebra lineal [14], y por otro, se la valora dentro del contexto de aplicaciones de álgebra lineal más que en sí misma. También es importante remarcar que tanto los algoritmos como la evaluación de rendimiento se analizarán desde las perspectivas de las leyes de Amdahl y Gustafson[2].

2 En breves palabras, la ley de Amdahl propone que el *speed up* de un programa paralelo no puede incrementarse proporcionalmente a la cantidad de procesadores utilizados, asumiendo fijo el tamaño del problema, dada la existencia de una parte serial constante en el mismo. Gustafson propuso una revisión, eliminando la restricción del tamaño fijo, por lo que aumentando el tamaño del problema, la parte proporcional serial del programa se reduce, y el *speed up* se mantiene al incrementar el número de procesadores utilizados.

2.- Procesamiento en Cada Computadora: ¿Secuencial o Paralelo?

Tanto para la optimización como para el análisis de rendimiento es importante conocer cuál es el elemento básico de procesamiento. Hasta no hace muchos años, el elemento básico de procesamiento ha sido una CPU (*Central Processing Unit*) con su/s correspondiente/s unidad/es de punto flotante necesarias/utilizadas para cómputo numérico. Si bien esta realidad se mantiene en los *clusters*, también es importante tener claro que las computadoras de escritorio actuales (en base a las que se construyen los *clusters*) ya no tienen una única CPU, básicamente por la utilización de multiprocesamiento simétrico con múltiples CPUs instaladas sobre una misma placa para compartir todos los recursos de procesamiento (memoria, entrada/salida, etc.) o, directamente por la utilización de procesadores con múltiples núcleos. Tanto la evolución del mercado como la de las empresas de microprocesadores estándares de relativamente bajo costo como Intel y AMD muestran una clara evolución hacia estos últimos procesadores, de forma tal que en poco tiempo no va a ser posible la adquisición de una computador con un único núcleo o CPU.

Desde la perspectiva de procesamiento intensivo, evidentemente las CPUs múltiples en cada computadora (o *nodo* de un *cluster*, a partir de este punto) son una ventaja, pero en el contexto de los algoritmos paralelos a ser utilizados en *clusters*, introducen una complejidad extra. Esto se debe a que normalmente el modelo de programación paralela en *clusters* ha sido el de pasaje de mensajes, y aunque es posible mantener este modelo independientemente de la cantidad de núcleos, esto puede generar una penalización importante de rendimiento. Es claro que, en términos de rendimiento, no es igual comunicar dos procesos ejecutándose en o asignados a dos CPUs de un mismo nodo que comunicar dos procesos en distintos nodos. Y la diferencia de rendimiento no es despreciable, es de varios órdenes de magnitud. En este punto, se podrían definir dos niveles o tipos de paralelismo: *intra* nodo con memoria compartida e *inter* nodos con memoria distribuida.

Sin embargo, el desarrollo de software optimizado para cómputo numérico ha evolucionado lo suficiente como para que el paralelismo *intra* nodo pueda considerarse resuelto de manera muy satisfactoria. Esto implica que se mantiene solamente la necesidad de algoritmos paralelos para computadoras de memoria distribuida, al menos en el dominio del cómputo numérico. Como ejemplo,

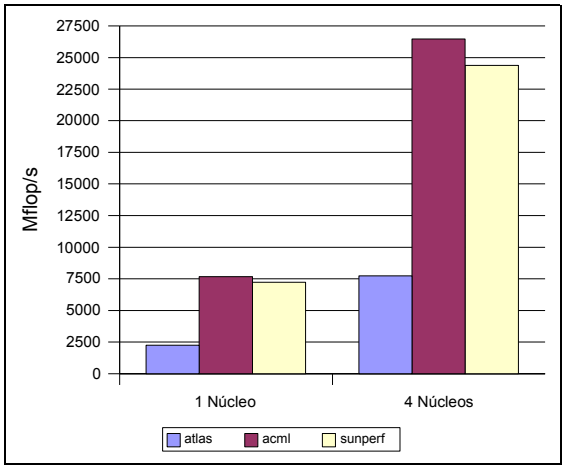
se presentan resultados en computadoras con procesamiento SMP de dos procesadores, donde además cada procesador tiene dos núcleos, cuyas características se resumen en la Tabla 1. Es claro que si cada nodo de un *cluster* tiene estas características, se tienen 4 CPUs por nodo y se puede/debe implementar alguna forma de cómputo paralelo que los utilize.

Sist. Operativo	**RAM**	**Procesadores**	**Cant. de Proc., GHz**	**Núcleos por Proc.**
Sun Solaris 10	4 GB	AMD Opteron 2200	2, 2.2 GHz	2

Tabla 1: Características de Procesamiento de Una Computadora.

Es interesante notar la cantidad de bibliotecas disponibles para cómputo numérico intensivo y para álgebra lineal en particular para estas computadoras: Sun tiene disponible la biblioteca sunperf (incluida en sunstudio) [18], AMD provee ACML (AMD Math Core Library) [1] y también existen alternativas de código abierto como ATLAS (Automatically Tuned Linear Algebra Software) [20]. En el contexto de procesadores de Intel, también se tiene disponible la biblioteca MKL (Intel Math Kernel Library) [13]. En todos los casos se tienen como mínimo todas las rutinas de BLAS (Basic Linear Algebra Subroutines) con rendimiento optimizado y, lo que es más interesante aún, el rendimiento está optimizado para el caso de múltiples procesadores/núcleos en una computadora.

A modo de ejemplo, se muestra en la Fig. 1 el rendimiento obtenido en Mflop/s para una multiplicación de matrices de 13000 × 13000 elementos en precisión simple, básicamente una llamada a la rutina sgemm() de BLAS.



	1 Núcleo	4 Núcleos
atlas	2.248	7.742
acml	7.675	26.470
sunperf	7.221	24.374

Valores de Mflop/s de cada biblioteca

Figura 1: Rendimiento Obtenido en Un Nodo/Computadora del *Cluster*.

La cantidad de núcleos a utilizar se puede definir en tiempo de ejecución en los casos de las bibliotecas sunperf y ACML (vía la cantidad de *threads* de OpenMP-Fortran), en el caso de ATLAS, solamente se puede elegir la opción de uso de un único núcleo o de todos los núcleos disponibles de manera estática en la etapa de enlazado (*link*) del binario. Claramente, todas las bibliotecas son muy eficientes en cuanto a la obtención del máximo rendimiento con múltiples procesadores y núcleos respecto de la opción secuencial (multiplica o escala por cuatro el rendimiento secuencial), y claramente también ATLAS no es la mejor opción en esta combinación de computadora y sistema operativo. Es importante destacar que el código fuente de la aplicación es exactamente el mismo, la diferencia se da en la generación del binario, con las diferentes bibliotecas disponibles.

Si bien siempre es posible intentar una optimización diferente de las provistas por las bibliotecas, es muy poco probable que se pueda mejorar la optimización de AMD para sus propios procesadores. Por otro lado, de acuerdo a los valores que se muestran en la Fig. 1, no hay mucho margen de ganancia dado que, por ejemplo para ACML, la opción de un único núcleo obtiene 7675 Mflop/s mientras que la opción de 4 núcleos obtiene 26470 Mflop/s, que implica una eficiencia de más del 85% en la paralelización *intra* nodo.

Teniendo en cuenta que la llamada a la rutina (o el programa mismo) no cambia por utilizar una versión optimizada y paralelizada para múltiples núcleos, en realidad es como si se estuviera programando secuencialmente en cada nodo. Es por esta razón que se puede pensar directamente en la paralelización orientada directamente al *cluster* como computadora paralela de memoria distribuida. Es importante recordar que esto no sucede únicamente para la multiplicación de matrices, sino para todas las rutinas provistas por las bibliotecas, que suelen incluir todo BLAS, LAPACK (Linear Algebra Package) y FFT (Fast Fourier Transforms), por ejemplo, en el caso de ACML y MKL.

3.- Rendimiento de Algoritmos Paralelos Sencillos

Aunque se podría recurrir a la utilización de los algoritmos paralelos clásicos como en el caso de la biblioteca ScaLAPACK [5], es relevante volver a la idea básica de algoritmo paralelo orientado a la arquitectura de procesamiento, para medir el rendimiento, tal como se comenta en la primera sección.

Más aún, en vez de utilizar algoritmos paralelos relativamente complejos, justamente por la adaptación del procesamiento paralelo a computadoras paralelas específicas, es interesante verificar o al menos experimentar con algoritmos paralelos sencillos, que tienden a tener menor complejidad en cuanto a desarrollo y a la utilización de los resultados, dado que la distribución de datos suele ser más directamente relacionada con la aplicación a resolver. Más específicamente, en vez de utilizar la distribución de datos por bloques cíclica y el algoritmo DIMMA (Distribution Independent Matrix Multiplication Algorithm) [7] de ScaLAPACK para la multiplicación de matrices, es bueno corroborar si no existen formas más sencillas y eficientes no solamente en cuanto a rendimiento obtenido en tiempo de ejecución sino en cuanto a la propuesta algorítmica para clusters. En principio, siguiendo la definición de la multiplicación de matrices, se puede decir que el problema es llegar a computar todos los valores de la matriz C, donde C = A × B, con A, B y C matrices cuadradas de orden *n* para simplificar la definición. Desde esta perspectiva, y dado que todos los elementos de la matriz C requieren la misma cantidad de operaciones, se puede pensar en distribuir el cómputo de diferentes partes de C en diferentes nodos del cluster. En este punto existen varias opciones, siendo una de las primeras, la distribución del cómputo en bloques de filas, tal como lo muestra la Fig. 2 para cuatro nodos: N_1, ..., N_4; cada uno con su correspondiente patrón de acceso a los datos de las matrices y donde todos los nodos acceden/utilizan todos los datos de la matriz B.

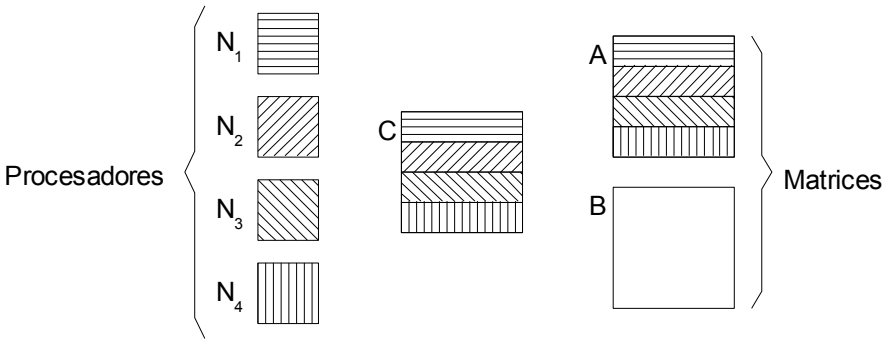
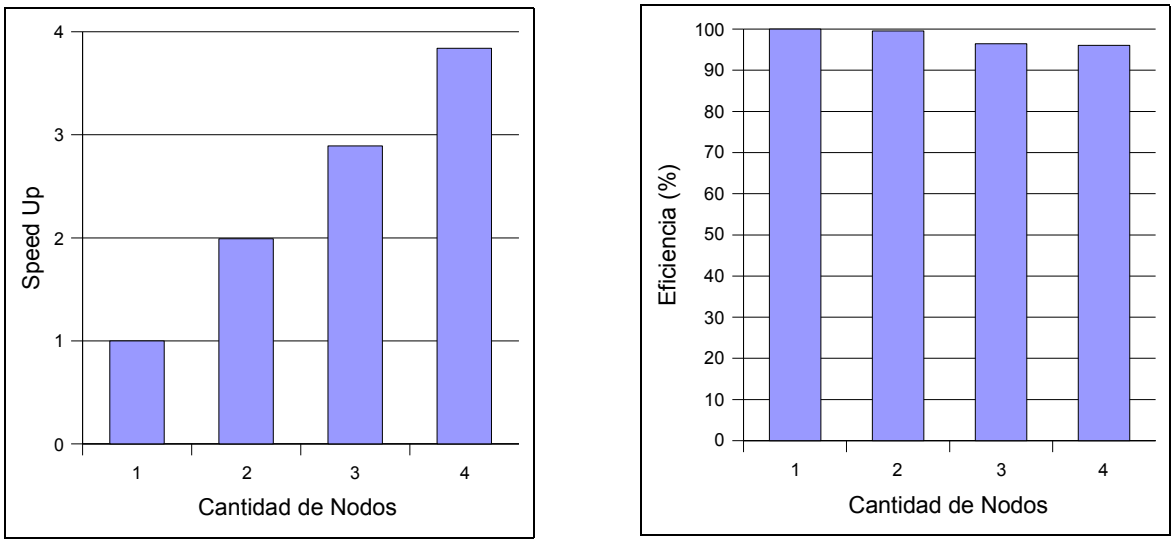


Figura 2: Distribución de Cómputo de la Multiplicación de Matrices.

Con esta idea en mente, queda definida la parte de cómputo de un programa distribuido entre *p* nodos (cuatro en el ejemplo), pero queda por definir cómo se distribuyen y/o acceden los datos de las matrices. Desde la perspectiva de los datos que se necesitan en cada uno de los nodos, se podría definir

un algoritmo paralelo que tenga todos los datos locales en *p* nodos: 1/*p* del total de filas de las matrices A y C, y toda la matriz B. Evidentemente, esta propuesta algorítmica tiene un problema básico, que se ha considerado inaceptable para casi cualquier algoritmo paralelo: los datos de la matriz B están replicados en todos los nodos. Esto tiene un problema más que conceptual: limita seriamente la escalabilidad del algoritmo, dado que al agregar un nodo, la memoria de este nodo se verá casi completamente ocupada por los datos de la matriz B, que nunca se distribuye y, por lo tanto, siempre debe ser almacenada en todos los nodos. Sin embargo, tal como se puntualiza en [11], la ley de Amdahl [2] se posiciona en la situación en la cual un mismo problema y con un mismo tamaño se resuelve utilizando cada vez más computadoras. Desde esta perspectiva, no habría inconvenientes con replicar los datos, dado que si es posible tener todas las matrices en un solo nodo, será posible también en varios nodos, con dos de las tres matrices (A y C) con los datos distribuidos. La Fig. 3 muestra los resultados de rendimiento de este algoritmo paralelo utilizando entre 1 y 4 nodos tales como los que se describen en la Tabla 1 anterior, donde se replican los datos de la matriz B de forma tal que cada nodo tiene todos los datos necesarios para computar su parte de la matriz resultado C.



a) Speed Up b) Eficiencia (%)

Figura 3: Rendimiento de la Multiplicación de Matrices en Paralelo con Datos Replicados.

Como era de esperar, y dado que no son necesarias las comunicaciones para el cómputo del resultado en cada nodo, el rendimiento es muy cercano al óptimo. Aunque no es relevante en este caso, el *cluster* (los cuatro nodos) está interconectado con una red Ethernet de 1 Gb/s. Tal como se puntualiza

en [11], en general no se resuelve el mismo problema cuando se utilizan más recursos, normalmente se resuelve el problema original pero con una cantidad de datos mayor. En el caso de los clusters y la multiplicación de matrices, esto significa que a medida que se utilizan más nodos, el tamaño de las matrices (el orden *n* de las matrices cuadradas) aumenta. Esto, a su vez implica la necesidad de distribuir todos los datos del problema a resolver.

Sin llegar a una distribución como la de ScaLAPACK y siguiendo con el razonamiento anterior de los datos necesarios para calcular la matriz resultado en cada nodo, se podría distribuir la matriz B de la operación C = A × B de forma tal que 1/*p* de las columnas de B sea asignado o almacenado en cada uno de los *p* nodos de un cluster. La Fig. 4 muestra esta distribución en cuatro nodos, y también cómo o qué partes de la matriz C se computan con los datos de las matrices A y B.

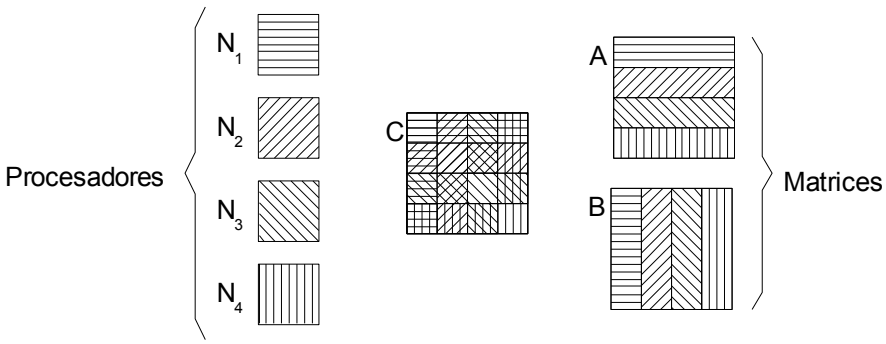


Figura 4: Distribución de Cómputo y Datos de la Multiplicación de Matrices.

Se puede notar que en el nodo N_1 sigue siendo posible computar todo el primer bloque de filas de C, pero ahora se necesitarán los datos de B que están distribuidos entre la totalidad de los nodos. Dado que todos los elementos de B son necesarios en todos los nodos, se puede pensar en un algoritmo donde cada nodo haga *broadcast* (o multidifusión) de sus datos y se compute en todos los nodos con los datos que se han recibido [19]. La Fig. 5 muestra los datos locales y el pseudocódigo del programa a ejecutar en el nodo N_i. Básicamente, se sigue una secuencia de *mensaje broadcast – cómputo parcial* del resultado en cada nodo. Aunque se podría pensar que esta secuencia genera penalización de rendimiento, dado que todo lo que sea esperar por rutinas de pasaje de mensajes (en este caso *broadcast*) implica tiempo de ejecución sin utilizar las unidades de punto flotante, la Fig. 6 muestra que no es el caso en el contexto de un cluster de cuatro nodos con cuatro cores cada uno de ellos, con las características dadas previamente en la Tabla 1.

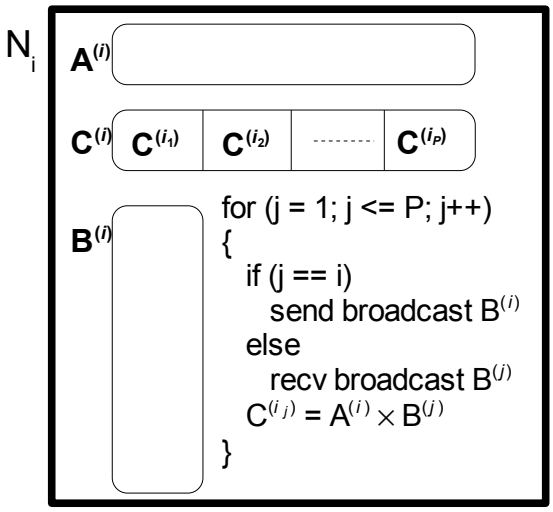
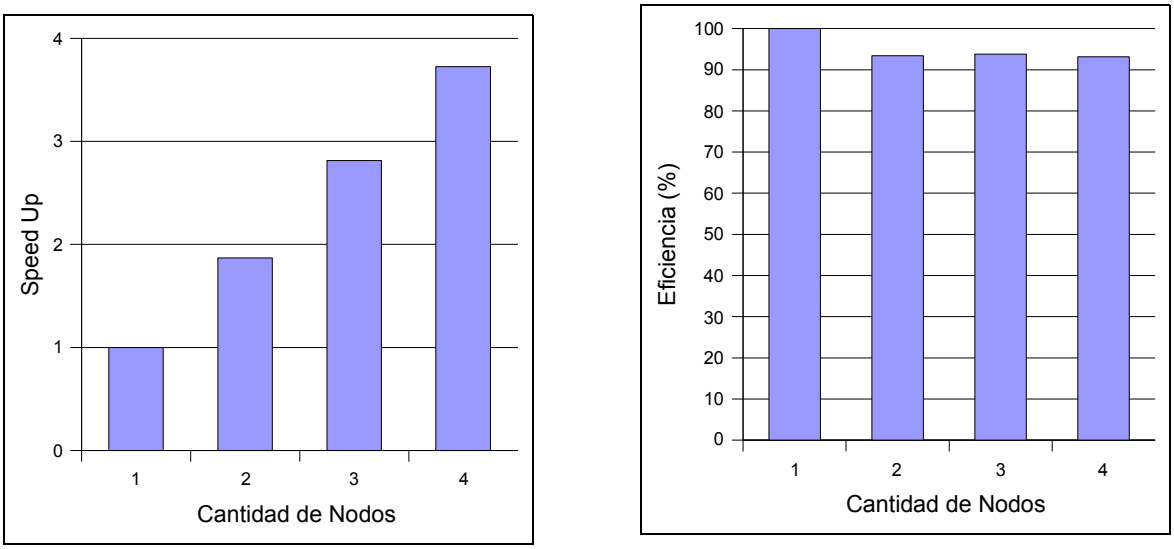


Figura 5: Algoritmo Paralelo Con Distribución de los Datos de las Matrices.

En este caso sí es relevante que la red de interconexión es Ethernet de 1 Gb/s (no siempre considerada adecuada para cómputo paralelo en clusters desde la perspectiva de rendimiento) y se utilizó la versión implementación de MPI provista por Sun, HPC Cluster tools[17], que está basada, a su vez, en la implementación Open MPI [10] de MPI (Message Passing Interface) [15]. Más allá de la comparación que se puede hacer de la Fig. 6 con la Fig. 3, la Fig. 6-b) muestra claramente que la eficiencia está por encima del 90%, es decir que más del 90% del tiempo de ejecución para el cálculo de la multiplicación de matrices se estuvieron utilizando los 16 núcleos disponibles en el *cluster*.



a) Speed Up b) Eficiencia (%)

Figura 6: Rendimiento de la Multiplicación de Matrices en Paralelo con Datos Distribuidos.

4.- Breve Análisis de los Algoritmos y de los Resultados Obtenidos

Si bien el algoritmo con replicación de datos da los mejores resultados (los de la Fig. 3), se deben destacar algunas características importantes:

- La visión de Amdahl de procesamiento paralelo no necesariamente es la mejor y ni siquiera es la más utilizada, dado que normalmente al utilizar más recursos de cómputo automáticamente también se procesan más datos [11]. Esto casi directamente invalida la posibilidad de replicar datos para mejorar el paralelismo en el programa.
- No siempre es posible mejorar el paralelismo replicando datos, como en el caso de la multiplicación de matrices. En el caso de las rutinas de LAPACK que computan factorizaciones (LU, QR, Cholesky, etc.), clasificadas como computacionales, necesariamente se debe realizar cómputo parcial de manera distribuida y comunicar estos resultados parciales, que son parte del avance del cómputo total. Esto no significa que deba descartarse la posibilidad de replicar para mejorar el rendimiento, sino que no siempre es posible.

El algoritmo con distribución de datos de la Fig. 5 tuvo muy buenos resultados, con eficiencia de más del 90% (Fig. 6). Se debe tener en cuenta, sin embargo, que la cantidad de nodos del cluster es pequeña, pero son todos los nodos disponibles actualmente en la instalación utilizada con múltiples propósitos, entre ellos la investigación. Esto por un lado no permite asegurar que este algoritmo será el mejor en todos los casos, pero al menos es bueno para relativamente pocos nodos de muy alto rendimiento cada uno de ellos (aproximadamente 26.5 Gflop/s cada nodo, mostrado en la Fig. 1). En este sentido, se puede confirmar la visión de la revisión de Gustafson a la ley de Amdahl: la mayoría de las veces que se escala el problema, se puede mantener rendimiento cercano al óptimo. Por supuesto, esto sigue dependiendo de la fracción de cómputo secuencial y paralelo del problema completo y la distribución de la parte paralelizable entre los recursos disponibles.

5.- Conclusiones y Trabajo Futuro

Se ha mostrado que, aunque los algoritmos sean extremadamente sencillos, son válidos para medir el rendimiento de clusters con nodos muy potentes en términos de procesamiento y con una red de

interconexión que no siempre es considerada adecuada para cómputo paralelo/intensivo. Esto significa que no necesariamente siempre se debe recurrir a algoritmos complejos, que implican distribuciones de datos difíciles de entender y manejar. En este sentido, se debe tener en cuenta que una multiplicación de matrices normalmente *no es el problema* a resolver, sino que *es parte de un problema* a resolver, y por esto es necesario conocer en detalle dónde están los datos y cómo acceder a ellos. Como casi siempre en estos casos, queda pendiente la experimentación con mayor cantidad de nodos y mejores redes de interconexión para verificar la validez en cuanto a escalabilidad de los algoritmos que se proponen en este artículo. Por supuesto que nunca se descarta la posibilidad de optimizaciones en cuanto a solapamiento de cómputo y comunicaciones, como en [19].

Referencias

[1]AMD Corp., ACML - AMD Core Math Library User Guide, http://developer.amd.com/cpu/Libraries/acml/downloads/Pages/default.aspx#docs

[2] G. M. Amdahl, “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities”, Proceedings AFIPS Conference Vol. 30, 1967, AFIPS Press, Reston, VA, pp. 483-485.

[3] M. Baker, R. Buyya, “Cluster Computing at a Glance”, in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 3-47, 1999.

[4] J. Beecroft, D. Addison, D. Hewson, M. McLaren, D. Roweth, F. Petrini, J. Nieplocha, “QsNet II: Defining High-Performance Network Design”, IEEE Micro. 25(4):34-47, Jul/Aug 2005.

[5] L. Blackford, J. Choi, A. Cleary, E. D' Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. Whaley, ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997.

[6] J. S. Bozman, V. Turner, “InfiniBand™ Architecture: Transforming I/O Technology”, IDC Whitepaper, Jan. 2001.

[7] J. Choi, “A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers”, Proc. of the High-Perf. Comp. on the Information Superhighway, IEEE, HPC-Asia '97.

[8] K. Dackland and E. Elmroth. “Design and Performance Modeling of Parallel Block Matrix Factorizations for Distributed Memory Multicomputers”, Proceedings of the Industrial Mathematics Week, pp 102-116, Trondheim, 1992.

[9] J. Dongarra, D. Walker, “Libraries for Linear Algebra”, in Sabot G. W. (Ed.), High Performance Computing: Problem Solving with Parallel and Vector Architectures, Addison-Wesley Publishing Company, Inc., pp. 93-134, 1995.

[10] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca, A. Lumsdaine, “Open MPI: A High-Performance, Heterogeneous MPI”, Proc. Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, Barcelona, Spain, 2006.

[11] J. L. Gustafson, “Re-evaluating Amdahl's Law”, Communications of the ACM, Vol. 31, No. 5, 1988, pp.

532-533.

[12] D. B. Gustavson, Q. Li, "The Scalable Coherent Interface (SCI)", IEEE Communications Magazine, Vol. 34, No. 8, August, 1996, pp. 52-63.

[13] Intel Corp., MKL Math Kernel Library, http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm

[14] B. Kågström, P. Ling, C. Van Loan, "Portable High-Performance GEMM-based Level 3 BLAS", R. F. Sincovec et al., Editors, Parallel Processing for Scientific Computing, Philadelphia, 1993, SIAM, pp. 339-346.

[15] MPI Forum, "MPI: a message-passing interface standard", International Journal of Supercomputer Applications, 8 (3/4), pp. 165-416, 1994.

[16] C. Seitz, "Myrinet: A Gigabit per second Local Area Network", IEEE Micro, Feb, 1995.

[17] Sun Microsystems, Inc., Sun HPC ClusterTools 7.1 - Overview, http://www.sun.com/software/products/clustertools/

[18] Sun Microsystems, Inc., Sun Performance Library User's Guide, 2007, http://docs.sun. com/source/819-5268/index.html

[19] Tinetti F. G., Cómputo Paralelo en Redes Locales de Computadoras, Tesis Doctoral (Doctorado en Informática), Universidad Autónoma de Barcelona, Marzo de 2004. Disponible en http://ftinetti.googlepages.com/tesisdoctoral

[20] R. C. Whaley, A. Petitet, "Minimizing development and maintenance costs in supporting persistently optimized BLAS", Software: Practice and Experience, Vol. 35, No. 2, pp. 101-121, Feb. 2005. http://math-atlas.sourceforge.net/