Factorización de Matrices Cholesky: Paralelización y Balance de Carga

Fernando G. Tinetti*, Fernando Romero
Facultad de Informática, UNLP
50 y 115, 1900, La Plata
Argentina
fernando@info.unlp.edu.ar, fromero@lidi.info.unlp.edu.ar

Resumen

En este artículo se presentan las ideas más importantes para la paralelización de la factorización de matrices Cholesky. Se discuten dos aspectos básicos: la distribución de los cálculos en distintos procesadores y la forma en que la distribución de estos cálculos sea similar en todos los procesadores. Para la distribución de los cálculos se tienen en cuenta específicamente las dependencias de datos y para la distribución de la carga de procesamiento se tienen en cuenta las características propias de la secuencia de avance de procesamiento del método de factorización. Las ideas relacionadas con el balance de carga son muy similares a otros métodos de factorización y por lo tanto se pueden reusar las formas de solución que sean satisfactorias. Finalmente, se muestran los resultados de rendimiento obtenidos con distintas cantidades de procesadores en un cluster de PCs.

1. Introducción

Los métodos numéricos han sido los primeros candidatos a ser resueltos en las plataformas de cómputo paralelo disponibles. De hecho, siempre se han estudiado no solamente las formas de tener un control estricto de los errores numéricos sino también la forma de optimizar el rendimiento. En el contexto de los usuarios de estos métodos para resolver sus propios problemas de modelización, la biblioteca LAPACK (Linear Algebra PACKage) se ha establecido como un estándar de facto y como mínimo en una referencia obligatoria [2] [7] [8]. Teniendo en cuenta la optimización de rendimiento y específicamente los tiempos de respuesta de las operaciones, se han desarrollado soluciones y optimizaciones desde múltiples puntos de vista:

- Las subrutinas de LAPACK se pueden clasificar de varias maneras enfocándose específicamente en la carga de procesamiento. Entre todas las subrutinas se han identificado claramente las que tienen mayores requerimientos de procesamiento y se las denomina "rutinas computacionales". Se han definido y estudiado claramente como las que se tienen que optimizar en cuanto a rendimiento y tiempo de respuesta.

*Investigador Asistente CICPBA

- Todas las rutinas de LAPACK (o como mínimo las rutinas computacionales) son implementadas en función de otro conjunto de rutinas denominadas BLAS (Basic Linear Algebra Subroutines) [10] [17] [11]. Estas rutinas son más sencillas y por lo tanto también más sencillas en cuanto a la optimización de rendimiento. Optimizando las rutinas de BLAS se optimizan las rutinas de LAPACK.

- Las rutinas de BLAS se han clasificado de acuerdo con los requerimientos de cómputo en tres niveles: BLAS nivel 1, BLAS nivel 2 y BLAS nivel 3. Las rutinas de nivel 1 acceden a $O(n)$ datos (vectores) y realizan $O(n)$ operaciones, las de nivel 2 acceden a $O(n^2)$ datos (matrices) y realizan $O(n^2)$ operaciones y las de nivel 3 acceden a $O(n^2)$ datos (matrices) y realizan $O(n^3)$ operaciones. Por lo tanto, la optimización de rendimiento debe ser aplicada a las rutinas de nivel 3 [9].

Es así que se podrían considerar dos niveles de complejidad en cuanto a la optimización de las rutinas sobre las cuales trabajar: las computacionales de LAPACK y las de nivel 3 de BLAS. Es claro que las computacionales de LAPACK son más complejas que las de nivel 3 de BLAS, pero el aprovechamiento de estas optimizaciones también es más directo. Se debe tener en cuenta que las rutinas computacionales de LAPACK son implementadas en función de las de BLAS, pero su complejidad y requerimientos de procesamiento no cambian.

Sin perder de vista las aplicaciones numéricas, las arquitecturas de procesamiento siempre han sido tema de estudio en cuanto a las posibles formas de procesamiento y sus correspondientes optimizaciones [12] [13]. Las arquitecturas de procesamiento paralelo son consideradas como la alternativa más fuerte en cuanto a la obtención de tiempo de respuesta razonable de las aplicaciones con mayores requerimientos de cómputo. De hecho, la evaluación de la capacidad de computadoras para hacer la clasificación TOP500 [6] [21] (donde se documentan las 500 máquinas más rápidas existentes/evaluadas) se lleva a cabo con un programa paralelo. Desde hace varios años, las redes locales de computadoras de escritorio (también denominadas *clusters*) se consideran como las plataformas más convenientes en cuanto a la relación costo/beneficio para cómputo paralelo [3] [4]. Los clusters tienen varias desventajas en cuanto a cómputo paralelo, básicamente derivadas de que ni las redes locales ni las computadoras de escritorio fueron ni son diseñadas para este tipo de procesamiento. Estas son buenas razones por las cuales la evaluación de rendimiento paralelo que se presenta en este artículo se lleva a cabo sobre un cluster utilizado para cómputo paralelo.

2. Factorización Cholesky

La factorización Cholesky se aplica a una matriz $A \in \mathbb{R}^{n \times n}$ simétrica definida positiva, y se utiliza para encontrar una matriz $L \in \mathbb{R}^{n \times n}$ triangular inferior, tal que [14]:

$$\overbrace{\begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} & \dots \\ a_{21} & a_{22} & a_{32} & a_{42} & \dots \\ a_{31} & a_{32} & a_{33} & a_{43} & \dots \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots \\ \dots & \dots & \dots & \dots & \ddots \end{pmatrix}}^{A} = \overbrace{\begin{pmatrix} l_{11} & 0 & \dots & \dots & \dots \\ l_{21} & l_{22} & 0 & \dots & \dots \\ l_{31} & l_{32} & l_{33} & 0 & \dots \\ l_{41} & l_{42} & l_{43} & l_{44} & \dots \\ \dots & \dots & \dots & \dots & \ddots \end{pmatrix}}^{L} \times \overbrace{\begin{pmatrix} l_{11} & l_{21} & l_{31} & l_{41} & \dots \\ 0 & l_{22} & l_{32} & l_{42} & \dots \\ \dots & 0 & l_{33} & l_{43} & \dots \\ \dots & \dots & 0 & l_{44} & \dots \\ \dots & \dots & \dots & \dots & \ddots \end{pmatrix}}^{L^T} \quad (1)$$

Por lo tanto, los elementos de la matriz L se calculan con $O(n^3/3)$ operaciones que, específicamente, son

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} \times l_{jk} \right) / \, l_{jj} \; ; \qquad 1 \leq j < i \leq n \tag{2}$$

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \; ; \qquad 1 \leq i \leq n \tag{3}$$

El patrón de dependencias en los cálculos es claro a partir de las ecuaciones anteriores. Según la Ec. (2), para el cálculo de l_{ij} con $j < i$ se utilizan los datos de la fila i y la fila j hasta la columna $i-1$ de ambas filas. Según la Ec. (3), para el cálculo de l_{ii} se utilizan los datos de la misma fila i desde la primera columna hasta la columna $i-1$. Puesto de otra manera, la primera fila no tiene ninguna dependencia de datos (se debe calcular solamente el primer elemento y se puede llevar a cabo inmediatamente a partir de la matriz a factorizar), y teniendo una fila k calculada se pueden calcular todos los elementos de la columna k de las filas siguientes y luego el elemento de la diagonal principal de la fila $k+1$.

3. Paralelización por Bloques de Filas

Con la idea anterior de dependencias, se puede llegar a una paralelización relativamente sencilla, avanzando por filas de la matriz y considerando a los procesos interconectados en un arreglo unidimensional. Distribuyendo la matriz a factorizar por bloques de filas, todos los procesadores tendrán un conjunto de filas consecutivas de la matriz original (y a factorizar) y tendrán datos de las columnas de la matriz original (y a factorizar) de acuerdo con las filas que tengan asignadas. La Fig. 1 muestra un ejemplo con una matriz de 8×8 elementos distribuida

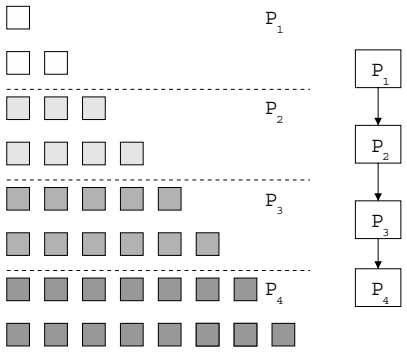

Figura 1: Distribución en Bloques: Arreglo de Procesadores.

entre cuatro computadoras, $P_1, \ldots, P_4$ (dos filas por procesador) interconectadas en un arreglo unidimensional de computadoras. Aunque la conectividad entre los procesadores puede ser bidireccional, para este caso en particular los datos (filas de L) se comunican en una sola dirección, tal como lo indican las flechas. Teniendo en cuenta esta distribución, la Fig. 2 muestra el pseudocódigo del procesamiento en cada computadora/procesador siguiendo el modelo de programación por pasaje de mensajes y el modelo de procesamiento SPMD (Single Program, Multiple Data). Se debe hacer notar que la cantidad de columnas con parte local depende del procesador, y esta cantidad de columnas con parte local es la que define la iteración principal de procesamiento. En la Fig. 1 se puede verificar claramente que el primer procesador, por

```
/* El primer procesador comienza de manera especial */
if (primer procesador)
  Computar el primer elemento de la diagonal

/* Columnas locales de las filas correspondientes */
for (k, columnas con parte local)
{
  if (Fila k de un proceso anterior)
    Recibir fila k del proceso anterior

  if (Hay proceso siguiente)
    Enviar fila k al proceso siguiente

  Computar parte local de columna k (local) con la fila k

  if (fila k+1 es local)
    Computar el elemento (k+1, k+1)
}
```

Figura 2: Distribución en Bloques: Cómputo y Comunicaciones.

ejemplo, tiene parte de solamente dos columnas y el último tiene parte de todas las columnas. Por otro lado, todo lo que se menciona como `computar` en el pseudocódigo corresponde a la implementación de la Ec. (2) y la Ec. (3) anteriores.

3.1. Breve Análisis de Rendimiento

En las bibliotecas de pasaje de mensajes las operaciones de transferencia de datos se llevan a cabo *en background* (como una operación de entrada/salida estándar desde el punto de vista del sistema operativo). Esto implica que el pseudocódigo anterior genera un procesamiento similar al de un *pipeline* ya que, por ejemplo, mientras el primer procesador opera sobre la columna k, el siguiente opera sobre la $k-1$ y en general el procesador P_c opera sobre la columna $k-c+1$. Sin embargo, en la paralelización por bloques de filas propuesta se pueden identificar problemas de rendimiento casi inmediatamente. El más evidente es el que surge de todas las factorizaciones "right-looking" que transforman algún tipo de matriz densa en una o más matrices triangulares, tales como LU, QR y Cholesky [14] [5]. En todas estas factorizaciones, las distribuciones por bloques de filas consecutivas producen inconvenientes claros de rendimiento: a medida que se avanza en el procesamiento, las primeras computadoras dejan de tener trabajo para realizar, comienzan a estar cada vez más ociosas y el desbalance de carga es immediato. En el caso de la factorización Cholesky, se agrega que, de hecho, las primeras computadoras tienen menor cantidad de datos para procesar que las últimas si la cantidad de filas por bloque es constante, tal como la que se propone (Fig. 1).

4. Paralelización con Distribución Cíclica de Filas

La forma más usual de resolver el problema de desbalance de carga de las factorizaciones right-looking es la distribución cíclica de los datos. La idea básica es que todas las computadoras

tengan datos de la matriz a procesar en la *mayoría* de las iteraciones. En el caso específico de la factorización Cholesky esto se logra de manera sencilla si todas las computadoras tienen filas que pertencen a distintos sectores de la matriz original, desde las primeras filas hasta las últimas. Y esto es justamente lo que se logra con la distribución cíclica de las filas entre las computadoras, que establece que la fila i de la matriz estará asignada a la computadora P_c si

$$c = i \text{ mód } cantc \qquad 0 \leq i \leq n-1 \tag{4}$$

donde se asume que las n filas se numeran desde 0 hasta $n-1$ y las *numtasks* computadoras se numeran desde 0 hasta $cantc - 1$. La Fig. 3 muestra esta distribución para una matriz de

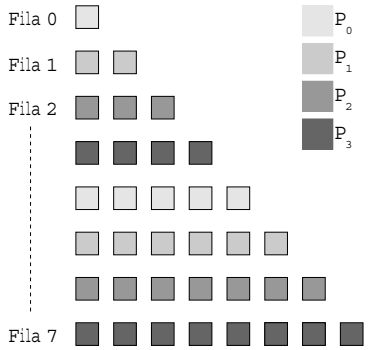

Figura 3: Distribución Cíclica de las Filas de una Matriz.

8×8 elementos a distribuir entre cuatro computadoras. La Fig. 4 muestra el pseudocódigo del proceso que se ejecuta en la computadora P_c en función de la distribución cíclica de las filas

```
/* El primer procesador comienza de manera especial */
if (primer procesador)
  Computar el primer elemento de la diagonal

/* Columnas locales de las filas correspondientes */
for (k, columnas con parte local)
{
  if (Fila k no es local)
    Recibir fila k del proceso c-1 mod numtasks

  if (Fila k no es local de c+1)
    Enviar fila k al proceso c+1 mod numtasks

  Computar parte local de columna k (local) con la fila k

  if (fila k+1 es local)
    Computar el elemento (k+1, k+1)
}
```

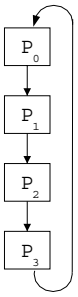

Figura 4: Procesamiento en la computadora P_c: Anillo de Procesadores.

de la matriz a factorizar. Siguiendo el ejemplo de la Fig. 3, y comparándolo con el algoritmo de la sección anterior, la computadora P_0 tendrá elementos de las filas 0 y 4 y por lo tanto

tendrá elementos a procesar por más de dos iteraciones. Dado que todas las computadoras tendrán elementos de filas no consecutivas, la computadora P_c recibirá filas de todas las demás computadoras, no solamente de las *anteriores*. Esto hace que la interconexión entre las computadoras en anillo sea más apropiada que la de un arreglo unidimensional simple. También en este caso, el procesamiento de la factorización Cholesky tiene características similares a la mayoría de los algoritmos paralelos definidos en el área de álgebra lineal: a) Programación con pasaje de mensajes, b) Procesamiento SPMD y c) Todas las comunicaciones son punto a punto. Por otro lado, todas las computadoras ahora tienen datos de *casi* todas las columnas y por lo tanto tendrán datos a procesar en la mayoría de las iteraciones lo cual, a su vez, balancea la carga de procesamiento.

5. Experimentación

Se realizaron experimentos con los algoritmos paralelos que se describen en la sección anterior, con el objetivo de evaluar su rendimiento. Los experimentos se llevaron a cabo en un cluster de PCs. La Tabla 1 describe sintéticamente las características de las computadoras utilizadas. La red de interconexión es Ethernet de 100 Mb/s y en el cableado se utiliza un único switch de

CPU	GHz	Memoria	Sistema Operativo
Intel Pentium 4	2.4	1 GB	Linux 2.4.18-14

Tabla 1: Rendimiento y Tiempo Estimado.

interconexión. El rendimiento se evalúa en función del speedup obtenido, que es una métrica muy conocida y útil en el contexto de los clusters homogéneos. Se utiliza la implementación MPICH [15] de MPI (Message Passing Interface) [18] para toda la comunicación de datos entre procesos de la aplicación paralela. La experimentación se llevó a cabo con matrices de 10000×10000 elementos, que es suficientemente grande como para que el tiempo de cómputo sea significativo y, por otro lado, el área de memoria necesaria es cercana a la disponible en las computadoras. El tamaño no se escala con la cantidad de computadoras para verificar el speedup de manera directa y para comparar el speedup de ambas formas de paralelizar basadas en las distintas formas de distribuir los datos.

La Fig. 5 muestra los resultados de rendimiento de los algoritmos paralelos que se describen anteriormente en el cluster utilizando distintas cantidades de computadoras. La serie de datos identificados con la etiqueta "Bloque"son los correspondientes al algoritmo paralelo con distribución de datos en bloques de filas. La serie de datos identificados con la etiqueta "Cíclico"son los correspondientes al algoritmo paralelo con distribución cíclica de las filas de la matriz a factorizar. Finalmente, la serie de datos identificados con la etiqueta "Optimo"son los correspondientes al speedup óptimo obtenible para cada cantidad de computadoras.

En principio, se puede notar a partir de los resultados obtenidos que el algoritmo con distribución cíclica de filas tiene mejor rendimiento que el de distribución de bloques de filas. Además, a medida que la cantidad de computadoras aumenta, la diferencia de rendimiento también aumenta en favor del algoritmo con distribución cíclica.

La Tabla 2 muestra los valores de speedup obtenido por los algoritmos para cada cantidad de computadoras, donde se puede apreciar mejor que, por ejemplo, para dos computadoras la diferencia de speedup es de aproximadamente 3.36 - 1.72 = 1.64 mientras que para ocho, la

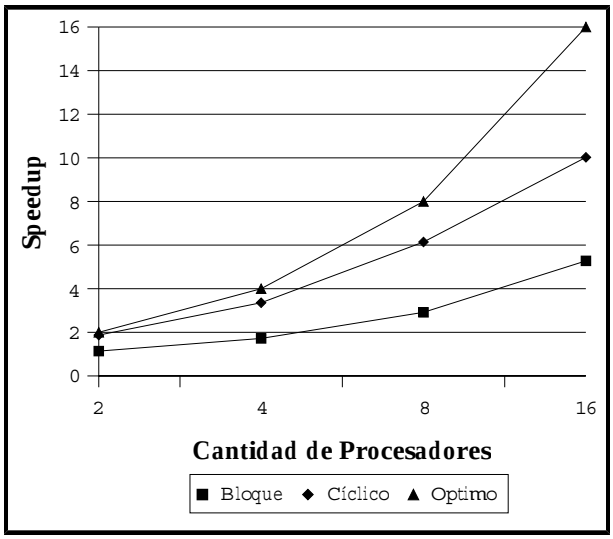

Figura 5: Experimentación con Matrices de 10000 × 10000 Elementos.

Cantidad de Computadoras	Speedup Bloque	Speedup Cíclico
2	1.14	1.88
4	1.72	3.36
8	2.92	6.14
16	5.28	10.03

Tabla 2: Valores de Speedup para Matrices de 10000 × 10000 Elementos.

diferencia de speedup es de aproximadamente 6.14 - 2.92 = 3.23. Esto significa que para dos computadoras el algoritmo de distribución cíclica aprovecha la capacidad de procesamiento de 1.64 computadoras más que el algoritmo de bloques y para ocho computadoras el aprovechamiento es mayor: hace uso de la capacidad de 3.23 computadoras más que el de bloque. La diferencia de rendimiento en detrimento del algoritmo de distribución de bloques de filas está basada en el desbalance de carga producido por las diferencias de cantidad de datos a calcular en cada computadora. Con el objetivo de confirmar que el problema de rendimiento del algoritmo de distribución de bloques es el desbalance de carga de procesamiento, se instrumentó el código de forma tal que se pueda identificar el tiempo de procesamiento de cada computadora independientemente de las comunicaciones. La Tabla 3 muestra los tiempos relativos en ejecución de procesamiento para el algoritmo de distribución de bloques. Se muestra el tiempo de procesamiento de cada computadora con respecto al tiempo total de procesamiento, que coincide con el tiempo de finalización de la computadora con el último bloque de filas. Con estos valores se puede cuantificar mejor el problema de desbalance. La computadora que recibe el primer bloque, por ejemplo, computa solamente el 0.2 % del tiempo total y el resto del tiempo permanece ociosa, dado que no hay más datos a procesar en su bloque de filas. A medida que el procesamiento avanza, más computadoras permanecen ociosas. Dado que, por ejemplo, la computadora con el bloque 9 solamente computa durante un 36.9 % del total, el 73.1 % del tiempo restante solamente hay procesamiento en seis computadoras, las que reciben los bloques 10 a 15 de la matriz original. En el caso del algoritmo con distribución cíclica de filas, la diferencia entre los tiempos de ejecución no supera el 5 %.

Computadora (Bloque de Filas)	Tiempo Relativo al Total (%)
0	0.2
1	1.0
2	2.8
3	5.3
4	8.7
5	12.9
6	17.4
7	23.1
8	29.6
9	36.9
10	45.0
11	53.9
12	63.6
13	74.1
14	85.3
15	100

Tabla 3: Tiempos Relativos de Procesamiento con Matrices de 10000×10000 Elementos.

Aunque los resultados de rendimiento del algoritmo con distribución cíclica de filas sean mejores que los de bloques de filas, esto no necesariamente significa que el rendimiento sea óptimo en sí mismo. De hecho, en la Fig. 5 se puede ver que la diferencia entre el speedup obtenido por este algoritmo y el speedup óptimo es cada vez mayor. Quizás para este análisis de rendimiento más exhaustivo del algoritmo con distribución cíclica conviene recurrir a la métrica de eficiencia paralela. La Tabla 4 muestra los valores de rendimiento en términos de

Cantidad de Computadoras	Eficiencia
2	0.94
4	0.84
8	0.77
16	0.63

Tabla 4: Eficiencia del Algoritmo con Distribución Cíclica, 10000×10000 Elementos.

eficiencia del algoritmo con distribución cíclica de filas para matrices de 10000×10000 elementos y distintas cantidades de computadoras. Si bien es cierto que al no escalar el tamaño de las matrices junto con la cantidad de computadoras se está inmediatamente bajo la ley de Amdhal [1] [16], también es posible que sean necesarias optimizaciones específicas orientadas a mejorar la escalabilidad. Una de las formas de optimización más directas a incluir es la distribución por bloques cíclica tal como la que se utiliza en [20] [2] para optimización de rendimiento secuencial y en [5] para optimización de rendimiento paralelo.

6. Conclusiones y Trabajo Futuro

En este artículo se han presentado ideas de fundamental importancia sobre la paralelización de la factorización de matrices Cholesky. También se comprobó la dependencia del rendimiento con respecto a la distribución de datos del algoritmo paralelo. El impacto de las factorizaciones *right-looking* (entre las que se puede incluir la factorización Cholesky), sobre el balance de carga en cómputo paralelo es fundamental. La experimentación presentada en referencia a los tiempos de cómputo relativos permite cuantificar la importancia del balance de carga relativa al rendimiento paralelo.

El paso inmediato siguiente incluye la aplicación de los principios de procesamiento con distribución cíclica de bloques a esta factorización. La idea es la combinación de la distribución cíclica que resuelve el problema de balance de carga con el procesamiento por bloques que optimiza el rendimiento de toda la jerarquía de memoria. También se debe adaptar el algoritmo paralelo, ahora incluyendo en la medida de lo posible las optimizaciones propias para la arquitectura paralela de un cluster homogéneo tal como en [19]. Otra de las tareas aún por resolver es la del balance de carga en los ambientes heterogéneos, que tiene su propia complejidad dada por las diferencias de capacidad de cómputo relativas entre las computadoras disponibles.

Referencias

[1] Amdhal G., "Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities", Proc. Mac. AFIPS 1967 Computer Conference, Vo. 30, pp. 483-485, Apr. 1967.

[2] Anderson E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users'Guide (Third Edition), ISBN:0-89871-447-8, SIAM Philadelphia, 1999.

[3] Anderson T., D. Culler, D. Patterson, and the NOW Team, "A Case for Networks of Workstations: NOW", IEEE Micro, Feb. 1995.

[4] Baker M., 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.

[5] Blackford L., 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, ISBN 0-89871-397-8, 1997.

[6] Dongarra J., "Performance of Various Computers Using Standard Linear Equations Software", University of Tennessee, Knoxville TN, 37996, Computer Science Technical Report Number CS - 89 - 85, January 2005, http://www.netlib.org/benchmark/ performance.ps.

[7] Chen Z., J. Dongarra, P. Luszczek, K. Roche, "Self adapting software for numerical linear algebra and LAPACK for clusters", Parallel Computing 29, pp. 1723-1743, Elsevier B.V., 2003.

[8] Chen Z., J. Dongarra, P. Luszczek, K. Roche, "The LAPACK for Clusters Project: an Example of Self Adapting Numerical Software", Proceedings of the 37th Hawaii International Conference on System Sciences, pp. 1-10, 0-7695-2056-1/04, IEEE, 2004.

[9] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, "A set of Level 3 Basic Linear Algebra Subprograms", ACM Trans. Math. Soft., 16 (1990), pp. 1–17.

[10] Dongarra J., J. Du Croz, S. Hammarling, R. Hanson, "An extended Set of Fortran Basic Linear Subroutines", ACM Trans. Math. Soft., 14 (1), pp. 1-17, 1988.

[11] Dongarra J., 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.

[12] Flynn M., "Very High Speed Computing Systems", Proc. IEEE, Vol. 54, 1966.

[13] Flynn M., "Some Computer Organizations and Their Affectiveness", IEEE Trans. on Computers, 21 (9), 1972.

[14] Golub G., C. Van Loan, Matrix Computations, 3rd Edition, The John Hopkins University Press, ISBN 0-8018-5413-X, 1996.

[15] Gropp W., E. Lusk, N. Doss, A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard", Parallel Computing, Vol. 22, No. 6, pp. 789-828, Sep. 1996.

[16] Gustafson J. L., "Reevaluating Amdhal's Law", Communications of the ACM, 31 (5) pp. 532-533, 1988.

[17] Lawson C., R. Hanson, D. Kincaid, F. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage", ACM Transactions on Mathematical Software 5, pp. 308-323, 1979.

[18] Snir M., S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra., MPI: The Complete Reference, Volume 1 - The MPI-1 Core, 2nd edition. The MIT Press, 1998.

[19] Tinetti F. G., "Parallel Linear Algebra on Clusters", VII Workshop de Investigadores en Ciencias de la Computación, WICC 2005, Universidad Nacional de Río Cuarto, Facultad de Ciencias Exactas, Físico-Matemáticas y Naturales, Departamento de Computación, pp. 301-305, 13 y 14 de Mayo de 2005.

[20] Whaley R. C., A. Petitet, J. J. Dongarra, Automated Empirical Optimization of Software and the ATLAS Project. Available at http://www.netlib.org/lapack/ lawns/lawn147.ps

[21] TOP500 Supercomputer Sites, http://www.top500.org