Paralelización de la Factorización de Matrices en Clusters

Fernando G. Tinetti
fernando@ada.info.unlp.edu.ar

Mónica Denham
mdenham@lidi.info.unlp.edu.ar

LIDI[1] - CeTAD[2]

Resumen

En este artículo se presenta un algoritmo paralelo para la resolución de la factorización de matrices del tipo LU específicamente diseñada para su implementación en redes de computadoras de escritorio (*Clusters*, *Networks of Workstations*). Además de su importancia en el ámbito de la resolución de grandes sistemas de ecuaciones, la factorización LU tiene el mismo patrón de procesamiento que otras factorizaciones también provenientes de las aplicaciones de álgebra lineal tales como QR, y en este sentido los principios de paralelización de LU son aplicables en general a los demás métodos conocidos para factorización de matrices. En este artículo también se analizan características importantes tanto para la paralelización de las factorizaciones como para el rendimiento secuencial optimizado de cada una de las computadoras que se utilizan. Se presentan los resultados obtenidos por el algoritmo en una red de computadoras homogéneas junto con el análisis de rendimiento correspondiente, que muestra la gran influencia del rendimiento de las comunicaciones (tanto a nivel físico como de la implementación de las rutinas de comunicaciones entre procesos) en el rendimiento del procesamiento paralelo en los clusters. También se mencionan las posibles extensiones y optimizaciones posibles a partir del algoritmo que se presenta.

Palabras Clave: Cómputo Paralelo en Clusters, Rendimiento Paralelo, Balance de Carga, Aplicaciones de Algebra Lineal, Factorización LU.

1.- Introducción

La relación costo-rendimiento de las computadoras de escritorio es sumamente favorable para su utilización en la resolución de los problemas de cómputo numérico en general y de aplicaciones de álgebra lineal en particular desde hace varios años. De hecho, existe un fuerte énfasis en cuanto a la construcción de computadoras paralelas-distribuidas basadas directamente en redes locales. En este sentido, desde el punto de vista del hardware de procesamiento estas plataformas pertenecen a la clase MIMD (Multiple Instruction - Multiple Data stream) [10] y el modelo de programación que se ha utilizado extensivamente es el de pasaje de mensajes, derivado de CSP (Communicating Sequential Processes) [13]. Las bibliotecas de uso libre para pasaje de mensajes entre procesos de una aplicación paralela destinados a este tipo de procesamiento paralelo-distribuido tales como PVM [8] e implementaciones de MPI [18] como LAM-MPI [24] y MPICH [25] han sido parte activa en todo el desarrollo y utilización satisfactoria de las redes de computadoras como máquinas paralelas.

Existen numerosas publicaciones con respecto a la utilización de clusters como arquitecturas paralelas, inicialmente con máquinas homogéneas, tales como las instalaciones denominadas

1 Laboratorio de Investigación y Desarrollo en Informática, Fac. de Informática, Calle 50 y 115, 1900 La Plata
2 Centro de Técnicas Analógico-Digitales, Fac. de Ingeniería, Calle 48 y 116, 1900 La Plata

Beowulf [5], NOW (Network of Workstations) [2] y *clusters* en general [3]. Sin embargo, aún existe un fuerte énfasis en cuanto a su utilización en el área de las aplicaciones denominadas *embarrassingly parallel* [11], que tienen muy pocas operaciones de comunicación y/o sincronizaciones durante el tiempo de cálculo. Específicamente en el área de las aplicaciones de cálculo numérico en general y de álgebra lineal en particular también el énfasis está puesto en la adaptación o directamente en la utilización de los algoritmos paralelos ya *clásicos*, que han sido el resultado de la evolución de la algorítmica paralela en las computadoras paralelas clásicas, también denominadas *supercomputadoras*. El gran problema reconocido en general es el de la relación entre la velocidad o capacidad de cómputo con la de comunicaciones, que es sumamente desfavorable en los *clusters* cuando se los compara con las máquinas paralelas tradicionales [23]. Una de los resultados en general reconocidos en la bibliografía es el de la biblioteca ScaLAPACK, Scalable LAPACK (Linear Algebra PACKage) [6], que es justamente la expresión de esta línea de investigación específicamente en el área de álgebra lineal.

En el área de álgebra lineal y a partir de la definición de la biblioteca LAPACK [1], la atención se ha dirigido, por un lado, a un subconjunto muy reducido pero muy importante desde el punto de vista del rendimiento de operaciones básicas denominadas BLAS (Basic Linear Algebra Subroutines) [7]. Por otro lado, también se reconoce que las operaciones incluidas en BLAS no son más que partes muy pequeñas de aplicaciones y que su utilización debería orientarse hacia problemas relativamente más complicados como los de las factorizaciones de matrices: LU, QR, etc., [12]. De hecho, el algoritmo propuesto en este artículo para la factorización LU de matrices está basado en un algoritmo básico para la multiplicación de matrices en clusters [20]. Como se aclara antes, el tipo de procesamiento de matrices establecido por los métodos clásicos de factorización de matrices (que son los que se utilizan) es similar y, por lo tanto, la metodología de paralelización de la factorización LU en realidad es común a todos los métodos.

La sección siguiente identificar de manera resumida las características de las redes locales que se deben utilizar y/o tener en cuenta para la paralelización de aplicaciones en general y de las aplicaciones provenientes del área de álgebra lineal en particular. La tercera sección describe el método de factorización LU con respecto a la utilización del procesamiento por bloques y presenta a nivel de pseudocódigo el algoritmo paralelo propuesto para ser utilizado en los clusters de computadoras de escritorio. La cuarta sección muestra los resultados obtenidos en un cluster en particular, que puede ser considerado una instalación Beowulf *clásica*. Finalmente se identifican las principales conclusiones a partir de los resultados obtenidos y a su vez se dan las extensiones inmediatas del algoritmo y de la tarea de investigación a desarrollar.

2. Principios de Paralelización de Aplicaciones de Algebra Lineal

Uno de los principales problemas que se debe enfrentar para la obtención de rendimiento *aceptable* del cómputo paralelo utilizando redes locales es el del propio origen u orientación del diseño de las redes locales. El hardware de base con el que se construye una red local o una instalación Beowulf es igual: computadoras de escritorio (PCs o estaciones de trabajo *clásicas*) con placas o interfases de interconexión en red (NIC: Network Interfase Card). Tanto las computadoras de escritorio que se utilizan como las propias placas de red no han sido ni en principio son diseñadas para cómputo paralelo. Se reconoce que el problema fundamental que aún no se ha resuelto totalmente desde el punto de vista del rendimiento es el de las comunicaciones. Aunque se han hecho numerosos esfuerzos para resolver este problema (desde la utilización de switches de interconexión y múltiples placas de red por computadora hasta la adaptación de los sistemas operativos) el problema aún sigue abierto y se siguen proponiendo soluciones más o menos específicas. Estas soluciones se proponen, en general, desde dos ámbitos: el de los algoritmos paralelos y el de las propias comunicaciones.

En [20] se resumen los principales algoritmos dedicados a la multiplicación de matrices en paralelo a utilizar en los clusters y se llega a la conclusión de que son demasiado influidos u orientados hacia las computadoras paralelas tradicionales. Esto implica que, si bien se pueden implementar y utilizar en los clusters, las dificultades para la obtención de rendimiento aceptable son muy difíciles de resolver aún desde el punto de vista mismo del análisis de los algoritmos, sin la necesidad de llegar a la experimentación. El mismo análisis se puede llevar a cabo en el ámbito de la paralelización de los métodos de factorización de matrices, aunque tanto los métodos como las paralelizaciones propuestas son mucho más acotados que en el caso de la multiplicación de matrices.

Las principales características de las redes locales que se deben tener en cuenta para la paralelización de aplicaciones en general y provenientes del álgebra lineal en particular son:

- Muy bajo acoplamiento. Esto implica, entre otras cosas, que la relación cómputo-comunicación de cada computadora es muy desfavorable para llevar a cabo cómputo paralelo.
- Mayoritariamente basadas en redes Ethernet [15]. Esto implica, entre otras cosas, que existe la posibilidad de broadcast físico de la información y además si se utilizan switches [19] [21] de interconexión en el cableado de la red también se pueden tener múltiples transferencias de información punto a punto.

Por lo tanto, los principios de paralelización de aplicaciones de álgebra lineal propuestos en [20] y que se mantienen para este algoritmo son:

- Distribución de datos sencilla. En el caso de computadoras homogéneas, que ha sido el más reportado hasta ahora y hacia el que se orienta este artículo, la distribución de datos debería ser trivial, dado que el hardware de procesamiento no implica ningún tipo de complejidad para esta tarea.
- Algoritmo paralelo basado en comunicaciones broadcast. Esto está directamente determinado por el hardware de comunicaciones Ethernet que tiene la posibilidad de realizar broadcast físico para las transferencias de datos

Además, los principios de paralelización que en general se mantienen en el área de procesamiento numérico también se utilizan para este algoritmo:

- Modelo de procesamiento SPMD (Single Program - Multple Data). Un único programa se replica y ejecuta en todas las computadoras.
- El balance de carga se implementa o es definido por la distribución de los datos.

3. Algoritmo de Factorización LU

El método de factorización de matrices denominado LU es ampliamente conocido y aceptado en cuanto a su utilidad como a sus propiedades de estabilidad numérica (específicamente cuando se utiliza pivoteo, al menos parcial) como requerimientos de cómputo y almacenamiento. La definición *inicial* del método se orienta a la resolución de sistemas de ecuaciones, y se basa de forma directa en lo que se conoce como Eliminación Gaussiana [17] [12]. Inicialmente se describirá el algoritmo secuencial de factorización LU por bloques y posteriormente se presentará el algoritmo paralelo propuesto para clusters de computadoras.

3.1 Algoritmo Secuencial de Factorización LU por Bloques

Dada una matriz cuadrada A de orden *n*, se buscan dos matrices denominadas usualmente L y U también cuadradas de orden *n* forma tal que

$$A = L \times U \quad (1)$$

y donde L es triangular inferior y U es triangular superior.

Se puede demostrar que si A es no singular, L y U existen y son únicas [12]. El método de factorización LU no hace más que aplicar sucesivamente pasos de eliminación gaussiana para que de manera iterativa se calculen los elementos de las matrices L y U. Normalmente y con el objetivo de estabilizar los cálculos desde el punto de vista numérico (básicamente para acotar el error) también se incorpora la técnica de pivoteo parcial dentro del método LU.

Desde el punto de vista de los requerimientos de memoria no se agrega ninguno más que el propio almacenamiento de la matriz A que se factoriza y por lo tanto se mantiene en el $O(n^2)$. Desde el punto de vista de los requerimientos de cómputo, la cantidad de operaciones de punto flotante que se necesitan para el cálculo de LU es $O(n^3)$. De esta manera se llega a la relación básica que se tiene para las operaciones BLAS de nivel 3 [7] [9], es decir que la cantidad de operaciones de punto flotante es $O(n^3)$ con $O(n^2)$ datos que se procesan.

Con el objetivo de aprovechar la arquitectura de cálculo subyacente en la mayoría de las computadoras, y específicamente la jerarquía de memoria con la inclusión de al menos un nivel de memoria cache se han definido la mayoría de las operaciones de álgebra lineal en términos de bloques de datos (o submatrices) [9]. Específicamente en el contexto del método LU se determina una partición de la matriz A tomando como base un bloque o submatriz de A, A_{00}, de $b \times b$ datos tal como lo muestra la Fig. 1.

A_{00}	A_{01}
A_{10}	A_{11}

Figura 1: División de una Matriz en Bloques.

Se busca la factorización LU de A, que expresada en función de la división en bloques anterior sería tal que

A_{00}	A_{01}		L_{00}	0		U_{00}	U_{01}
A_{10}	A_{11}	=	L_{10}	L_{11}	×	0	0	U_{11}

donde

$$A_{00} = L_{00} U_{00} \quad (2)$$
$$A_{01} = L_{00} U_{01} \quad (3)$$
$$A_{10} = L_{10} U_{00} \quad (4)$$
$$A_{11} = L_{10} U_{01} + L_{11} U_{11} \quad (5)$$

y las matrices L_{ij} son triangulares inferiores y U_{kl} ($0 \leq i, j, k, l \leq 1$) son triangulares superiores. Si se aplica la factorización LU de manera directa (*clásica*) al bloque de A A_{00}, se tienen L_{00} y U_{00} tal que se verifica la Ec. (2). Utilizando la Ec. (3) se tiene que $L_{00}\ U_{01} = A_{01}$, y como L_{00} es triangular inferior se puede aplicar de manera directa el método de resolución de un sistema de ecuaciones *triangular* con múltiples resultados (*multiple right hand side*) para obtener cada una de las columnas de U_{01}. De la misma manera, o de manera similar se utiliza la Ec. (4) para la obtención de L_{10}, dado que $L_{10}\ U_{00} = A_{10}$ y en este caso U_{00} es triangular superior y se puede aplicar también de manera directa la solución de un sistema de ecuaciones *triangular* con múltiples resultados (*multiple right hand side*). Solamente faltaría el cálculo de L_{11} y U_{11} para tener toda la factorización de la matriz A. En este caso, se utiliza la Ec. (5) llegando a que $L_{11}\ U_{11} = A_{11} - L_{10}\ U_{01}$, es decir que hallar las matrices L_{11} y U_{11} implica la aplicación del mismo método LU por bloques a la (sub)matriz resultado de $A_{11} - L_{10}\ U_{01}$. Lo único que no se ha mencionado de manera explícita es el procesamiento necesario asociado a los pivotes, que básicamente implica seleccionar el pivote en cada fila o columna (según lo que se elija, y es análogo en cuanto a estabilidad numérica) y el intercambio de filas o columnas que corresponden.

Con el método por bloques explicado no solamente se pueden definir todas las operaciones en términos de submatrices de la matriz original A sino que también se tienen dos características de procesamiento que pueden ser utilizadas satisfactoriamente en cuanto a optimización de código:

1. La mayor parte de las operaciones de punto flotante se ejecutan para resolver $A_{11} - L_{10}\ U_{01}$, que es básicamente una multiplicación de matrices.
2. Todo el método de factorización LU puede ser resuelto utilizando dos rutinas definidas en BLAS, que son la de resolución de sistemas de ecuaciones triangulares y multiplicación de matrices (_xtrsm y _xgemm respectivamente en términos de la interfase C de BLAS) y una de LAPACK (_xgetrf, en términos de la interfase C de LAPACK).

3.2 Algoritmo Paralelo de Factorización LU

Para la definición del algoritmo paralelo se utiliza de manera directa la idea de procesamiento por bloques como en las propuestas de factorizaciones que existen hasta el momento [9]. Como en la mayoría de los algoritmos paralelos de procesamiento numérico provenientes del área de álgebra lineal, se analiza por un lado la distribución de los datos (en este caso la matriz a factorizar) y por el otro el procesamiento de los datos que están distribuidos entre los procesadores (en este caso las computadoras del cluster a utilizar como computadora paralela).

Distribución de Datos. Dado que se tiende a que todas las comunicaciones sean del tipo broadcast, las distribuciones de datos *unidimensionales* son favorecidas por sobre las bidimensionales o las que tienen en cuenta la interconexión en hipercubos, por ejemplo. La comunicación (o las transferencias de datos) entre las computadoras que se tiende a aprovechar es la definición misma del estándar Ethernet en cuanto a la interconexión lógica de las computadoras con un único bus. En este sentido la distribución de datos es unidimensional aunque la interconexión de las computadoras no es la de un anillo, que se considera *clásico* en el contexto de la organización unidimensional de los procesadores [16].

Una vez que se restringe la distribución de los datos a las unidimensionales, se tienen solamente dos alternativas que son análogas: por filas o por columnas. Si bien inicialmente lo que se puede proponer es la división de la matriz completa en tantas partes como procesadores (computadoras) disponibles, esto implica una pérdida directa del balance de carga de procesamiento. Si, por ejemplo, se tienen cuatro procesadores P_0, P_1, P_2, y P_3, y se divide la matriz en cuatro bloques de filas tal como muestra la Fig. 2, cuando se terminan de procesar las filas que se asignan al

procesador P_0, de hecho el procesador P_0 no tiene ninguna tarea de procesamiento más. Esto significa que a partir de ese instante toda la tarea de factorización de la matriz se realiza sin la capacidad de procesamiento de P_0. Algo similar ocurre cuando se llegan a procesar posteriormente las filas asignadas a P_1, a partir de lo cual todo el procesamiento siguiente se hace solamente en los procesadores P_2 y P_3, y esto implica claramente que el procesamiento no está balanceado entre los cuatro procesadores.

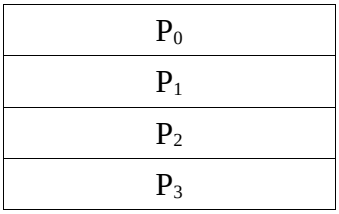

Figura 2: Partición y Asignación de una Matriz por Bloques de Filas.

En este momento se utiliza directamente la idea de procesamiento por bloques, así como las distribución que se ha llamado cíclica por bloques. Se establece un tamaño de bloques que sea relativamente pequeño con respecto al tamaño total de la matriz y la distribución se realiza por bloques de este tamaño elegido. En el caso de las distribuciones unidimensionales, este tamaño de bloques es la cantidad de filas o columnas que se distribuyen como una unidad. Si, por ejemplo, se determina que se tienen ocho bloques en total y cuatro procesadores, la distribución cíclica por bloques de columnas se realiza como lo muestra la Fig. 3, donde el bloque *i* se asigna al procesador *i* mod *P* donde *P* es la cantidad total de procesadores. Mientras mayor es la cantidad de bloques mayor es también el balance de carga resultante. Dado que normalmente la cantidad de filas y columnas de las matrices a procesar es mucho mayor que la cantidad de procesadores el balance de carga implementado de esta forma no presenta inconvenientes.

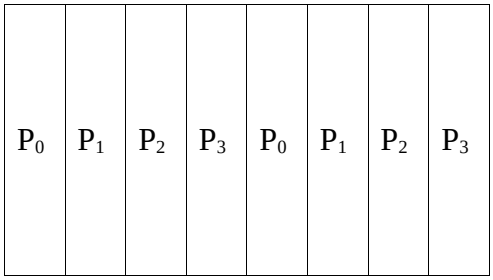

Figura 3: Distribución Cíclica por Bloques de Columnas.

De esta manera, se llega a cumplir el primero de los principios de paralelización enumerados en la sección anterior: distribución de datos sencilla. A la vez se logra con la distribución de datos el balance de carga necesario para obtener rendimiento aceptable en el procesamiento de la factorización LU.

Procesamiento. Como la gran mayoría de las aplicaciones numéricas provenientes del área de álgebra lineal, el modelo de procesamiento es SPMD, todas las computadoras en el cluster ejecutan el mismo programa. El procesamiento que se realiza en cada una de las computadoras del cluster, se muestra a nivel de pseudocódigo para la máquina C_i ($0 \leq i \leq P$-1) en la Fig. 4. Como se puede notar en el pseudocódigo, todas las comunicaciones son del tipo broadcast y por lo tanto si se optimiza este tipo de transferencias de datos entre los procesos de una aplicación paralela (utilizando la

capacidad de broadcast físico de las redes Ethernet, por ejemplo) se optimiza casi de manera directa el rendimiento del procesamiento paralelo para la factorización LU. De esta manera, también se llega a satisfacer el segundo de los principios de paralelización enumerados en la sección anterior: algoritmo paralelo basado en comunicaciones broadcast.

C_i

```
for (j = 0; j < Cant_Blq; j++)
{
  if (i == (j mod P))
  {
    Factorización del Bloque j en LU (_xgetrf)
    Broadcast_send de los pivotes-intercambios de filas o columnas
    Broadcast_send del bloque j factorizado (en LU)
  }
  else
  {
    Broadcast_receive de los pivotes-intercambios de filas o columnas
    Broadcast_receive del bloque j factorizado (en LU)
  }
  Actualización del resto de la matriz de acuerdo con los pivotes
  Resolver L10 U00 = A10 (_xtrsm)
  Resolver A11 - L10  U01 (_xgemm)
}
```

Figura 4: Pseudocódigo del Algoritmo Paralelo de Factorización LU.

Se debe notar en el pseudocódigo de la Fig. 4 que se incorpora explícitamente todo lo que corresponde al manejo de las filas o columnas de los pivotes porque ahora la matriz está distribuida entre los procesadores y por lo tanto los intercambios que se producen por los pivotes deben ser explícitamente distribuidos desde la computadora que hace la factorización de un bloque. Por otro lado, asumiendo por ejemplo que la matriz se divide en bloques de filas, al hacer la factorización de un bloque de filas se tienen calculados los bloques que en la Fig. 1 se muestran como L_{00}, U_{00} y U_{01} y por lo tanto lo único que se debe calcular son los bloques L_{10} y $A_{11} - L_{10}\ U_{01}$ (sobre el cual continúa la aplicación del método por bloques).

4. Experimentación

La experimentación se realizó sobre lo que se podría considerar una instalación Beowulf clásica: ocho computadoras iguales con 64 MB de memoria principal interconectadas con placas de red Ethernet de 100 Mb/s y en el cableado se utiliza un switch también con capacidad de 100 Mb/s. Se utilizó PVM como la biblioteca de pasaje de mensajes entre procesos de la aplicación paralela. Los experimentos se orientaron en varios sentidos. Los parámetros a considerar fueron:

- Tamaño de matrices a factorizar. Se tomó como tamaño de referencia el mayor posible tal que la factorización secuencial (en una sola máquina) no debe recurrir al espacio de memoria swap para realizar los cálculos. De esta manera, se logra un tamaño relativamente grande de matrices que justifica la necesidad de paralelización y también se evita la utilización de matrices muy pequeñas con respecto a los tamaños de memoria cache que pueden desvirtuar los resultados en cuanto a rendimiento obtenido.
- Tamaño de bloque. Se utilizaron diferentes tamaños de bloque para realizar todas las operaciones necesarias en la factorización, tal como se explica en la sección anterior. Los valores varían entre

1 y 256, y para evitar una gran cantidad de pruebas muy similares se tomaron las potencias de 4 entre estos dos valores. Dado que se suele mencionar que 32 es el tamaño de bloque más usual en las computadoras de escritorio [4] [6] también se utilizó este tamaño de bloque. Resumiendo, los diferentes tamaños de bloque utilizados fueron: 1, 4, 16, 32, 64 y 256.

- Cantidad de computadoras. Para verificar el comportamiento del algoritmo a medida que se utilizan mayor cantidad de computadoras se llevaron a cabo experimentos con todas las cantidades posibles de computadoras, es decir: 1, 2, 3, ...8.

El código secuencial utilizado es totalmente optimizado siguiendo los lineamientos dado por la biblioteca denominada ATLAS (Automatically Tuned Linear Algebra Software) [22], y el principal índice de rendimiento obtenido es directamente el tiempo total de ejecución.

La Fig. 1 muestra los resultados obtenidos para matrices cuadradas de orden 3500. Sobre el eje **x** se muestran las diferentes cantidades de computadoras que se utilizaron (una tarea de cómputo en cada computadora), el eje **y** del gráfico muestra los diferentes valores para el tamaño de bloques y sobre el eje **z** se muestra el tiempo total de ejecución dado en segundos.

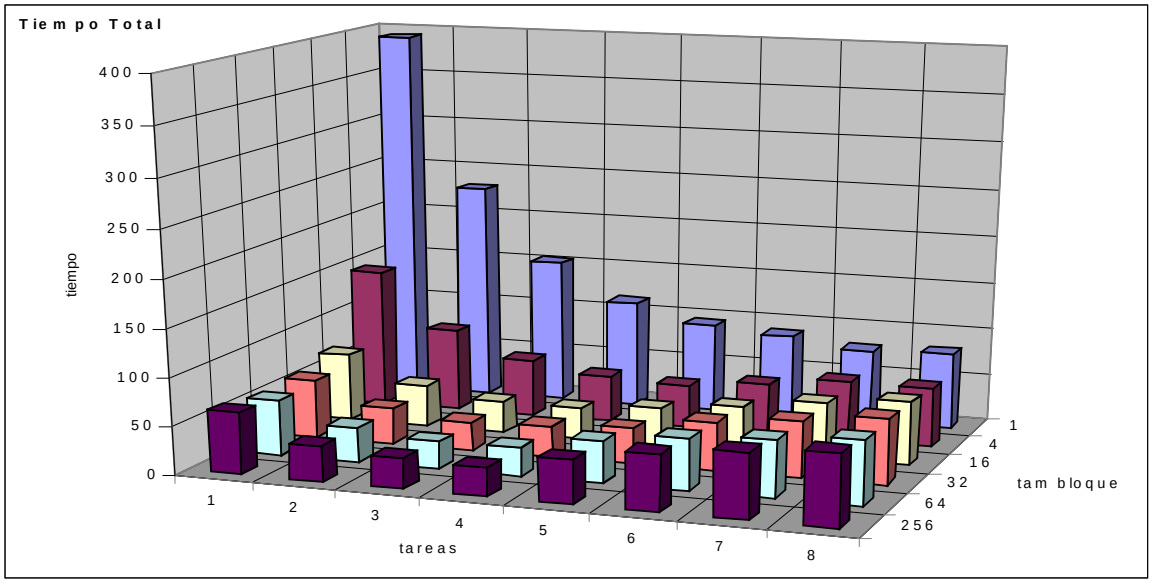

Figura 1: Tiempos de Ejecución de la Factorización LU.

Teniendo en cuenta los resultados obtenidos para la ejecución secuencial, es decir en una sola computadora y con diferentes tamaños de bloques se puede verificar el impacto que los tamaños de bloque muy pequeños tienen sobre el rendimiento. De hecho, se muestra que para tamaño de bloque 1 (una fila completa, en este caso) el código secuencial está lejos de obtener el máximo rendimiento posible y por lo tanto se pierde toda capacidad de optimización que el procesamiento por bloques permiten en las aplicaciones secuenciales.

Tomando como referencia el tiempo de ejecución con tamaño de bloque 1, se puede notar que el rendimiento mejora notablemente a medida que se utiliza mayor cantidad de máquinas hasta 4 computadoras. Desde el agregado de la quinta computadora en adelante, la mejora no es significativa. Si se comparan estos resultados con los que se tienen para el mejor tamaño de bloque (64), además de las diferencias absolutas de tiempos de ejecución, las relaciones de ganancia por computadora agregada son diferentes. Cuando se utiliza el mejor tamaño de bloque solamente se tiene mejor rendimiento hasta el agregado de la tercera computadora, al utilizar mayor cantidad de computadoras no solamente no mejora el rendimiento sino que en realidad es peor. Esta característica es bastante usual en las aplicaciones de cálculo intensivo con código secuencial

optimizado, y no se verifica la degradación de rendimiento a medida que se agregan computadoras para el tamaño de bloque 1 solamente porque el tamaño de bloque 1 implica que el código secuencial (que se ejecuta en cada computadora) no es totalmente optimizado. De hecho, esta es la razón básica para utilizar el mejor algoritmo secuencial para el cálculo de speedup [14] de las aplicaciones paralelas, dado que la utilización de código no optimizado tiende a obtener mucho mayor speedup que con código totalmente optimizado.

Más allá del análisis anterior que se puede repetir para los distintos tamaños de bloque, el análisis del algoritmo paralelo propuesto debe hacerse para el tamaño de bloque 64 que es el de mayor rendimiento secuencial. Los resultados de la experimentación que se muestran en la Fig. 1 en realidad no son aceptables por dos razones:

1. No se logra utilizar de manera eficiente la capacidad de todas las computadoras a partir de la inclusión de la cuarta máquina hasta la octava.
2. A priori no se tiene una idea clara de cuál es la mayor cantidad de computadoras que se pueden utilizar satisfactoriamente, o para las cuáles el rendimiento es proporcional a la capacidad de cálculo de todas las computadoras entre las cuales se distribuyen los cálculos.

Esto puede ser debido a que:

- El algoritmo propuesto falla en la distribución de los datos, en el balance de carga, en la forma de comunicar los datos o en alguna otra característica que penaliza el rendimiento paralelo.
- La granularidad de la aplicación hace que a partir de la cuarta máquina en adelante el tiempo de comunicaciones sea mayor que el de procesamiento. Si esta es la razón, se tendría en este caso un índice importante para la determinación *a priori* de la máxima cantidad de computadoras que se pueden utilizar con rendimiento *aceptable*.

Sin embargo, hasta ahora no se disponen de datos suficientes como para identificar la/s causa/s de la degradación de rendimiento a partir de la inclusión de la cuarta computadora.

Se puede recurrir en este caso a los principales datos con los cuales se determina lo que sucede en tiempo de ejecución: tiempo dedicado específicamente a cálculo y a comunicaciones en cada uno de los procesos. La Tabla 1 muestra cuánto tiempo se utiliza para cómputo y cuánto para comunicaciones en cada proceso (se mantiene el tamaño de bloque de 64).

Computadoras	***Cómputo (seg.)***	***Comunicaciones (seg.)***
1	57.82	-
2	30.29	6.03
3	20.37	8.37
4	15.67	14.54
5	13.07	30.34
6	10.60	43.02
7	8.36	51.62
8	7.15	59.98

Tabla 1: Cómputo y Comunicaciones en cada Computadora.

A partir de la información de la Tabla 1, se pueden analizar tres aspectos relacionados con el rendimiento del algoritmo propuesto:

1. Tiempo de cómputo relacionado con la cantidad de máquinas y el rendimiento por máquina.
2. Tiempo de comunicaciones relacionado con el rendimiento de la red de comunicaciones y con la

implementación de los mensajes broadcast.
3. Tiempo de cómputo relacionado con el tiempo de comunicaciones, con el que se tiene una idea de la granularidad de la aplicación.

Tiempo de cómputo. Es el que se espera a priori, dado que en general disminuye proporcionalmente con la cantidad de computadoras que se utilizan. Cuando dos computadoras computan en paralelo el tiempo de cómputo es aproximadamente la mitad del tiempo secuencial, cuando se incluye la tercera el tiempo es aproximadamente un tercio y se continúa con esta tendencia. Esto implica que el cómputo se distribuye adecuadamente y además cada computadora siempre realiza el procesamiento secuencial a su máxima capacidad de cálculo.

Tiempo de comunicaciones. Es el problema real de rendimiento. En realidad se tienen dos problemas relacionados con las comunicaciones: rendimiento e implementación de los mensajes broadcast. En cuanto a rendimiento de las comunicaciones: para matrices cuadradas de orden 3500, la cantidad aproximada de bytes es 49 MB, si se tiene en cuenta que la red de comunicaciones es de 100 Mb/s, el tiempo óptimo de comunicaciones sería aproximadamente 4 segundos, dado que para dos computadoras se utilizan un poco más de 6 segundos ya se tiene una degradación relativamente importante de rendimiento con respecto al óptimo. Sin embargo, esta degradación no sería tan importante si se mantuviera invariante con respecto a la cantidad de computadoras, pero esto no sucede.

A pesar de que la cantidad de datos que se tienen que comunicar es siempre la misma (básicamente, toda la matriz), a medida que la cantidad de máquinas utilizadas es mayor el tiempo de comunicaciones aumenta proporcionalmente. Esto relaciona enfoca directamente la forma en que se implementan los mensajes broadcast: se traduce (en PVM) a múltiples mensajes punto a punto. Es decir que PVM no aprovecha la capacidad de broadcast de las redes Ethernet ni tampoco aprovecha la posibilidad de múltiples mensajes punto a punto con los cuales podría implementar los mensajes broadcast. Esto a su vez implica que la mayor parte del tiempo total de ejecución del algoritmo paralelo lo único que se hace es comunicar los (mismos) datos entre las máquinas.

Granularidad. Dado que la degradación se debe completamente a la implementación de los mensajes broadcast, no es necesario analizar la relación cómputo-comunicaciones en este punto. Sin embargo, teniendo una implementación de los mensajes broadcast *razonable* en términos de rendimiento, se mantiene la posibilidad de un análisis *a priori* de granularidad para la determinación de la cantidad de computadoras *óptima*.

De esta manera, se llega a que un algoritmo sencillo puede tener rendimiento aceptable y hasta optimizado si los mensajes broadcast se implementan de manera también adaptada a las redes de interconexión Ethernet disponibles en la mayoría de los clusters.

5. Conclusiones y Trabajo Futuro

Se ha presentado un algoritmo directamente orientado al procesamiento paralelo en clusters. Si bien se lo puede relacionar con los algoritmos propuestos para las máquinas paralelas tradicionales, dos características son distintivas: distribución de los datos y utilización de mensajes broadcast únicamente en lo que se refiere a las comunicaciones entre procesos.

La experimentación muestra claramente que la distribución de datos propuesta así como el balance de carga obtenido es aceptable para este tipo de procesamiento paralelo-distribuido. En este sentido, no se hace más que comprobar que las propuestas hechas en la bibliografía son apropiadas

para el procesamiento paralelo en clusters y por lo tanto es válido aprovecharlas.

También la experimentación desarrollada muestra claramente que el bajo rendimiento obtenido por el algoritmo se debe a la implementación que la biblioteca PVM hace de los mensajes broadcast, y que es inapropiada desde el punto de vista del rendimiento en las redes Ethernet. Aprovechando la capacidad de los switches y/o directamente la definición misma de la norma Ethernet podría mejorar de manera directa y sustancial el rendimiento de los mensajes broadcast y también del algoritmo mismo.

Como mínimo se debería utilizar una biblioteca que aproveche algunas de las capacidades disponibles en las redes Ethernet para realizar el broadcast de los datos para que el algoritmo tenga rendimiento aceptable. En este sentido, algunas de las implementaciones de MPI indican que en su implementación utilizan múltiples mensajes punto a punto simultáneos para la implementación de los mensajes broadcast [24]. Utilizar una de estas bibliotecas mejoraría directamente el rendimiento del algoritmo propuesto.

Otras extensiones que no son tan sencillas aunque útiles incluyen el estudio y adaptación del algoritmo propuesto para los clusters con computadoras heterogéneas. En este caso se debería establecer muy claramente cómo se adaptan tanto el tamaño de bloques utilizado como la distribución de los datos entre máquinas con diferentes capacidades de cálculo. La adaptación y/o propuestas de algoritmos para múltiples clusters interconectados siempre es un objetivo que no se debe dejar de lado, aunque su complejidad es mucho mayor.

Bibliografía

[1] Anderson E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide (Second Edition), SIAM Philadelphia, 1995.

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

[3] 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.

[4] Beaumont O., V. Boudet, A. Petitet, F. Rastello, Y. Robert, “A Proposal for Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)”, IEEE Transactions on Computers, vol. 50, No. 10, pp. 1052-1070, Oct. 01.

[5] Becker D. J., T. Sterling, D. Savaresse, J. E. Dorband, U. A. Ranawak, C. W. Packer, “Beowulf: A Parallel Workstation for Scientific Computation”, Proc. of the International Concerence on Parallel Processing, vol. 1, pp. 11-14, Boca Raton, Florida, Aug. 1996.

[6] 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, 1997.

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

[8] Dongarra J., A. Geist, R. Manchek, V. Sunderam, Integrated pvm framework supports heterogeneous network computing, Computers in Physics, (7)2, pp. 166-175, April 1993.

[9] 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.

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

[11] Foster I., Designing and Building Parallel Programs, Addison-Wesley, Inc., 1995. *Versión html* disponible en http://www-unix.mcs.anl.gov/dbpp
[12] Golub G. H., C. F. Van Loan, Matrix Computation, Second Edition, The John Hopkins University Press, Baltimore, Maryland, 1989.
[13] Hoare C., Communicating Sequential Processes, Englewood Cliffs, Prentice-Hall, 1986.
[14] Hockney R., C. Jesshope, Parallel Computers 2, Adam Hilger, Bristol and Philadelphia, IOP Publishing Ltd., 1988.
[15] Institute of Electrical and Electronics Engineers, Local Area Network - CSMA/CD Access Method and Physical Layer Specifications ANSI/IEEE 802.3 - IEEE Computer Society, 1985.
[16] Leighton F, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes", Morgan Kaufman Publishers, 1992.
[17] Luenberger D. G., Linear and Nonlinear Programming, Second Ed., Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, USA, 1984.
[18] MPI Forum, "MPI: a message-passing interface standard", International Journal of Supercomputer Applications, 8 (3/4), pp. 165-416, 1994.
[19] Spurgeon C., Ethernet Configuration Guidelines, Peer-to-Peer Communications, San Jose, CA, 1996.
[20] Tinetti F., A. Quijano, A. De Giusti, E. Luque, "Heterogeneous Networks of Workstations and the Parallel Matrix Multiplication", Proceedings Euro PVM/MPI 2001, Santorini, Greece, Apr. 2002.
[21] Trulove J., LAN Wiring, Mc-Graw Hill, New York, 1997
[22] Whaley R., J. Dongarra, "Automatically Tuned Linear Algebra Software", Proceedings of the SC98 Conference, Orlando, FL, IEEE Publications, November, 1998.
[23] Wilkinson B., Allen M., Parallel Programming: Techniques and Applications Using Networking Workstations, Prentice-Hall, Inc., 1999.
[24] LAM/MPI (Local Area Computing / Message Passing Interface) Home Page
http://www.mpi.nd.edu/lam
[25] MPICH Home Page http://www-unix.mcs.anl.gov/mpi/mpich/