Balance de Carga para Factorización LU de Matrices en Clusters Heterogéneos.

Fernando G. Tinetti[1], Mónica Denham
III-LIDI (Instituto de Investigación en Informática LIDI)
Facultad de Informática, UNLP
50 y 115, 1900 La Plata
Argentina
{fernando, mdenham}@lidi.info.unlp.edu.ar
Tel.: +54 221 4227707, Fax: +54 221 4273235

Resumen

En este trabajo se intenta resolver de forma correcta el balance de carga para la factorización LU de matrices en clusters heterogéneos. Se utiliza una solución paralela a la factorización LU especialmente desarrollada para obtener buen rendimiento en clusters heterogéneos. Se proponen distintas alternativas para distribuir los datos entre los procesos, las cuales se han implementado y evaluado. Se muestran los resultados obtenidos, su análisis y las conclusiones a las que se llegaron.

1. Introducción

El cómputo paralelo es utilizado principalmente en áreas donde existe gran requerimiento de cómputo o donde se debe procesar gran volumen de datos. Así, el objetivo del cómputo paralelo es obtener soluciones más rápidas, obtener más respuestas en un tiempo determinado u obtener respuestas en tiempos razonables. Para esto es necesario que las aplicaciones paralelas tengan buen rendimiento.

Para obtener aplicaciones paralelas con buen rendimiento, es conveniente considerar las características de la arquitectura para la cual se está desarrollando la solución. De esta forma, si se consideran las características de la arquitectura en el momento mismo del análisis del problema, se pueden diseñar e implementar soluciones que consideren estas características maximizándose el aprovechamiento de las que puedan aumentar el rendimiento y a su vez evitar las características que lo penalicen.

En la actualidad existen diversas arquitecturas paralelas, las cuales han evolucionado año tras año, gracias al avance tecnológico de los componentes que la integran. Este avance, obliga también a los desarrolladores a mejorar las soluciones adaptándolas a las nuevas características de las arquitecturas.

Los clusters heterogéneos son una arquitectura muy conveniente desde el punto de vista de su costo y disponibilidad. Un cluster heterogéneo está formado por un conjunto de computadoras de cualquier tipo, interconectadas por una red Ethernet [4]. El tipo de

[1] Investigador Asistente CICPBA

computadoras suelen ser PCs comunes de escritorio, workstations o SMPs. Así, se puede utilizar una red LAN como cluster para cómputo paralelo.

Esta arquitectura es de muy bajo costo si se la compara con una arquitectura paralela tradicional: las computadoras y el tipo de red utilizados son los más económicos y estándares del mercado y su relación Mflop/costo es muy conveniente [6]. Una consecuencia directa del costo es la alta disponibilidad de esta arquitectura para cómputo paralelo (dado por el gran número de redes LAN ya instaladas y en uso o por el bajo costo que tendría armar un cluster comprando todos sus componentes).
Además se puede ver que los componentes de una computadora de escritorio común son cada vez de mayor capacidad (tanto de procesamiento, almacenamiento, velocidad de acceso, etc). Así, se tiene la posibilidad de obtener muy buen rendimiento a muy bajo costo. Además, un cluster heterogéneo es una arquitectura de muy fácil mantenimiento y alta escalabilidad. Si se tiene que agregar un componente nuevo, tanto para reemplazar uno existente o para aumentar la capacidad del cluster, simplemente se agrega un componente que mínimamente cumpla con los requerimientos deseados, sin ser necesario que este nuevo componente sea igual (o similar) al hardware original ya que se está frente a una arquitectura heterogénea.

En este trabajo se desarrolla una solución paralela a la factorización LU de matrices. Esta factorización es una operación de álgebra lineal, dentro del conjunto de operaciones de cómputo numérico. Las operaciones de álgebra lineal son muy utilizadas en campos tales como medicina, física, astronomía, simulaciones, etc., todos campos con aplicaciones con gran requerimiento de cómputo. Por lo tanto, es conveniente disponer de versiones optimizadas de estas operaciones. Existe ya mucho trabajo y esfuerzo puesto en esto y también es importante considerar que mientras las arquitecturas sigan avanzando y cambiando sus características, las soluciones también deberían cambiar, adaptándose a estos cambios y así obtener buen rendimiento en estas arquitecturas con nuevas características.

Este trabajo es la continuación de [16] y [15]. En el primero se desarrolla una solución a la factorización LU paralela para un tipo de cluster heterogéneo específico. En [15] se extiende el trabajo anterior desarrollando una solución paralalela a la factorización orientada a clusters heterogéneos en general. Esto es, una solución que obtenga buen rendimiento en clusters heterogéneos, sin importar la heterogeneidad del mismo. En dicho trabajo se pone especial atención al método de distribución de carga, y en este trabajo se sigue en esta dirección: se proponen diferentes modificaciones al método de distribución de carga, con los cuales se experimentó y se analizaron los resultados obtenidos. Estos resultados mostraron comportamientos no esperados en los tiempos de comunicación los cuales penalizaron el tiempo total de resolución.

Durante este trabajo se explicará brevemente el método de la factorizacion LU y se mencionarán las principales características de la solución propuesta (lo cual se explica con mayor detalle en [16] y [15]). Luego se explicarán las modificaciones propuestas en el método de distribución de carga, se explicarán las principales características de las pruebas realizadas, se mostrarán y analizarán los resultados obtenidos y se enumerarán las conclusiones a las que se arribaron a partir de dichos resultados.

2. Factorización LU

La factorización LU pertenece a la librería LAPACK (Linear Algebra PACKage) [1][5] y se utiliza para resolver sistemas de ecuaciones lineales de la forma:

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1}$$

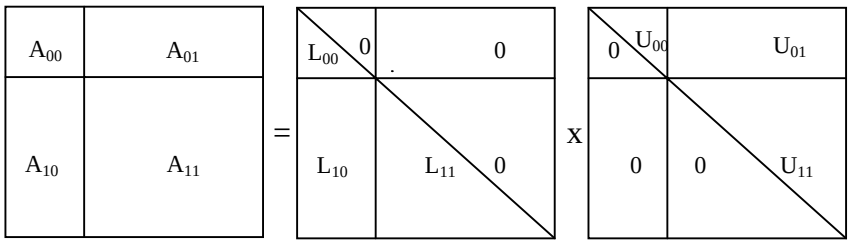
Donde $\boldsymbol{A}$ es la matriz formada por los coeficientes de las ecuaciones, y se utilizan los valores de $\boldsymbol{b}$ para hallar los valores de las incógnitas representadas por $\boldsymbol{x}$ en la ecuación 1. Dada la simpleza de resolver un sistema de ecuaciones triangular, el objetivo de la factorización LU es encontrar un sistema de ecuaciones triangular equivalente al original, para luego resolver de forma más simple el sistema de ecuaciones original. Así, realizar la factorización LU sobre $\boldsymbol{A}$ implica hallar las matrices $\boldsymbol{L}$ y $\boldsymbol{U}$ tales que:

$$\boldsymbol{A} = \boldsymbol{L}\ \boldsymbol{x}\ \boldsymbol{U} \tag{2}$$

Siendo $\boldsymbol{L}$ una matriz triangular inferior y $\boldsymbol{U}$ una matriz triangular superior. Así, se pueden utilizar $\boldsymbol{L}$ y $\boldsymbol{U}$ para obtener los valores de las incógnitas $\boldsymbol{x}$ de la ecuación 1.

Para hallar los valores de $\boldsymbol{L}$ y de $\boldsymbol{U}$ se aplican sucesivos pasos de eliminación Gaussiana a los elementos de $\boldsymbol{A}$. Cuando la matriz $\boldsymbol{A}$ es de n x n elementos, se realizan $O(n^3)$ operaciones sobre los $O(n^2)$ datos de $\boldsymbol{A}$ para hallar $\boldsymbol{L}$ y $\boldsymbol{U}$. De esta forma, esta operación es una de las que requieren el máximo de procesamiento ($O(n^3)$ operaciones sobre $O(n^2)$ datos) entre las operaciones de su tipo. Existen muchas operaciones con estos requerimientos de cómputo y es usual definir estas operaciones en términos de bloques. Esto implica dividir la matriz en bloques y realizar la operación entre los bloques y no entre los datos individuales de la matriz. Esto permite hacer un mejor uso de la jerarquía de memoria instalados en las computadoras [2][8][5].

Realizar la factorización LU por bloques implica dividir la matriz $\boldsymbol{A}$ como se muestra en la figura 1:



$$A_{00} = L_{00}U_{00} \tag{3}$$
$$A_{01} = L_{00}U_{01} \tag{4}$$
$$A_{10} = L_{10}U_{00} \tag{5}$$
$$A_{11} = L_{10}U_{01} + L_{11}U_{11} \tag{6}$$

Figura 1: división por bloqes de la matriz.

De esta forma, factorizar la matriz $\boldsymbol{A}$ implica hallar los valores de $\boldsymbol{L}_{00}$, $\boldsymbol{L}_{10}$, $\boldsymbol{L}_{11}$, $\boldsymbol{U}_{00}$, $\boldsymbol{U}_{01}$ y $\boldsymbol{U}_{11}$, para ello se utilizan las ecuaciones (3) a (6) las cuales se obtienen por la definición de multiplicación de matrices (pues se está resolviendo $\boldsymbol{A} = \boldsymbol{L}$ x $\boldsymbol{U}$). Para hallar los valores de

los bloques mencionados, primero se hallan $\boldsymbol{L}_{00}$ y $\boldsymbol{U}_{00}$ aplicando el método de factorización LU directamente a los datos de $\boldsymbol{A}_{00}$ (pasos de eliminación Gaussiana), resolviéndose la ecuación (3). Luego, utilizando la ecuación (4) se hallan los valores de $\boldsymbol{U}_{01}$ resolviendo un sistema de ecuaciones triangular (usando $\boldsymbol{A}_{01}$ y $\boldsymbol{L}_{00}$, esta última es triangular inferior). De la misma forma, se hallan los valores de $\boldsymbol{L}_{10}$ resolviendo la ecuación (5), siendo $\boldsymbol{U}_{00}$ una matriz triangular superior. De esta forma, sólo resta encontrar los valores de $\boldsymbol{L}_{11}$ y $\boldsymbol{U}_{11}$. Para esto se utiliza la ecuación (6), la cual se la reescribe:

$$\boldsymbol{L}_{11}\boldsymbol{U}_{11} = \boldsymbol{A}_{11} - \boldsymbol{L}_{10}\boldsymbol{U}_{01} \quad (7)$$

y de esta forma, se puede ver que el problema se reduce a aplicar el mismo método al resultado de la ecuación (7). Esto es, volver a dividir la parte restante de la matriz en bloques, y aplicar los mismos pasos. Este procedimiento se aplica tantas veces como bloques haya en la matriz obteniéndose así la factorización completa de la misma.

El método utilizado para resolver esta factorización es el presentado en [16], para el cual se han propuesto distintos métodos de distribución de los bloques con el fin de obtener balance de carga en ambientes heterogéneos.

En las siguientes subsecciones se describirán las principales características de la arquitectura utilizada las cuales se tienen en cuenta en el diseño del algoritmo y en el diseño de los distintos métodos de distribución de carga.

3. Clusters Heterogéneos

Como ya se ha mencionado, los clusters heterogéneos componen una arquitectura muy conveniente para cómputo paralelo desde el punto de vista del costo, disponibilidad, escalabilidad, rendimiento, etc.

Pero esta arquitectura tiene ciertas características que no son apropiadas para cómputo paralelo ya que pueden penalizar fuertemente el rendimiento total de las aplicaciones. Dentro de estas características, las principales son:

- Bajo acoplamiento: los componentes que integran un cluster (principalmente sus nodos) no tienen ningún tipo de relación entre sí, no comparten nada más que la red que los interconecta. Esto hace que todas las comunicaciones y toda forma de sincronización entre los procesos deba hacerse utilizando esta red.
- Red de interconexión Ethernet: el objetivo tradicional de una red Ethernet es compartir algún tipo de recurso (datos, bases de datos, impresoras), y no cómputo paralelo. Una red Ethernet tiene muy bajo rendimiento dado principalmente por su alto tiempo de startup y su bajo ancho de banda. Y como se mencionó en el primer punto, un cluster es una arquitectura débilmente acoplada, por lo que todas las comunicaciones y sincronizaciones se hace a través de la red, por lo que, si la red tiene bajo rendimiento, esto puede llegar a ser un cuello de botella en las aplicaciones paralelas, penalizando así el rendimiento.
- Heterogeneidad de cómputo: como ya se ha mencionado, el conjunto de nodos puede ser de cualquier tipo, por lo tanto las capacidades de cómputo varían entre los

distintos nodos. Esto implica que se debe poner especial atención a cómo se distribuyen los datos para lograr que todos los nodos tengan tiempos similares de cómputo (lo que no significa igual cantidad de datos) y evitar que todo el tiempo dependa del nodo que más tarde en resolver su parte asignada del problema.

Estas desventajas surgen principalmente del objetivo con el que se han creado los componentes: no es cómputo paralelo [14]. Como ejemplo se puede ver que las redes de interconexión de cualquier máquina paralela tradicional (creada para cómputo paralelo) es una red con altas prestaciones. Además también es fácil ver que todas las unidades de procesamiento de estas máquinas tienen las mismas características (homogéneas).

Así, es conveniente tener en cuenta todas las características de la arquitectura desde el momento mismo del diseño de las aplicaciones, de esta forma es posible evitar las penalizaciones que éstas puedan llegar a producir como así tambíen potenciar el uso de las características que puedan llegar a ayudar a obtener mejor rendimiento. En este trabajo se trabaja de esta forma y se desarrolló una solución a la factorización teniendo en cuenta todas estas características.

4. Métodos propuestos, experimentación y resultados

La solución utilizada en este trabajo sigue el algoritmo descrito en [16] el cual tiene las siguientes características:

- Resolución de la factorización por bloques. El tamaño de bloque se ha fijado en 64 filas cada uno, en se [16] comprobó que este es el tamaño de bloque óptimo para esta factorización en este ambiente de trabajo (clusters heterogéneos).
- Programa paralelo donde todos los nodos ejecutan el mismo algoritmo (SPMD).
- Todas las comunicaciones entre los nodos se hacen a través de mensajes broadcasts. En este caso se utiliza una rutina de comunicación que aprovecha el bradcast físico de la red y que además permite solapar cómputo con comunicación.
- Procesamiento secuencial optimizado en cada uno de los nodos. Se utilizan métodos optimizados para resolver la factorización LU, resolución de sistemas de ecuaciones triangulares y multiplicación de matrices.
- Balance de carga definido por la distribución de los datos (distribución de los bloques entre los distintos nodos). Sobre la distribución de bloques es donde más se trabajó con el objetivo de obtener balance de carga.
- El algoritmo es tal que se solapa la factorización de todos los bloques (excepto el primero) con la actualización del resto de la matriz. Esto permite tener el bloque factorizado tal vez antes de necesitarlo [16].

4.1 Métodos de distribución de carga

Desde [15] se menciona la necesidad de dedicar especial atención al balance de carga. En esta aplicación, el balance de carga se determina por la forma de distribuir los bloques entre los nodos. Obtener balance de carga implica que todos los nodos tarden tiempos similares en resolver el problema. En un ambiente homogéneo, el balance de carga se resuelve directamente distribuyendo la misma cantidad de trabajo entre los nodos. Al tener la misma capacidad y la misma cantidad de datos, el tiempo de cómputo será similar. Pero en un

ambiente heterogéneo esto se hace más difícil ya que los nodos tienen distintas capacidades de cómputo.

En un ambiente heterogéneo, se logra balance de carga si se distribuye el trabajo en función a la capacidad de cada nodo. Así, a los nodos que más capacidad tienen se les asignará más trabajo que a los de menor capacidad. Esto hace que todos los nodos tarden tiempos similares en resolver los bloques que le han sido asignados. Obtener balance de carga implica utilizar las capacidades de todos los nodos de la mejor forma posible. Esto es, que todos los nodos trabajen durante toda la factorización (o al menos la mayor parte) evitando sobrecarga en algunos nodos y que otros queden ociosos y así, el tiempo total no dependa del tiempo del nodo que más tarda (no penaliza el tiempo total).

En [15] se enumeran las características de distintos métodos de distribución de bloques los cuales tienen en cuenta las distintas capacidades de cómputo de cada nodo. Se proponen 4 métodos de los cuales el último es el que más ventajas mostró en este ambiente. Las principales características del método son: división de todos los bloques en grupos [5], distribución de los bloques de cada grupo entre todos los nodos y esta distribución se hace en función a la capacidad de cómputo de cada nodo. Además se realizan ciertos cálculos sobre las potencias de cada nodo con el objetivo de obtener potencias más chicas pero relativas a las originales (que sigan representando las diferencias de capacidad de cómputo de los nodos) y de esta forma tener grupos con menos bloques cada uno. Dentro de cada uno de los grupos la distribución de los bloques se hace de forma cíclica. En este trabajo se continúa utilizando este método con todas estas características, pero además se muestran distintas modificaciones que tienen como objetivo mejorar aun más el balance de carga.

Se han propuesto distintas modificaciones. En las pruebas realizadas en [15] se utilizan clusters compuestos por 2 grupos de nodos con distintas capacidades. En el mismo trabajo se logra balancear los nodos de un mismo tipo, pero se puede notar una pequeña diferencia de tiempos entre los nodos de los distintos tipos. Para eliminar (o al menos reducir) este desbalance se proponen las siguientes modificaciones.

4.1.1. Distribución por número de grupo

Asumiendo que pueda llegar a existir una relación entre cantidad de trabajo y ubicación del bloque en la matriz, la primer modificación se basa en la idea de evitar que los nodos siempre reciban los mismos bloques de cada uno de los grupos.
Los nodos del cluster (denominados $p_0..p_{n-1}$ considerando un cluster formado por n nodos) están ordenados por capacidad de procesamiento (de mayor capacidad a menor capacidad), y la distribución de los bloques de todos los grupos se resuelve de la misma forma. Esta primera modificación cambia esto último, se intenta que los nodos obtengan distintos bloques de cada uno de los grupos. Se asume una numeración de los grupos comenzando desde el primer grupo (1, impar) hasta el último grupo (par o impar, dependiendo de la cantidad de grupos total).

Principalmente, la modificación propuesta es distribuir los bloques de un grupo desde el nodo p_0 hasta el nodo p_{n-1} y distribuir los bloques del próximo grupo desde el nodo p_{n-1} hasta el p_0. Entonces se alterna la forma de distribución entre los grupos y esto se repite

para cada uno de los grupos en que se dividen los bloques. En la figura 2 se muestra la distribución para una matriz donde se distribuyen los primeros 2 grupos. Se asume un cluster formado por 3 nodos ($p_0..p_2$) con potencias 3, 2 y 1 repectivamente. De esta forma, cada grupo tiene 6 bloques.

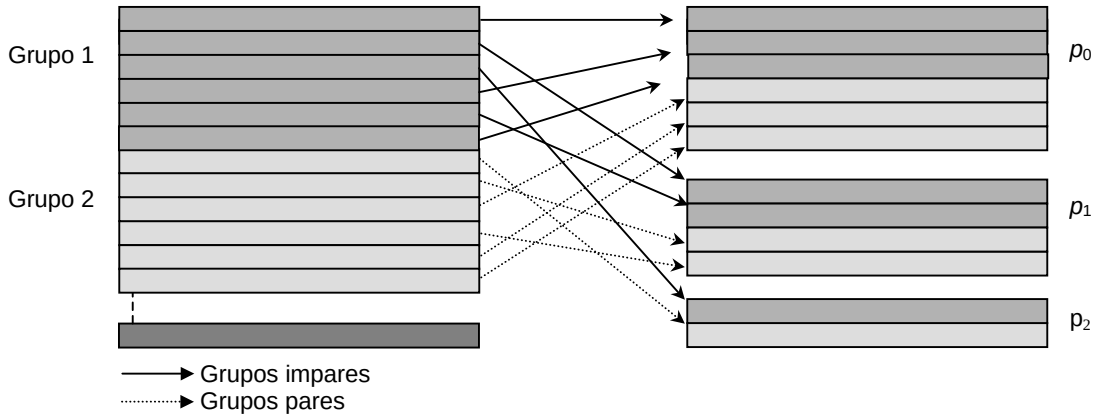


Figura 2: Distribución de bloques alternando por grupo.

Esta modificación se implementó y se realizaron distintas experimentaciones. Se pudo comprobar que no se obtiene ninguna mejora en el balance de carga. Por lo tanto, esta modificación no soluciona el problema. Por lo tanto, se puede decir que es suficiente con la distribución cíclica de los bloques dentro de cada grupo.

4.1.2. Redistribución de los últimos grupos

La segunda modificación sobre el algoritmo de distribución de bloques tiene como objetivo distribuir los últimos bloques de forma que se puede elegir el trabajo total que se le asigna a cada nodo. Con esta modificación se realizaron pruebas donde se distribuyen los bloques y se logra balancear el tiempo de cómputo entre todos los nodos.

Se realizaron distintas pruebas para distintos tamaños de matrices. Para todos los tamaños de matrices se pudo observar el mismo comportamiento, el cual se explica mostrando los resultados para matrices de 25088 x 25088 elementos (se trabaja con matrices de orden divisible por el tamaño de bloque). El cluster utilizado tiene las siguientes características:

Cantidad	CPU	Frec. Reloj	Memoria	Mflops LU por bloques
4	Pentium 4	2400 MHz	1GB	6188
4	Pentium III	700 MHz	256 MB	1561

Tabla 1: configuración del cluster utilizado.

Para simplificar las siguientes tablas y gráficos, se usará w_i para nombrar a la computadora *Watson*$_i$ (Pentium 4) y lp_j a la computadora *Lidipar*$_j$ (Pentium 3).

Utilizando la distribución de [15] más la primer modificación (distribución alternando la asignación de los bloques de los grupos pares y los impares), se obtuvo la siguiente distribución y los siguientes tiempos:

Nombre	CPU	Filas asignadas	Trabajo ideal	Trabajo real	Carga real (cuando 100% es lo que debería recibir)
$w_{4..7}$	Pentium 4	4992	19.96%	19.89%	99.64%
$lp_{65..68}$	Pentium 3	1280	5.03%	5.10%	101.39%

Tabla 2: Distribución obtenida con el método de distribución de carga sin redistribución de bloques.

Nodo	Total Filas	Tiempo total	Tiempo de comunicación	Tiempo de cómputo
w4	4992	556,84	60,55	496,30
w5	4992	556,64	27,71	528,93
w6	4992	556,51	25,47	531,04
w7	4992	556,52	28,46	528,06
lp65	1280	558,09	80,18	477,90
lp66	1280	558,09	80,58	477,52
lp67	1280	558,09	78,18	479,92
lp68	1280	558,09	77,76	480,33

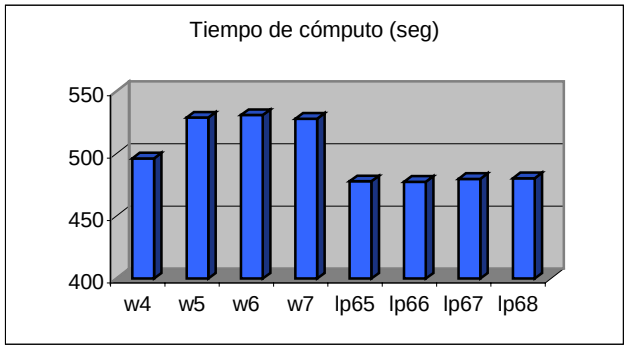


Figura 2: Tiempos obtenidos sin redistribución de bloques para matrices de 25088 x 25088 flotantes.

Se puede observar que aunque el balance de carga es aproximado al ideal (última columna de la Tabla 2), el tiempo de cómputo de los primeros 4 nodos es significativamente mayor que el de los nodos Pentium 3. Con el mismo ejemplo, utilizando la modificación 2 sobre el algoritmo de distribución, se distribuyen los últimos 4 bloques a los nodos Pentium 3 con el objetivo de balancear los tiempos de cómputo y se obtuvo:

Nombre	CPU	Filas asignadas	Trabajo ideal	Trabajo real	Carga real (cuando 100% es lo que debería recibir)
$w_{4..7}$	Pentium 4	4928	19.96%	19.64%	98.39%
$lp_{65..68}$	Pentium 3	1344	5.03%	5.35%	106.36%

Tabla 3: Distribución obtenida con el método de distribución de carga redistribuyendo los últimos 4 bloques.

Con esta distribución se obtuvieron los siguientes tiempos:

Nodo	Total filas	Tiempo Total	Tiempo de Comunicación	Tiempo de Cómputo
w4	4928	574,56	90,24	484,32
w5	4928	574,51	58,56	515,95
w6	4928	574,23	58,11	516,12
w7	4928	574,36	56,94	517,43
lp65	1344	575,32	51,25	524,07
lp66	1344	575,32	51,25	524,07
lp67	1344	575,33	50,97	524,36
lp68	1344	575,21	50,68	524,53

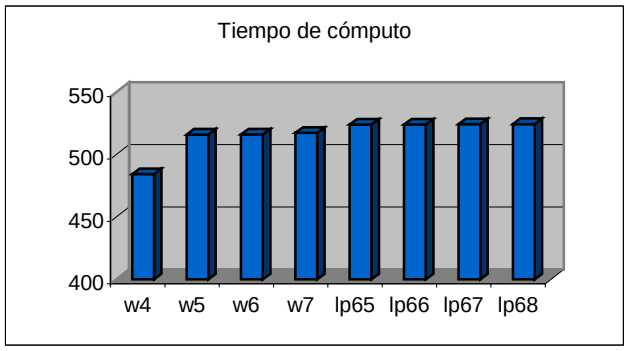


Figura 3: Tiempos obtenidos redistribuyendo los últimos 4 bloques para matrices de 25088 x 25088 flotantes.

Con esta modificación se observó que se logran balancear los tiempos de cómputo entre los distintos tipos de nodos. Con esto se espera que el tiempo total de cómputo disminuya, pues

si todos los nodos tardan tiempos similares (balanceados), ningún nodo tardaría mucho más que otro o estaría ocioso por mucho tiempo esperando que el más lento termine. Pero esto no es lo que sucede en la experimentación. Particularmente se puede observar que aunque los tiempos de cómputo se balencean (factorización LU y actualización de los bloques), el tiempo total de resolución no se reduce cuando se logra balance de carga, por el contrario crece, y esto se repite para otros tamaños de matrices.

En la mayoría de las pruebas (distintos tamaños de matrices) balancear la carga implicó asignar más bloques a los nodos de menor capacidad (Pentium 3). La cantidad reasignada es la mínima (los últimos 4 bloques o en algunos casos se llegó a reasignar los últimos 8 bloques). Esta reasignación no sólo implica dar más trabajo a los nodos de menor capacidad, sino también asignar menos a los de mayor capacidad (se sacan bloques de los nodos más potentes para asignarse a los nodos de menor capacidad). Esto permitió ver un comportamiento interesante en los tiempos de cómputo como así también en los de comunicación:

- Tiempos de cómputo:

    a. El tiempo de cómputo de los nodos $w_{4..7}$ decrece muy poco (2.56% aprox).
    b. Aumenta mucho el tiempo de cómputo de los nodos $lp_{65..68}$ (8.42% aprox).

- Tiempos de comunicación

    a. Crece mucho en los nodos $w_{4..7}$ (51.4% aproximadamente)
    b. Disminuye en los nodos $lp_{65..68}$ (36% aproximadamente).

Esto hace que los tiempos de cómputo se balanceen pero que los tiempos totales de toda la factorización crezcan. Esto se puede observar en las figuras 2 y 3, comparando los tiempos de las tablas y los resultados mostrados en los gráficos.

Una posible explicación a esto es que en cada iteración se realiza un broadcast (comunicación el bloque factorizado) donde todos los nodos se sincronizan. El broadcast no se resuelve hasta que todos los nodos involucrados en la comunicación reciben el mensaje. Ahora, ¿Cómo influye esto en los tiempos totales haciendo que aunque se mejore el balance de carga los tiempos totales sean mayores? La respuesta está en que cuando los nodos más veloces ($w_{4..7}$) tienen más datos para computar, tardan más y los más lentos ($lp_{65..68}$) esperan los datos de los más rápidos, por lo tanto los nodos más lentos y con menos datos tienen mucho tiempo de comunicación esperando que los nodos más rápidos terminen de actualizar sus datos (muchos) y lleguen al broadcast (punto de sincronización). Entonces, lo que hace que los nodos más lentos tengan mucho tiempo de comunicación (tabla de la Figura 2) es la espera a que los nodos más rápidos terminen de actualizar sus bloques.
Cuando se les asigna más bloques a los nodos $lp_{65..68}$, esto se balancea, el tiempo de cómputo (de actualización principalmente) es similar entre los nodos, por lo tanto ahora son las más veloces las que esperan a que las más lentas actualicen sus datos (aumentando el tiempo de comunicación de las más veloces). Esto se muestra en la tabla de la Figura 3.

De esta forma, se puede suponer que la comunicación penaliza los tiempos totales. Los tiempos de sincronización están incluidos en los de comunicación, y la sincronización que existe en cada una de las iteraciones a lo largo de toda la factorización estaría influyendo en la penalización total. Se puede entonces suponer que es conveniente esperar a algún nodo a que llegue al punto de sincronización y no balancear en todos los nodos el tiempo de cómputo y también el de comunicación (ya que este balance puede implicar un aumento en la suma de estos dos tiempos).

Ante estas observaciones, se quiso determinar si en esto está influyendo la factorización solapada con la actualización de la parte activa de la matriz. Para esto, se realizaron pruebas similares con un algoritmo donde no se solapa la factorización con la actualización del resto de la matriz. Los resultados mostraron que:

a. El tiempo total es mucho mayor cuando no se solapa la factorización del bloque con la actualización. Se han realizado distintas pruebas y se pudo ver que el algoritmo que solapa la factorización del bloque con la actualización de la parte activa de la matriz logra mejoras del 26% al 36% en los tiempos totales (para mismo tamaño de matriz y misma distribución final de bloques entre los nodos). Por lo tanto, es conveniente solapar la factorización y actualización ya que la mejora producida es importante.
b. El comportamiento de los tiempos de cómputo y de comunicación son iguales a la versión que solapa la factorización con la actualización. Por lo que no influye la factorización por adelantado del bloque corriente en cada interación.

5. Conclusiones

Se ha implementado una solución a la factorización LU de matrices especialmente orientada a clusters heterogéneos. Aunque se reconoce que orientar una aplicación para ser utilizada en un tipo específico de computadora significa perder generalidad y hasta portabilidad, se reconoce que esto es conveniente (casi fundamental) si se quiere lograr buen rendimiento en la aplicación. Y este es uno de los objetivos del cómputo paralelo.
Las aplicaciones numéricas y en particular las operaciones de álgebra lineal son muy utilizadas y existen en la actualidad distintas versiones de soluciones a estos problemas. Estas distintas soluciones tienen como objetivo común lograr buen rendimiento, y la mayoría está orientada a un tipo de arquitectura particular o a un conjunto de arquitecturas que comparten características específicas (memoria compartida o distribuida, nodos fuertemente o debilmente acoplados, disposición de nodos en grilla, toro, lineal, etc) [7][8]. Es fácil ver como el avance en las arquitecturas paralelas y la búsqueda de buen rendimiento obligó esta evolución en las soluciones existentes de estas operaciones.

Este trabajo es la continuación de [16] y [15] en los cuales se desarrolla una solución a la factorización LU de matrices orientada específicamente a clusters heterogéneos y se proponen distintos métodos de distribución de carga. En este trabajo se han propuesto y evaluado modificaciones al último método de distribución de carga propuesto en [15] con el objetivo de mejorar el balance de carga y los tiempos totales de la operación. Pero las distintas pruebas realizadas con estas modificaciones mostraron que:

a. Se logra balancear los tiempos de cómputo: todos los nodos tienen tiempos similares de procesamiento. Esto es, se repartieron los bloques de forma tal que todos los nodos, aunque tengan distintas capacidades de cómputo, tengan tiempos similares en la factorización LU y en la actualización de la parte activa de la matriz.
b. Aunque se logra el objetivo propuesto, se pudo ver que esto no implica mejorar los tiempos totales, por el contrario, aumentan. Esto es así debido a los tiempos de comunicación, los cuales aumentan y penalizan el tiempo total. Así, es hasta conveniente tener un cierto desbalance de carga ya que evita la penalización producida por la comunicación y sincronización de los procesos. Se concluye que es conveniente esperar a algún nodo a que llegue al punto de sincronización y no balancear los tiempo de cómputo y comunicación ya que este balance puede implicar un aumento en la suma de estos dos tiempos (por lo tanto, en el tiempo total de la resolución de la factorización).
c. Para el mismo tamaño de matriz el tiempo total es menor utilizando el algoritmo que solapa la factorización del bloque con la actualización de la parte activa de la matriz.

Como resultado de este trabajo se obtuvieron distintos métodos de distribución de carga que distribuyen los bloques en función a la capacidad de procesamiento de cada uno de los nodos que mostraron que, en este contexto (factorización LU de matrices en clusters heterogéneos), es conveniente un cierto desbalance de carga entre los nodos para evitar que el tiempo de comunicación penalice el tiempo total.

A partir de esto, sería conveniente analizar cuál es el desbalance óptimo, o sea, cuál es el desbalance que logra minimizar los tiempos de comunicación y sincronización y por lo tanto, logre minimizar la penalización de éstos sobre el tiempo total.

6. Referencias

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

[2] Anderson E., Z. Bai, C.Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen. “LAPACK: A Portable Linear Algebra Library for High-Performance Computers”, Proceedings of Supercomputing ’90, pages 1-10, IEEE Press, 1900

[3] Anderson T., D. Culler, D. Patterson, and de 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 Perfomormance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 3-47, 1999.

[5] Barbosa J., J. Tavares and A. J. Padilha. “Linear Algebra Algorithms in a Heterogeneous Cluster of Personal Computers”. Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) pp: 147-159.

[6] Denham M., “Paralelización de la Factorización LU de Matrices para Clusters Heterogéneos”, Tesina de Grado, abril 2005.

[7] 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.
[8] 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.
[9] Flynn M. “Very High Speed Computing Systems” Proc. IEEE, Vol 54, 1966.
[10] Flynn M “Some Computer Organization and Their Effectiveness”, IEEE Trans. On Computers, 21 (9) 1972.
[11] Institute of Electrical and Electronics Engineers, Local Area Network – CSMA/CD Access Method and Phisical Layer Specifications ANSI/IEEE 802.3 – IEEE Computer Society, 1985.
[12] Lawson C. R. Hanson, D. Kincaid, F. Krogh, “Basic Linear Algebra Subprograms for Fortran Usage”, ACM Transaction on Mathematical Software 5, pp. 308-323, 1979.
[13] Kumar V., A. Grama, A. Gupta, G. Karypis, Introducción to Parallel Computing. Desing and Analisys of Algorithms, The Benjamin/Cummings Publishing Company, Inc 1994.
[14] Tinetti F. G. “Cómputo Paralelo en Redes Locales de Computadoras”, Tesis Doctoral.Universidad Autónoma de Barcelona, Facultad de Ciencias. Marzo 2004.
[15] Tinetti F. G., M. Denham, “Álgebra Lineal en Paralelo: Factorizaciones en Clusters Heterogéneos”, Proceedings X Congreso Argentino de Ciencias de la Computación (CACIC 2004), Universidad Nacional de La Matanza. Publicado en el CD del Congreso.
[16] Tinetti F. G., M. Denham, “Paralelización de la Factorización LU de Matrices en Clusters Heterogéneos”, Proceedings IX Congreso Argentino de Ciencias de la Computación (CACIC 2003), Facultad de Informática de la Universidad Nacional de La Plata, La Plata, Argentina, p 385-396. Oct. 2003.
[17] Tinetti F. G., E. Luque, “Parallel Matrix Multiplication on Heterogeneous Networks of Workstations”, Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, p 122 Oct. 2002.
[18] Tinetti F. G., A. Quijano, A. De Giusti, E. Luque, “Heterogeneous Networks of Workstations and the Parallel Matrix Multiplication”, Y. Cotronis and J. Dongarra (Eds.): EuroPVM/MPI 2001, LNCS 2131, pp 296-303, Springer-Verlag, 2001.
[19] Wilkinson B., Allen M., Parallel Programing: Techniques and Applications Using Networked Workstations, Prentice-Hall, Inc., 1999.