Capítulo 5: Comparación con ScaLAPACK

En este capítulo se presentan dos aspectos importantes en cuanto a la validez y utilización de los aportes de esta tesis: 1) Aplicación de los principios de paralelización en ambientes homogéneos dedicados a cómputo paralelo para dos casos específicos: multiplicación de matrices y factorización LU de matrices, y 2) Comparación de resultados obtenidos por experimentación en cuanto a rendimiento con respecto a la biblioteca ScaLAPACK. Esta biblioteca está dedicada específicamente a las plataformas de cómputo paralelo homogéneas con memoria distribuida, y es aceptada en general como la que implementa los mejores algoritmos paralelos existentes en cuanto a escalabilidad y optimización de rendimiento.

La utilización de la biblioteca ScaLAPACK hasta ahora se orienta *solamente* a hardware homogéneo y por lo tanto se tiene en cuenta *inicialmente* la red homogénea con la que se ha trabajado hasta ahora. Sin embargo, se ha tenido la posibilidad, para esta comparación a una segunda red homogénea que también se describe (al menos en cuanto a lo mínimo necesario) en este capítulo. En cierto modo, varias características de experimentación cambian en este capítulo por dos razones: 1) La experimentación es *clásica* y *sencilla* en cuanto a los experimentos. Por ejemplo, no hay análisis de tamaños de problemas fuera de los límites de la memoria de cada máquina dado que lo que se intenta es la comparación directa de los algoritmos; 2) Solamente se tienen en cuenta redes de computadoras homogéneas, dado que de otro modo la comparación con los algoritmos implementados en ScaLAPACK no sería posible o podría ser “sesgada”.

En este capítulo se agrega también el problema de factorización LU de matrices, a la cual se aplican los mismos conceptos de paralelización que a la multiplicación de matrices. El objetivo no ha sido en este caso el examen exhaustivo de algoritmos, implementaciones, etc. como en el caso de la multiplicación de matrices sino directamente la aplicación de los principios de paralelización a este problema específico. Por esta razón la descripción del problema mismo y de los algoritmos paralelos no será exhaustiva.

5.1 Características Generales de ScaLAPACK

Es muy interesante resaltar que el análisis de algoritmos y propuestas de algoritmos para resolver los problemas de álgebra lineal en paralelo sigue una estrecha relación con las computadoras paralelas *tradicionales* (tales como las pertenecientes a las clases de computadoras paralelas "A*d Hoc*" y "Comerciales" del Capítulo 1, Figura 1.2). Por lo tanto, todo el análisis realizado para el caso de la multiplicación de matrices es aplicable a las demás operaciones incluidas en ScaLAPACK y/o las utilizadas para resolverlas.

Desde el punto de vista de la implementación de ScaLAPACK, se pueden identificar varias "capas" de software [21], que se muestran de manera simplificada en la Figura 5.1 desde el punto de vista del software a utilizar en cada computadora.

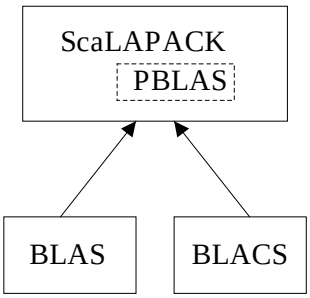


Figura 5.1: Una Visión en Capas de ScaLAPACK.

Como parte *integrante* de ScaLAPACK se hace mucho énfasis en las operaciones básicas (tal como en el contexto de LAPACK), que se denominan PBLAS (Parallel Basic Linear Algebra Subroutines). Todo lo relacionado con el cómputo local en términos de cálculo u operaciones de cálculo matricial, se mantiene directamente relacionado con BLAS (Basic Linear Algebra Subroutines) y de esta manera se aprovecha directamente el todo el esfuerzo y el desarrollo de varios años de investigación. Por otro lado, se debe tener en cuenta la necesidad de comunicación entre procesos de las computadoras paralelas en general y de las computadoras paralelas con memoria distribuida en particular, y por lo tanto se agrega BLACS (Basic Linear Algebra Communication Subroutines).

Es interesante el agregado de BLACS desde dos puntos de vista:

- Como se ha mencionado, se reconoce la necesidad de comunicación entre procesos en un ambiente de memoria distribuida. En este sentido, se tiende a descartar la posibilidad de memoria compartida (sea físicamente distribuida o no) y al modelo de programación por pasaje de mensajes de las arquitecturas paralelas distribuidas. Esta decisión está, a su vez, directamente orientada a la obtención de rendimiento optimizado en cuanto a cómputo y comunicaciones.
- No se utiliza directamente ninguna de las bibliotecas de pasaje de mensajes disponibles: ni las de uso libre (PVM, e implementaciones de MPI) ni las provistas por las empresas comerciales de computadoras paralelas y que son específicamente optimizadas para sus redes de interconexión (IBM, por ejemplo para sus máquinas SP). De hecho, existen implementaciones de BLACS de uso libre que hacen uso directo de PVM o MPICH (una de las implementaciones de MPI).

Los algoritmos paralelos implementados en ScaLAPACK están fuertemente influenciados por los que han sido propuestos en el área de la multicomputadoras con organización o interconexión bidimensional de procesadores o con redes de interconexión más complejas aún, como las de topología de hipercubo [2] [3] [21]. Esta decisión se orienta a utilizar y aprovechar directamente todo lo que se ha investigado y los resultados obtenidos hasta el momento. Sin embargo, este tipo de algoritmos tiende a tener una alta penalización de rendimiento en el contexto de los clusters, dado que los clusters basados en redes de interconexión Ethernet y con sistemas operativos de propósito general (Linux, AIX, IRIX, Solaris, etc.) la relación cómputo-comunicación es varios órdenes de magnitud peor (para el rendimiento de las aplicaciones paralelas) que en las multicomputadoras tradicionales. La razón más importante para que esto sea verdadero en la gran mayoría de los casos es que ni las redes Ethernet ni los sistemas operativos tradicionales están pensados para cómputo paralelo a diferencia de las multicomputadoras, construidas con el propósito de hacer cómputo paralelo.

Por lo expuesto anteriormente se puede comprender rápidamente por qué, en general, los clusters sobre los cuales se utilizan bibliotecas como ScaLAPACK y se resuelven tareas con cómputo paralelo en general normalmente están “restringidos” a cableado completamente “switched”. Con este tipo de cableado es posible llevar a cabo múltiples comunicaciones punto-a-punto, y esto hace que el hardware sea similar al de las multicomputadoras. Sin embargo, se debe notar que el costo de las redes completamente “switched” crece mucho más que linealmente en función de la cantidad de computadoras interconectadas.

5.2 Paralelización de la 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 [27]. Inicialmente se describirá el algoritmo secuencial de factorización LU por bloques y posteriormente se presentará el algoritmo paralelo propuesto para clusters de computadoras, siguiendo los mismos principios que se utilizaron para la multiplicación de matrices.

5.2.1 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

$$A = L \times U \tag{5.1}$$

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

Se puede demostrar que si A es no singular, L y U existen [59]. 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 [84]. 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 misma relación básica que se tiene para las operaciones BLAS de nivel 3, 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). 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 Figura 5.2.

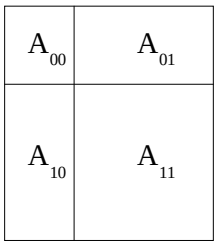


Figura 5.2: División de una Matriz en Bloques.

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

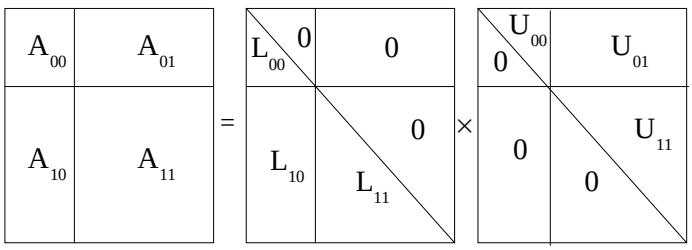


Figura 5.3: Factorización LU en Términos de Bloques.

donde se deben cumplir las siguientes ecuaciones teniendo en cuenta las submatrices de A, A_{00}, A_{01}, A_{10} y A_{11}, las submatrices de L distintas de cero, L_{00}, L_{10} y L_{11}, y las submatrices de U también distintas de cero, U_{00}, U_{01} y U_{11}

$$A_{00} = L_{00} U_{00} \quad (5.2)$$
$$A_{01} = L_{00} U_{01} \quad (5.3)$$
$$A_{10} = L_{10} U_{00} \quad (5.4)$$
$$A_{11} = L_{10} U_{01} + L_{11} U_{11} \quad (5.5)$$

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 las matrices triangulares L_{00} y U_{00} tal que se verifica de manera directa (por la aplicación del método de factorización LU) la Ecuación (5.2). Utilizando la Ecuación (5.3) 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 Ecuación (5.4) 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 Ecuación (5.5) 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 y realizar 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 la actualización de la matriz $A_{11} - L_{10} U_{01}$, 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 (_trsm y _gemm respectivamente en términos de la interfase C de BLAS) y una de LAPACK (_getrf, en términos de la interfase C de LAPACK).

5.2.2 Algoritmo Paralelo de Factorización LU *para* Multicomputadoras

El método que se describió antes para la factorización LU se implementa casi de manera directa en las computadoras secuenciales, pero tiene inconvenientes de rendimiento en las computadoras paralelas de memoria distribuida. La distribución de los datos tiene una importancia decisiva en cuanto a rendimiento desde dos puntos de vista: a) balance de

carga, y b) escalabilidad.

Es relativamente sencillo identificar que a medida que se avanza en las iteraciones del algoritmo la esquina derecha superior de la matriz queda calculada y esta parte calculada es cada vez (iteración) mayor. Esto implica directamente que esos datos no necesitan ser actualizados y además ni siquiera intervienen en los cálculos subsiguientes. Teniendo en cuenta que los datos están distribuidos en diferentes computadoras, todas las computadoras que tengan asignados estos datos no los volverán a utilizar ni para actualizarlos ni para calcular otros en función de ellos. Para que no se produzca desbalance de carga a medida que avanzan las iteraciones, las bibliotecas como ScaLAPACK y PLAPACK utilizan la distribución denominada descomposición cíclica de bloques bidimensional (*two-dimensional block cyclic decomposition*), tal como se la menciona en [26] [21] [45] y que se detalló sobre el final del Capítulo 3. Las principales ideas en las que se basa esta distribución de datos son:

- Tener muchos más bloques de datos que procesadores, y de esa forma la distribución genera que un mismo procesador tiene bloques que son actualizados en casi todas las iteraciones. Por lo tanto todos los procesadores tienden a tener bloques que son actualizados en todas las iteraciones y todos procesan en todas la iteraciones.
- Se asume que las distribuciones bidimensionales son suficientemente escalables al menos para las aplicaciones de álgebra lineal.
- El trabajo experimental normalmente se lleva acabo en multicomputadoras o en clusters homogéneos con redes de interconexión completamente “switched” y de alto rendimiento (en cierta forma *ad hoc* para procesamiento paralelo).

Por lo tanto, el algoritmo paralelo utilizado para la factorización LU está directamente basado en la descripción dada para el cálculo de LU en función de bloques, pero con distribución de los datos de la matriz de manera bidimensional y cíclica de bloques.

5.2.3 Algoritmo Paralelo de Factorización LU *para* Clusters

Los principios para la paralelización de la factorización LU de matrices son básicamente los mismos que se utilizaron para la multiplicación de matrices:

- Modelo de programación por pasaje de mensajes: procesos que computan localmente y se comunican por medio de mensajes con los demás.
- Modelo de ejecución SPMD (Single Program, Multiple Data): un mismo programa que todas las computadoras ejecutan utilizando diferentes datos.
- Distribución de datos unidimensional: la matriz completa es dividida por filas o por columnas (no de ambas maneras, lo que conduciría a distribuciones bidimensionales).
- Comunicación entre procesos solamente por mensajes broadcast: la mayor parte de las transferencias entre procesos (o todas) se hace vía mensajes broadcast para intentar aprovechar al máximo las características de las redes Ethernet en cuanto a tasa de transferencia y escalabilidad.

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 [82].

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 ws_0, ws_1, ws_2, y ws_3, y se divide la matriz en cuatro bloques de filas tal como muestra la Figura 5.4, cuando se terminan de procesar las filas que se asignan al procesador ws_0, de hecho el procesador ws_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 ws_0. Algo similar ocurre cuando se llegan a procesar posteriormente las filas asignadas a ws_1, a partir de lo cual todo el procesamiento siguiente se hace solamente en los procesadores ws_2 y ws_3, y esto implica claramente que el procesamiento no está balanceado entre los cuatro procesadores.

Figura 5.4: 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 Figura 5.5, donde el bloque *i* se asigna al procesador *i* mod *P* donde *P* es la cantidad total de computadoras.

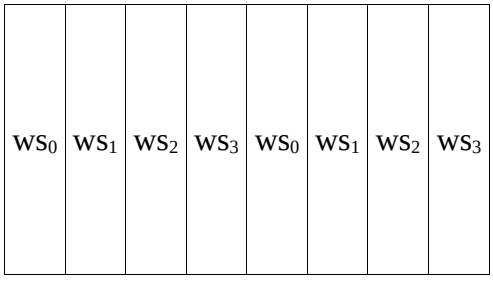


Figura 5.5: Distribución Cíclica por Bloques de Columnas.

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 computadoras el balance de carga implementado de esta forma no presenta inconvenientes. Por un lado, la distribución de datos es sencilla, y con pocos parámetros a definir (solamente el tamaño del bloque) y por el otro se logra 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. Una primera propuesta, con períodos de cómputo y comunicación que se ejecutan de manera secuencial en cada una de las computadoras del cluster se muestra en pseudocódigo para la máquina ws_i ($0 \leq i \leq P$-1) en la Figura 5.6 [127]. 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 *completo* del procesamiento paralelo para la factorización LU.

ws_i

```
for (j = 0; j < nblocks; j++)
{
  if (i == (j mod P))       /* Current block is local */
  {
    Factorize block j
    broadcast_send (factorized block j and pivots)
  }
  else
    broadcast_receive (factorized block j and pivots)
  Apply pivots                  /*                          */
  Update L                      /*  Update local blocks     */
  Update trailing matrix        /*                          */
}
```

Figura 5.6: Pseudocódigo de un Algoritmo Paralelo de Factorización LU.

Se debe notar en el pseudocódigo de la Figura 5.6 que se incorpora explícitamente todo lo que corresponde al manejo 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 que corresponden a L_{10} (L en la Figura 5.6) y a A_{11} - L_{10} U_{01} (“trailing matrix” en la Figura 5.6, sobre los que continúa la aplicación del método por bloques).

Como en el caso de la multiplicación de matrices, el pseudocódigo de la Figura 5.6 impone

una secuencia bien definida y estricta de pasos de cómputo local y comunicaciones con los demás procesos/procesadores. También como en el caso de la multiplicación de matrices se puede organizar el cómputo de manera tal que se puedan llevar a cabo comunicaciones solapadas con cómputo local (siempre que sea posible) y de esta manera intentar la reducción de penalización de rendimiento que imponen las comunicaciones en los clusters. Esta propuesta es la que se muestra en la Figura 5.7 en forma de pseudocódigo.

```
                                                        ws_i
if (i == 0)
  Factorize and broadcast_send block 0
for (j = 0; j < nblocks; j++)
{
  if (i == (j mod P))  /* Current block is local */
    Update local blocks
  else if (i == ((j+1) mod P))  /* Next block is local */
  {
    broadcast_recv_b (factorized block j)
    Update and Factorize block j+1
    broadcast_send_b (factorized block j+1)
    Update local blocks (block j+1 already updated)
  }
  else /* ws_i does not hold block j nor block j+1 */
  {
    broadcast_recv_b (factorized block j)
    Update local blocks (block j+1 already updated)
  }
}
```

Figura 5.7: Factorización LU en Paralelo con Cómputo y Comunicación Solapados.

El algoritmo de la Figura 5.7 agrega la idea de “bloque siguiente” al algoritmo clásico de la Figura 5.6. Dado que la factorización LU de un bloque y su correspondiente operación de comunicación send_broadcast_b imponen la espera de todos los demás procesadores, el *siguiente* bloque es factorizado y enviado “en background” para que en la siguiente iteración ya esté disponible para la actualización del resto de la matriz.

5.3 Experimentación

Inicialmente, se deben definir los parámetros con los cuales se llevaron a cabo los experimentos, específicamente en el caso de la biblioteca ScaLAPACK deben definirse:

- Biblioteca de soporte de comunicaciones (PVM, implementación de MPI, etc.).
- Grilla de procesadores (arreglo bidimensional de procesadores).
- Tamaño de Bloques.

Dado que ScaLAPACK se utiliza en ambientes homogéneos ya no es necesario hacer referencia a la potencia de cálculo relativa de los procesadores para el balance de carga.

Además, se puede utilizar directamente al *speedup* como índice de rendimiento para comparar los algoritmos paralelos implementados en ScaLAPACK con otros, tales como los que se proponen en este capítulo y en el capítulo 3.

Con el fin de evitar confusiones, y dado que la comparación de este capítulo tiende a cuantificar las diferencias entre los algoritmos implementados en ScaLAPACK con los que se proponen en este capítulo y el capítulo 3, se utilizarán los algoritmos propuestos que tienden a aprovechar al máximo la capacidad de cómputo solapado con comunicaciones entre computadoras. De esta manera, ya queda un único algoritmo propuesto para resolver cada una de las tareas a llevar a cabo en paralelo: multiplicación y factorización LU de matrices.

5.3.1 Conjunto de Experimentos

Hardware. La primera red homogénea que se utilizará es la única en estas condiciones de las que se han utilizado: la red local LIDI. Sin embargo, de manera temporal se tuvo acceso a otra red local que se pudo utilizar y además interconectar a la red local LIDI. Esta *segunda* red está compuesta por 8 PCs con procesador Duron con mayor capacidad de cómputo y almacenamiento que las PCs del LIDI, con placas de interconexión Ethernet de 10/100 Mb/s. Además, dado que se pudieron utilizar otros dos switches Ethernet (además del que ya tenía la red local LIDI) de 100 Mb/s de 8 bocas, se pueden hacer experimentos con dos "máquinas paralelas" más: una con las ocho PCs con procesadores Duron completamente interconectados por un switch y otra con la combinación de las dos redes con tres switches. Finalmente, en la última de estas secciones de experimentación, se mostrarán los resultados obtenidos en el último de los clusters homogéneos en el cual se llevaron a cabo más experimentos específicamente orientados a comparar el rendimiento de los algoritmos propuestos en esta tesis y los que implementa ScaLAPACK.

En el caso de la red con las ocho PCs con procesadores Duron y un único switch (que se denominará LIDI-D de aquí en más), se tiene un cluster homogéneo *clásico*, como la misma red LIDI, pero con una diferencia importante con la red LIDI: la relación cómputo comunicación. Dado que las computadoras de la red LIDI-D tienen mayor capacidad de cómputo y almacenamiento y la *misma* red de interconexión que las de la red LIDI se puede cuantificar la diferencia de la relación cómputo-comunicación entre ambas directamente utilizando las diferencias de capacidad de cómputo entre las PCs. En la Tabla 5.1 se muestran las características de las PCs del cluster LIDI-D de manera resumida, junto con su capacidad de cómputo expresada en términos de Mflop/s.

Procesadores	Frec. Reloj	Memoria	Mflop/s
AMD Duron	850 MHz	256 MB	1200

Tabla 5.1: Características de las Computadoras del Cluster LIDI-D.

Dado que las PCs de LIDI-D tienen rendimiento de aproximadamente 1200 Mflop/s, poco más del doble de las computadoras de la red LIDI, se puede afirmar que la relación

cómputo comunicación de la red LIDI-D es aproximadamente dos veces peor para cómputo paralelo que la de la red LIDI. Expresado de otra manera, en un mismo intervalo de tiempo:

- Las computadoras de la red LIDI-D pueden hacer el doble de cálculos que las de la red LIDI.
- las computadoras de la red LIDI-D pueden transmitir la misma cantidad de datos que las de la red LIDI.

La cantidad de memoria RAM instalada en las computadoras de la red LIDI-D es de 256 MB, con lo que se pueden hacer pruebas con matrices mayores que en la red LIDI.

En el caso de la combinación de las dos redes con tres switches de ocho bocas, la situación no es tan común por dos razones: a) no hay homogeneidad en las dieciséis PCs, y b) no se tiene la capacidad de "switching" completo, es decir que no son posibles todas las combinaciones de comunicaciones simultáneas de dos PCs al mismo tiempo. La Figura 5.8 muestra cómo interconectar dieciséis PCs ($PIII_0$, ..., $PIII_7$ de la red LIDI y D_0, ..., D_7 de la red LIDI-D), con tres switches de ocho bocas, donde se puede ver claramente cómo se pierde la capacidad de "switching" completo entre las dieciséis PCs. De aquí en más, esta red local con dieciséis PCs se denominará LIDI-16.

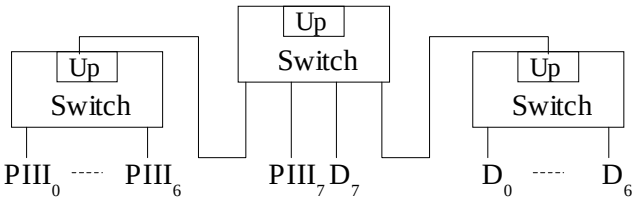


Figura 5.8: Redes LIDI y LIDI-D Interconectadas con Tres Switches.

En la red LIDI-16 y desde el punto de vista de rendimiento, se tendrá en cuenta que todas las computadoras tienen la capacidad de cómputo de las PCs de la red LIDI, es decir que se consideran homogéneas. El rendimiento de ScaLAPACK y de los algoritmos propuestos será teniendo en cuenta la capacidad de las computadoras de la red LIDI. Sin embargo, no hay una solución tan simple para el caso de las comunicaciones, dado que ScaLAPACK tiende a asumir que los arreglos bidimensionales de procesadores se comportan como una red estática con enlaces directos, y esto no es necesariamente posible con la organización de las interconexiones con tres switches como se muestra en la Figura 5.8. De todas maneras, se utilizará al menos como una idea de lo que puede suceder con una red de dieciséis computadoras, intentando identificar las características de escalabilidad de ScaLAPACK y de los algoritmos propuestos. Resumiendo, se llevarán a cabo experimentos en tres redes de PCs *homogéneas*: LIDI, LIDI-D y LIDI-16 interconectadas con switches Ethernet de 100 Mb/s.

Software de Base. En todos los casos el cómputo local que se lleva a cabo en cada PC se hace completamente optimizado. En el caso de ScaLAPACK, se tienen varias alternativas para las rutinas de comunicaciones de datos (implementaciones posibles de BLACS): PVM e implementaciones posibles de MPI. Para evitar comparar a priori cuál es la mejor de ellas se utilizaron dos: PVM y MPICH. La elección implementación de MPI elegida es MPICH dado que en la documentación y la instalación de ScaLAPACK tiende a utilizar esta biblioteca como *referente* de MPI. En el caso de los algoritmos propuestos para multiplicación y factorización LU de matrices se utilizará el mensaje broadcast

explícitamente implementado para el aprovechamiento de las redes Ethernet. Por lo tanto, se tienen dos alternativas de implementación de ScaLAPACK: con PVM y con MPICH, y una única alternativa de software de base para los algoritmos propuestos.

Parámetros de Ejecución. En el caso de ScaLAPACK se deben definir las grillas de procesadores y los tamaños de bloque. En el caso de las redes de 8 procesadores, se definieron grillas de: 8x1, 1x8, 2x4 y 4x2 procesadores. En el caso de la red de 16 procesadores, se definieron grillas de 8x2, 2x8 y 4x4 procesadores. En cuanto a tamaños de bloque, la idea es experimentar con tamaños pequeños y grandes de bloques de datos, más algunos intermedios considerados "clásicos" en el contexto de ScaLAPACK. Por lo tanto, los tamaños con los que se llevó a cabo la experimentación fueron (siempre bloques cuadrados, submatrices cuadradas de las matrices a procesar): 16, 32, 64, 126, 256, 512 y 1024 elementos.

Para los algoritmos propuestos en este capítulo y en el capítulo 3 no es necesaria la definición de ningún tipo de grilla, simplemente se utilizan todos los procesadores disponibles. En cuanto al tamaño de bloques, solamente son necesarios en la factorización LU de matrices y fueron los mismos que los utilizados para la biblioteca ScaLAPACK. Se agregaron algunos intermedios (no potencias de dos) como tamaños de bloques de 100 para identificar si había alguna variación de rendimiento diferente de la esperada.

Una última decisión se refiere a los tamaños de matrices, y en todos los casos se utilizaron los tamaños máximos de matrices, que dependen de la cantidad total de memoria de las computadoras y del problema mismo. En el caso de la multiplicación de matrices se tienen que almacenar tres matrices y en el caso de la factorización LU solamente una. Cuando se utiliza la red LIDI-16 se considera que **todas** las máquinas tienen 64 MB de RAM. Los tamaños específicos para cada problema y red de computadoras se muestran a continuación en la Tabla 5.2.

Cluster	Multiplicación de Matrices	Factorización LU de Matrices
LIDI	5000	9000
LIDI-D	10000	20000
LIDI-16	8000	13000

Tabla 5.2: Tamaños de Problemas en Cada Red Local.

Resumen de los Experimentos. En el caso de ScaLAPACK se tienen, para cada problema (multiplicación y factorización LU de matrices), dos conjuntos de resultados en cada red de PCs, dependiendo de la biblioteca de transporte de comunicaciones: PVM Y MPICH. En el caso de los algoritmos propuestos hay una única posibilidad, dado que tanto los algoritmos como la rutina de comunicaciones broadcast es *única*. La Tabla 5.3 resume los experimentos realizados, donde:

- ScaMM denota la multiplicación de matrices implementada en ScaLAPACK, más específicamente en PBLAS.
- PropMM es el algoritmo propuesto para multiplicar matrices con cómputo y comunicaciones solapadas.

- ScaLU es el algoritmo implementado por ScaLAPACK para hacer la factorización LU en paralelo.
- PropLU es el algoritmo propuesto para hacer la factorización de matrices LU en paralelo.
- Bcast-UDP es la rutina de comunicaciones broadcast específicamente optimizada para aprovechar las características del broadcast de las redes Ethernet.

Comm.	ScaMM	PropMM	ScaLU	PropLU
PVM	#blq, Grilla		#blq, Grilla	
MPICH	#blq, Grilla		#blq, Grilla	
Bcast-UDP		1		#blq

Tabla 5.3: Experimentos con ScaLAPACK y con los Algoritmos Propuestos.

Las entradas vacías en la Tabla 5.3 corresponden a experimentos que no tiene sentido o no se pueden llevar a cabo:

- La multiplicación y la factorización de matrices de ScaLAPACK no se puede llevar a cabo de manera inmediata con la rutina implementada para aprovechar las características del broadcast de las redes Ethernet porque ScaLAPACK necesita la biblioteca BLACS completa, no *solamente* una rutina de broadcast entre procesos.
- Ya se mostró en el capítulo anterior que el rendimiento obtenido con la multiplicación de matrices con el algoritmo propuesto y la biblioteca PVM está lejos de ser aceptable. Se eligió, por lo tanto, experimentar con los algoritmos propuestos descartando la biblioteca PVM y todas las implementaciones de MPI (incluyendo MPICH) que, a priori, no aseguran la implementación optimizada de los mensajes broadcast.

Las entradas en la Tabla 5.3 que contienen “#blq”, indican que el rendimiento depende al menos del tamaño de bloques de datos con los que se trabaje. Si, además, la entrada tiene “Grilla” se está indicando que es necesaria la definición de un arreglo bidimensional de los procesadores que se utilizan. Hay una única entrada en la Tabla 5.3 que contiene un “1” y esto indica que no hay ningún parámetro a definir para esta alternativa. Expresado de otra manera, el algoritmo propuesto para multiplicar matrices no depende de nada más que de las computadoras a utilizar. En las dos subsecciones que siguen se muestran los resultados en función de ScaLAPACK y comparándolos directamente con los resultados obtenidos por los algoritmos propuestos, dado que ese es el objetivo de toda la experimentación en este capítulo.

5.3.2 Resultados: ScaLAPACK-PVM

La instalación de ScaLAPACK *sobre* PVM es muy sencilla, y se pueden probar rápidamente las diferentes alternativas (que son paramétricas) de ejecución. En el eje **y** de las figuras siguientes se muestra el rendimiento en términos del Speedup obtenido por cada algoritmo y en el eje **x** de la misma figura se muestran los algoritmos y parámetros utilizados para su ejecución. Los resultados obtenidos con ScaLAPACK se muestran con las barras más claras e identificando cada barra con los tres parámetros con se se obtuvo:

tamaño de bloque - cantidad de filas de procesadores de la grilla y cantidad columnas de procesadores de la grilla. Los resultados obtenidos con los algoritmos propuestos se muestran con las barras más oscuras e identificados con “Prop”.

La Figura 5.9 muestra los mejores cinco mejores resultados de rendimiento con las diferentes combinaciones de parámetros con ScaLAPACK-PVM y el rendimiento del algoritmo propuesto en el capítulo 3 (cómputo solapado con comunicaciones) implementado con los mensajes broadcasts optimizados en la red LIDI. El mejor resultado de rendimiento obtenido con ScaLAPACK-PVM en la red LIDI (última barra en gris claro de izquierda a derecha de la Figura 5.9, identificada con 32-4-2) corresponde a la ejecución con bloques de datos de 32x32 elementos y la grilla de procesadores de 4x2. El algoritmo propuesto, con las comunicaciones optimizadas se muestra identificado como “Prop” y la barra que le corresponde es más oscura en la figura.

El mejor resultado de rendimiento en términos de Speedup obtenido por ScaLAPACK-PVM es de poco menos de 5.8 y el rendimiento obtenido por el algoritmo propuesto es de casi 7.2, la mejora porcentual con respecto a ScaLAPACK-PVM es de aproximadamente 25%. Es decir que, con el mismo hardware, el algoritmo propuesto tiene aproximadamente 25% mejor speedup que el de ScaLAPACK-PVM. El speedup óptimo es 8.

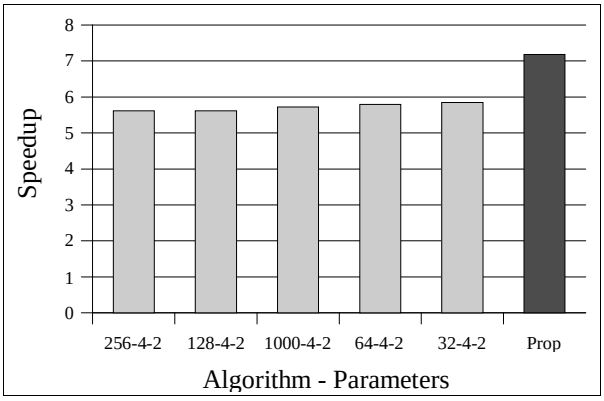


Figura 5.9: Multiplicación de Matrices en LIDI, ScaLAPACK-PVM.

La Figura 5.10 muestra los mejores cinco mejores resultados de rendimiento con las diferentes combinaciones de parámetros con ScaLAPACK-PVM y el rendimiento del algoritmo propuesto en el capítulo 3 (cómputo solapado con comunicaciones) más los mensajes broadcasts optimizados en la red LIDI-D.

El mejor resultado de rendimiento obtenido con ScaLAPACK-PVM en la red LIDI (última barra en gris claro de izquierda a derecha de la Figura 5.10, identificada con 64-4-2), corresponde a la ejecución con bloques de datos de 64x64 elementos y la grilla de procesadores de 4x2. El algoritmo propuesto, con las comunicaciones optimizadas se muestra identificado como “Prop” y la barra que le corresponde es más oscura en la figura. El mejor resultado de rendimiento en términos de Speedup obtenido por ScaLAPACK-PVM es de poco menos de 5.6 y el rendimiento obtenido por el algoritmo propuesto es de casi 7, la mejora porcentual con respecto a ScaLAPACK-PVM es de poco más del 25%. Es decir que, con el mismo hardware, el algoritmo propuesto tiene aproximadamente 25%

mejor speedup que el de ScaLAPACK-PVM. El speedup óptimo es 8.

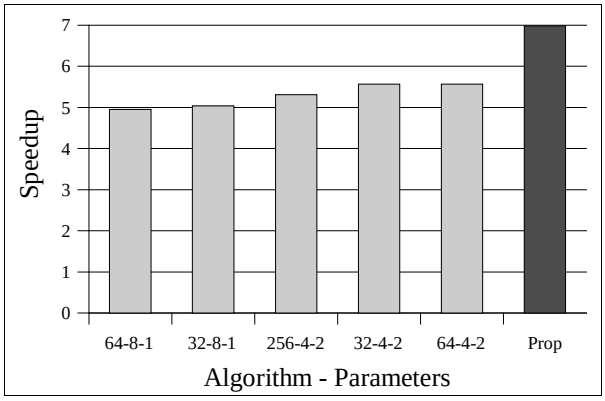


Figura 5.10: Multiplicación de Matrices en LIDI-D, ScaLAPACK-PVM.

La Figura 5.11 muestra los mejores cinco mejores resultados de rendimiento con las diferentes combinaciones de parámetros con ScaLAPACK-PVM y el rendimiento del algoritmo propuesto en el capítulo 3 (cómputo solapado con comunicaciones) más los mensajes broadcasts optimizados en la red LIDI-16.

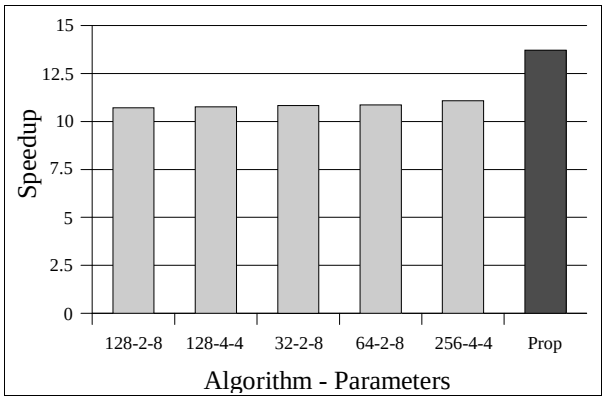


Figura 5.11: Multiplicación de Matrices en LIDI-D, ScaLAPACK-PVM.

El mejor resultado de rendimiento obtenido con ScaLAPACK-PVM en la red LIDI (última barra en gris claro de izquierda a derecha, identificada con 256-4-4), corresponde a la ejecución con bloques de datos de 256x256 elementos y la grilla de procesadores de 4x4. El algoritmo propuesto, con las comunicaciones optimizadas se muestra identificado como “Prop” y la barra que le corresponde es más oscura en la figura.

El mejor resultado de rendimiento en términos de Speedup obtenido por ScaLAPACK-PVM es de poco más de 11 y el rendimiento obtenido por el algoritmo propuesto es de casi 13.8, la mejora porcentual con respecto a ScaLAPACK-PVM es de casi 24%. Es decir que, con el mismo hardware, el algoritmo propuesto tiene casi 24% mejor speedup que el de ScaLAPACK-PVM. El speedup óptimo en este caso es de 16.

Dado que para factorización LU de matrices se pueden tener diferentes tamaños de bloque

para el algoritmo propuesto, los resultados se muestran a continuación tienen las siguientes características:

- Los distintos tamaños de bloques se muestran en cada barra de las figuras a continuación que corresponden al algoritmo propuesto de factorización LU, que ahora están identificadas con “Prop-tamaño de bloque”.
- Se muestran los tres mejores resultados de rendimiento para ScaLAPACK y para el algoritmo propuesto, con lo que cada figura tiene tres barras en gris claro y tres barras en gris oscuro.

La Figura 5.12 muestra los mejores tres mejores resultados de rendimiento con las diferentes combinaciones de parámetros con ScaLAPACK-PVM y los tres mejores resultados de rendimiento del algoritmo propuesto en este capítulo (cómputo solapado con comunicaciones, diferentes tamaños de bloques) implementado con los mensajes broadcasts optimizados en la red LIDI. El mejor resultado de rendimiento obtenido con ScaLAPACK-PVM en la red LIDI (última barra en gris claro de izquierda a derecha de la figura, identificada con 32-1-8) corresponde a la ejecución con bloques de datos de 32x32 elementos y la grilla de procesadores de 1x8. El mejor rendimiento del algoritmo propuesto corresponde al tamaño de bloque 128 y se muestra identificado como “Prop-128” en la última barra de izquierda a derecha en la figura.

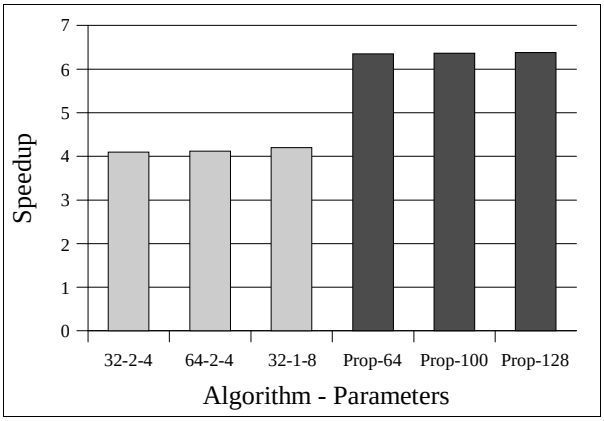


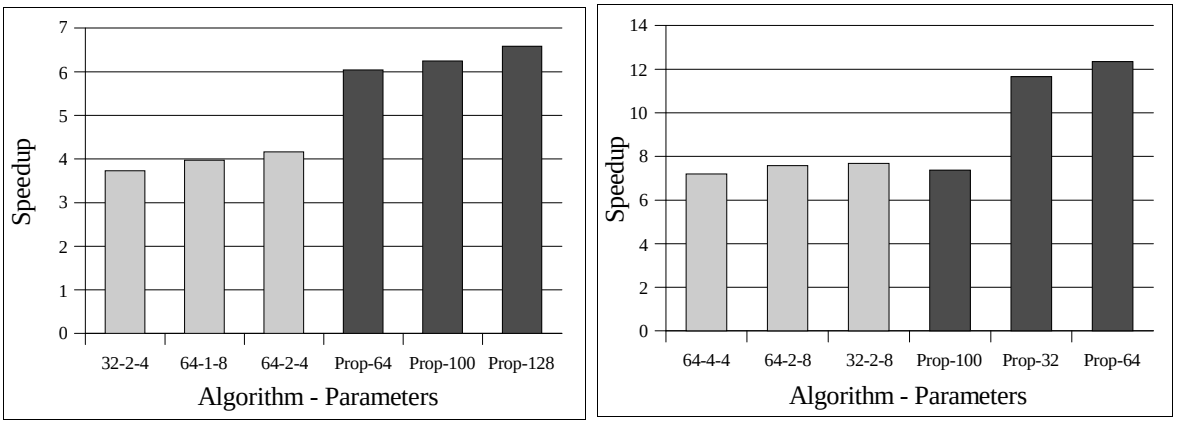
Figura 5.12: Factorización LU de Matrices en LIDI, ScaLAPACK-PVM.

El mejor resultado de rendimiento en términos de Speedup obtenido por ScaLAPACK-PVM es de aproximadamente 4.2 y el rendimiento obtenido por el algoritmo propuesto es de casi 6.4, la mejora porcentual con respecto a ScaLAPACK-PVM es de casi 52%. Es decir que, con el mismo hardware, el algoritmo propuesto tiene casi 52% mejor speedup que el de ScaLAPACK-PVM. El speedup óptimo es 8.

En la Figura 5.13 se muestran los resultados de speedup obtenido en las redes LIDI-D y LIDI-16. En todos los casos la mejora con el algoritmo propuesto está entre 50% y 60% con respecto al rendimiento obtenido por ScaLAPACK-PVM.

Una de las primeras conclusiones preliminares que se puede derivar de los resultados que se han mostrado, es que, en todos los casos, los algoritmos propuestos tienen mejor rendimiento que ScaLAPACK-PVM. Dado que, como se ha visto en el capítulo anterior, el rendimiento de PVM puede ser inaceptable, es posible que el rendimiento de ScaLAPACK

sea ampliamente penalizado por la utilización de PVM para la comunicación entre sus datos. Es por eso que se hicieron los experimentos que están a continuación, que son *exactamente* los mismos pero se cambia la biblioteca PVM por una implementación de MPI, MPICH.

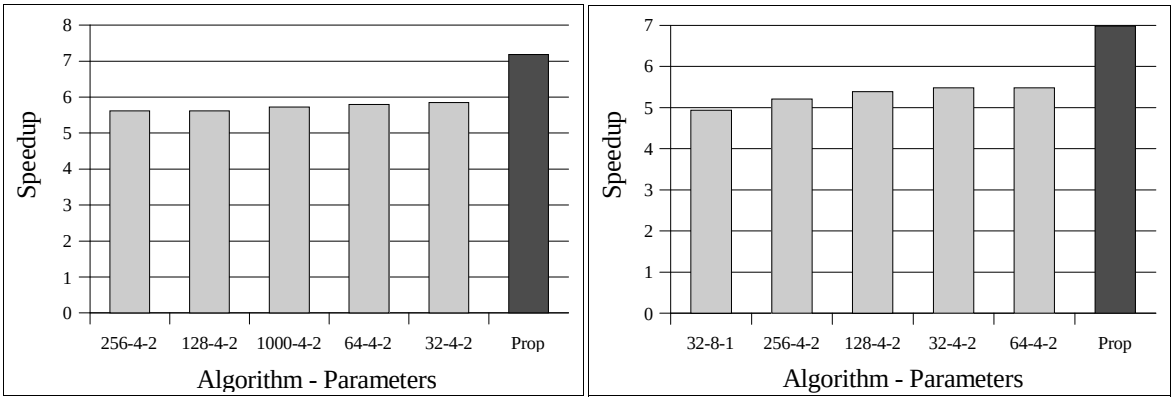


a) LIDI-D b) LIDI-16

Figura 5.13: Factorización LU de Matrices en LIDI-D y LIDI-16, ScaLAPACK-PVM.

5.3.3 Resultados: ScaLAPACK-MPICH

La instalación de ScaLAPACK *sobre* MPICH también es muy sencilla, y se pueden probar rápidamente las diferentes alternativas (que son paramétricas) de ejecución. Dado que el formato de las figuras con las que se muestra el resultado obtenido es el mismo, se pondrán los resultados sin una descripción tan detallada como en los casos anteriores. En todos los casos se repite el rendimiento obtenido con los algoritmos propuestos en esta tesis para tener una comparación visual más inmediata. En la Figura 5.14 se muestran los resultados de los experimentos en las dos redes de ocho PCs: LIDI y LIDI-D.



a) LIDI b) LIDI-D

Figura 5.14: Multiplicación de Matrices en LIDI y LIDI-D, ScaLAPACK-MPICH.

En el cluster LIDI el algoritmo propuesto para multiplicar matrices tiene casi 23% mejor resultado que la biblioteca ScaLAPACK (en términos de valores de Speedup) y en el

cluster LIDI-D supera en algo más de 27%.

Para completar los datos, se muestran en la Figura 5.15 los resultados de rendimiento obtenido en la red LIDI-16.

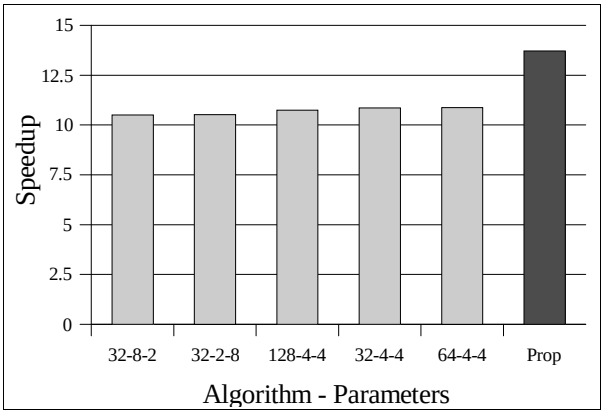
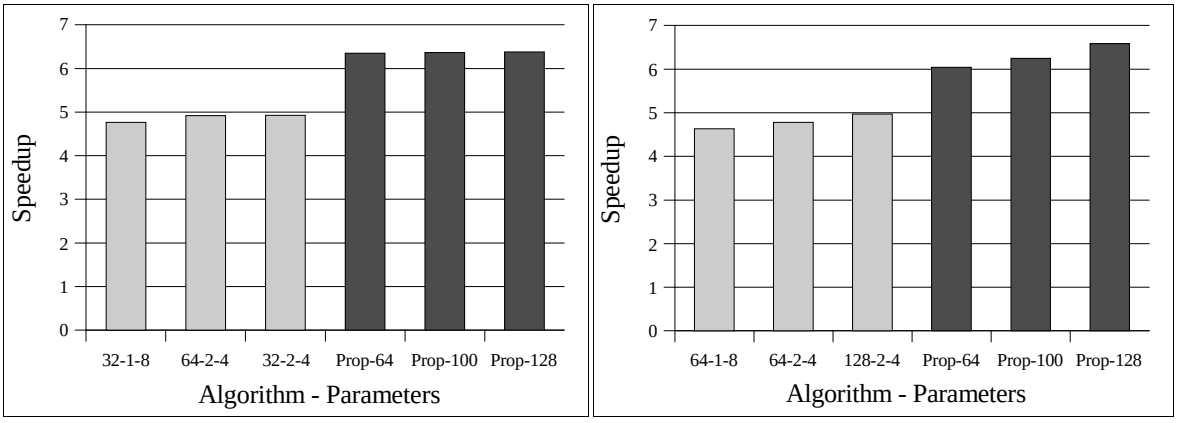


Figura 5.15: Multiplicación de Matrices en LIDI-16, ScaLAPACK-MPICH.

En este caso el algoritmo propuesto en esta tesis para multiplicar matrices en clusters supera a la multiplicación de matrices de ScaLAPACK por algo más del 26%. Resumiendo, los resultados obtenidos con ScaLAPACK-MPICH son muy similares a los obtenidos con ScaLAPACK-PVM y por lo tanto la comparación de los resultados de rendimiento con el algoritmo de multiplicación de matrices propuesto es similar también.

La situación cambia bastante cuando se trata de la factorización LU. La Figura 5.16 muestra los valores de Speedup obtenidos con ScaLAPACK-MPICH y repite los obtenidos con el algoritmo propuesto en este capítulo para las dos redes de ocho PCs.



a) LIDI b) LIDI-D

Figura 5.16: Factorización LU de Matrices en LIDI y LIDI-D, ScaLAPACK-MPICH.

Comparando los valores de Speedup de ScaLAPACK-MPICH que aparecen en la Figura 5.16 con los que se tienen de ScaLAPACK-PVM en la Figura 5.12 y Figura 5.13-a) se puede notar que ScaLAPACK tiene bastante mejor rendimiento de la factorización LU cuando el transporte de los datos está a cargo de MPICH. De hecho, la ganancia del

algoritmo propuesto con respecto a ScaLAPACK-PVM varía entre 50% y 60%, pero con respecto a ScaLAPACK-MPICH varía entre 30% y 32%. Dos de las conclusiones inmediatas a partir de esta información son:

- PVM tiene en algunos casos una fuerte penalización de rendimiento de comunicaciones, que *se transforma* en penalización de rendimiento total.
- Los algoritmos paralelos propuestos tienen una ganancia "casi constante" con respecto a los algoritmos que implementa ScaLAPACK. Esta ganancia, en términos porcentuales, es de aproximadamente 25% en el caso de la multiplicación de matrices y 30% en el caso de la factorización LU de matrices. Es muy interesante, porque son dos problemas que se paralelizan utilizando los mismos principios y la ganancia es superior para ambos, es decir que los principios de paralelización para redes locales se "confirman" como mejores al menos para estos dos casos: multiplicación y factorización LU de matrices.

La Figura 5.17, además de completar los datos de la factorización de matrices en paralelo para la red de dieciséis PCs, agrega información valiosa respecto del rendimiento de ScaLAPACK cuando la red no es totalmente "switched" como en el caso de esta red en particular. Como se puede apreciar, el rendimiento de ScaLAPACK-MPICH es muy similar al de ScaLAPACK-MPICH (y, por lo tanto, la diferencia con el algoritmo de factorización de matrices propuesto en este capítulo vuelve a ser bastante grande). Se podrían dar al menos dos causas para que el rendimiento de ScaLAPACK-MPICH "empeore":

- El algoritmo de factorización de matrices de ScaLAPACK no es escalable y por lo tanto cuando se utilizan más computadoras el rendimiento es peor.
- La red no totalmente *switched* genera una penalización extra de rendimiento que no tienen las redes totalmente *switched* y por lo tanto el rendimiento paralelo total empeora notablemente.

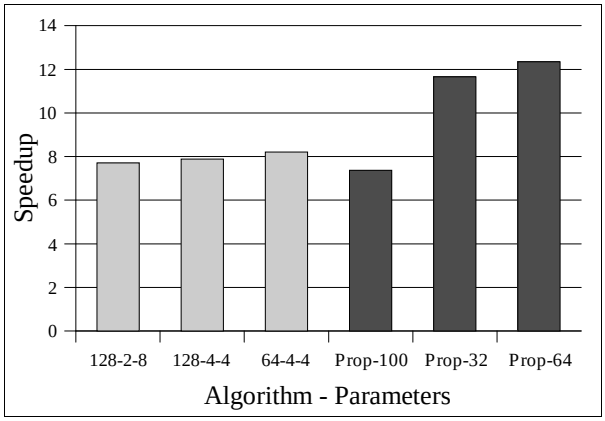


Figura 5.17: Factorización LU de Matrices en LIDI-16, ScaLAPACK-MPICH.

Se debe tener en cuenta en este punto que el Speedup obtenido en las dos redes interconectadas con un único switch es de aproximadamente 63% del óptimo mientras que el obtenido en la red LIDI-16 (que no tiene un único switch sino tres "en cascada") es aproximadamente 50% del óptimo. Por lo tanto, dado que

- el rendimiento de ScaLAPACK es fuertemente influido por el rendimiento de las comunicaciones,

- el rendimiento relativo que se obtiene en las dos redes interconectadas por un único switch es bastante superior al que se obtiene en una red que no tiene esa característica de interconexión,

se puede descartar, al menos en principio, la falta de escalabilidad de ScaLAPACK.

5.3.4 ScaLAPACK-MPICH y Escalabilidad

En esta última subsección se mostrarán los resultados obtenidos en la mayor de los clusters en los que se pudo realizar experimentación para comparar los algoritmos propuestos en esta tesis con los implementados en ScaLAPACK para llevar a cabo la misma tarea. Este cluster está constituido por 20 PCs interconectadas por un único switch Ethernet de 100 Mb/s, es decir que se tiene switching completo con las 20 máquinas interconectadas por Ethernet de 100 Mb/s. En la Tabla 5.4 se muestran de manera resumida las características de las PCs del cluster, que se denominará a partir de ahora como Cl-20, junto con su capacidad de cómputo expresada en términos de Mflop/s.

Procesadores	Frec. Reloj	Memoria	Mflop/s
Intel P4	2.4 GHz	1 GB	5000

Tabla 5.4: Características de las Computadoras del Cluster Cl-20.

Por otro lado, la Tabla 5.5 muestra los tamaños de matrices utilizados en los problemas de multiplicación y factorización LU respectivamente.

Multiplicación de Matrices	Factorización LU de Matrices
38000	65000

Tabla 5.5: Tamaños de Problemas en el Cluster Cl-20.

Dado que la biblioteca ScaLAPACK obtiene sus mejores resultados de rendimiento al utilizar la biblioteca de pasaje de mensajes MPICH, todos los experimentos que se llevaron a cabo con ScaLAPACK se utiliza MPICH para pasaje de mensajes. En cierta forma, los experimentos que se presentan a continuación tienen dos características que los distinguen de los anteriores:

1. Son los que utilizan la mayor cantidad de máquinas, y en cierta forma muestran de manera acotada (hasta 20 computadoras) la escalabilidad de los algoritmos propuestos.
2. Son los que utilizan las computadoras con la mayor potencia de cálculo, manteniendo la red de interconexión a 100 Mb/s, con lo que se tiene la peor relación entre el rendimiento de cómputo local y el de comunicaciones.

Dado que se tienen 20 computadoras y que las recomendaciones de ScaLAPACK para la obtención de rendimiento optimizado es mantener la grilla de P×Q con P lo más similar a Q posible [21], las grillas utilizadas en los experimentos fueron de 4×5 y 5×4 procesadores.

La Figura 5.18 muestra los valores de rendimiento obtenido con ScaLAPACK-MPICH y

con el algoritmo propuesto en esta tesis (cómputo solapado con comunicaciones) para la multiplicación de matrices de 38000×38000 elementos. El mejor valor obtenido por ScaLAPACK-MPICH se tiene con tamaño de bloque 32 y con las máquinas que se interconectan como en una grilla de 5×4 procesadores. El valor absoluto de Speedup obtenido por ScaLAPACK-MPICH en este caso es de aproximadamente 10. En el caso del algoritmo propuesto, se tiene Speedup de algo más de 16, lo que representa una mejora de alrededor del 62% con respecto a ScaLAPACK-MPICH. En este caso, el valor óptimo de Speedup es 20, la cantidad de computadoras utilizadas.

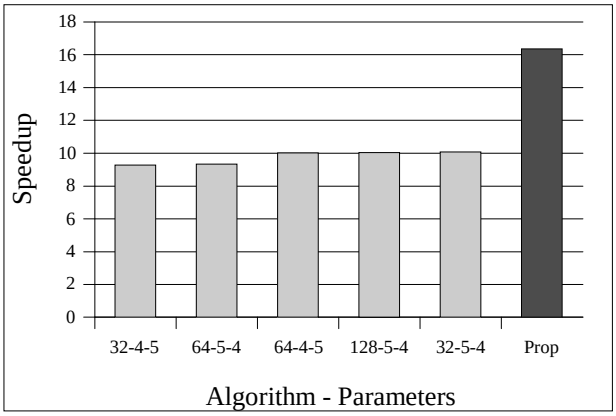


Figura 5.18: Multiplicación de Matrices en Cl-20, ScaLAPACK-MPICH.

La Figura 5.19 muestra los valores de rendimiento obtenido con ScaLAPACK-MPICH y con el algoritmo propuesto en esta tesis (cómputo solapado con comunicaciones) para la factorización LU de matrices de 65000×65000 elementos. El mejor valor obtenido por ScaLAPACK-MPICH se tiene con tamaño de bloque 64 y con las máquinas que se interconectan como en una grilla de 4×5 procesadores. El valor absoluto de Speedup obtenido por ScaLAPACK-MPICH en este caso es de aproximadamente 10. En el caso del algoritmo propuesto, se tiene Speedup de algo más de 18, lo que representa una mejora de alrededor del 80% con respecto a ScaLAPACK-MPICH. También en este caso, el valor óptimo de Speedup es 20, la cantidad de computadoras utilizadas.

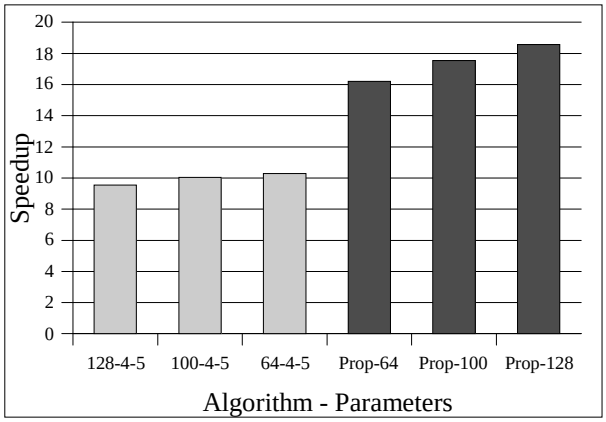


Figura 5.19: Factorización LU de Matrices en Cl-20, ScaLAPACK-MPICH.

Es muy interesante notar que, más allá de la comparación con ScaLAPACK, se tienen resultados de Speedup muy cercanos al máximo absoluto. En el caso de la multiplicación de matrices se tiene casi el 82% del rendimiento óptimo y en el caso de la factorización LU se tiene casi el 93% del óptimo, lo cual es muy satisfactorio, teniendo en cuenta que, por ejemplo, se tiene una red de interconexión de *solamente* 100 Mb/s y con alta latencia.

5.4 Resumen de la Comparación con ScaLAPACK

La Tabla 5.6 muestra de forma resumida la comparación de rendimiento obtenido por los algoritmos propuestos en esta tesis con los implementados por/en ScaLAPACK. Par cada uno de los clusters utilizados (LIDI, LIDI-D, LIDI-16 y Cl-20) se da

- El rendimiento de ScaLAPACK en términos de Speedup (columna **Sca**).
- El rendimiento de los algoritmos propuestos en esta tesis en términos de Speedup (columna **Prop**).
- El porcentaje de mejora en Speedup que se tiene con la utilización de los algoritmos propuestos en esta tesis (columna %**Prop**).

	LIDI 8 PIII - 100 Mb/s			LIDI-D 8 D - 100 Mb/s			LIDI-16 16 PCs - 100 Mb/s*			Cl-20 20 PCs - 100 Mb/s		
	Sca	**Prop**	**%Prop**	**Sca**	**Prop**	**%Prop**	**Sca**	**Prop**	**%Prop**	**Sca**	**Prop**	**%Prop**
MM	5.84	7.18	+23%	5.48	6.98	+27%	10.87	13.72	+26%	10.08	16.35	+62%
LU	4.93	6.38	+30%	4.97	6.59	+33%	8.2	12.35	+51%	10.28	18.56	+81%

*La única red Ethernet con combinación de switches *en cascada*, todas las demás tienen *switching* completo.

Tabla 5.6: Resumen de la Comparación con ScaLAPACK.

Si bien la ganancia en términos de Speedup es notable, es más importante aún que en clusters con mayor cantidad de computadoras y con peor relación de rendimiento de cómputo local con respecto a comunicaciones, la ganancia tiende a ser mayor. Más aún, independientemente de la comparación con ScaLAPACK, la Tabla 5.7 muestra los valores absolutos de Speedup obtenido por los algoritmos propuestos en esta tesis (columna **Prop**) junto con el porcentaje del óptimo que esos valores representan (columna **%Op**).

	LIDI 8 PIII - 100 Mb/s		LIDI-D 8 D - 100 Mb/s		LIDI-16 16 PCs - 100 Mb/s*		Cl-20 20 PCs - 100 Mb/s	
	Prop	**%Op**	**Prop**	**%Op**	**Prop**	**%Op**	**Prop**	**%Op**
MM	7.18	90%	6.98	87%	13.72	86%	16.35	82%
LU	6.38	80%	6.59	82%	12.35	77%	18.56	93%

*La única red Ethernet con combinación de switches *en cascada*, todas las demás tienen *switching* completo.

Tabla 5.7: Relación de los Algoritmos Propuestos con el Optimo Absoluto.

Todos los valores, excepto para la factorización LU en LIDI-16 superan el 80% del óptimo absoluto que es altamente satisfactorio, más aún recordando que se utiliza una red de

interconexión de muy bajo costo y que no depende de la utilización de switches para la interconexión. Aún cuando la excepción de LU en LIDI-16 podría utilizarse como indicación de que el algoritmo no es suficientemente escalable, esta presunción se podría descartar en base a los valores obtenidos para mayor cantidad de computadoras y de mayor potencia de cálculo del cluster Cl-20.