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.

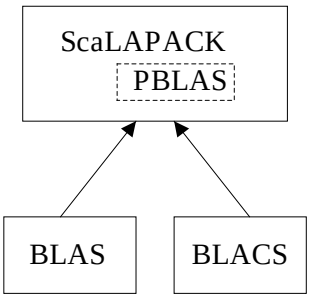


Figura 5.1: Una Visión en Capas de ScaLAPACK.

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.

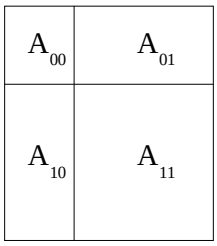


Figura 5.2: División de una Matriz en Bloques.

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

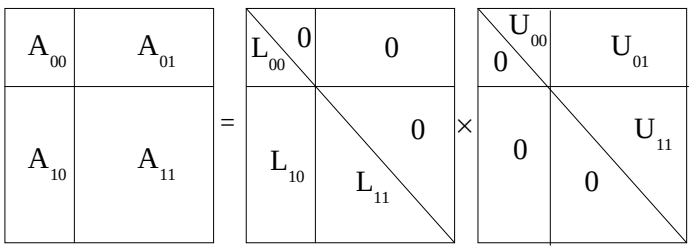


Figura 5.3: Factorización LU en Términos de Bloques.

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.

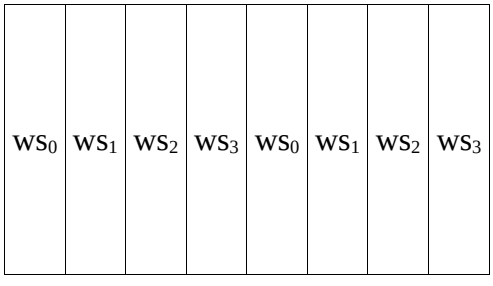


Figura 5.5: Distribución Cíclica por Bloques de Columnas.

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.

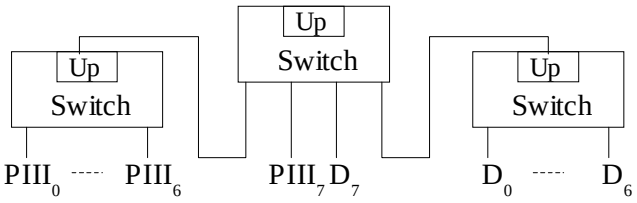


Figura 5.8: Redes LIDI y LIDI-D Interconectadas con Tres Switches.

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.

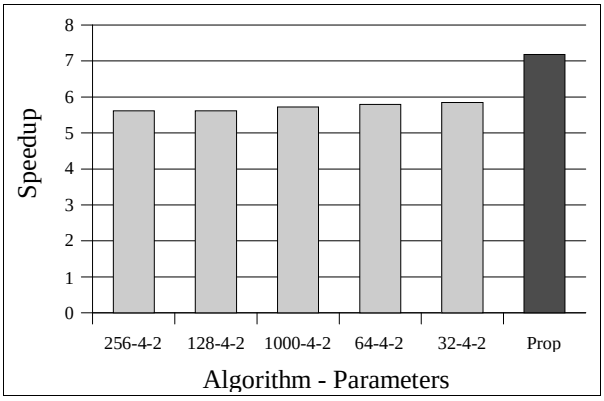


Figura 5.9: Multiplicación de Matrices en LIDI, ScaLAPACK-PVM.

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.

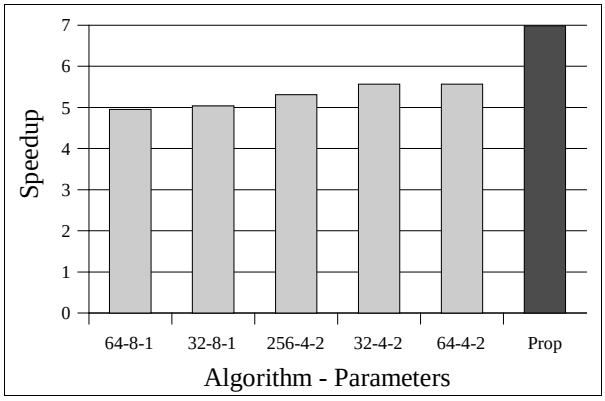


Figura 5.10: Multiplicación de Matrices en LIDI-D, ScaLAPACK-PVM.

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.

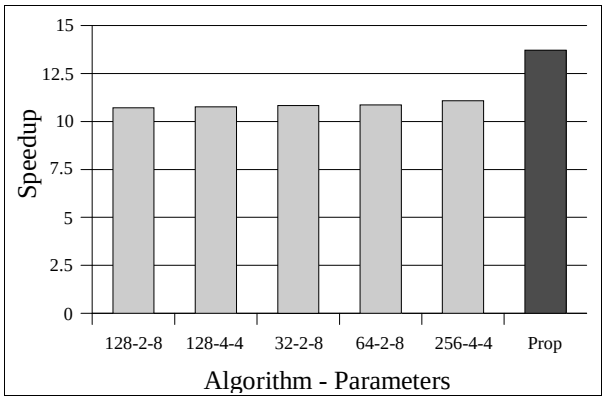


Figura 5.11: Multiplicación de Matrices en LIDI-D, ScaLAPACK-PVM.

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.

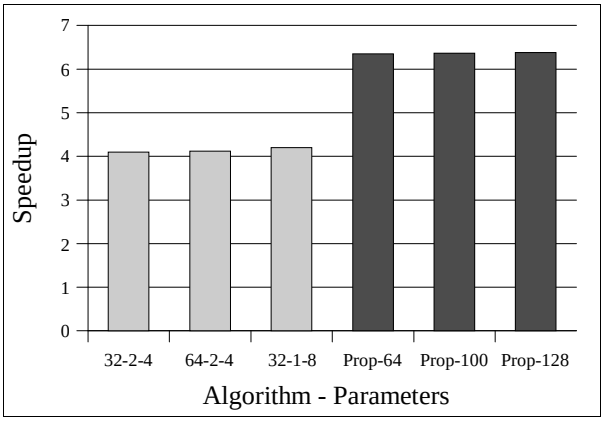


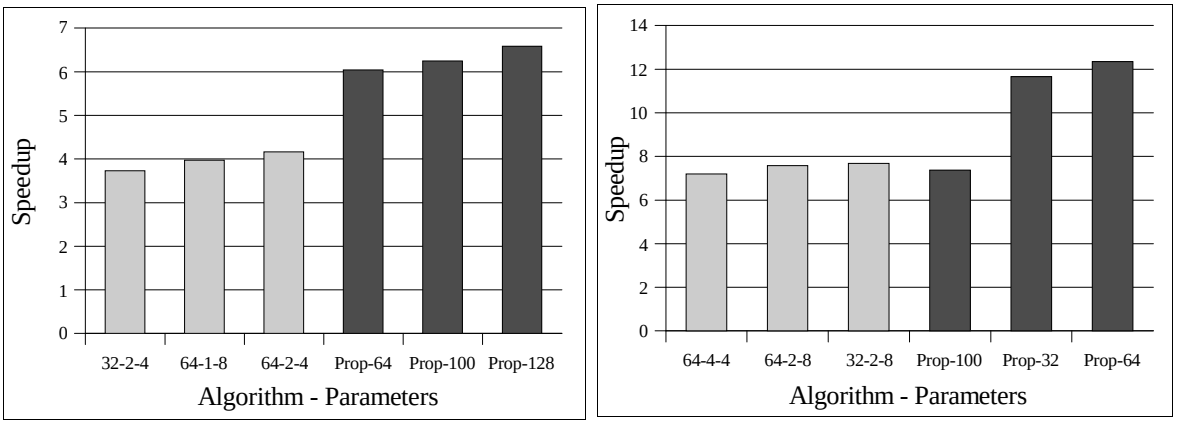
Figura 5.12: Factorización LU de Matrices en LIDI, ScaLAPACK-PVM.

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.

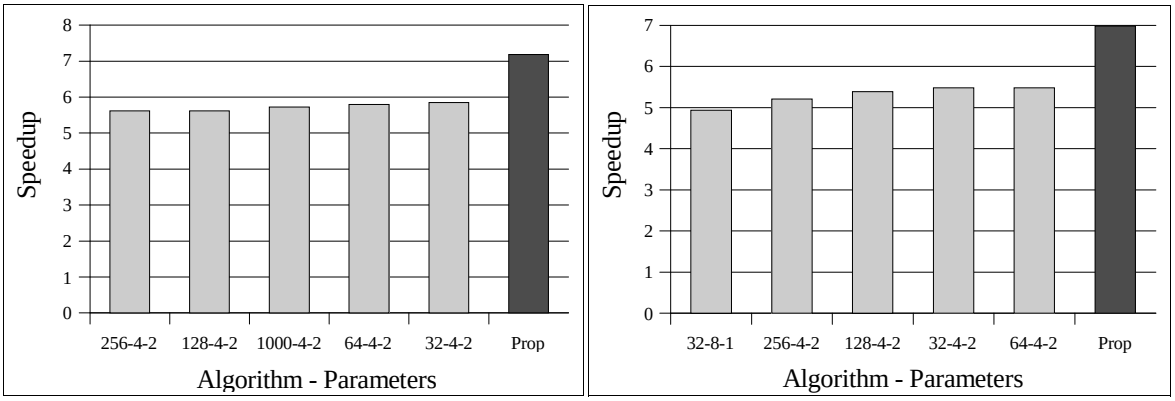


a) LIDI-D b) LIDI-16

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.

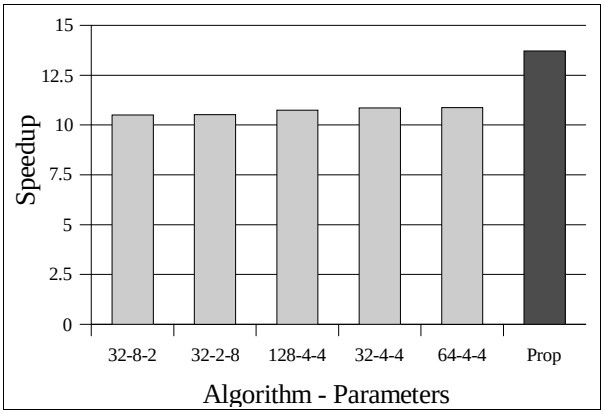
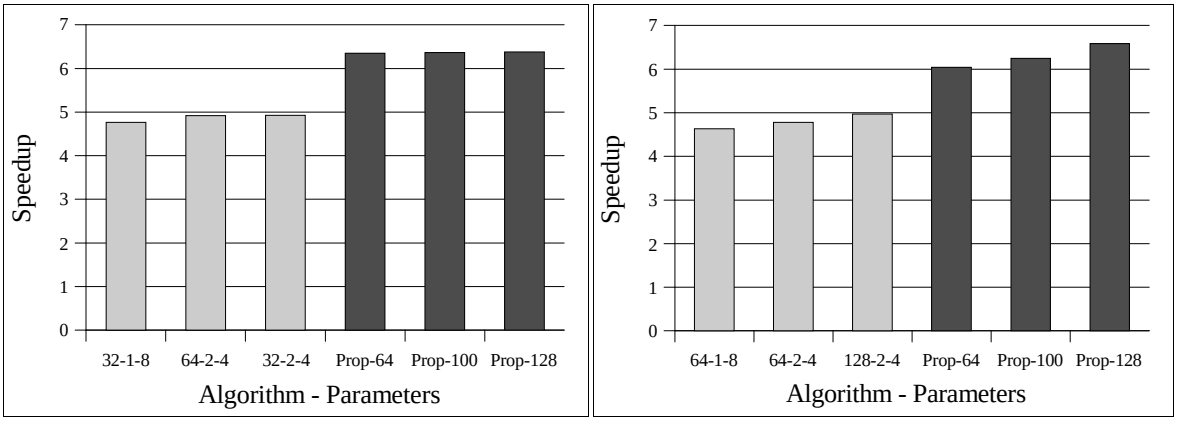


Figura 5.15: Multiplicación de Matrices en LIDI-16, ScaLAPACK-MPICH.

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.

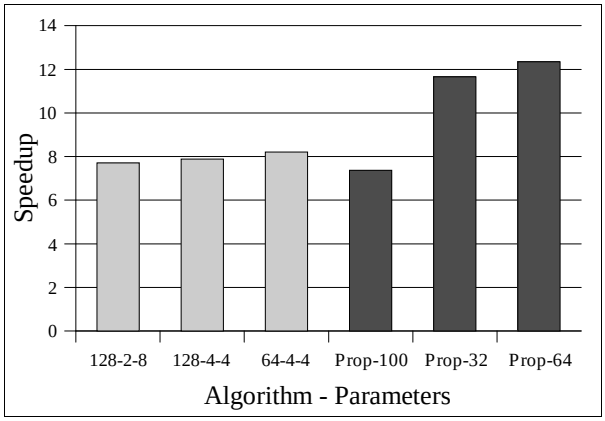


Figura 5.17: Factorización LU de Matrices en LIDI-16, ScaLAPACK-MPICH.

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.

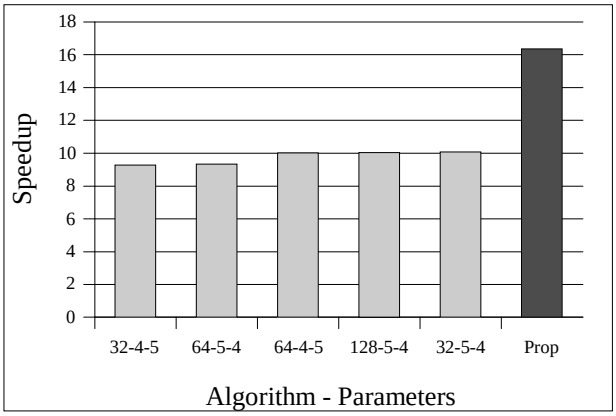


Figura 5.18: Multiplicación de Matrices en Cl-20, ScaLAPACK-MPICH.

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.

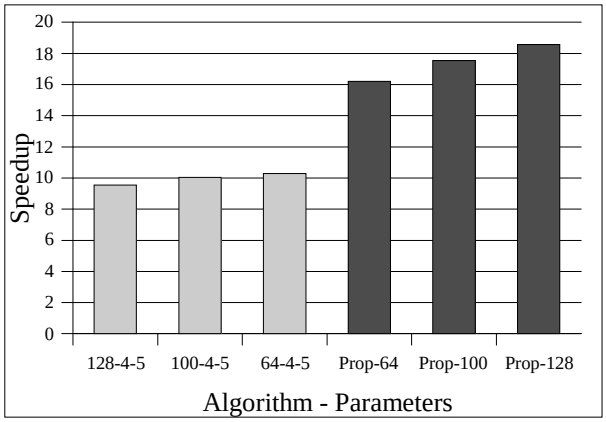


Figura 5.19: Factorización LU de Matrices en Cl-20, ScaLAPACK-MPICH.

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.

Capítulo 6: Conclusiones y Trabajo Futuro

En este capítulo se presentan las principales conclusiones de esta tesis a partir de los capítulos anteriores con un énfasis especial en la dirección de identificar problemas y soluciones para la obtención de máximo rendimiento con cómputo paralelo en redes locales de computadoras. El contexto inicial de hardware de procesamiento paralelo lo dan las redes locales de computadoras que están instaladas y que se pueden aprovechar para resolver problemas en paralelo. También se presenta un breve resumen de los aportes de esta tesis y su relación con las publicaciones que se han hecho a lo largo del desarrollo de la misma.

También en este capítulo se presentan algunas consideraciones respecto de la continuación de la investigación en esta área. Si bien es muy difícil una estimación precisa de las extensiones sí se pueden identificar con bastante claridad algunos problemas inmediatos que se pueden resolver en este contexto de cómputo paralelo y también algunas alternativas de utilización de hardware más allá de una red local de computadoras.

6.1 Conclusiones

La evolución de las computadoras paralelas ha sido clara en varias direcciones, una de las cuales es la utilización de hardware de cómputo estándar. En este sentido, los microprocesadores de uso masivo en las computadoras de bajo costo como las estaciones de trabajo y PCs son utilizados en las computadoras paralelas que se reportan entre las de mayor capacidad de cálculo absoluta [86]. En el escalafón más bajo en cuanto a costo de cómputo paralelo se pueden ubicar a las redes locales de computadoras, que tienen el mismo tipo de procesadores de base pero que en principio no han sido orientadas a cómputo paralelo. De hecho, la mejor relación costo/rendimiento de hardware paralelo es el de las redes locales ya instaladas porque no tienen ningún tipo de costo de instalación ni de mantenimiento dado que su existencia es independiente del cómputo paralelo. Sin embargo, no se pueden dejar de lado otros costos adicionales que tienen estas redes, tales como el de instalación y mantenimiento del software específico para cómputo paralelo y el de la baja disponibilidad de las computadoras que, como se ha mencionado, no tienen como prioridad la ejecución de programas paralelos.

Tanto las redes de computadoras ya instaladas que se pueden utilizar para cómputo paralelo como las instalaciones del tipo Beowulf y/o clusters homogéneos que evolucionan en el sentido de reposición y/o agregado de computadoras suelen tener hardware de cómputo heterogéneo y hardware de comunicaciones homogéneo. La heterogeneidad de las computadoras de las redes locales instaladas es más o menos “natural” teniendo en cuenta tanto el tiempo de instalación y consiguiente evolución, como las distintas funciones o tipo de problemas hacia los cuales están orientadas cada una de las computadoras de la red local. La heterogeneidad de las computadoras de las instalaciones del tipo Beowulf y/o clusters homogéneos es una consecuencia no buscada de la evolución del hardware de bajo costo utilizado como base: las PCs. El tiempo de disponibilidad en el mercado de los componentes básicos tales como un tipo de procesador, memoria y disco rígido de PCs, por ejemplo, es muy corto. Por lo tanto, siempre que se necesita mayor potencia de cálculo (más PCs) o reemplazar una computadora que deja de funcionar, la probabilidad de tener heterogeneidad en el hardware es relativamente grande comparada con la que tiene una computadora paralela “tradicional” bajo las mismas circunstancias. Y la probabilidad crece a medida que transcurre el tiempo y los componentes no están disponibles de manera inmediata en el mercado.

La homogeneidad del hardware de comunicaciones está dada también por el bajo costo, en este caso de las placas de interfase (NIC: Network Interface Cards) de comunicaciones Ethernet. El estándar definido como Ethernet en sus varias versiones se ha instalado como el de más bajo costo y aparentemente va a seguir con esta tendencia en todas sus versiones definidas hasta el momento: 10 Mb/s, 10/100 Mb/s, 1 Gb/s y 10 Gb/s. De hecho, las redes locales mayoritariamente utilizan Ethernet de 10 y de 10/100 Mb/s dado que ha probado ser muy útil para la mayoría de las aplicaciones de oficina. Además, las instalaciones Beowulf se recomiendan con hardware de comunicaciones de 100 Mb/s y con cableado basado en switches de comunicaciones Ethernet de 100 Mb/s y/o 1 Gb/s. El bajo costo de estas redes incluye no solamente el mismo hardware de interconexión de las computadoras (NICs), sino también todo el personal técnico ya capacitado y con experiencia, con el que los demás tipos de redes utilizadas no cuentan.

Tanto la heterogeneidad de cómputo como las redes Ethernet de interconexión de procesadores (desde el punto de vista de una máquina paralela) tienen características muy bien definidas y no necesariamente apropiadas para cómputo paralelo. La heterogeneidad de cómputo plantea un problema que raramente (sino *nunca*) se debe enfrentar en las computadoras paralelas tradicionales, que es el desbalance dado por las capacidades de cálculo de los procesadores que son diferentes. El hardware de interconexión Ethernet plantea problemas quizás más serios:

- Ethernet no está orientado *a priori* a cómputo paralelo y por lo tanto índices de rendimiento tales como latencia y ancho de banda son bastante mayores que los de las redes de interconexión de las computadoras paralelas. Expresado de otra manera, el rendimiento de las redes de comunicación Ethernet no está *balanceado* de acuerdo con la capacidad de procesamiento de las computadoras.
- El método de acceso al "único" medio de comunicaciones definido por el estándar, CSMA/CD (Carrier Sense-Multiple Access/Collision Detect), hace que el rendimiento de la red de interconexión sea altamente dependiente del tráfico y del cableado (con la utilización de switches, por ejemplo).

Por lo tanto, es necesaria una revisión bastante exhaustiva de los algoritmos paralelos para identificar problemas y soluciones en el contexto de este *nuevo* hardware paralelo proporcionado por las redes de computadoras heterogéneas. En ningún caso se debe perder de vista que la razón de ser del procesamiento paralelo es el aumento del rendimiento con respecto al proporcionado por el procesamiento secuencial. La revisión de los algoritmos paralelos tiende a ser caso por caso al menos en términos de áreas de aplicaciones o de problemas a ser resueltos utilizando procesamiento paralelo.

Las aplicaciones de álgebra lineal constituyen una de las grandes áreas de problemas que tradicionalmente han sido resueltos aprovechando el rendimiento que proporcionan las arquitecturas de cómputo paralelo disponibles. Dentro de las aplicaciones del álgebra lineal se han identificado un conjunto de operaciones o directamente rutinas de cómputo que se han considerado básicas y de utilización extensiva en la mayoría de los problemas incluidos dentro de esta área. Tales rutinas se han denominado BLAS (Basic Linear Algebra Subroutines) y tanto para su clasificación como para la identificación de requerimientos de cómputo y de memoria de cada una de ellas se las divide en tres niveles: nivel 1, nivel 2 y nivel 3 (Level 1 o L1 BLAS, Level 2 o L2 BLAS y Level 3 o L3 BLAS). Desde el punto de vista del rendimiento, las rutinas de nivel 3 (L3 BLAS) son las que se deben optimizar para obtener rendimiento cercano al óptimo de cada máquina y de hecho, muchas empresas de microprocesadores estándares proveen bibliotecas BLAS con marcado énfasis en la optimización y el consiguiente rendimiento de las rutinas incluidas en BLAS de nivel 3.

La multiplicación de matrices puede considerarse el pilar o la rutina a partir de la cual todas las demás incluidas en BLAS de nivel 3 se pueden definir. Quizás por esta razón y/o por su simplicidad la mayoría de los reportes de investigación en esta área de procesamiento paralelo comienza por el "problema" de la multiplicación de matrices en paralelo. Expresado de otra manera, al optimizar la multiplicación de matrices de alguna manera se optimiza todo el nivel 3 de BLAS y por lo tanto se tendrían optimizadas la mayoría de las aplicaciones basadas en álgebra lineal y que dependen de la optimización de

las rutinas que llevan acabo las operaciones provenientes del álgebra lineal. Aunque esta optimización no sea necesariamente directa, sí se puede afirmar que el tipo procesamiento que se debe aplicar para resolver la multiplicación de matrices es muy similar al del resto de las rutinas definidas como BLAS de nivel 3 e incluso muy similar también a los problemas más específicos que se resuelven recurriendo a operaciones del álgebra lineal. En este sentido, es muy probable que lo que se haga para optimizar la multiplicación de matrices (en paralelo o no) sea utilizable y/o aprovechable en otras operaciones. Es por esta razón que la orientación de toda esta tesis ha sido a la multiplicación de matrices en paralelo con algunos comentarios hacia la generalización.

Enfocando específicamente la multiplicación de matrices en paralelo, al analizar los algoritmos propuestos hasta ahora se llega a la conclusión de que son bastante orientados hacia las computadoras paralelas tradicionales. De hecho, cada algoritmo paralelo de multiplicación de matrices se puede identificar como especialmente apropiado para las computadoras paralelas de memoria compartida (denominados multiprocesadores) o para las computadoras paralelas con memoria distribuida o de pasaje de mensajes (denominados multicomputadoras).

Los algoritmos paralelos orientados a multiprocesadores no son apropiados para los sistemas de cómputo paralelo de memoria distribuida. Más aún, las redes de computadoras en general (instalaciones Beowulf, sistemas heterogéneos, etc.) son especialmente inapropiadas dado el bajo nivel de acoplamiento o más específicamente la distribución y separación del hardware disponible para cómputo paralelo.

Los algoritmos paralelos orientados a multicomputadoras siguen teniendo una base muy cercana al hardware de las computadoras paralelas tradicionales. Más específicamente, al proponer estos algoritmos se han asumido varias características subyacentes del hardware paralelo tales como:

- Interconexión de los procesadores en forma de malla o toro bidimensional, arreglos de árboles o hipercubos. Esto significa tener la posibilidad de múltiples conexiones punto a punto y múltiples caminos opcionales de la red de interconexión para la transferencia de datos entre dos procesadores.
- Elementos de procesamiento homogéneos. Esto implica que el balance de carga es trivial y directamente dado por la distribución de la misma cantidad de datos de las matrices involucradas a todos los procesadores.

Ninguna de las dos características anteriores es posible de mantener en las redes locales de computadoras heterogéneas. Por lo tanto, es necesario desarrollar algoritmos que hagan uso eficiente de las características de estas *nuevas* arquitecturas paralelas. Por un lado, estos algoritmos deben estar preparados para las diferencias de capacidad de cómputo de las máquinas interconectadas por las redes locales y por el otro deben aprovechar al máximo el rendimiento y las características de las redes de interconexión Ethernet. Y estas son las dos bases sobre las que se apoyan los algoritmos paralelos de multiplicación de matrices propuestos en esta tesis (de hecho, se pueden considerar como dos variantes de un mismo algoritmo paralelo):

- Balance de carga dado por la distribución de datos que a su vez se hace de acuerdo con la capacidad de cálculo relativa de cada computadora.
- Comunicaciones de tipo broadcast únicamente, de tal manera que se aprovecha al

máximo la capacidad de las redes Ethernet.
- Distribución de los datos de manera unidimensional, siguiendo casi de manera unívoca el propio hardware de interconexión física definido por el estándar Ethernet.

Tal como se muestra en el capítulo de experimentación, el sólo hecho de proponer un algoritmo "apropiado" no garantiza obtener rendimiento aceptable ni escalable. De hecho, tal cual lo muestran los resultados de los experimentos, al utilizar la biblioteca de comunicaciones PVM se tiene que agregando máquinas para llevar a cabo cómputo paralelo (aumentar la capacidad de cálculo) y resolver el mismo problema, el rendimiento se reduce. Más aún, dependiendo de las computadoras esta reducción del rendimiento puede ser drástica a punto tal que se tiene peor rendimiento que con una única computadora. En este caso, el tiempo de cómputo paralelo termina dominado por el tiempo necesario para los mensajes broadcast.

Dado que el rendimiento del cómputo local es satisfactorio, es necesario mejorar el rendimiento de los mensajes broadcast para tener rendimiento aceptable para este problema. Tal como se ha discutido, es muy difícil asegurar *a priori* que la implementación de los mensajes broadcast propuestos por las bibliotecas de "propósito general" tales como PVM y las implementaciones de MPI se hagan específicamente para aprovechar la capacidad de broadcast de las redes Ethernet. Por lo tanto, se propone una *nueva* rutina de mensajes broadcast entre procesos basada directamente en el protocolo UDP que es tanto o más utilizado que las propias redes Ethernet. Aunque esta rutina de mensajes broadcast sea extendida a toda una biblioteca de comunicaciones colectivas, en principio el propósito no es definir una *nueva* biblioteca ni reemplazar a las bibliotecas existentes. Sí es necesario el aprovechamiento máximo de las redes Ethernet y por lo tanto la implementación de las rutinas más utilizadas y/o de las que depende el rendimiento se deberían adecuar a las características y capacidades de estas redes de interconexión de computadoras.

El rendimiento de las multiplicaciones de matrices en paralelo es aceptable en las redes locales heterogéneas cuando el algoritmo propuesto específicamente para éstas se implementa utilizando una rutina de broadcast que aprovecha las capacidades de las redes Ethernet. Por lo tanto, y como era de esperar, al menos dos aspectos se combinan desde el punto de vista del rendimiento para obtener el máximo de las redes locales instaladas que se pueden utilizar para cómputo paralelo: algoritmo e implementación en general, y en particular la implementación de los mensajes broadcast. Expresado de otra manera, sin un algoritmo apropiado no se puede obtener buen rendimiento, y aún con un algoritmo apropiado el rendimiento no es satisfactorio si la implementación es inadecuada. En este caso, la parte más problemática de la implementación ha sido la de los mensajes broadcast. Dado que ninguna biblioteca de pasaje de mensajes implementa de manera apropiada los mensajes broadcast o por lo menos no se puede asegurar *a priori* que lo haga, se ha desarrollado una rutina específica que resuelve el problema de rendimiento. Una vez más, se debe recordar que el rendimiento es la razón de ser del procesamiento paralelo en general o como mínimo del procesamiento paralelo que se utiliza para resolver los problemas numéricos en general y las operaciones de álgebra lineal en particular.

La red del LIDI en particular muestra que los algoritmos propuestos también son apropiados para las instalaciones del tipo Beowulf, y/o con hardware de procesamiento homogéneo y red de interconexión con mejor rendimiento que el de las redes locales

instaladas. En el caso de las computadoras paralelas *tradicionales* la utilización no es tan inmediata o incondicional. En los multiprocesadores, *a priori* parece innecesaria la utilización de un "nuevo" algoritmo paralelo dado que los propuestos son muy adecuados o por lo menos más adecuados que cualquiera propuesto para multicomputadoras. En el caso de las multicomputadoras se debe ser muy cuidadoso sobre todo en la implementación y rendimiento de los mensajes broadcast. En este sentido, las redes de interconexión estáticas (con enlaces punto a punto limitados y predefinidos) suelen imponer ciertos límites a la escalabilidad y consecuente rendimiento de los mensajes broadcast. En todo caso, los múltiples esfuerzos de investigación en la dirección de mejorar el rendimiento de las comunicaciones colectivas y de los mensajes broadcast en particular en estas computadoras es aprovechable por los algoritmos propuestos para multiplicar matrices en paralelo.

La experimentación que se llevó a cabo para comparar los algoritmos propuestos para la multiplicación y factorización LU de matrices con respecto a los que implementa ScaLAPACK fue muy satisfactoria. De hecho, la ganancia mínima de los algoritmos propuestos es mayor al 20% del rendimiento obtenido con ScaLAPACK, es decir que con el mismo hardware y el mismo código de cómputo local totalmente optimizado, los algoritmos propuestos obtienen en todos los casos más del 20% mejor rendimiento que ScaLAPACK. Se debe recordar que se considera a ScaLAPACK como una de las bibliotecas que implementa los mejores algoritmos paralelos para el área de álgebra lineal, tanto en rendimiento paralelo como en escalabilidad.

Desde el punto de vista del problema de multiplicación de matrices resuelto (y aún considerando el problema de factorización LU de matrices) se deben tener en cuenta dos consideraciones muy importantes:

- El problema no es significativo en sí mismo, dado que es muy poco frecuente que todo lo que se necesita resolver sea una multiplicación de matrices. Normalmente la multiplicación de matrices es parte de o se utiliza entre otras operaciones para resolver un problema en general.
- Tal como se ha explicado al principio, la multiplicación de matrices es representativa en cuanto a procesamiento de todo el nivel 3 de BLAS y por lo tanto lo que se obtiene con la multiplicación de matrices se puede utilizar en todas las rutinas incluidas en el nivel 3 de BLAS. Dado que en general lo más importante en cuanto a rendimiento se relaciona con estas rutinas (L3 BLAS), la optimización de la multiplicación de matrices se transforma en un aporte significativo para todas o la mayoría de las aplicaciones de álgebra lineal. Esta ha sido la tendencia en general, sea en procesadores secuenciales, computadoras paralelas en general y multiprocesadores, multicomputadoras y/o redes locales de computadoras en particular.

Dado que

- la comunicación entre procesos se resuelve siempre con mensajes broadcast
- la implementación de estos mensajes se hizo aprovechando las capacidades de las redes Ethernet

el rendimiento es escalable al menos hasta el límite dado por la granularidad mínima. De todas maneras, no se debe olvidar que esta granularidad mínima es bastante grande en el caso de las redes locales y dependiente del hardware de comunicaciones (10 Mb/s, 100 Mb/s, 1 Gb/s, etc.).

Desde otro punto de vista, la misma rutina de mensajes broadcast basada en UDP que se ha implementado muestra que no necesariamente se debe trasladar la heterogeneidad del hardware de cómputo de las redes locales al rendimiento de las comunicaciones. Más específicamente:

- El ancho de banda asintótico y/o el tiempo de transferencia de mensajes relativamente grandes es independiente de las computadoras involucradas y depende de la capacidad de las redes de comunicaciones.
- El tiempo de latencia de las comunicaciones es dependiente de la capacidad de cómputo de las máquinas involucradas en una transferencia de datos.

Tal como se ha mostrado tanto en la experimentación con la multiplicación de matrices misma, como en el Apéndice C específicamente para los mensajes punto a punto; las bibliotecas de comunicaciones de propósito general tales como PVM y las implementaciones de uso libre de MPI tienen la tendencia a hacer el rendimiento de las comunicaciones dependiente de la heterogeneidad de las computadoras. Esto se debe a las capas de software que deben agregar para resolver las múltiples rutinas de comunicaciones entre procesos que normalmente implementan y que implican una sobrecarga (overhead) considerable de procesamiento.

Más aún, la misma rutina de mensajes broadcast muestra que es posible tener un mensaje broadcast entre procesos de usuario que cumpla con:

- Rendimiento cercano al óptimo absoluto proporcionado por el hardware de comunicaciones. Aunque la rutina está destinada a los mensajes broadcast el rendimiento de las comunicaciones punto a punto (entre dos computadoras) también es altamente satisfactorio.
- Rendimiento escalable, es decir que el tiempo de broadcast es *casi* independiente de la cantidad de máquinas involucradas. Evidentemente la sincronización y la forma utilizada para confirmación de la llegada de los mensajes a cada computadora implican la existencia de un costo por computadora que interviene en un mensaje broadcast, pero este costo es mucho menor en tiempo de ejecución que la replicación completa de todo el mensaje a cada proceso (máquina) receptor.
- Portabilidad, dado que los únicos requisitos para que esta rutina se pueda utilizar son la conectividad IP (TCP y UDP) y un compilador del lenguaje C. De hecho, al utilizar protocolos estándares y utilizados extensivamente hasta se logra independencia del hardware de comunicaciones. Aunque inicialmente orientada al aprovechamiento de las redes Ethernet, la rutina para llevar a cabo mensajes broadcast es portable a cualquier ambiente con conectividad IP. Aunque no se han hecho pruebas específicas, es bastante probable que en redes de interconexión de computadoras que no tienen la posibilidad de broadcast de datos en el hardware (tales como las redes ATM), el rendimiento de esta rutina de todas maneras sea satisfactorio.
- Ningún requisito adicional desde el punto de vista de un usuario de las redes locales de computadoras que se utilizan para cómputo paralelo. En particular, no son necesarias alteraciones del sistema operativo ni prioridades o procesos con prioridades más allá de las disponibles para los procesos de usuario.
- Sencillez de utilización. De hecho, en los programas con los cuales se realizó la experimentación los cambios a nivel de código fuente no fueron mucho más allá de el reemplazo de la rutina de PVM utilizada para los mensajes broadcast. Todo el resto de las comunicaciones (que no tienen influencia sobre el rendimiento o su influencia es mínima) se continuaron haciendo con rutinas provistas por PVM.

- Manejo de la heterogeneidad en la representación de los datos de las computadoras que normalmente están interconectadas en una red local. En la misma experimentación se utilizaron diferentes tipos de máquinas con diferentes procesadores y sus propias representaciones de los tipos de datos numéricos.
- Interfase de utilización común a la de las demás bibliotecas de pasaje de mensajes de propósito general. De hecho, la implementación de esta rutina hace suponer que las demás rutinas que comúnmente se incluyen dentro de las comunicaciones colectivas es relativamente simple como para tener una biblioteca completa de comunicaciones colectivas.

El algoritmo que resuelve las multiplicaciones de matrices en paralelo con los períodos de procesamiento local y de mensajes broadcast de manera secuencial es simple y confiable en cuanto a estimación de rendimiento. En este sentido, se tiene un modelo de máquina paralela que es capaz de

- Ejecutar simultáneamente en cada procesador tal como cualquier otra perteneciente a la clase MIMD de memoria distribuida. No hay ningún tipo de interferencia entre distintos procesadores (máquinas) para resolver cómputo local.
- Llevar a cabo mensajes broadcast de manera relativamente independiente de la cantidad de máquinas involucradas.
- La interferencia de los mensajes sobre el rendimiento de cómputo local es poco significativa.
- No hay interferencia del cómputo local sobre el rendimiento de las comunicaciones.

Y por otro lado, se tiene un algoritmo de cómputo paralelo que además de aprovechar todas estas características involucra un tipo de procesamiento que es altamente regular. Si bien no se puede asegurar que todos los problemas numéricos tienen procesamiento tan regular, sí se puede afirmar que es una característica similar de una gran parte de las rutinas y aplicaciones provenientes del álgebra lineal. La combinación de este modelo de máquina con este tipo de algoritmos paralelos hace muy sencilla y relativamente confiable la estimación de rendimiento que se puede obtener. Quizás como una consecuencia de esto, también es posible identificar con bastante claridad cuándo comienzan los problemas de rendimiento debido a la granularidad de los problemas que se resuelven. Por lo tanto, el propio programa de multiplicación de matrices en paralelo con los períodos de cómputo y comunicaciones ejecutados u organizados de manera secuencial puede ser utilizado como un benchmark para la identificación de la granularidad mínima de un conjunto de computadoras interconectadas en una red local.

Más allá de obtener mejor rendimiento, el algoritmo de multiplicación de matrices en paralelo que está diseñado para solapar cómputo con comunicaciones es particularmente útil para identificar problemas de rendimiento. Específicamente, con la implementación de este algoritmo es posible identificar con claridad las computadoras que efectivamente pueden solapar cómputo con comunicaciones y hasta cuál es la penalización en términos de rendimiento que esto produce. En este sentido, en los ambientes heterogéneos se pueden tener distintas penalizaciones en diferentes máquinas y con la cuantificación de esta penalización se puede mejorar el balance de carga para compensar las diferencias. Una herramienta de este tipo se torna valiosa cuando funciona en múltiples computadoras y proporciona información que es muy difícil de obtener por otros medios.

Las computadoras involucradas en una red local quizás son las que tienen menor capacidad

tanto en procesamiento como en memoria principal instalada entre todas las disponibles en el mercado. En este sentido, la ganancia obtenida por el uso de una red local procesando en paralelo puede ser muy grande. La experimentación que se hizo involucrando problemas que iban más allá de la capacidad de almacenamiento de la mejor computadora de cada red local intenta cuantificar esta ganancia. La idea básica en esta dirección es: aún utilizando la mejor computadora de una red local pueden haber problemas de rendimiento dado que no es suficiente para resolver un problema dado, principalmente por la cantidad de memoria disponible en esa computadora. Si bien gracias al manejo de memoria swap es posible almacenar una gran cantidad de datos más allá de la memoria instalada, el rendimiento puede sufrir una gran penalización. Por lo tanto, la utilización de las demás computadoras de la red local no solamente proveen memoria para almacenar datos sino que también permiten que *todas* las computadoras lleven a cabo su procesamiento a la máxima velocidad. Es decir que en todas las computadoras se pueden aprovechar los recursos disponibles de manera óptima u optimizada.

Específicamente en términos del valor de speedup como métrica de rendimiento se ha mostrado algo que es relativamente sencillo pero muy poco frecuente en cuanto a los reportes de investigación: el rendimiento en los ambientes heterogéneos no está directamente relacionado con la *cantidad* de computadoras (o procesadores) que se utilizan. En este sentido, las máquinas paralelas tradicionales, con su hardware de procesamiento homogéneo ha establecido que el máximo valor de speedup posible de obtener es igual a la cantidad de procesadores que se utilizan. En los ambientes heterogéneos esto queda sin sustento dado que los procesadores no necesariamente tienen la misma capacidad de cálculo. De hecho, la recta $y = x$ con la que se ha relacionado tradicionalmente el máximo valor de speedup ha permitido la interpolación de valores intermedios y esta interpolación de valores intermedios también queda sin sustento en los ambientes de procesamiento paralelo con procesadores heterogéneos.

Una de las bases para obtener rendimiento satisfactorio y *predecible* con procesamiento paralelo es la utilización del mejor código secuencial para cómputo local. Además, si se utiliza código de cómputo local no optimizado se llega a que la estimación de rendimiento dada por el factor de speedup pierde casi todo su significado, dado que el rendimiento paralelo se obtiene como una combinación de

- Rendimiento local de cada computadora.
- Cantidad de operaciones que se pueden realizar simultáneamente.
- Rendimiento de las comunicaciones.

Se muestra con bastante detalle en el Apéndice B que cuando se utiliza código no optimizado el rendimiento de cada computadora es altamente dependiente del tamaño del problema, básicamente por la relación que existe entre la cantidad de datos a procesar y la capacidad de la memoria cache de los procesadores (específicamente de la memoria cache de primer nivel). En general, en todas las arquitecturas paralelas de memoria distribuida y en el caso particular de las redes locales de computadoras que se utilizan para cómputo paralelo, aumentar la cantidad de procesadores implica que cada procesador tiene problemas cada vez menores en cuanto a cantidad de datos a procesar. Esta menor cantidad de datos tiene una mayor probabilidad de aprovechar mejor el espacio de memoria cache y por lo tanto el rendimiento de cómputo se mejora notablemente. Así se llega a que cuando las rutinas de de cómputo local no son optimizadas el rendimiento paralelo no

necesariamente mejora por utilizar más computadoras sino porque cada computadora resuelve un problema con menor cantidad de datos y por lo tanto el rendimiento de cómputo local es significativamente mayor. Por otro lado, el código de cómputo totalmente optimizado hace que el rendimiento sea relativamente independiente del tamaño de problema que se resuelve y por lo tanto toda ganancia obtenida por cómputo paralelo es

- "Real", dado que no hay otra forma de obtener mejor rendimiento de las computadoras secuenciales que se utilizan porque el rendimiento secuencial con el que se compara es el óptimo.
- Debida únicamente a la utilización de mayor cantidad de computadoras o procesadores, ya que el tamaño del problema no influye significativamente en el rendimiento local de cada máquina.

6.2 Resumen de Aportes y Publicaciones Relacionadas con Esta Tesis

Se pueden enumerar de manera resumida los aportes de esta tesis también relacionados con las publicaciones que se han hecho al respecto. Inicialmente, se debe identificar con cierta precisión el problema básico de rendimiento paralelo en las redes locales de computadoras y clusters y proponer algún tipo de solución. Estos dos aportes iniciales puede ser resumidos como:

1. **Análisis de los algoritmos de multiplicación de matrices en paralelo para su utilización en redes locales de computadoras que se pueden aprovechar para cómputo paralelo.**
2. **Propuesta de los principios de paralelización que se utilizaron para diseñar los algoritmos propuestos en esta tesis.**

Y estos aportes están directamente relacionados con las publicaciones:

- [135] Tinetti F., A. Quijano, A. De Giusti, "Heterogeneous Networks of Workstations and SPMD Scientific Computing", 1999 International Conference on Parallel Processing, The University of Aizu, Aizu-Wakamatsu, Fukushima, Japan, September 21 - 24, 1999, pp. 338-342.
- [137] Tinetti F., Sager G., Rexachs D., Luque E., "Cómputo Paralelo en Estaciones de Trabajo no Dedicadas", VI Congreso Argentino de Ciencias de la Computación, Ushuaia, Argentina, Octubre de 2000, Tomo II, pp. 1121-1132.

Donde se presenta experimentación específicamente orientada a mostrar que los algoritmos tradicionales no necesariamente son útiles en las redes locales de computadoras. Por otro lado, las publicaciones (en orden cronológico):

- [116] Tinetti F., "Aplicaciones Paralelas de Cómputo Intensivo en NOW Heterogéneas", Workshop de Investigadores en Ciencias de la Computación (WICC 99), San Juan, Argentina, 27 y 28 de Mayo de 1999, pp. 17-20.
- [117] Tinetti F., "Performance of Scientific Processing in Networks of Workstations", Workshop de Investigadores en Ciencias de la Computación (WICC 2000), La Plata, Argentina, 22 y 23 de Mayo de 2000, pp. 10-12.
- [124] Tinetti F., Barbieri A., Denham M., "Algoritmos Paralelos para Aprovechar Redes

Locales Instaladas", Workshop de Investigadores en Ciencias de la Computación (WICC 2002), Bahía Blanca, Argentina, 17-18 de Mayo de 2002, pp. 399-401.
- [128] Tinetti F., Denham M., "Algebra Lineal en Clusters Basados en Redes Ethernet", Workshop de Investigadores en Ciencias de la Computación (WICC 2003), Tandil, Argentina, 22-23 de Mayo de 2003, pp. 575-579.
- [134] Tinetti F., Quijano A., "Costos del Cómputo Paralelo en Clusters Heterogéneos", Workshop de Investigadores en Ciencias de la Computación (WICC 2003), Tandil, Argentina, 22-23 de Mayo de 2003, pp. 580-584.

Están más orientadas a presentar las ideas como líneas de investigación abiertas y/o en desarrollo. Se debe notar que cada uno de los años en que se ha participado en este congreso se han reportado los avances de la línea de investigación con respecto al año anterior.

Una vez identificados los inconvenientes y alguna propuesta de solución en general, es necesario probar la propuesta. La alternativa elegida ha sido hacerlo de forma específica en el área de las aplicaciones de álgebra lineal y de las operaciones básicas. En este contexto se aporta en esta tesis:

3. **Propuesta de algoritmos específicos de multiplicación de matrices en clusters, diseñados siguiendo los principios de paralelización mencionados antes.**
4. **Utilización del algoritmo de multiplicación de matrices en paralelo que está diseñado para solapar cómputo con comunicaciones para identificar problemas de rendimiento (como *benchmark*, en cierta forma).**

Estos algoritmos han sido presentados junto con experimentación que avala su validez en las publicaciones:
- [118] Tinetti F., "Performance of Scientific Processing in NOW: Matrix Multiplication Example", JCS&T, Journal of Computer Science & Technology, Special Issue on Computer Science Research, Vol. 1 No. 4, March 2001, pp. 78-87.
- [131] Tinetti F., Luque E., "Parallel Matrix Multiplication on Heterogeneous Networks of Workstations", Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 15 al 18 de Octubre de 2002, p. 122.
- [132] Tinetti F., Luque E., "Efficient Broadcasts and Simple Algorithms for Parallel Linear Algebra Computing in Clusters", Workshop on Communication Architecture for Clusters, International Parallel and Distributed Processing Symposium (IPDPS '03), Nice Acropolis Convention Center, Nice, France April 22-26, 2003.
- [136] Tinetti F., A. Quijano, A. De Giusti, E. Luque, "Heterogeneous Networks of Workstations and the Parallel Matrix Multiplication", Recent Advances in Parallel Virtual Machine and Message Passing Interface, 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23-26, 2001, Proceedings, Yannis Cotronis, Jack Dongarra (Eds.), Lecture Notes in Computer Science 2131 Springer 2001, ISBN 3-540-42609-4, pp. 296-303.

En muchas de las publicaciones anteriores también se presenta otro de los aportes de esta tesis, específicamente orientado al aprovechamiento de las redes Ethernet, aporte que se puede resumir en:

5. Propuesta de una rutina de mensajes broadcast basada en el protocolo UDP para optimizar la utilización de las redes Ethernet.

En este caso, las publicaciones más específicamente relacionadas son:

- [120] Tinetti F., Barbieri A., "Collective Communications for Parallel Processing in Networks of Workstations", Proceedings SCI 2001, Volume XIV, Computer Science and Engineering: Part II, Nagib Callaos, Fernando G. Tinetti, Jean Marc Champarnaud, Jong Kun Lee, Editors, International Institute of Informatics and Systemics, Orlando, Florida, USA, ISBN 980-07-7554-4, July 2001, pp. 285-289.
- [123] Tinetti F., Barbieri A., "An Efficient Implementation for Broadcasting Data in Parallel Applications over Ethernet Clusters", Proceedings of the 17th International Conference on Advanced Information Networking and Applications (AINA 2003), IEEE Press, ISBN 0-7695-1906-7, March 2003.

También en esta tesis se tratan aspectos de rendimiento de cómputo paralelo en clusters homogéneos que se pueden resumir como:

6. Propuesta de algoritmos específicos de multiplicación de matrices y factorización LU de matrices en clusters homogéneos, diseñados siguiendo los principios de paralelización mencionados antes. En realidad, el algoritmo de multiplicación de matrices es el mismo que el presentado para los clusters heterogéneos, mostrando de esta manera su utilización directa en clusters homogéneos.

Estos algoritmos en el contexto de los clusters homogéneos se presentaron junto con experimentación específica y/o de comparación con ScaLAPACK en algunas de las publicaciones anteriores y en:

- [127] Tinetti F., Denham M., "Paralelización de la Factorización de Matrices en Clusters", Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 15 al 18 de Octubre de 2002, p. 121.
- [130] Tinetti F., Denham M., De Giusti A., "Parallel Matrix Multiplication and LU Factorization on Ethernet-based Clusters", High Performance Computing. 5th International Symposium, ISHPC 2003, Tokyo-Odaiba, Japan, October 20-22, 2003, Proceedings. Series: Lecture Notes in Computer Science, Vol. 2858. Veidenbaum, A.; Joe, K.; Amano, H.; Aiso, H. (Eds.), 2003, XV, 566 p. ISBN: 3-540-20359-1
- [129] Tinetti F., Denham M., "Paralelización de la Factorización LU de Matrices en Clusters Heterogéneos", Proceedings IX Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Informática, Universidad Nacional de La Plata, La Plata, Argentina, 6 al 10 de Octubre de 2003, p. 385-396.

Donde la última publicación muestra los primeros resultados obtenidos al utilizar los principios de paralelización para la factorización LU de matrices en clusters heterogéneos.

Aunque la evaluación de las comunicaciones es bastante conocida, en esta tesis tiene especial relevancia dado que se ha mostrado la penalización excesiva que puede llegar a

imponer sobre algoritmos paralelos específicamente diseñados para la obtención de rendimiento optimizado. También se presenta en el Apéndice C toda la metodología y los resultados obtenidos en términos de

7. **Evaluación de rendimiento de las comunicaciones desde la perspectiva de cómputo paralelo en clusters heterogéneos (operaciones punto a punto y colectivas).**

Que se ha reflejado en las publicaciones:

- [119] Tinetti F., Barbieri A., "Cómputo y Comunicación: Definición y Rendimiento en Redes de Estaciones de Trabajo", Workshop de Investigadores en Ciencias de la Computación (WICC 2001), San Luis, Argentina, 22-24 de Mayo de 2001, pp. 45-48.
- [121] Tinetti F., Barbieri A., "Análisis del Rendimiento de las Comunicaciones sobre NOWs", Proceedings VII Congreso Argentino de Ciencias de la Computación (CACIC), El Calafate, Santa Cruz, Argentina, 16 al 20 de Octubre de 2001, pp. 654-656.
- [122] Tinetti F., Barbieri A., "Cómputo Paralelo en Clusters: Herramienta de Evaluación de Rendimiento de las Comunicaciones", Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 15 al 18 de Octubre de 2002, p. 123.
- [125] Tinetti F., D' Alessandro A., Quijano A., "Communication Performance of Installed Networks of Workstations for Parallel Processing", Proceedings SCI 2001, Volume XIV, Computer Science and Engineering: Part II, Nagib Callaos, Fernando G. Tinetti, Jean Marc Champarnaud, Jong Kun Lee, Editors, International Institute of Informatics and Systemics, Orlando, Florida, USA, ISBN 980-07-7554-4 July 2001, pp. 290-294.
- [133] Tinetti F., Quijano A., "Capacidad de Comunicaciones Disponible para Cómputo Paralelo en Redes Locales Instaladas", Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 15 al 18 de Octubre de 2002, p. 125.

Aunque no directamente relacionado con el contexto de las redes locales instaladas se han llevado a cabo algunos estudios relacionados con el rendimiento de la multiplicación de matrices en supercomputadoras o al menos computadoras paralelas tradicionales, que se ha publicado en

- [126] Tinetti F., Denham M., "Paralelización y Speedup Superlineal en Supercomputadoras. Ejemplo con Multiplicación de Matrices", Proceedings VII Congreso Argentino de Ciencias de la Computación (CACIC), El Calafate, Santa Cruz, Argentina, 16 al 20 de Octubre de 2001, pp. 765-774.

Muchas de las conclusiones a las que se llega en esta publicación tiene relación directa con el Apéndice B, dedicado a mostrar el rendimiento secuencial de las computadoras utilizadas. En particular, la noción distorsionada de rendimiento que se puede llegar a obtener cuando el código de los programas utilizados no son optimizados específicamente para la aplicación resuelta y la arquitectura de cómputo utilizadas.

6.3 Trabajo Futuro

Como se explica antes, el problema de la multiplicación de matrices no es significativo en sí mismo sino representativo de un conjunto de problemas de procesamiento numérico de datos. En el contexto de las rutinas BLAS de nivel 3, la extensión inmediata es:

- Utilizar directamente la multiplicación de matrices para la implementación de todas las rutinas incluidas en BLAS de nivel 3.
- Utilizar los principios de paralelización utilizados en la multiplicación de matrices para resolver las demás rutinas de BLAS nivel 3.

La primera opción tiene la ventaja de no tener mucho más costo que codificar las rutinas en términos de multiplicaciones de matrices. Aunque la segunda opción no tiene la ventaja anterior sino que involucra el costo asociado de paralelización caso por caso, tiene la ventaja de permitir un rango más amplio de posible ganancia debida al cómputo paralelo. Tal cual están definidas, las rutinas BLAS de nivel 3 son una cantidad bastante reducida y en la paralelización caso por caso se pueden aprovechar mejor las características propias de procesamiento para obtener mejor rendimiento. Cualquiera sea la alternativa elegida, o incluso en la experimentación con ambas, el aprovechamiento de las ideas o principios de paralelización de esta tesis es bastante directo.

Como un paso un poco mayor en cuanto a la extensión de esta tesis es directamente atacar un problema completo proveniente del álgebra lineal. Como un ejemplo se puede mencionar el mismo método de factorización LU presentado en el Capítulo 5, que se puede utilizar para la resolución del problema de sistemas de ecuaciones lineales. En un contexto un poco más general se podría avanzar en la dirección de los problemas que resuelve la biblioteca LAPACK como para experimentar con una gama relativamente amplia de problemas provenientes del álgebra lineal. La ventaja asociada a la experimentación con LAPACK es que la biblioteca misma ha sido utilizada hasta el momento y por lo tanto existe una cantidad relativamente grande de usuarios potenciales. Se puede afirmar que hasta este punto, es decir manteniéndose en operaciones y aplicaciones de álgebra lineal el tipo de procesamiento, es bastante similar respecto del procesamiento de la multiplicación de matrices. Si bien existen muchas particularidades, la gran mayoría de las operaciones son:

- Bastante sencillas en cuanto a codificación.
- Muy conocidas en cuanto a métodos de solución.
- Con un alcance muy bien definido de dependencia de datos y también con subconjuntos de datos que se pueden calcular de manera independiente.

Esta extensión del trabajo está avalada por el hecho de haber encontrado que la paralelización de operaciones como la multiplicación y la factorización de matrices con los principios relativamente sencillos de esta tesis proporciona código optimizado para las redes locales interconectadas por Ethernet. De hecho, la experimentación que se llevó a cabo con el objetivo de comparar los algoritmos propuestos con los implementados por ScaLAPACK avalan esta línea de investigación futura.

El siguiente nivel de extensiones, un poco más complejo, lo representan las aplicaciones numéricas en general y en particular todo lo que involucra procesamiento no lineal. Es más

complejo desde dos puntos de vista:

- Codificación de los métodos de solución de problemas específicos.
- Relaciones de dependencia de cálculos, que no son tan estructuradas como en la mayoría de las operaciones de álgebra lineal.

Un área específica es la de procesamiento de señales, que tiene múltiples aplicaciones y donde los métodos de solución a los problemas específicos son numerosos y muchas veces muy dispares entre sí. El cálculo conocido y relativamente simple en este contexto de una FFT (Fast Fourier Transform) involucra por ejemplo un patrón de acceso a datos que es en cierta forma regular pero tan específico que ha dado lugar directamente a modos de direccionamiento de datos *ad hoc* en los procesadores diseñados para procesamiento de señales digitales o DSP (Digital Signal Processor). Si bien los principios de paralelización en esta área son los mismos, dado que están pensados para el aprovechamiento optimizado de los recursos de cómputo de las redes locales y no para un área de procesamiento en particular, la aplicación de estos principios no es tan sencilla como en el caso de la multiplicación de matrices o las demás operaciones o rutinas relacionadas con álgebra lineal.

En otro eje de investigación, siempre es posible pensar extensiones o al menos experimentar con la posibilidad de utilización de más de una red local. En este sentido, y para las aplicaciones de álgebra lineal con sus características de procesamiento fuertemente acoplado, es muy importante hasta qué punto es posible la ganancia de rendimiento con la utilización de más de una red local. Más específicamente, la cuantificación de la penalización (por ejemplo en cuanto a granularidad mínima) por la distribución de los datos en múltiples redes locales es útil para caracterizar *a priori* la utilidad de uso de más de una red local para solucionar un problema en paralelo.

En el caso de múltiples redes locales también puede ser significativo el aporte de otros métodos simples pero efectivos de procesamiento tales como el *pipelining* (similar a una línea de ensamblaje tradicional) o el establecimiento de “servidores” específicos para tareas especialmente penalizadas por la distribución física de las computadoras que se utilizan. Se debe tener especial cuidado en este contexto con todas las comunicaciones que sean *remotas* en el sentido de transferir datos entre dos o más computadoras que pertenecen a distintas redes locales. El caso específico de los mensajes broadcast, por ejemplo, sigue siendo sumamente útil y sencillo en una red local y en todas las redes locales que se utilizan, pero la implementación de estos mensajes cuando están involucradas varias computadoras de distintas redes locales se debe pensar con mucho cuidado en cuanto a tráfico, congestión (competencias) de los enlaces intermedios de transporte entre redes, tiempo de latencia, etc. La estrategia a seguir ya no es tan inmediata, aunque el rendimiento tan satisfactorio que se puede obtener en cada red local en cierta forma favorece esta línea de investigación.

La utilización de las tres redes locales sobre las que se llevó a cabo toda la experimentación sigue siendo posible y podrían diseñarse un conjunto de experimentos para analizar los resultados y a partir de allí proponer alternativas de aprovechamiento de cada una de las computadoras. De alguna manera, la posibilidad de utilizar más de una red local aumenta considerablemente el rango de tamaños de problemas que se pueden resolver (independientemente de que el problema sea multiplicar matrices o cualquier otro) pero también agrega problemas bastante “desconocidos” al menos en este contexto de las

aplicaciones de álgebra lineal tales como el impacto sobre la granularidad mínima y la escalabilidad ahora a nivel de redes locales. Otro de los problemas en este contexto es el de rendimiento vs. capacidad de almacenamiento en memoria principal: ¿Es preferible una red local con mayor capacidad de almacenamiento en memoria principal o con mayor capacidad de procesamiento? Es bastante probable que las redes locales que tienen mayor capacidad de almacenamiento (sumando las capacidades de cada una de las computadoras interconectadas) sean también las de mayor capacidad de procesamiento, pero esto no se puede asegurar dado que las redes locales no necesariamente están diseñadas para cómputo paralelo y aún más para cómputo paralelo con otras redes.

En una extensión de esta tesis que se podría denominar “a gran escala” se pueden combinar los dos tipos de extensiones que se han mencionado hasta este punto:

- Extensión en cuanto a otros problemas a resolver
- Extensión en cuanto a la utilización de mayor cantidad de máquinas a utilizar involucrando más de una red local.

Quizás en ambos casos los problemas serán mucho mayores con respecto a procesamiento necesario como a cantidad de datos a procesar, pero se pueden seguir utilizando al menos inicialmente los principios básicos de paralelización de matrices. En todo caso, a partir de los problemas que se identifican vía experimentación se pueden proponer otros más específicos y apropiados para obtener el máximo rendimiento posible a partir de los recursos disponibles.

Bibliografía

[1] Aho A., J. Hopcroft, J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts, 1974.

[2] Akl S., The Design and Analysis of Parallel Algorithms, Prentice-Hall, Inc., 1989.

[3] Akl S., Parallel Computation: Models and Methods, Prentice-Hall, Upple Saddle River, 1997.

[4] Alpatov P., G. Baker, C. Edwards, J. Grunnels, G. Morrow, J. Overfelt, R. van de Geijn, Y. Wu, "PLAPACK: Parallel Linear Algebra Package - Design Overview", Proc. Supercomputing '97, SC97, 1997. Disponible en
http://www.cs.utexas.edu/users/rvdg/conference.html

[5] Alpern B., L. Carter, J. Ferrante, "Space-limited procedures: A methodology for portable high-performance", International Working Conference on Massively Parallel Programming Models, 1995.

[6] Amdhal G., "Validity of the Single-Processor Approachto Achieving Large-Scale Computing Capabilities", Proc. 1967 AFIPS Conference, Vo. 30, p. 483, 1967.

[7] 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, 1990.

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

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

[10] Andrews G, Concurrent Programming: Principles and Practice, The Benjamin/ Cummings Publishing Company, Inc., 1991.

[11] Bailey D. H., "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers", RNR Technical Report RNR-91-020, June 11, 1991. Disponible en http://unix.hensa.ac.uk/parallel/papers/surveys/

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

[13] Baker M., G. Fox, "Metacomputing: Harnessing Informal Supercomputers", in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 154-185, 1999.

[14] Bala V., J. Bruck, R. Cypher, P. Elustondo, A. Ho, C. Ho, S. Kipnis, M. Snir, “CCL: A Portable and Tunable Collective Communication Library for Scalable Parallel Computing”, Proc. of the 8th International Conference on Parallel Processing, IEEE, April 1994.

[15] Banikazemi M., V. Moorthy, D. Panda, “Efficient Collective Communication on Heterogeneous Networks of Workstations”, Proc. International Conference on Parallel Processing, pp. 460-467, 1998.

[16] Barnett M., S. Gupta, D. Payne, L. Shuler, R. van de Geijn, J. Watts, “Interprocessor Collective Communication Library (InterCom)”, Proc. of the Scalable High-Performance Computing Conference '94, Knoxville, TN, USA, IEEE Computer Society Press, pp. 357-364, May 1994.

[17] Barreiro Paz M., V. M. Gulias, “Cluster Setup and its Administration”, in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 48-67, 1999.

[18] Basney J., M. Livny, “Deploying a High Throughput Computing Cluster”, in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 116-134, 1999.

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

[20] Bilmes J., K. Asanovic, C. Chin, J. Demmel, “Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology”, Proceedings of the International Conference on Supercomputing, Vienna, Austria, July 1997, ACM SIGARC.

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

[22] Blackford L., Cleary A., Demmel J., Dhillon I., Dongarra J., Hammarling S., Petitet A., Ren H., Stanley K., Whaley R., “LAPACK Working Note 112: Practical Experience in the Dangers of Heterogeneous Computing”, UT, CS-96-334, 1996. Disponible en http://www.netlib.org/lapack/lawns/index.html

[23] Cannon L. E., A Cellular Computer to Implement the Kalman Filter Algorithm, Ph.D. Thesis, Montana State University, Bozman, Montana, 1969.

[24] Catlett C., L. Smarr, “Metacomputing”, Communications of the ACM, Vol. 35 (6), pp. 44-52, 1992.

[25] Chiola G., G. Ciaccio, “Lightweigth Messaging Systems”, in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 246-269, 1999.

[26] Choi J., “A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers”, Proceedings of the High-Performance Computing on the Information Superhighway, IEEE, HPC-Asia '97.

[27] Choi J., J. Dongarra, R. Pozo, D. Walker, “ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers”, Proc. 4th Symposium on the Frontiers of Massively Parallel Computation, Ieee Computer Society Press, pp. 120-127, 1992.

[28] Choi J., J. Dongarra, D. Walker, “The Design of Scalable Software Libraries for Distributed Memory Concurrent Computers”, Proc. of Environment and Tools for Parallel Scientific Computing Workshop, Saint Hilaire du Touvet, France, Elsevier Science Publishers, Sep. 1992.

[29] Choi J., J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, R. Whaley, “LAPACK Working Note 95. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance”, 1996. Disponible en http://www.netlib.org/lapack/lawns/index.html

[30] Choi J., J. Dongarra, D. Walker, “PUMMA: Parallel Universal Matrix Multiplication Algorithm on Distributed Memory Concurrent Computers”, in Concurrency: Practice and Experience, 6:543-570, 1994.

[31] Ciaccio G., “Optimal Communication Performance on Fast Ethernet with GAMMA”, Proceedings Workshop PC-NOW, IPPS/SPDP'98, Orlando, FL, LNCS No. 1388, Springer, pp. 534-548, April 1998.

[32] Comer D., Internetworking with TCP/IP: Vol. I: Principles, Protocols, and Architecture, 3rd Ed., Prentice-Hall, Englewood Cliffs, 1993.

[33] DeCegama A., Parallel Processing Architectures and VLSI Hardware, Volume I, Prentice-Hall International, Inc., Englewood Cliffs, 1989.

[34] Deering S., “Host Extensions for IP Multicasting”, RFC 1112, USC/Information Sciences Institute, no. 1112, Aug. 1989.

[35] Dekel E., D. Nassimi, S. Sahni, “Parallel matrix and graph algorithms”, SIAM Journal on Computing, 10:657-673, 1981.

[36] Demmel J., J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, D. Sorensen, Prospectus for the Development of a Linear Algebra Library for High-Performance Computers, ANL, MCS-TM-97, Sep. 1987, disponible en
http://www.netlib.org/lapack/lawns

[37] Dietz H., Linux Parallel Processing HOWTO, Jan 98, disponible en http://www.linuxdoc.org/HOWTO/Parallel-Processing-HOWTO.html

[38] Dietz H., W. Cohen, T. Muhammad, T. Mattox, “Compiler Techniques For Fine-Grain Execution on Workstation Clusters Using PAPERS”, 7th Annual Workshop on Languages and Compilers for Parallel Computing, Cornell University, Aug 1994.

[39] Dietz H., R. Hoare, T. Mattox, “A fine-Grain Parallel Architecture Based on Barrier Synchronization”, Proc. International Conference on Parallel Processing, Vol. I, pp. 247-250, August 1996.

[40] Dietz H., T. Muhammad, J. Sponaugle, T. Mattox, PAPERS: Purdue's Adapter for Parallel Execution and Rapid Synchronization, Purdue University School of Electrical Engineering, Technical Report TR-EE-94-11, March 1994.

[41] Donaldson V., F. Berman, R. Paturi, “Program Speedup in a Heterogeneous Computing Network”, Journal of Parallel and Distributed Computing, 21:3, pp. 316-322, June 1994.

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

[43] 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.

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

[45] Dongarra J., van de Geijn R., Walker D., “Scalability Issues Affecting the Design of a Dense Linear Algebra Library”, Journal of Parallel and Distributed Computing, Vol. 22, No. 3, pp. 523-537, Sep. 1994.

[46] 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.

[47] Edwards C., P. Geng, A. Patra, R. van de Geijn, “Parallel matrix distributions: Have we been doing it all wrong?”, Technical Report TR-95-39, Department of Computer Sciences, The University of Texas at Austin, 1995. Disponible en http://www.cs.utexas.edu/users/rvdg/abstracts/PBMD.html

[48] Feitelson D., “Scheduling Parallel Jobs on Clusters”, in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 519-533, 1999.

[49] Feitelson D., A. Batat, G. Benhanokh, D. Er-El, Y. Etsion, A. Kavas, T. Klainer, U. Lublin, M. Volovic, “The ParPar System: A Software MPP”, in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 754-770, 1999.

[50] Flynn M., “Very High Speed Computing Systems”, Proc. IEEE, Vol. 54, 1966.

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

[52] Foster I., Designing and Building Parallel Programs, Addison-Wesley, Inc., 1995. *Versión html* disponible en http://www-unix.mcs.anl.gov/dbpp

[53] Foster I., “Internet Computing and the Emerging Grid”, Nature, Macmillan Publishers, Dec 7, 2000, disponible en http://www.nature.com/nature/webmatters/grid/grid.html

[54] Foster I., C. Kesselman, Eds., The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kauffmann, 1999.

[55] Foster I., C. Kesselman, S. Tuecke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International Journal Supercomputer Applications, 2001. Disponible en httP://www.globus.org/research/papers/anatomy.pdf

[56] Fox G., M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, Solving Problems on Concurrent Processors, Vol. I, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[57] Fox G., W. Otto, J. Hey, “Matrix Algorithms on a Hypercube I: Matrix Multiplication”, Parallel Computing, 4: 17-31, 1987.

[58] Fox G., R. Williams, P. Messina, Parallel Computing Works!, Morgan Kaufmann Publishers, Inc., 1994.

[59] Golub G. H., C. F. Van Loan, Matrix Computation, Second Edition, The John Hopkins University Press, Baltimore, Maryland, 1989.

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

[61] Gustafson J., “Computational Verifiability and Feasibility of the ASCI Program”, IEEE Computational Science and Engineering, pp. 36-44, January-March 1998.

[62] Hake J.-F., “Parallel Algorithms for Matrix Operations and Their Performance on Multiprocessor Systems”, in Advances in Parallel Algorithms, L. Kronosjo, D. Shumsheruddin, eds., Haslted Press, New York, 1993.

[63] Hennessy J. L., D. A. Patterson, Computer Architecture. A Quantitative Approach, Morgan Kaufmann, San Francisco, 1996.

[64] Henning J. L., SPEC CPU2000: Measuring CPU Performance in the New Millenium, Computer, IEEE Computer Society, July 2000.

[65] Hipper G., D. Tavangarian, “A Concurrent Communication Architecture for Workstation Clusters”, Proceedings of the ISMM International Conference on Intelligent

Information Management Systems, Washington, D. C., 1995.

[66] Hoare C., Communicating Sequential Processes, Englewood Cliffs, Prentice-Hall, 1986.

[67] Hoare R., T. Mattox, H. Dietz, TTLPAPERS 96081. The Modularity Scalable, Field Upgradable, Implementation of Purdue's Adapter for Parallel Execution and Rapid Synchronization, Purdue University West Lafayette, May 2000, disponible en http://www.aggregate.org/AFN/96081/Index.html

[68] Hockney R., M. Berry (eds.), "Public International Benchmarks for Parallel Computers", Scientific Programming 3(2), pp. 101-146, 1994.

[69] Hockney R., C. Jesshope, Parallel Computers 2, Adam Hilger, Bristol and Philadelphia, IOP Publishing Ltd., 1988.

[70] Horowitz E., A. Zorat, "Divide-and-Conquer for Parallel Processing", IEEE Trans. on Comp., Vol. C-32, No. 6, pp. 582-585, 1983

[71] Hwang K., Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, Inc., 1993.

[72] Institute of Electrical and Electronics Engineers, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1984, 1984.

[73] Institute of Electrical and Electronics Engineers, Local Area Network - CSMA/CD Access Method and Physical Layer Specifications ANSI/IEEE 802.3 - IEEE Computer Society, 1985.

[74] Intel Corporation, Streaming SIMD Extensions - Matrix Multiplication, AP-930, Order Number: 245045-001, June 1999.

[75] Johnston W., D. Gannon, W. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid", Proc. 8th Symposium on High Performance Distributed Computing, IEEE Computer Society Press, 1999.

[76] Kadambi J., I. Crayford, M. Kalkunte, Gigabit Ethernet, Prentice Hall, Upper Saddle River, NJ, 1998.

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

[78] Kim J., Lilja D., "Multiple Path Communication", in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 364-382, 1999.

[79] Kumar V, Grama A, Gupta A, Karypis G, Introduction to Parallel Computing. Design and Analysis of Algorithms, The Benjamin/Cummings Publishing Company, Inc., 1994.

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

[81] Lawson C., R. Hanson, D. Kincaid, F. Krogh, "Algorithm 539: Basic Linear Algebra Subprograms for Fortran Usage", ACM Transactions on Mathematical Software 5, pp. 324-325, 1979.

[82] Leighton F, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes", Morgan Kaufman Publishers, 1992.

[83] Lu P., "Using Scoped Behavior to Optimize Data Sharing Idioms", in R. Buyya Ed., High Performance Cluster Computing: Programming an Applications, Vol. 2, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 113-130, 1999.

[84] Luenberger D. G., Linear and Nonlinear Programming, Second Ed., Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, USA, 1984.

[85] McCalpin J. D., M. Smotherman, "Automatic benchmark generation for cache optimization of matrix algorithms", Proceedings of the 33rd Annual Southeast Conference, R. Geist and S. Junkins editors, Association for Computing Machinery, New York, March 1995.

[86] Meuer H., E. Strohmaier, J. Dongarra, H. Simon, "TOP500 Supercomputer Sites", 18th Edition, Nov. 2001. Disponible en http://www.top500.org

[87] Miguel J., A. Arruabarrena, R. Beivide, J. A. Gregorio, "Assessing the Performance of the New IBM SP2 Communication Subsystem", IEEE Parallel & Distributed Computing, pp. 12-22, Winter 1996.

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

[89] Murhammer M., et al., TCP/IP: Tutorial and Technical Overview, Prentice Hall, Upper Saddle River, NJ, 1998.

[90] Mutka M., M. Livny, "The Available Capacity of a Privately Owned Workstation Environment", Performance Evaluation 12, 269-284, 1991.

[91] Nagendra B., Rzymianowicz, "High Speed Networks", in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 204-245, 1999.

[92] Pacheco P., Parallel Programming with MPI, Morgan Kaufmann, San Francisco, California, 1997.

[93] Partridge C., S. Pink, “A Faster UDP”, IEEE/ACM Trans. on Networking, Aug. 1993.

[94] PARKBENCH committee, ParkBench 2.0 Release Notes, April 1996, Available at http://www.netlib.org/parkbench

[95] Postel J., “User Datagram Protocol”, RFC 768, USC/Information Sciences Institute, Aug. 1980.

[96] Postel J., “Internet Protocol”, RFC 791, USC/Information Sciences Institute, Sep. 1981.

[97] Postel J., “Transmission Contol Protocol”, RFC 793, USC/Information Sciences Institute, Sep. 1981.

[98] Preparata F., J. Vuillemin, “Area-time optimal VLSI networks for matrix multiplication”, Proceedings of the 14th Princeton Conference on Information Science and Systems, pp. 300-309, 1980.

[99] Radajewski J., D. Eadline, Beouwulf HOWTO, Nov 1998, disponible en http://www.linuxdoc.org/HOWTO/Beowulf-HOWTO.html

[100] Ranka S., S. Sahni, Hypercube Algorithms for Image Processing and Pattern Recognition, Springer-Verlag, New York, 1990.

[101] Rice J., Matrix Computations and Mathematical Software, McGraw-Hill, New York, 1981.

[102] Rice J., Numerical Methods, Software,and Analysis, 2nd. Ed., Academic Press, San Diego, 1992.

[103] Ridge D., D. Becker, P. Merkey, T. Sterling, “Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs, Proceedings IEEE Aerospace, 1997, disponible también en http://www.beowulf.org/papers/papers.html

[104] Saavedra R., W. Mao, D. Park, J. Chame, S. Moon, “The combined effectiveness of unimodular transformations, tiling, and software prefetching”, Proceeding of the 10th International Parallel Symposium, IEEE Computer Society, April, 1996.

[105] Seifert R., Gigabit Ethernet, Addison-Wesley, Reading, MA, 1998.

[106] Sidani M., B. Harrod, “LAPACK Working Note 116: Parallel Matrix Distributions: Have we been doing it all right?”, Nov. 1996. Disponible en http://www.netlib.org/lapack/lawns/index.html

[107] Snir M., S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI: The Complete Reference, The MIT Press, Cambridge, Massachusetts, 1996.

[108] Spurgeon C., Ethernet Configuration Guidelines, Peer-to-Peer Communications, San

Jose, CA, 1996.

[109] Stallings W., Local and Metropolitan Local Area Networks, 6th Ed., Prentice-Hall, Upper Saddle River, NJ, 2000.

[110] Stallings W., Business Data Communications, 4th Ed., Prentice-Hall, Upper Saddle River, NJ, 2001.

[111] Sterling T., J. Salmon, D. Becker, D. Savaresse, How to build a Beowulf: a guide to the implementation and application of PC clusters, The MIT Press, Cambridge, Massachusetts, 1999.

[112] Stevens R., TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley Longman, Inc., 1994.

[113] Stone H., High-Performance Computer Architecture, Third Edition, Addison-Wesley Publishing Company, 1993.

[114] Strassen V., Gaussian Elimination Is Not Optimal, Numerische Mathematik, Vol. 13, pp. 353-356, 1969.

[115] Suciu O., C. Fetzer, “Determining the User-Level Transmission Delay in Networks of Workstations”, Proc. of the IASTED International Conference on Parallel and Distributed Systems, Universidad Autónoma de Barcelona, Spain, June 9-11, 1997.

[116] Tinetti F., “Aplicaciones Paralelas de Cómputo Intensivo en NOW Heterogéneas”, Workshop de Investigadores en Ciencias de la Computación (WICC 99), San Juan, Argentina, 27 y 28 de Mayo de 1999, pp. 17-20.

[117] Tinetti F., “Performance of Scientific Processing in Networks of Workstations”, Workshop de Investigadores en Ciencias de la Computación (WICC 2000), La Plata, Argentina, 22 y 23 de Mayo de 2000, pp. 10-12.

[118] Tinetti F., “Performance of Scientific Processing in NOW: Matrix Multiplication Example”, JCS&T, Journal of Computer Science & Technology, Special Issue on Computer Science Research, Vol. 1 No. 4, March 2001, pp. 78-87.

[119] Tinetti F., Barbieri A., “Cómputo y Comunicación: Definición y Rendimiento en Redes de Estaciones de Trabajo”, Workshop de Investigadores en Ciencias de la Computación (WICC 2001), San Luis, Argentina, 22-24 de Mayo de 2001, pp. 45-48.

[120] Tinetti F., Barbieri A., “Collective Communications for Parallel Processing in Networks of Workstations”, Proceedings SCI 2001, Volume XIV, Computer Science and Engineering: Part II, Nagib Callaos, Fernando G. Tinetti, Jean Marc Champarnaud, Jong Kun Lee, Editors, International Institute of Informatics and Systemics, Orlando, Florida, USA, ISBN 980-07-7554-4, July 2001, pp. 285-289.

[121] Tinetti F., Barbieri A., “Análisis del Rendimiento de las Comunicaciones sobre

NOWs", Proceedings VII Congreso Argentino de Ciencias de la Computación (CACIC), El Calafate, Santa Cruz, Argentina, 16 al 20 de Octubre de 2001, pp. 654-656.

[122] Tinetti F., Barbieri A., "Cómputo Paralelo en Clusters: Herramienta de Evaluación de Rendimiento de las Comunicaciones", Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 15 al 18 de Octubre de 2002, p. 123.

[123] Tinetti F., Barbieri A., "An Efficient Implementation for Broadcasting Data in Parallel Applications over Ethernet Clusters", Proceedings of the 17th International Conference on Advanced Information Networking and Applications (AINA 2003), IEEE Press, ISBN 0-7695-1906-7, March 2003.

[124] Tinetti F., Barbieri A., Denham M., "Algoritmos Paralelos para Aprovechar Redes Locales Instaladas", Workshop de Investigadores en Ciencias de la Computación (WICC 2002), Bahía Blanca, Argentina, 17-18 de Mayo de 2002, pp. 399-401.

[125] Tinetti F., D' Alessandro A., Quijano A., "Communication Performance of Installed Networks of Workstations for Parallel Processing", Proceedings SCI 2001, Volume XIV, Computer Science and Engineering: Part II, Nagib Callaos, Fernando G. Tinetti, Jean Marc Champarnaud, Jong Kun Lee, Editors, International Institute of Informatics and Systemics, Orlando, Florida, USA, ISBN 980-07-7554-4 July 2001, pp. 290-294.

[126] Tinetti F., Denham M., "Paralelización y Speedup Superlineal en Supercomputadoras. Ejemplo con Multiplicación de Matrices", Proceedings VII Congreso Argentino de Ciencias de la Computación (CACIC), El Calafate, Santa Cruz, Argentina, 16 al 20 de Octubre de 2001, pp. 765-774.

[127] Tinetti F., Denham M., "Paralelización de la Factorización de Matrices en Clusters", Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 15 al 18 de Octubre de 2002, p. 121.

[128] Tinetti F., Denham M., "Algebra Lineal en Clusters Basados en Redes Ethernet", Workshop de Investigadores en Ciencias de la Computación (WICC 2003), Tandil, Argentina, 22-23 de Mayo de 2003, pp. 575-579.

[129] Tinetti F., Denham M., "Paralelización de la Factorización LU de Matrices en Clusters Heterogéneos", Proceedings IX Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Informática, Universidad Nacional de La Plata, La Plata, Argentina, 6 al 10 de Octubre de 2003, p. 385-396.

[130] Tinetti F., Denham M., De Giusti A., "Parallel Matrix Multiplication and LU Factorization on Ethernet-based Clusters", High Performance Computing. 5th International Symposium, ISHPC 2003, Tokyo-Odaiba, Japan, October 20-22, 2003, Proceedings. Series: Lecture Notes in Computer Science, Vol. 2858. Veidenbaum, A.; Joe, K.; Amano, H.; Aiso, H. (Eds.), 2003, XV, 566 p. ISBN: 3-540-20359-1

[131] Tinetti F., Luque E., “Parallel Matrix Multiplication on Heterogeneous Networks of Workstations”, Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 15 al 18 de Octubre de 2002, p. 122.

[132] Tinetti F., Luque E., “Efficient Broadcasts and Simple Algorithms for Parallel Linear Algebra Computing in Clusters”, Workshop on Communication Architecture for Clusters, International Parallel and Distributed Processing Symposium (IPDPS '03), Nice Acropolis Convention Center, Nice, France April 22-26, 2003.

[133] Tinetti F., Quijano A., “Capacidad de Comunicaciones Disponible para Cómputo Paralelo en Redes Locales Instaladas”, Proceedings VIII Congreso Argentino de Ciencias de la Computación (CACIC), Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, 15 al 18 de Octubre de 2002, p. 125.

[134] Tinetti F., Quijano A., “Costos del Cómputo Paralelo en Clusters Heterogéneos”, Workshop de Investigadores en Ciencias de la Computación (WICC 2003), Tandil, Argentina, 22-23 de Mayo de 2003, pp. 580-584.

[135] Tinetti F., A. Quijano, A. De Giusti, “Heterogeneous Networks of Workstations and SPMD Scientific Computing”, 1999 International Conference on Parallel Processing, The University of Aizu, Aizu-Wakamatsu, Fukushima, Japan, September 21 - 24, 1999, pp. 338-342.

[136] Tinetti F., A. Quijano, A. De Giusti, E. Luque, “Heterogeneous Networks of Workstations and the Parallel Matrix Multiplication”, Recent Advances in Parallel Virtual Machine and Message Passing Interface, 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23-26, 2001, Proceedings, Yannis Cotronis, Jack Dongarra (Eds.), Lecture Notes in Computer Science 2131 Springer 2001, ISBN 3-540-42609-4, pp. 296-303.

[137] Tinetti F., Sager G., Rexachs D., Luque E., “Cómputo Paralelo en Estaciones de Trabajo no Dedicadas”, VI Congreso Argentino de Ciencias de la Computación, Ushuaia, Argentina, Octubre de 2000, Tomo II, pp. 1121-1132.

[138] Trulove J., LAN Wiring, Mc-Graw Hill, New York, 1997

[139] Yokokawa M., S. Shingu, S. Kawai, K. Tani, H. Miyoshi, “Performance Estimation of the Earth Simulator”, Towards Teracomputing, Proc. of 8th ECMWF Workshop, World Scientific, pp. 34-53, 1998.

[140] Yoshida K., S. Shingu, “Research and Development of the Earth Simulator”, Proc. of 9th ECMWF Workshop, World Scientific, pp. 1-13, 2000.

[141] van de Geijn R., Using PLAPACK: Parallel Linear Algebra Package, The MIT Press, 1997.

[142] van de Geijn R., J. Watts, SUMMA Scalable Universal Matrix Multiplication

Algorithm, LAPACK Working Note 99, Technical Report CS-95-286, University of Tenesse, 1995.

[143] Warren M. S., T. C. Germann, P. S. Lomdahl, D. M. Beazley, J. K. Salmon, "Avalon: An Alpha/Linux Cluster Achieves 10 Gflops for $150k". Disponible en http://cnls.lanl.gov/avalon/avalon_bell98/ propuesto para el 1998 Gordon Bell Price/Performance Prize.

[144] Whaley R., J. Dongarra, "Automatically Tuned Linear Algebra Software", Proceedings of the SC98 Conference, Orlando, FL, IEEE Publications, November, 1998.

[145] Wilkinson J, C. Reinsch, Handbook of Automatic Computation: Volume II - Linear Algebra, Springer-Verlag, New York, 1971.

[146] Wilkinson B., Allen M., Parallel Programming: Techniques and Applications Using Networking Workstations, Prentice-Hall, Inc., 1999.

[147] Zhang X., Y. Yan, "Modeling and characterizing parallel computing performance on heterogeneous NOW", Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing, *(SPDP'95)*, IEEE Computer Society Press, San Antonio, Texas, October 1995, pp. 25-34

[ATLAS] ATLAS Home Page http://www.netlib.org/atlas

[BEOWULF] Beowulf Home Page http://www.beowulf.org

[BLAS] BLAS Home Page http://www.netlib.org/blas

[CXML] Compaq Extended Math Library (CXML) Home Page
http://www.compaq.com/math/documentation/cxml

[EA] Earth Simulator Home Page http://www.es.jamstec.go.jp

[GAMMA] GAMMA Home Page http://www.disi.unige.it/project/gamma/

[LAM/MPI] LAM/MPI (Local Area Computing / Message Passing Interface) Home Page
http://www.mpi.nd.edu/lam

[LAPACK] LAPACK Home Page http://www.netlib.org/lapack

[LinuxRH] Linux RedHat Home Page, http://www.redhat.com

[MPICH] MPICH Home Page http://www-unix.mcs.anl.gov/mpi/mpich/

[PAPERS] PAPERS home page http://garage.ecn.purdue.edu/~papers

[PLAPACK] PLAPACK Home Page http://www.cs.utexas.edu/users/plapack

[PVM] PVM Home Page http://www.emm.ornl.gov/pvm/pvm_home. html

[ScaLAPACK] ScaLAPACK Home Page
http://www.netlib.org/scalapack/scalapack_home.html

[SCSL1] SGI SCSL (Scientific Computing Software Library) Home Page
http://www.sgi.com/software/scsl.html

[SCSL2] SGI SCSL (Scientific Computing Software Library) Data Sheet
http://www.sgi.com/Products/PDF/3024.pdf

[SML] Intel Pentium III Small Matrix Library
http://developer.intel.com/design/pentiumiii/sml/245045.htm

[SPEC] SPEC Home Page http://www.spec.org

[TOP500] TOP500 Supercomputer Site Home Page http://www.top500.org

[WinLinux] Winlinux Home Page, http://www.winlinux.net

Apéndice A: Características de las Redes Locales

En este apéndice se muestran las principales características de hardware y software de las redes locales y de las computadoras que las componen y que se utilizaron en la experimentación. Las dos razones más importantes para incluir un apéndice dedicado a descripción del hardware y software utilizado son:

1. Hay varios capítulos dedicados a experimentación para la observación de distintos índices de rendimiento, y por lo tanto repetir toda la información en cada capítulo sería redundante.

2. Existen muchas características de las que depende el rendimiento de una computadora, y por lo tanto la sola enumeración de cada una de ellas puede resultar en el desvío de la atención respecto de lo que se expone en cada capítulo de experimentación.

En el contexto de la ejecución de aplicaciones paralelas sobre redes de computadoras es importante conocer la topología de la red de interconexión de las computadoras. Por esta razón también se incluyen algunos detalles del hardware de las redes locales en las cuales están incluidas las computadoras con las que se lleva a cabo la experimentación.

A.1 Introducción: Características de Hardware y Software

El rendimiento de una computadora normalmente depende de muchos factores de diseño de su arquitectura y también de la forma en que el software haga uso del hardware disponible (nivel de optimización). En este caso, *software* abarca desde el sistema operativo hasta el mismo programa de aplicación que se implementa para ser ejecutado sobre la computadora.

Las características de hardware que se consideran elementales en cuanto al rendimiento de una computadora suelen ser:
1. (Micro)Procesador.
2. Frecuencia de reloj.
3. Tamaño de memoria principal.

En la lista anterior se solía incluir el tamaño de memoria cache del primer nivel (*L1 cache size*), pero a medida que las tecnologías de integración han avanzado, los procesadores ya incluyen *de facto* este nivel de memoria y por lo tanto está definido unívocamente por el mismo procesador. Más aún, una gran cantidad de procesadores estándares incluyen también el segundo nivel de memoria (*L2 cache*) en el mismo circuito integrado y por lo tanto también este segundo nivel de memoria cache queda unívocamente determinado por el procesador. De todas maneras, como la jerarquía de memoria completa (que incluye todos los niveles de memoria cache disponibles) tiene un impacto tan importante en el rendimiento de las computadoras y también para tener una visión rápida y completa de la arquitectura de cada máquina, se incluyen en este apéndice los datos de:
1. nivel de memoria cache (L1, L2, etc.),
2. tamaño de memoria cache. Si la memoria cache está dividida se dan los tamaños para datos (D) e instrucciones (I) por separado, si es unificada se da el tamaño total,
3. ubicación de memoria cache (interna o externa con respecto al procesador),

de todos los niveles de memoria cache disponibles de cada computadora. También con referencia a la jerarquía de memoria, se incluye para cada máquina el espacio (tamaño) de *swap* configurado.

Para tener una referencia *completa* (o más general) del tipo de máquina, se incluyen los datos de:
1. Fabricante de la computadora (*Marca*), en el caso de algunas PCs se menciona que son armadas por partes.
2. Modelo, indicando el *tipo* de computadora (PC o modelo determinado por el fabricante-marca).
3. Nombre: identificación de la computadora dentro de la red local. Estos nombres son los asignados previamente en cada red local por el administrador de cada computadora y por lo tanto no necesariamente tienen relación con cómputo paralelo.

Las características de software más importantes que se detallan para computadora son:
1. Sistema operativo.
2. Compilador utilizado.
3. Opciones de optimización utilizadas en la compilación. En el caso de no utilizar estas opciones de optimización se mencionarán explícitamente qué opciones de compilación

se utilizaron en el contexto apropiado.

Si bien existen muchas más características de hardware y de software que se pueden mencionar para cada computadora, se considera que los que se han detallado pueden dar una idea bastante completa y explícita de máquina.

Resulta interesante hacer un pequeño comentario respecto a la bibliografía y/o documentación de las características de las computadoras. Varios sistemas operativos derivados de Unix poseen comando/s propio/s para la descripción del hardware, aunque en general no son comunes a todas las versiones de Unix y por lo tanto se deben buscar específicamente en los manuales correspondientes.

En otro nivel de detalle, es relativamente sencillo encontrar información de los procesadores en las páginas de Internet de cada fabricante, así como también encontrar información de las computadoras que son provistas por empresas reconocidas. Sin embargo, las dificultades crecen de forma drástica dependiendo de:

- Antigüedad de la computadora-procesador. Como es de esperar, para computadoras de más de cinco años (dependiendo del fabricante) es bastante difícil encontrar información detallada.
- PCs construidas por partes. También como es de esperar, la información de cada PC es muy específica del tiempo en que se construyó y del proveedor mismo. Detalles tales como la velocidad del bus del sistema y de la memoria RAM dependen usualmente del estado del mercado respectivo en el momento de la construcción de la computadora.

A.2 Redes Locales

Las computadoras están ubicadas en tres redes locales, que se denominarán según su pertenencia a los laboratorios en los que están ubicadas:

1. CeTAD: Centro de Técnicas Analógico-Digitales, Departamento de Electrotecnia, Facultad de Ingeniería, Universidad Nacional de La Plata. Es la que está instalada desde hace más tiempo y las computadoras que la componen son utilizadas con múltiples propósitos.
2. LQT: Laboratorio de Química Teórica, CEQUINOR, Departamento de Química, Facultad de Ciencias Exactas, Universidad Nacional de La Plata. Es una red destinada a la resolución de problemas numéricos, fue instalada hace varios años y se ejecutan trabajos secuenciales y paralelos desarrollados con PVM y Linda.
3. LIDI: perteneciente al Laboratorio de Investigación y Desarrollo en Informática, Facultad de Informática, Universidad Nacional de La Plata. Está dedicada a enseñanza de programación paralela e investigación. Puede considerarse directamente una instalación más del tipo Beowulf, aunque no de las más costosas en cuanto a cantidad de máquinas y red de interconexión.

Las dos primera redes locales están conectadas a Internet, siendo de hecho subredes de una red clase B de Internet. Las subsecciones que siguen muestran los nombres de las computadoras junto con la configuración física de cada una de las redes locales.

A.2.1 Red Local del CeTAD

La red local ubicada en el CeTAD es quizás de las más representativas entre las redes locales que han evolucionado durante varios años. Además, las máquinas tienen múltiples propósitos, que abarcan trabajo administrativo, prototipación de algoritmos de procesamiento de señales y diseño de circuitos integrados de propósito específico. Comenzó con tres o cuatro computadoras interconectadas con cable coaxial y a partir de su creación ha evolucionado en cuatro direcciones:

1. Actualización de las computadoras: reemplazadas o actualizadas por partes.
2. Agregado de computadoras.
3. Impresora en red compartida.
4. Discos locales de cada computadora compartidos.

La Fig. A.1 muestra esquemáticamente la interconexión física de las computadoras junto con sus nombres.

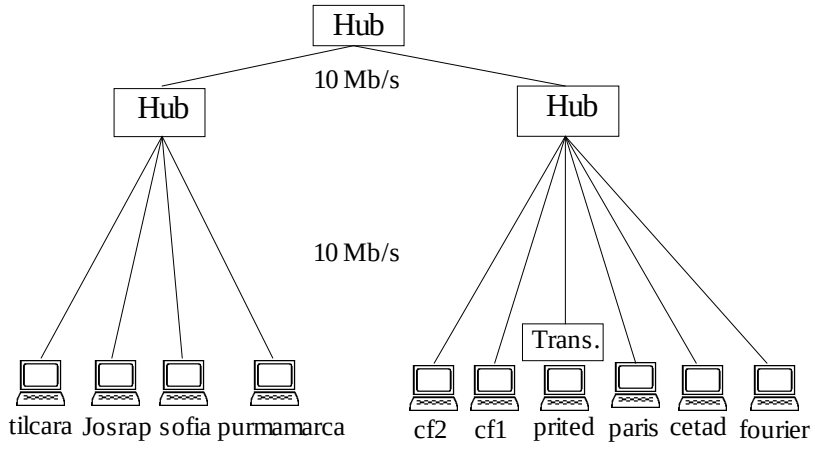


Figura A.1: Red Local del CeTAD.

Las computadoras que aparecen en la Figura A.1 con los nombres **cf1** y **cf2** en realidad tienen asignados los nombres **cetadfomec1** y **cetadfomec2** respectivamente, pero se abreviaron (y en general se abreviarán) por razones de espacio.

En la Figura A.1 se muestran *solamente* las computadoras de la red local utilizadas para cómputo paralelo. Se excluyeron, por lo tanto:

- otras nueve computadoras que están interconectadas con las que se muestran pero que no se utilizan para cómputo paralelo,
- una computadora dedicada a *router-gateway* y *spooler* de impresión común,
- la impresora compartida, también conectada directamente a la red.

Como en toda red local basada en hubs, la islación de un subconjunto de computadoras del resto depende de la cantidad y distribución de los hubs. En el caso específico de esta red local, la aislación de las diez computadoras que se muestran en la Figura A.1 de las demás es bastante sencillo dado que todas las máquinas y los hubs están en una misma habitación.

A.2.2 Red Local del LQT

Las computadoras de la red local del LQT son utilizadas con exclusividad para aplicaciones de cómputo intensivo y para algunas aplicaciones de cómputo paralelo. Por ser una red con relativamente pocas computadoras no tiene una gran complejidad en cuanto a su cableado de interconexión. De hecho, solamente tienen instalado lo mínimo necesario para tal fin, sin herramientas de oficina consideradas *clásicas* como editores/formateadores de texto o planillas de cálculo. Sin embargo, a semejanza de la red local del CeTAD, está instalada y en uso desde hace varios años y consecuentemente ha sido actualizada (y *aumentada*) varias veces.

La Figura A.2 muestra esquemáticamente la interconexión física y los nombres de las seis máquinas de la red utilizadas para cómputo paralelo. También en este caso se omitieron otras tres computadoras que no se utilizan para cómputo paralelo y una computadora dedicada a *router-gateway* de la red.

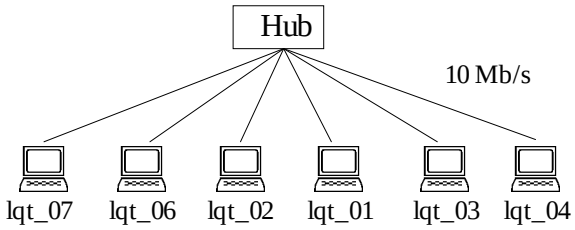


Figura A.2: Red Local del LQT.

A.2.3 Red Local del LIDI

A diferencia de las dos redes locales anteriores:

- La red local del LIDI fue planificada y construida exclusivamente para cómputo paralelo y por esta razón también coincide con una instalación Beowulf.
- No llega a tener un año de instalada y no ha tenido ningún cambio desde su instalación, manteniéndose homogénea.
- Las placas de interconexión (NIC: Network Interface Card) de cada computadora son Ethernet de 10/100 Mb/s.
- Se utiliza un switch Ethernet de 10/100 Mb/s. Por lo tanto, las computadoras no solamente se comunican a 100 Mb/s sino que además pueden realizarse múltiples transferencias de datos punto a punto simultáneas.

La Figura A.3 muestra esquemáticamente la interconexión física y los nombres de las seis máquinas de la red utilizadas para cómputo paralelo. Aunque desde el punto de vista de la asignación de números IP (Internet Protocol) esta red local es parte de otra con varias computadoras, impresoras, discos compartidos, etc., también se la puede considerar de manera aislada simplemente manteniendo desconectado el “uplink” del switch de comunicaciones. De hecho, la aislación de esta red local de otras computadoras y su tráfico en la red asociado es bastante más sencilla que en el caso de las redes locales del CeTAD y del LQT.

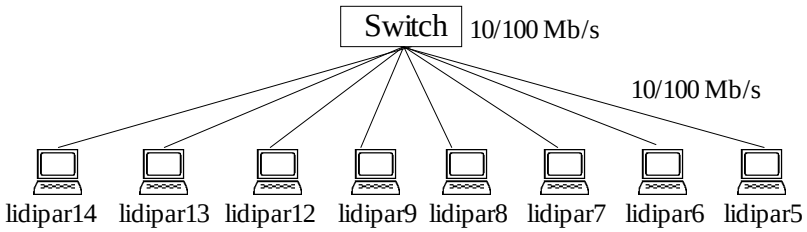


Figura A.3: Red Local del LIDI.

A.3 Detalle de las Computadoras

En esta sección se proporcionan las características mencionadas en la primera sección de cada una de las computadoras pertenecientes a cada red local.

A.3.1 Computadoras de la Red Local del CeTAD

1) Nombre: purmamarca
Marca: Construida por partes
Modelo: PC
Procesador: Pentium II
Reloj: 400 MHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna
Cache L2 Tamaño: 512 KB Unificada Ubicación: Interna
Mem. principal: 64 MB
Espacio de swap: 128 MB
Sistema operativo: Linux Red Hat 6.0 (kernel 2.2.5-15)
Compilador: gcc egcs-2.91.66 / egcs-1.1.2
Opciones de Optimización: -O3 -mcpu=pentiumpro -march=pentiumpro

2)-3) Nombre: cetadfomec1, cetadfomec2 (usualmente mencionadas como cf1 y cf2)
Marca: IBM
Modelo: PC 300GL
Procesador: Celeron 300A
Reloj: 300 MHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna
Cache L2 Tamaño: 128 KB Unificada Ubicación: Interna
Mem. principal: 32 MB
Espacio de swap: 64 MB
Sistema operativo: Linux Red Hat 6.0 (kernel 2.2.5-15)
Compilador: gcc egcs-2.91.66 / egcs-1.1.2
Opciones de Optimización: -O3 -mcpu=pentiumpro \march=pentiumpro

4) Nombre: sofia

Marca: IBM
Modelo: RS/6000 43P-140
Procesador: Power PC 604e
Reloj: 200 MHz
Cache L1 Tamaño: 32 KB datos + 32 KB instr. Ubicación: Interna
Cache L2 Tamaño: 1 MB Unificada Ubicación: Externa
Mem. principal: 64 MB
Espacio de swap: 128 MB
Sistema operativo: AIX 4.3
Compilador: gcc 2.8.1
Opciones de Optimización: -O3 -mcpu=604

5) Nombre: fourier
Marca: Construida por partes
Modelo: PC
Procesador: Pentium MMX
Reloj: 200 MHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna
Cache L2 Tamaño: 512 KB Unificada Ubicación: Externa
Mem. principal: 32 MB
Espacio de swap: 64 MB
Sistema operativo: Winlinux 2000 (kernel 2.2.13)
Compilador: gcc egcs-2.91.66 / egcs-1.1.2
Opciones de Optimización: -O3 -mcpu=pentium -march=pentium

6) Nombre: Josrap
Marca: Construida por partes
Modelo: PC
Procesador: AMD K6-2
Reloj: 450 MHz
Cache L1 Tamaño: 32 KB datos + 32 KB instr. Ubicación: Interna
Cache L2 Tamaño: 512 KB Unificada Ubicación: Externa
Mem. principal: 62 MB
Espacio de swap: 64 MB
Sistema operativo: Winlinux 2000 (kernel 2.2.13)
Compilador: gcc egcs-2.91.66 / egcs-1.1.2
Opciones de Optimización: -O3 -mcpu=pentium -march=pentium

7) Nombre: tilcara
Marca: Construida por partes
Modelo: PC
Procesador: Pentium
Reloj: 133 MHz
Cache L1 Tamaño: 8 KB datos + 8 KB instr. Ubicación: Interna
Cache L2 Tamaño: 256 KB Unificada Ubicación: Externa
Mem. principal: 32 MB
Espacio de swap: 64 MB
Sistema operativo: Linux Red Hat 6.0 (kernel 2.2.5-15)

Compilador: gcc egcs-2.91.66 / egcs-1.1.2
Opciones de Optimización: -O3 -mcpu=pentium -march=pentium

8) Nombre: paris
Marca: Sun
Modelo: SPARCstation 4
Procesador: MicroSPARC-II
Reloj: 110 MHz
Cache L1 Tamaño: 8 KB datos + 16 KB instr. Ubicación: Interna
Mem. principal: 96 MB
Espacio de swap: 160 MB
Sistema operativo: SunOS 5.5.1 - Solaris 2.5.1
Compilador: gcc 2.8.1
Opciones de Optimización: -O3 -mv8

9) Nombre: cetad
Marca: Sun
Modelo: SPARCstation 5
Procesador: MicroSPARC-II
Reloj: 85 MHz
Cache L1 Tamaño: 8 KB datos + 16 KB instr. Ubicación: Interna
Mem. principal: 96 MB
Espacio de swap: 143640 KB
Sistema operativo: SunOS 4.1.4
Compilador: gcc 2.8.1
Opciones de Optimización: -O3 -mv8

10) Nombre: prited
Marca: Sun
Modelo: SPARCstation 2
Procesador: CY7C601
Reloj: 40 MHz
Cache L1 Tamaño: 64 KB Unificada Ubicación: Externa
Mem. principal: 32 MB
Espacio de swap: 65520 KB
Sistema operativo: SunOS 4.1.3
Compilador: gcc 2.8.1
Opciones de Optimización: -O3

A.3.2 Computadoras de la Red Local del LQT

1) Nombre: lqt_01
Marca: Construida por partes
Modelo: PC
Procesador: Pentium III - 550E
Reloj: 550 MHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna

Cache L2 Tamaño: 256 KB Unificada Ubicación: Interna
Mem. principal: 512 MB
Espacio de swap: 64 MB
Sistema operativo: Linux Red Hat 7.2 (kernel 2.4.2-2)
Compilador: gcc egcs-2.95.3
Opciones de Optimización: -O3 -mcpu=pentiumpro -march=pentiumpro

2) Nombre: lqt_02
Marca: Construida por partes
Modelo: PC
Procesador: Celeron
Reloj: 700 MHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna
Cache L2 Tamaño: 128 KB Unificada Ubicación: Interna
Mem. principal: 512 MB
Espacio de swap: 70 MB
Sistema operativo: Linux Red Hat 7.2 (kernel 2.4.2-2)
Compilador: gcc 2.95.3
Opciones de Optimización: -O3 -mcpu=pentiumpro -march=pentiumpro

3) Nombre: lqt_03
Marca: Construida por partes
Modelo: PC
Procesador: Pentium II
Reloj: 400 MHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna
Cache L2 Tamaño: 512 KB Unificada Ubicación: Interna
Mem. principal: 512 MB
Espacio de swap: 128 MB
Sistema operativo: Linux Red Hat 7.2 (kernel 2.4.2-2)
Compilador: gcc 2.95.3
Opciones de Optimización: -O3 -mcpu=pentiumpro -march=pentiumpro

4) Nombre: lqt_04
Marca: Construida por partes
Modelo: PC
Procesador: Pentium II
Reloj: 400 MHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna
Cache L2 Tamaño: 512 KB Unificada Ubicación: Interna
Mem. principal: 512 MB
Espacio de swap: 128 MB
Sistema operativo: Linux Red Hat 7.2 (kernel 2.4.2-2)
Compilador: gcc 2.95.3
Opciones de Optimización: -O3 -mcpu=pentiumpro -march=pentiumpro

6)-7) Nombre: lqt_06, lqt_07
Marca: Construida por partes

Modelo: PC
Procesador: Pentium III
Reloj: 1 GHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna
Cache L2 Tamaño: 256 KB Unificada Ubicación: Interna
Mem. principal: 512 MB
Espacio de swap: 1 GB
Sistema operativo: Linux Red Hat 7.2 (kernel 2.4.2-2)
Compilador: gcc 2.95.3
Opciones de Optimización: -O3 -mcpu=pentiumpro -march=pentiumpro

A.3.3 Computadoras de la Red Local del LIDI

Nombres: lidipar14, lidipar13, lidipar12, lidipar9, lidipar8, lidipar7, lidipar6, lidipar5
Marca: Construida por partes
Modelo: PC
Procesador: Pentium III
Reloj: 700 MHz
Cache L1 Tamaño: 16 KB datos + 16 KB instr. Ubicación: Interna
Cache L2 Tamaño: 256 KB Unificada Ubicación: Interna
Mem. principal: 64 MB
Espacio de swap: 128 MB
Sistema operativo: Linux Red Hat 6.1 (kernel 2.2.12-20)
Compilador: gcc egcs-2.91.66
Opciones de Optimización: -O3 -mcpu=pentiumpro -march=pentiumpro

Apéndice B: Rendimiento de Procesamiento Secuencial de las Computadoras

Este Apéndice se dedica a la descripción del rendimiento de las computadoras que se obtiene en el campo de aplicación definido de los problemas de álgebra lineal, específicamente los que se pueden expresar en función de multiplicaciones de matrices.

La caracterización del rendimiento de las computadoras es un problema muy conocido y estudiado por varias razones, esencialmente para la estimación del tipo de problemas que se pueden resolver en términos de tamaño y tiempo de ejecución.

Si bien es importante la caracterización del rendimiento de las computadoras, también se describe en este Apéndice cómo afecta al rendimiento la optimización del código que se lleva a cabo. Además, se muestra cómo esta optimización tiene efectos variables dependiendo de las características del hardware de cada computadora.

En el caso específico de aplicaciones paralelas, conocer el rendimiento secuencial con precisión es esencial para caracterizar con esa misma precisión la ganancia obtenida por el aporte del procesamiento paralelo. De la misma manera, se puede estimar o al menos se tiene la información mínima necesaria para estimar la relación del costo-beneficio de las computadoras paralelas en general y de las redes de computadoras procesando en paralelo en particular.

B.1 Introducción

La caracterización del rendimiento de las computadoras se ha utilizado con varios propósitos, entre los cuales se pueden mencionar [7]:

- Estimación de la capacidad de resolución de problemas, tanto en lo referente al *tamaño* de los problemas que se pueden resolver como al *tiempo de ejecución* necesarios.
- Verificación del costo de las computadoras, no solamente en cuanto al hardware sino también en cuanto al software de base y de aplicación necesarios.
- Elección de la computadora más adecuada para el problema o clase de problemas que se deben resolver. Implícitamente en este caso se utiliza el índice de rendimiento como un parámetro de comparación de las computadoras a utilizar.

Tradicionalmente, la capacidad de cómputo numérico de una computadora se ha caracterizado con la cantidad de operaciones de punto flotante por unidad de tiempo (Mflop/s: millones de operaciones de punto flotante por segundo) o por un número que lo identifique de forma unívoca (SPEC: [6] [15]). Tradicionalmente también se han tomado dos líneas generales para el cálculo de este índice de rendimiento:

1. Análisis del hardware de procesamiento: unidad/es de punto flotante, diseño de las unidades de punto flotante (pipelines, registros internos, etc.), memoria/s cache (niveles, tamaños, etc.), capacidad de memoria principal, etc.
2. Ejecución de un programa o conjunto de programas específicos de cálculo denominados *benchmarks*.

El análisis del hardware de procesamiento normalmente da lugar a lo que se conoce como *rendimiento pico*, o rendimiento máximo *teórico* de la computadora. Esta línea de caracterización del rendimiento ha sido adoptada normalmente por los fabricantes de las computadoras y también ya es aceptado que es muy poco probable de obtener por una aplicación específica.

La utilización de benchmarks se hizo cotidiana dada la separación existente entre el rendimiento pico y el rendimiento real que las aplicaciones obtienen normalmente en su ejecución en las computadoras. Es muy difícil la elección de un conjunto de programas que logren reunir las características de, o representar a, *toda* la gama de posibles programas que se pueden ejecutar sobre una computadora y por lo tanto existen muchos benchmarks utilizados y aún muchos más propuestos.

Si está bien definido el tipo de aplicaciones específicas sobre el cual se utilizarán las computadoras, sigue siendo muy útil la caracterización en este campo de aplicaciones sin utilizar los benchmarks más generales. Este es el caso de las aplicaciones definidas en términos de las multiplicaciones de matrices [2] [4], y por lo tanto lo más preciso que se puede obtener en este campo es el rendimiento de la multiplicación de matrices misma, que se tomará como el *benchmark* de referencia para la experimentación.

Utilizar un *benchmark* tan específico y tan cercano a la aplicación que se debe resolver tiene, en el contexto de la aplicación paralela, una ventaja más: define con precisión la velocidad relativa de las computadoras para el procesamiento local. Si bien este índice (velocidad relativa de cómputo) no es tan necesario ni importante en el contexto de las

computadoras paralelas con elementos de procesamiento homogéneos, se torna indispensable para el cómputo paralelo con elementos de procesamiento heterogéneos. Sin este tipo de información es muy difícil llegar a tener balance equilibrado de la carga computacional.

B.2 Computadoras Utilizadas

Elegir computadoras para caracterizar rendimiento secuencial e intentar representar con esta caracterización *todos* los casos posibles es casi tan difícil como elegir el conjunto de programas que permitan comparar el rendimiento de las computadoras. En el caso del cómputo paralelo sobre redes locales ya instaladas, está claro que se deben analizar computadoras interconectadas en una red local, pero la variedad de posibilidades es muy grande.

Las tres redes locales sobre las cuales se trabajará son las que se describieron en detalle en el Apéndice A:

- CeTAD: Centro de Técnicas Analógico-Digitales, Departamento de Electrotecnia, Facultad de Ingeniería, Universidad Nacional de La Plata. Es la que está instalada desde hace más tiempo y las computadoras que la componen son utilizadas con múltiples propósitos.
- LQT: Laboratorio de Química Teórica, CEQUINOR, Departamento de Química, Facultad de Ciencias Exactas, Universidad Nacional de La Plata. Es una red destinada a la resolución de problemas numéricos, fue instalada hace varios años y se ejecutan trabajos secuenciales y paralelos desarrollados con PVM y Linda.
- LIDI: perteneciente al Laboratorio de Investigación y Desarrollo en Informática, Facultad de Informática, Universidad Nacional de La Plata. Está dedicada a enseñanza de programación paralela e investigación. Puede considerarse directamente una instalación más del tipo Beowulf, aunque no de las más costosas en cuanto a cantidad de máquinas y red de interconexión.

Tanto la red del CeTAD como la del LQT tienen varias características interesantes para el estudio de rendimiento con el objetivo de utilizarlas para cómputo paralelo:

- Preexistencia: no se construyeron para ser estudiadas sino para ser utilizadas. En este sentido son *reales* y han tenido la evolución en el tiempo que suelen tener las redes locales con la actualización e inclusión de computadoras.
- Heterogeneidad: se puede considerar como una consecuencia de su preexistencia al presente trabajo de investigación, pero es importante remarcar que ambas redes de computadoras tienen máquinas bastante diferentes entre sí al menos en términos de velocidad de cómputo, y en el caso de la red del CeTAD incluso a nivel de arquitectura de las computadoras.
- Dedicadas a cómputo secuencial: si bien en ambas redes locales se ejecutan programas paralelos, al menos en su instalación las computadoras estaban dedicadas a cómputo secuencial. De hecho, varias de las computadoras de la red local del CeTAD existen para tareas estrictamente secuenciales y fueron adaptadas para cómputo paralelo instalando el software necesario.

- Hardware estándar y de bajo costo: tanto las computadoras como las redes de interconexión son ampliamente conocidas y de bajo costo. De hecho, más de una computadora utilizada puede considerarse lista para ser dada de baja.

La red del LIDI provee el marco estándar de estudio de procesamiento paralelo en redes de computadoras. Más específicamente: el hardware de cómputo y de comunicaciones es homogéneo y la red de interconexión puede considerarse apropiada dado que coincide con las ideas básicas de una instalación Beowulf.

En el Apéndice A se describen las tres redes locales con todas las características consideradas importantes de cada una de las computadoras. Por otro lado, es importante (y de alguna manera es uno de los objetivos más importantes de este Apéndice), identificar las características generales de rendimiento en las redes de computadoras más allá de los detalles propios de cada máquina utilizada. Por lo tanto, se intentará llegar a conclusiones más generales de rendimiento sobre hardware de cómputo heterogéneo.

B.3 Descripción de los Experimentos

Se llevaron a cabo tres grandes grupos de experimentos, dependiendo de la forma de codificar-diseñar-implementar la solución al problema:

1. Código sin optimizar. La multiplicación de matrices se codifica con las tres iteraciones clásicas, sin ningún esfuerzo por parte de la implementación ni por el compilador.
2. Código optimizado por el compilador. En este caso, se compila el código sin optimizar (las tres iteraciones) con todas las opciones de optimización disponibles del compilador utilizado para el procesador de la computadora.
3. Código optimizado a nivel del programa fuente. En este caso, el esfuerzo de optimización del código es realizado a nivel del código fuente (sin recurrir a código de máquina), además opcionalmente se pueden utilizar las opciones de optimización del compilador pero no se considera necesario.

Para cada tipo de optimización se calcula el rendimiento obtenido para varios tamaños de matrices posibles. La variación del tamaño del problema a resolver (en requerimientos de memoria y cómputo) es usual en este tipo de *benchmarks* y tiene por objetivo la identificación de posibles dependencias del rendimiento sostenido de una computadora con respecto al tamaño de la aplicación.

En las subsecciones que siguen se describen:

- Las características más importantes junto con el/los objetivos de cada uno de los tipos de optimizaciones. Siempre en el contexto de procesamiento secuencial y de los problemas numéricos en general y de procesamiento de matrices en particular.
- Los tamaños de matrices elegidos para estimar el rendimiento de las computadoras, y los criterios por los cuales se eligen.
- Las características más importantes de la experimentación en cada computadora, que de alguna manera es una visión de mayor nivel de abstracción para los dos puntos anteriores.

B.3.1 Código sin Optimización

El rendimiento obtenido con la multiplicación de matrices sin optimizar en realidad no tiene gran utilidad como índice de rendimiento ya que sin ningún esfuerzo por parte del programador se puede obtener algo mejor utilizando correctamente las opciones de compilación disponibles. En este caso es necesario solamente conocer bien el compilador y es muy útil conocer las características del procesador.

El rendimiento obtenido con el código sin optimizar se utilizará con dos objetivos:
1. Como referencia de ganancia en cada computadora para los otros dos casos de optimización: del compilador y del código fuente.
2. Como referencia para comparación de los efectos de optimización en computadoras heterogéneas.

B.3.2 Optimizaciones del Compilador

En el caso de las optimizaciones del compilador, no se tienen grandes desventajas a nivel conceptual. Como se ha explicado, se debe conocer el compilador y las características del procesador relacionadas con las operaciones aritméticas y/o diseño de la/s memoria/s cache. Sin embargo, siempre es conveniente tener en cuenta que usualmente el mejor compilador para cada computadora es el que provee el fabricante. De acuerdo a la evolución en el desarrollo y comercialización de las computadoras, inicialmente el compilador era provisto por la empresa fabricante sin costo adicional. Desde hace algunos años, el compilador es optativo y debe comprarse la licencia al fabricante (del código ejecutable y las librerías propias) por separado. Esta situación, tiene al menos dos consecuencias en la actualidad:
1. Muchas de las computadoras instaladas no tienen el compilador (más apropiado) propietario de la empresa fabricante de la computadora.
2. Se hizo más popular aún la utilización de compiladores de uso libre tales como gcc/gcc-egcs para el lenguaje C, que se pueden obtener vía Internet en código binario y/o fuente.

Desde el principio se ha establecido que se pretenden utilizar las computadoras tal como están instaladas, y en todo caso con el mínimo costo de instalación de software (en particular, compiladores) adicional. Por lo tanto, no solamente es recomendable utilizar las computadoras con un compilador de uso libre como gcc/gcc-egcs sino que además se debería cuantificar su eficiencia en cada máquina. El hecho de caracterizar el rendimiento de las computadoras con un compilador en particular de alguna manera también está caracterizando el rendimiento del compilador mismo. Más allá de las conclusiones particulares a las que se llegue respecto del compilador, y que en principio no son materia de estudio en esta tesis, se llega a que:
- En general, el compilador que instalado en cada máquina será el utilizado. Se asume que es lo *mejor* que puede haber o por lo menos lo que tiene mínimo costo.
- La combinación computadora-compilador utilizados puede no ser la mejor y por lo tanto es útil para caracterizar la computadora en particular, pero no para comparar máquinas

en general (a nivel de modelos de computadoras, por ejemplo).

De acuerdo con esta última conclusión se adopta en todos los gráficos de caracterización de rendimiento (en este Apéndice y en todos los de experimentación) la norma de hacer referencia a las computadoras por sus nombres (*hostnames*), sin referencia a marcas y/o modelos. Si bien los nombres de las máquinas no tienen ninguna información (*a priori*) relacionada con el rendimiento y eso puede oscurecer un poco la interpretación de resultados, no es correcto caracterizar rendimiento a nivel de marcas-modelos de computadoras cuando el compilador utilizado no es el *mejor* (asumiendo que el *mejor* es el provisto por el fabricante de la máquina) que se puede utilizar en cada computadora. Por otro lado, en el Apéndice A se dan *todos* los detalles de cada una de las computadoras y en la interpretación de los resultados se hará mención explícita a cada uno de los que se consideren importantes.

B.3.3 Optimización del Código Fuente

Es muy difícil asegurar *a priori* que la optimización del código fuente que se haga es efectiva. Sin embargo, en el campo del cómputo numérico en general existe mucha experiencia (y publicaciones) respecto de las formas de aprovechar al máximo las características del hardware de procesamiento. Algunas de las optimizaciones que ya se consideran estándares a nivel de código fuente son [3] [1] [10] [11] [5]:

- Utilización intensiva de los registros internos dedicados a datos numéricos.
- Procesamiento por bloques para el aprovechamiento intensivo de los datos asignados en memoria/s cache/s.
- Aprovechamiento máximo de las unidades de ejecución implementadas con *pipelines* (identificación de los saltos y dependencias).
- Intercalación de instrucciones que operan con datos de punto flotante y con datos enteros, para el máximo aprovechamiento posible de los procesadores superescalares.
- Identificación y *separación* de las operaciones con dependencias de datos que reducen el rendimiento de los procesadores superescalares.

En el caso particular de las multiplicaciones de matrices, ya se tienen disponibles numerosas fuentes de optimización, y numerosas fuentes de información, inclusive a nivel de código fuente disponible en Internet. Por un lado, se puede llegar a esperar que cada procesador tenga provisto su propio conjunto de rutinas de cómputo numérico, como el caso particular del Pentium III y Pentium 4 de Intel, que disponen de numerosas rutinas en código fuente (aunque en lenguaje de ensamblador, a nivel de instrucciones del procesador) [9] disponibles, que se pueden obtener vía Internet [13].

Si bien las bibliotecas optimizadas por las propias empresas fabricantes de procesadores pueden ser muy atractivas (y de uso libre), de todas maneras subsisten al menos dos inconvenientes:

- No todos los procesadores tienen disponible tal tipo de código-bibliotecas.
- El código fuente suele ser bastante dependiente de un compilador en particular, en el caso anterior de Intel se necesitan compiladores propietarios (cuya utilización depende de comprar una licencia) tanto del lenguaje C como de lenguaje de ensamblador.

Por otro lado, ya hay disponible también en Internet numerosas librerías que generan código optimizado en general, normalmente en lenguaje C o FORTRAN para ser compiladas localmente en las computadoras independientemente del compilador instalado [3]. Uno de los proyectos más significativos al respecto, y aún en desarrollo es el denominado ATLAS (Automatically Tuned Linear Algebra Software) [12] [14]. Como casi la mayoría de los paquetes en el campo del álgebra lineal comenzó dedicado exclusivamente a la multiplicación (secuencial) de matrices y posteriormente se extendió a todo LAPACK Level 3 [4]. Esta alternativa es satisfactoria, por varias razones:

- Es de uso libre, el *único* costo es el de instalación. Esto normalmente implica instalar código que se obtiene en Internet y que ejecuta un conjunto de programas (*scripts*) para identificar el hardware de procesamiento y optimizar el código de acuerdo a lo estimado por estos programas. Finalmente quedan disponibles un conjunto de bibliotecas que contienen las rutinas optimizadas y que se pueden utilizar desde los programas.
- Los conceptos de optimización son suficientemente claros como para ser implementados independientemente de una biblioteca o distribución en particular. Esto significa que ni siquiera es necesario utilizar una biblioteca disponible en Internet sino que se puede desarrollar el código apropiado con su consiguiente costo.
- Es medianamente independiente del procesador, ya que se puede hacer en código fuente como C o FORTRAN y por lo tanto se puede compilar localmente en cada computadora.

Dado que el código se optimiza sin hacer uso específico de un compilador, las opciones de optimizaciones propias del compilador no suelen mejorar mucho el código ejecutable. Una vez más, es necesario recordar que estas optimizaciones son apropiadas para este tipo de procesamiento matricial y no necesariamente puede hacerse en todos los casos o para todas las aplicaciones en general.

La optimización de código fuente también será denominada *optimización completa*, dado que en general es lo mejor que se puede hacer para obtener rendimiento óptimo o cercano al óptimo de cada computadora.

B.3.4 Tamaños de Matrices a Multiplicar

Los tamaños elegidos de las matrices a multiplicar (se asumen matrices cuadradas, de $n \times n$ elementos) tienen relación con las características del procesamiento de matrices y del subsistema de memoria de cada computadora. A modo de ejemplo: si las matrices son suficientemente pequeñas como para ser asignadas completamente en el primer nivel de memoria cache (L1 Cache), el rendimiento sostenido será muy satisfactorio y con valores cercanos al óptimo teórico del procesador. Si, por el contrario, las matrices no pueden ser asignadas en ninguno de los niveles de memoria cache, el rendimiento dependerá en forma casi directa con el patrón de acceso a los datos para ser procesados.

Dada la heterogeneidad de las computadoras con las que se experimenta (Apéndice A), y en particular los diferentes tamaños y niveles de memoria cache de las computadoras, se tomaron como referencia varios tamaños de matrices relativamente pequeños con respecto al tamaño de memoria principal: matrices de orden n = 100, 200, 400. Los datos numéricos se representan con números en punto flotante de precisión simple (norma IEEE 754 [8]) de

cuatro bytes. Por lo tanto, para n = 100, la cantidad de datos necesaria para almacenar una matriz será de $100^2 \times 4$ bytes, un poco menos de 40 KB de datos.

Dos valores se tomaron como representativos de los tamaños de matrices que se pueden manejar en memoria principal de 32 MB: matrices de orden n = 800 y de orden n = 1600. Con matrices de 800×800 elementos, la cantidad de memoria requerida para contener las tres matrices que intervienen en una multiplicación ($C = A \times B$) es de aproximadamente 7.3 MB (22.8% del total de 32 MB de memoria principal aproximadamente). En el caso de matrices de 1600×1600 elementos, la cantidad aproximada de memoria que se requiere es 29.3 MB, lo que representa el 91.6% del total de 32 MB de memoria principal.

En el caso de las computadoras con 32 MB de memoria principal se puede considerar suficiente el tamaño del problema con n = 1600, o por lo menos es suficientemente grande como para que el tamaño de la memoria cache no sea suficiente para contener una parte relativamente grande del problema. Sin embargo, teniendo en cuenta que hay máquinas con 512 MB de memoria principal, se buscaron valores de n suficientemente grandes para ocupar casi completamente la memoria principal.

Por lo tanto, en todas las computadoras se realizaron los experimentos con matrices cuadradas de orden n = 100, 200, 400, 800 y 1600. En el caso de las computadoras con memoria principal de 64 MB o 512 MB, y para tener valores de referencia para ser utilizados en el cálculo de speedup, se llevaron a cabo experimentos con matrices mayores. Además, dado que siempre se tiende a la utilización de las computadoras en el límite de su capacidad, también se llevaron a cabo experimentos con el máximo posible en cuanto a tamaño de las matrices. Como es de suponer, esto depende no solamente del tamaño de la memoria principal instalada sino también del espacio de swap configurado en el sistema.

Para las computadoras de 64 MB de memoria principal, los tamaños considerados representativos de los problemas que requieren una buena parte o toda la memoria principal corresponden a valores de n = 1900, 2000, 2200 y 2400. Estos tamaños de matrices implican los porcentajes aproximados de requerimientos de memoria (asumiendo 64 MB en total) de: 65%, 72%, 87%, y 103% respectivamente. Se debe recordar que es posible experimentar con los valores cercanos y superiores al 100% de requerimientos de memoria principal dependiendo del tamaño de memoria swap configurada.

En las computadoras de 64 MB de memoria principal, el tamaño máximo con el cual se pudo llevar a cabo la multiplicación de matrices es para n = 3200, y como referencia se hicieron también experimentos con n = 3000. Estos tamaños de matrices implican los porcentajes aproximados de requerimientos de memoria (asumiendo 64 MB en total) de: 183% y 161% respectivamente. Como se puntualizó antes, los tamaños máximos del problema dependen de tres aspectos:

- memoria principal instalada.
- espacio de swap configurado.
- sistema operativo, ya que es éste el que en última instancia decide cuándo cancelar un proceso por falta de memoria.

Y estos tres aspectos coinciden al menos en la máquinas más rápidas de la red del CeTAD y de la red del LIDI.

Para las computadoras de 512 MB de memoria principal, los tamaños considerados representativos de los problemas que requieren una buena parte o toda la memoria principal corresponden a valores de n = 4000, 5000, 6000 y 7000. Estos tamaños de matrices implican los porcentajes aproximados de requerimientos de memoria (asumiendo 512 MB en total) de: 36%, 56%, 80%, y 110% respectivamente. Se debe notar que es posible experimentar con los valores cercanos y superiores al 100% de utilización de memoria principal para almacenar los datos dependiendo del tamaño de memoria swap configurada.

En las computadoras de 512 MB de memoria principal, el tamaño máximo con el cual se pudo llevar a cabo la multiplicación de matrices es para n = 9000, y como referencia se hicieron también experimentos con n = 8000. Estos tamaños de matrices implican los porcentajes aproximados de requerimientos de memoria (asumiendo 512 MB en total) de: 181% y 143% respectivamente.

En resumen, en las computadoras más rápidas de cada red local se experimentó en los límites de la capacidad total de memoria. En todos los casos, cuando se muestra el rendimiento de cada computadoras se muestra también cuál es el máximo tamaño de matrices que se puede resolver sin hacer uso del *swapping* de páginas de memoria.

B.4 Rendimiento de la Multiplicación de Matrices

En las subsecciones que siguen se muestran los tiempos de ejecución y también el rendimiento expresado en Mflop/s obtenidos en cada una de las computadoras que resuelven una multiplicación de matrices según el tipo de optimización realizada. Cuando se considera necesario, se incluyen también algunos comentarios que explican los valores de los índices de rendimiento de las computadoras. Dado que las ocho computadoras de la red local del LIDI son iguales, se muestran todos los datos de la experimentación solamente para una de ellas. Finalmente, se incluye una subsección más dedicada a la comparación del rendimiento entre los distintos tipos de optimización elegidos.

B.4.1 Rendimiento sin Optimización

La Figura B.1 muestra los tiempos de ejecución (registrados en segundos) obtenidos para la multiplicación de matrices (cuadradas) de diferentes tamaños en las computadoras del CeTAD. La Figura B.2, muestra el tiempo de ejecución en las computadoras del LQT. En el eje **x** del gráfico se muestra el valor de n (tamaño de las matrices) y en el eje **y** se muestra el tiempo de ejecución en valores logarítmicos.

Si bien tanto la Figura B.1 como la Figura B.2 permiten tener una idea aproximada de los tiempos de ejecución necesarios en cada una de las máquinas, la escala logarítmica y el mismo índice de rendimiento definido como tiempo de ejecución puede complicar la interpretación de los resultados. De todas maneras, se puede identificar rápidamente que para un tamaño de matrices definido, las diferencias de velocidad de cómputo son notables.

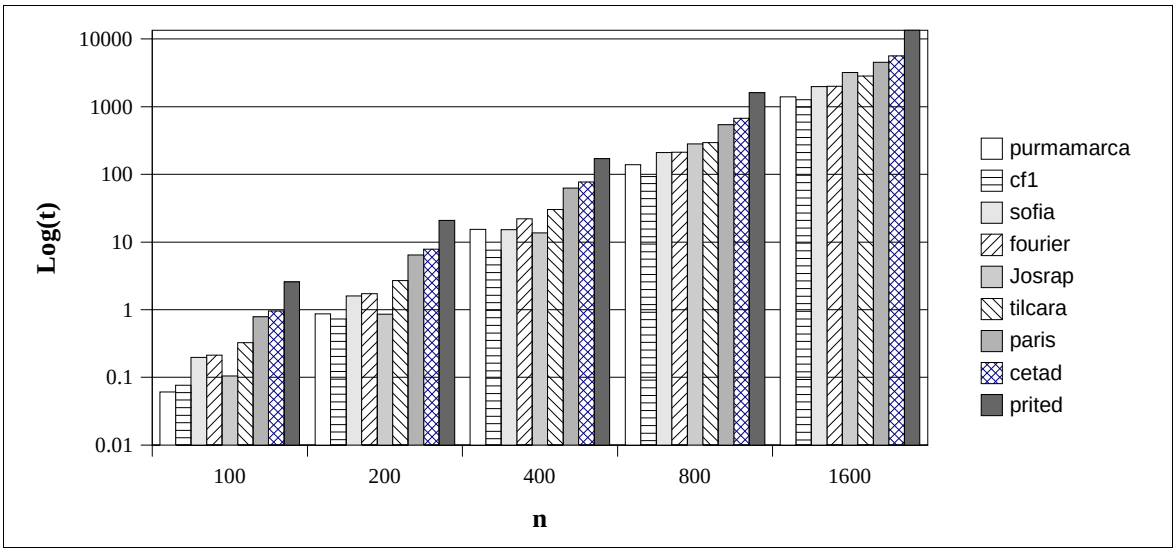


Figura B.1: Tiempos de Ejecución en el CeTAD sin Optimización.

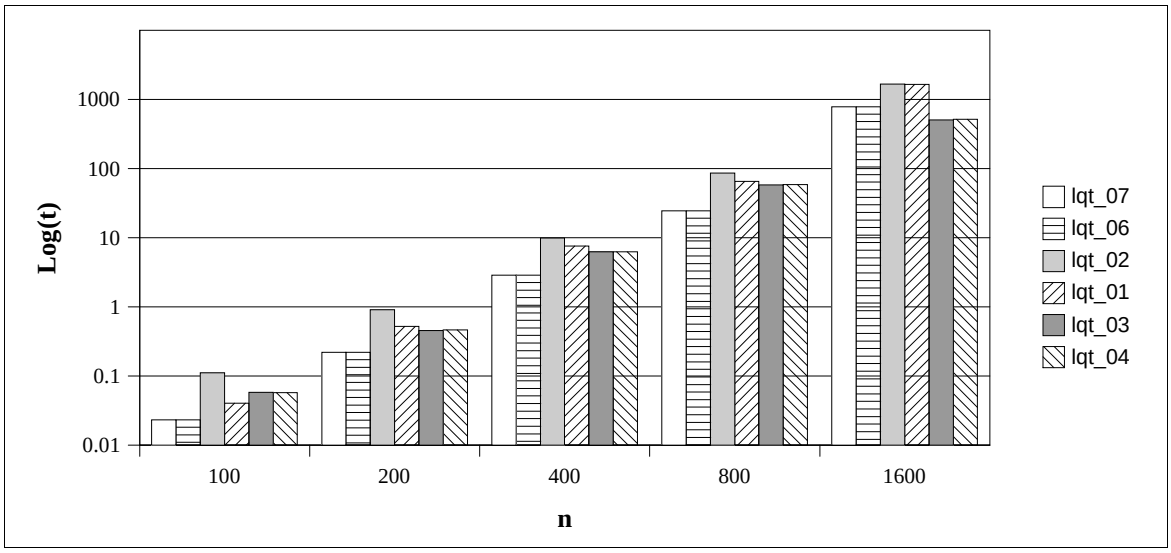


Figura B.2: Tiempos de Ejecución en el LQT sin Optimización.

La Figura B.3 así como la Figura B.4 muestran los mismos experimentos pero identificando los Mflop/s obtenidos en computadora. Lo más notable que se puede identificar con facilidad son las diferencias de velocidad de cómputo de cada máquina que queda bastante enmascarada en el caso de los gráficos de tiempo con escala logarítmica.

En todas las computadoras es notable el efecto que tiene el tamaño de la memoria cache y el tamaño del problema a resolver sobre el rendimiento. En todos los casos, a medida que la cantidad de datos crece, la probabilidad de reusar un dato asignado en la memoria cache disminuye dado que no se establece a priori ningún patrón de acceso a los datos para aprovechar la jerarquía de memoria con uno o más niveles de memoria cache.

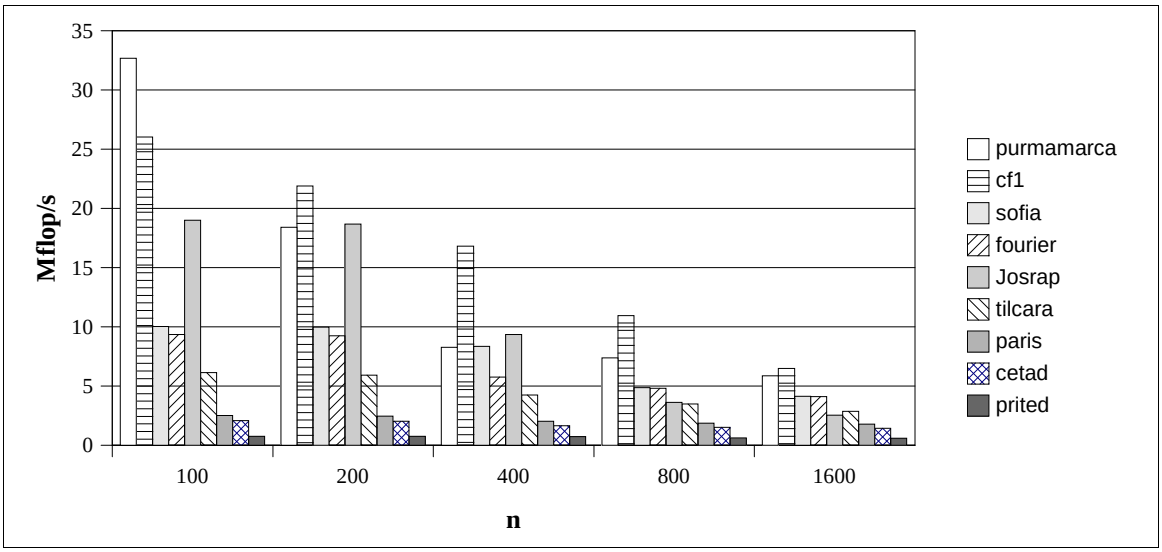


Figura B.3: Mflop/s en el CeTAD sin Optimización.

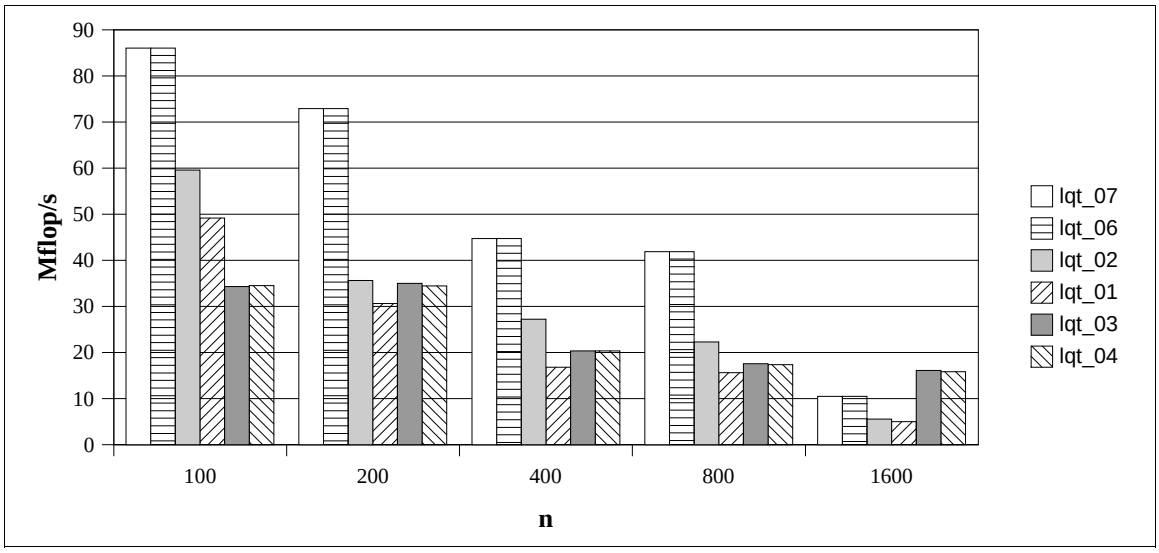


Figura B.4: Mflop/s en el LQT sin Optimización.

Es interesante notar que a medida que los procesadores son más veloces, el impacto sobre la disminución del rendimiento cuando el problema crece es mayor, dado que la diferencia relativa de velocidad de memoria y de procesamiento crece proporcionalmente. Por ejemplo, **lqt_01** pasa de casi 50 Mflop/s con matrices de 100x100 a 5 Mflop/s con matrices de 1600x1600 lo que implica utilizar solamente una décima parte del rendimiento posible tal como fue medido para los problemas con matrices de orden n = 100.

Las diferencias de velocidad relativa entre el acceso a memoria y el procesamiento generan a su vez diferencias relativas de velocidad entre las computadoras que dependen del tamaño del problema a resolver. Por ejemplo, para n = 100, la computadora con denominada **paris** (Figura B.3), tiene casi el doble de la capacidad de cálculo de **Josrap** y para n = 1600 **Josrap** tiene mayor rendimiento que **paris**. Si bien para cómputo secuencial esto da una idea de velocidad relativa diferente para distintos tamaños de problemas,

cuando se trata de cómputo paralelo tiene impacto directo en el balance de carga computacional, que debería ser resuelto por la aplicación en función del tamaño del problema de procesamiento de datos.

B.4.2 Rendimiento con Optimizaciones del Compilador

Como se explicó anteriormente, los programas de aplicación que se ejecutan normalmente tienen el grado de optimización mínimo que provee el compilador para la computadora (más específicamente, para el procesador). En todas las máquinas reportadas, el compilador utilizado es *gcc/gcc-egcs* y por lo tanto las opciones no varían de forma significativa en las diferentes máquinas. Sin embargo, se debe tener en cuenta que en un ambiente heterogéneo la variedad de compiladores puede llegar a ser igual a la cantidad de máquinas que se utilizan y por lo tanto conocer todos los detalles de los compiladores (procesadores-optimizaciones) es igualmente complejo.

La Figura B.5 y la Figura B.6 muestran los tiempos de ejecución en cada una de las máquinas del CeTAD y del LQT respectivamente. Como en las figuras anteriores donde se muestran tiempos de ejecución, el eje **x** del gráfico corresponde a distintos tamaños de matrices (desde matrices de orden n = 100, hasta matrices de orden n = 1600), y el eje **y** del gráfico muestra el tiempo de ejecución con escala logarítmica.

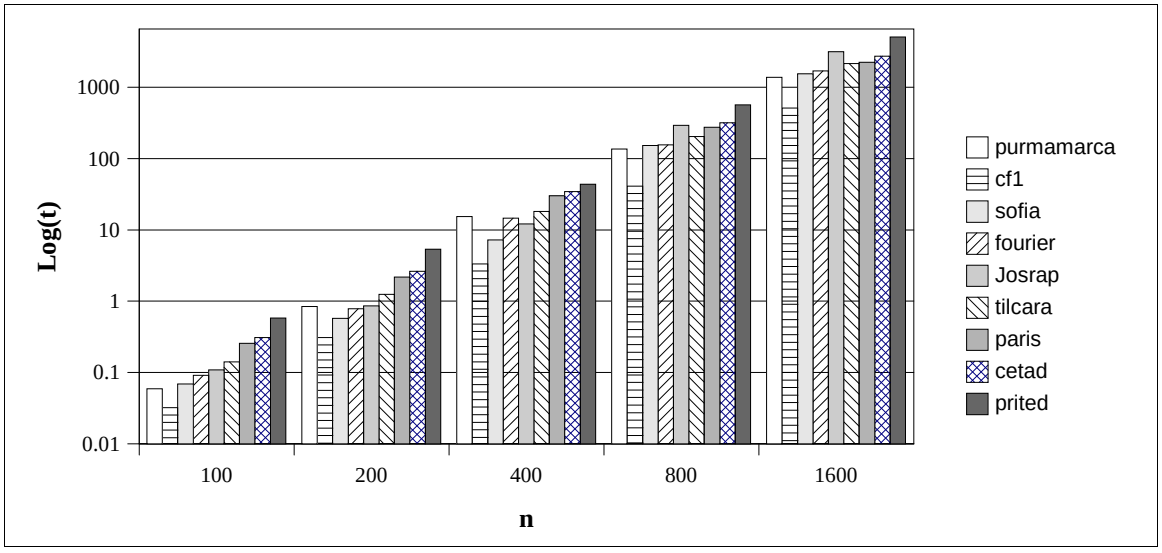


Figura B.5: Tiempos de Ejecución en el CeTAD con Optimización del Compilador.

Comparando la Figura B.5 con la Figura B.1 se puede notar que en muchas computadoras el tiempo total de ejecución disminuye para todos los tamaños de matrices con los cuales se realizaron los experimentos. De la misma manera se puede comparar la Figura B.6 con la Figura B.2. Una vez más, las diferencias relativas de velocidad son difíciles de identificar con precisión por la escala logarítmica en la que se muestran los tiempos de ejecución.

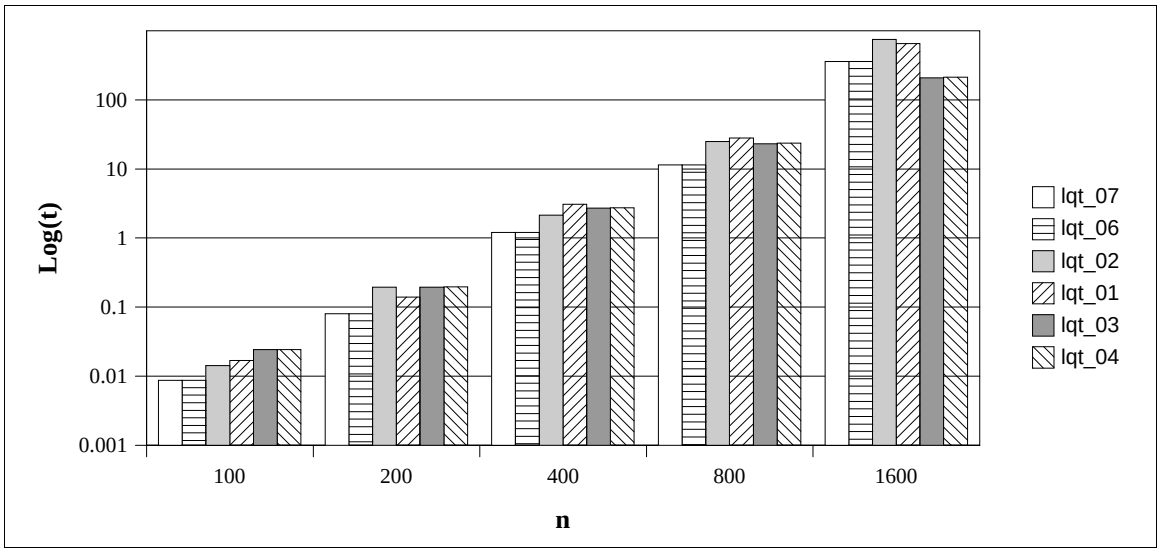


Figura B.6: Tiempos de Ejecución en el LQT con Optimización del Compilador.

La Figura B.7 y la Figura B.8 muestran el rendimiento de cada computadora en Mflop/s, donde se pueden identificar mejor las diferencias con respecto a la ejecución sin ninguna optimización. Dependiendo de la computadora y del tamaño del problema que se resuelve el rendimiento mejora en algunos casos más del 100%: **lqt_01** pasa de poco menos de 50 Mflop/s para *n* = 100 (Figura B.4) a poco menos de 120 Mflop/s para el mismo tamaño de problema (Figura B.8).

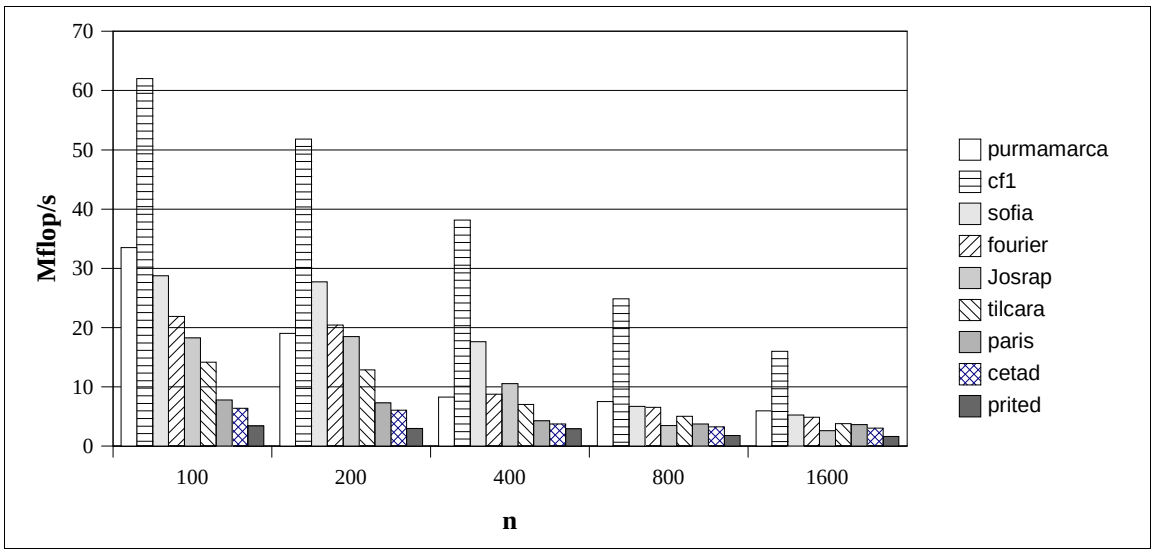


Figura B.7: Mflop/s en el CeTAD con Optimización del Compilador.

Aunque el rendimiento mejora de forma notable en algunos casos como el mencionado y en general mejora en todas las computadoras sigue siendo importante el peso del tamaño del problema en el rendimiento obtenido. De la misma manera, la disminución de rendimiento que tienen las máquinas a medida que aumenta el tamaño del problema tiene distintas características en cada una de ellas y por lo tanto se sigue verificando que las

diferencias de velocidad relativa de las máquinas es dependiente del tamaño del problema que se resuelve.

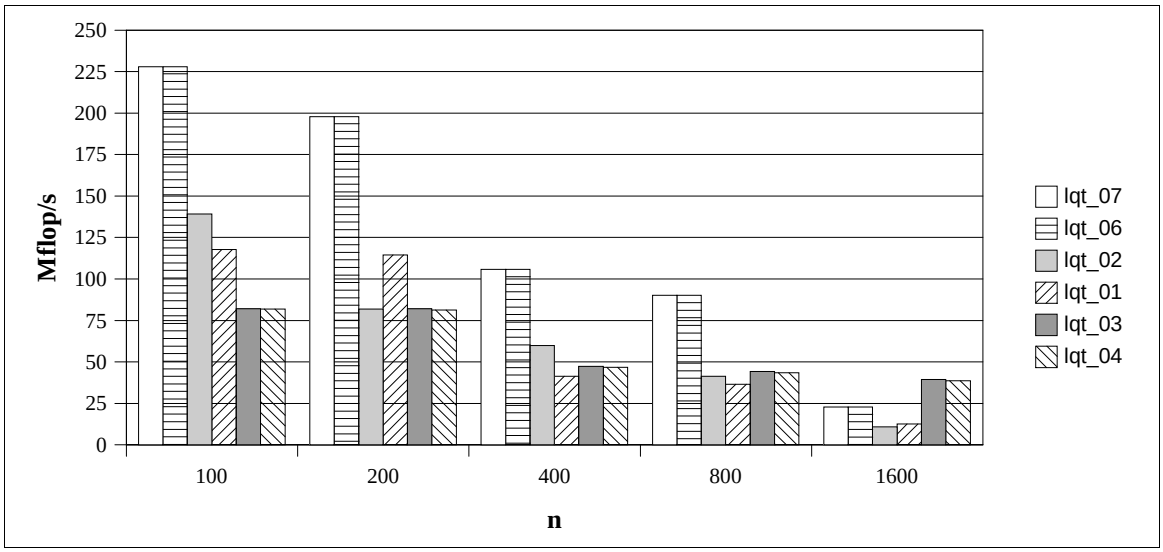


Figura B.8: Mflop/s en el LQT con Optimización del Compilador.

Dado que es muy difícil que un compilador pueda realizar todas las optimizaciones posibles [12], es muy importante contar con código especialmente optimizado para las computadoras que se utilizan. Más ventajoso aún es aprovechar código disponible especialmente optimizado para las computadoras. En la subsección que sigue se muestran los experimentos realizados con este tipo de código y se muestran diferencias muy importantes a nivel de aumento de rendimiento y a nivel conceptual de rendimiento en general de las máquinas que ejecutan código de procesamiento para realizar operaciones provenientes de álgebra lineal.

B.4.3 Rendimiento con Optimización del Código Fuente

La Figura B.9 y la Figura B.10 muestran los tiempos de ejecución de las computadoras del CeTAD y del LQT respectivamente, para cada uno de los tamaños de matrices cuando el código fuente se optimiza para obtener el mejor rendimiento posible. Comparando la Figura B.9 con la Figura B.5 y la Figura B.10 con la Figura B.6 se puede comprobar rápidamente una disminución general del tiempo total de ejecución (en todas las computadoras y para todos los tamaños de problema) muy importante.

A modo de ejemplo, en la Figura B.5 se muestra que la computadora del CeTAD **purmamarca** utiliza aproximadamente un segundo para resolver una multiplicación de matrices de 200×200 elementos. En la Figura B.9 se muestra que el mismo problema en la misma computadora se resuelve en menos de una décima de segundo. Por otro lado, en la Figura B.6 se muestra que las computadoras del LQT **lqt_06** y **lqt_07** utilizan bastante más de cien segundos para resolver una multiplicación de matrices de 1600×1600 elementos. En la Figura B.10 se muestra que el mismo problema en las mismas computadoras se

resuelve en poco más de diez segundos.

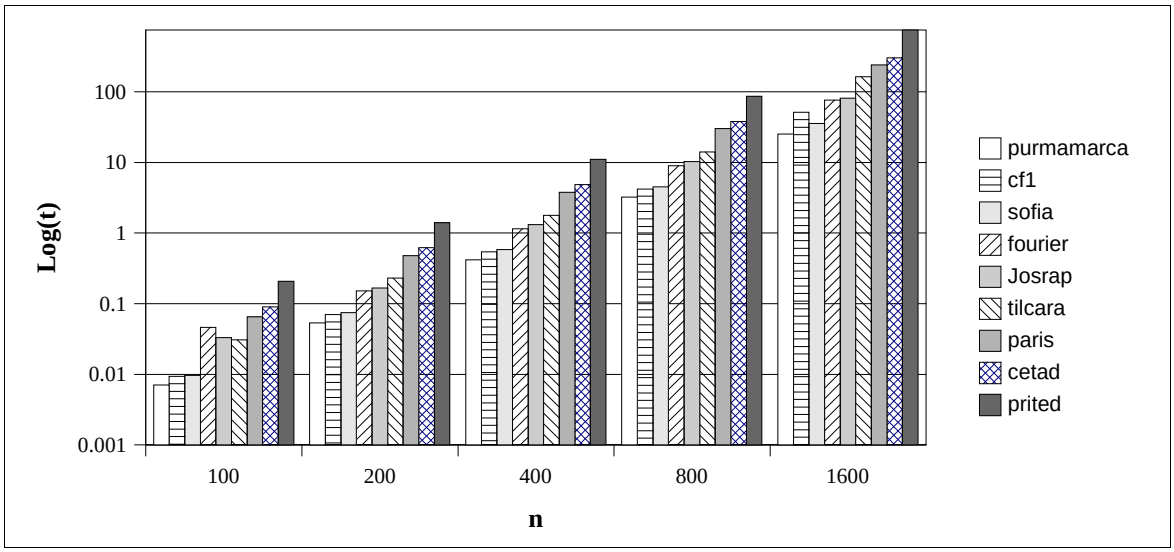


Figura B.9: Tiempos de Ejecución en el CeTAD con Optimización Completa.

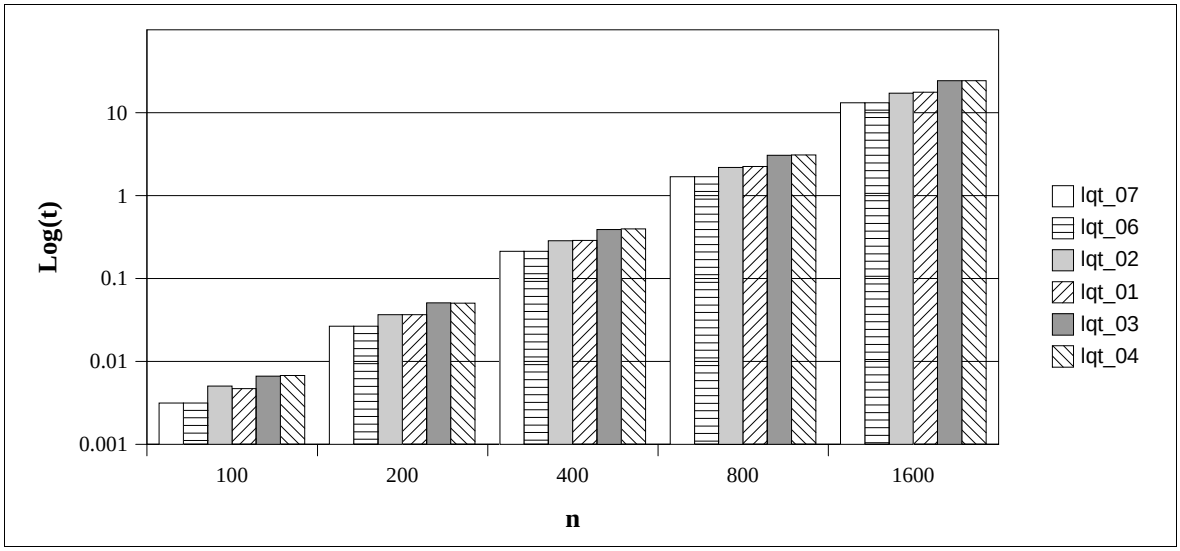


Figura B.10: Tiempos de Ejecución en el LQT con Optimización Completa.

El rendimiento de cada una de las computadoras expresado en Mflop/s se muestra en la Figura B.11 para las máquinas del CeTAD y en la Figura B.12 para las máquinas del LQT. Si se comparan los valores de rendimiento de las máquinas del CeTAD que se muestran en la Figura B.11 (código completamente optimizado) con los de la Figura B.7 (código optimizado por el compilador) y con los de la Figura B.3 (código sin ninguna clase de optimización), las diferencias del rendimiento son muy notables. Estas diferencias no solamente se pueden verificar en cuanto a los valores absolutos, que es idéntico a lo que sucede con el tiempo de ejecución, sino en cuanto a las variaciones de rendimiento dependiendo del tamaño del problema que se resuelve y la relación entre las capacidades relativas de cómputo de cada máquina.

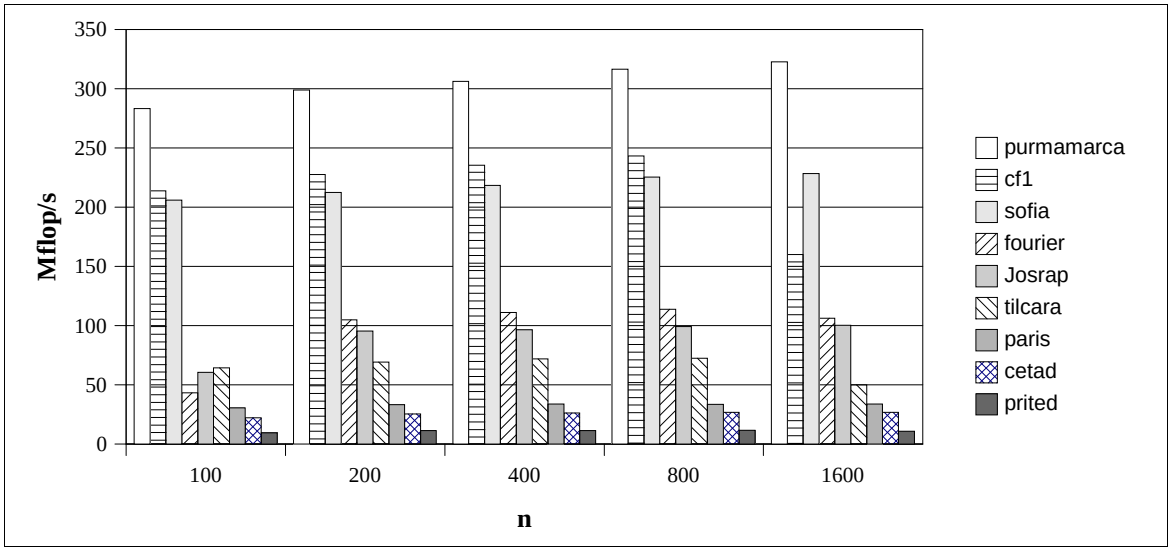


Figura B.11: Mflop/s en el CeTAD con Optimización Completa.

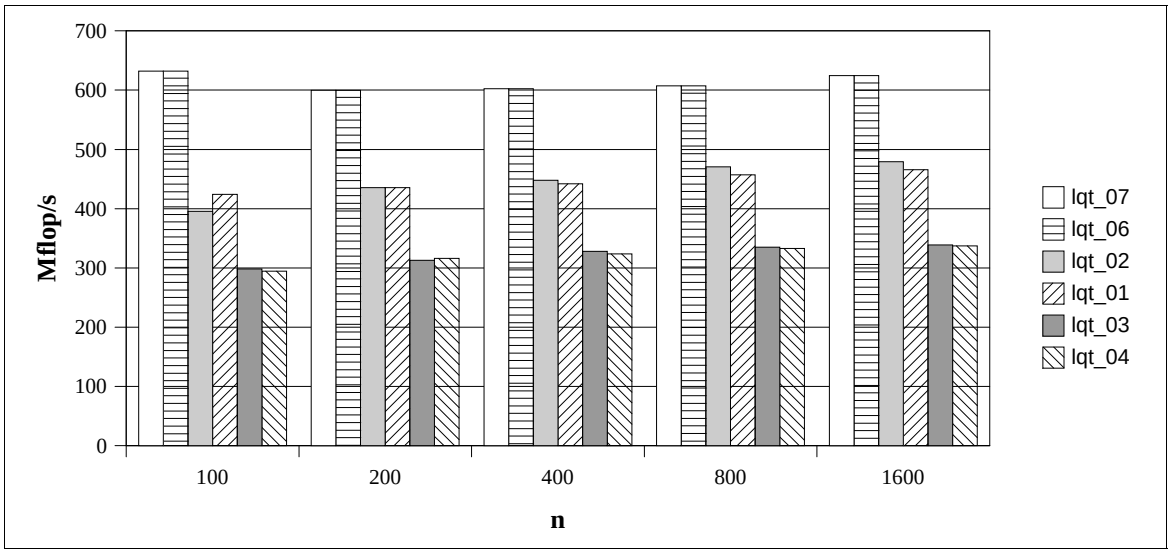


Figura B.12: Mflop/s en el LQT con Optimización Completa.

A partir de estas dos últimas figuras, se pueden identificar características que son casi tan importantes como el aumento de los valores absolutos del rendimiento de todas las máquinas:

El Rendimiento no disminuye a medida que el tamaño del problema crece: las dos figuras lo muestran claramente para casi todas las computadoras. A pesar de la gran heterogeneidad, el rendimiento sigue esta "regla" sin importar procesadores ni ciclos de reloj. Las excepciones que se pueden identificar aparecen en la Figura B.11 y son las computadoras **tilcara** y **cf1**. En ambos casos el rendimiento se degrada notablemente para el valor de n = 1600. En el caso de **tilcara**, pasa de poco más de 70 Mflop/s para n = 800 a 50 Mflop/s (disminución de casi 30% del rendimiento) para n = 1600 y **cf1** pasa de poco

más de 240 Mflop/s a aproximadamente 160 Mflop/s (disminución de casi el 35%) para los tamaños de problema $n = 800$ y $n = 1600$ respectivamente. En ambos casos se debe a la relación entre el tamaño de memoria principal y el tamaño del problema. Para $n = 800$, la cantidad de memoria requerida para las matrices es de poco más de 7 MB y por lo tanto hay espacio suficiente en todas las máquinas, mientras que para $n = 1600$ la cantidad de memoria requerida es de más de 29 MB. Tanto en **tilcara** como en **cf1** (**cetadfomec1**) sucede algo similar: el reporte de actividad del sistema operativo muestra que deben recurrir al espacio de memoria swap aunque su memoria principal es de 32 MB y en principio pueden contener todos los datos que se procesan. Se debe recordar que en la memoria principal se debe contener el kernel del sistema operativo (Linux), partes de memoria que no es posible desalojar al espacio de swap (buffers del sistema operativo, por ejemplo), y un mínimo de código de la aplicación misma. Además, dado que la potencia de cálculo del procesador de **cf1** (Celeron 300 MHz) es más de tres veces mayor que la potencia de **tilcara** (Pentium 133 MHz), es de esperar que el impacto en la disminución del rendimiento sea mayor.

El rendimiento es casi constante o aumenta levemente cuando el problema es mayor. Se puede comprobar en ambas figuras con las excepciones mencionadas, para cada una de las computadoras (una vez más, independientemente de los procesadores y de la heterogeneidad de las máquinas). La base para el cumplimiento de esta regla es el procesamiento por bloques que se realiza sobre los datos. Cada vez que se asigna un dato en memoria cache es reutilizado al máximo, dado que el patrón de acceso a los datos (en memoria) se codifica especialmente con este fin. Esto implica que aumentar la cantidad de datos no aumenta la cantidad de fallos en memoria cache, y más aún, aumentar la cantidad de datos en el caso de las operaciones matriciales y en particular en las multiplicaciones de matrices implica que se puede aumentar la cantidad de accesos a memoria cache con los mismos datos. Es esta manera se puede explicar el incremento en el rendimiento cuando el tamaño de las matrices crece, ya que la asignación de datos a cache sigue siendo un acceso a memoria principal pero una vez en memoria cache se puede reutilizar más veces (aumentando de esta manera la cantidad de “hits” de memoria cache) porque hay más operaciones a realizar por cada dato.

La relación entre las velocidades de cálculo de las computadoras es casi constante, sin variaciones mayores al 10%. Esta característica se puede identificar como una consecuencia del rendimiento casi constante o aumento leve del rendimiento que se explicó anteriormente, pero en el contexto de las aplicaciones paralelas que se resuelven en computadoras heterogéneas es muy importante y necesariamente se debe analizar. Por ejemplo, en la Figura B.12 se puede notar que la computadora **lqt_02** tiene una capacidad de cálculo aproximada de un tercio mayor que lqt_03 en el procesamiento de las matrices de 100×100, y no sólo esto sino que esta relación de velocidades se mantiene aproximadamente constante de manera independiente de los tamaños de matrices que se procesan.

Desde el punto de vista de las aplicaciones paralelas esta última característica que se nota en la experimentación implica una simplificación notable en la forma en que se resuelve el balance de carga computacional. Si se acepta un desbalance de carga computacional del 10% por ejemplo, la forma en que se distribuye la carga se puede implementar de manera independiente del tamaño del problema que se debe resolver en cada computadora, y por lo

tanto se elimina un parámetro de sintonización o al menos se disminuye notablemente el rango de variación.

B.4.4 Matrices Mayores en las Computadoras Con Mayor Capacidad de Cómputo

Muchas de las aplicaciones de cálculo intensivo normalmente tienden a ocupar una gran cantidad de memoria (procesamiento de grandes volúmenes de datos). Por otro lado, también es de esperar que las aplicaciones aprovechen todos (o la mayoría de) los recursos de las computadoras, en particular la memoria principal disponible. Es por esto que se realizaron experimentos para caracterizar el rendimiento de las máquinas cuando el problema a resolver ocupa toda la memoria e incluso cuando los requerimientos son mayores que la memoria principal instalada y se debe recurrir a la utilización del espacio de *swap* configurado.

Las máquinas elegidas para la realización de estos experimentos fueron las de mayor potencia de cálculo de cada una de las redes, es decir **purmamarca** (Pentium II 400 MHz, 64 MB) del CeTAD y **lqt_07** (Pentium III 1 GHz, 512 MB) del LQT. Los experimentos se diseñaron con dos propósitos, utilizando código completamente optimizado:

- Caracterizar el rendimiento para el mayor tamaño de problema que pueda contener la memoria principal.
- Caracterizar el rendimiento para el mayor tamaño de problema que se pueda resolver (incluyendo espacio de memoria *swap*).

La Figura B.13 muestra el rendimiento de **purmamarca** para distintos tamaños de matrices, incluyendo los que implican la utilización de memoria swap durante los cálculos de la multiplicación de matices $C=A\times B$. A partir de las matrices de orden n = 1600 inclusive se muestra la proporción de memoria principal aproximada que se requiere para contener todos los datos del problema. Por ejemplo para n = 1900, el 65% (aproximadamente) de la memoria principal se utiliza para contener los datos de las matrices.

En la Figura B.13 también se muestra el mayor valor de n para el cual todos los datos pueden residir en memoria principal (n = 2000) sin recurrir a memoria swap durante los cálculos. Es decir que a partir de n = 2200 la computadora debe recurrir al *swapping* de páginas de memoria para resolver el problema. De esta manera se explica la disminución del rendimiento para los valores de n = 2200, 2400, 3000 y 3200 con respecto al rendimiento obtenido con n = 2000.

La Figura B.14 muestra el rendimiento de **lqt_07** para distintos tamaños de las matrices que se multiplican. También se muestra el mayor tamaño que se puede contener completamente en memoria principal (n = 5000) y a partir de n = 4000 inclusive la proporción aproximada de memoria principal necesaria para contener todos los datos del problema.

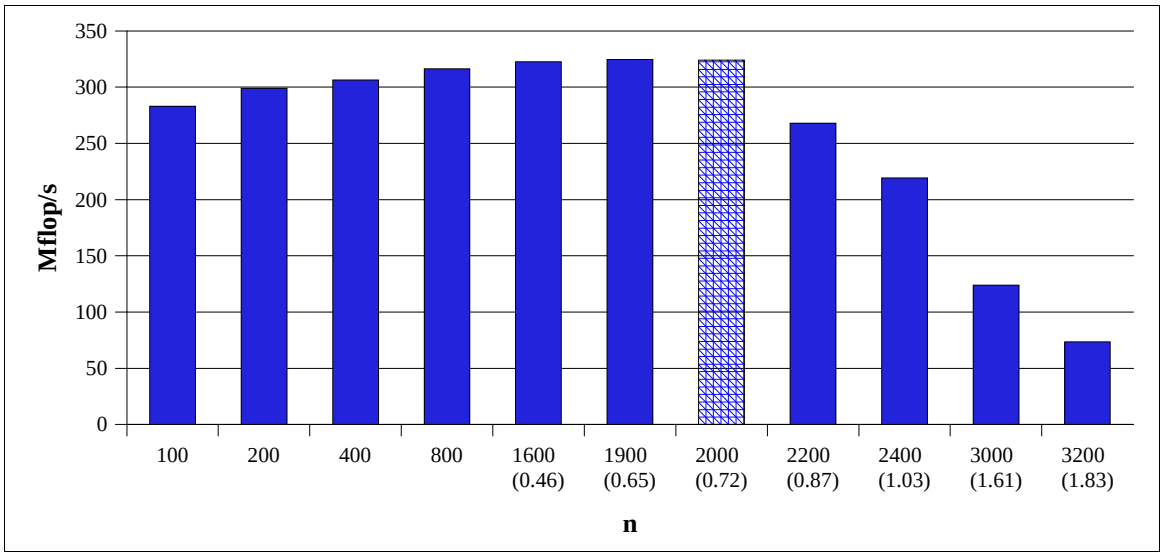


Figura B.13: Mflop/s en **purmamarca**.

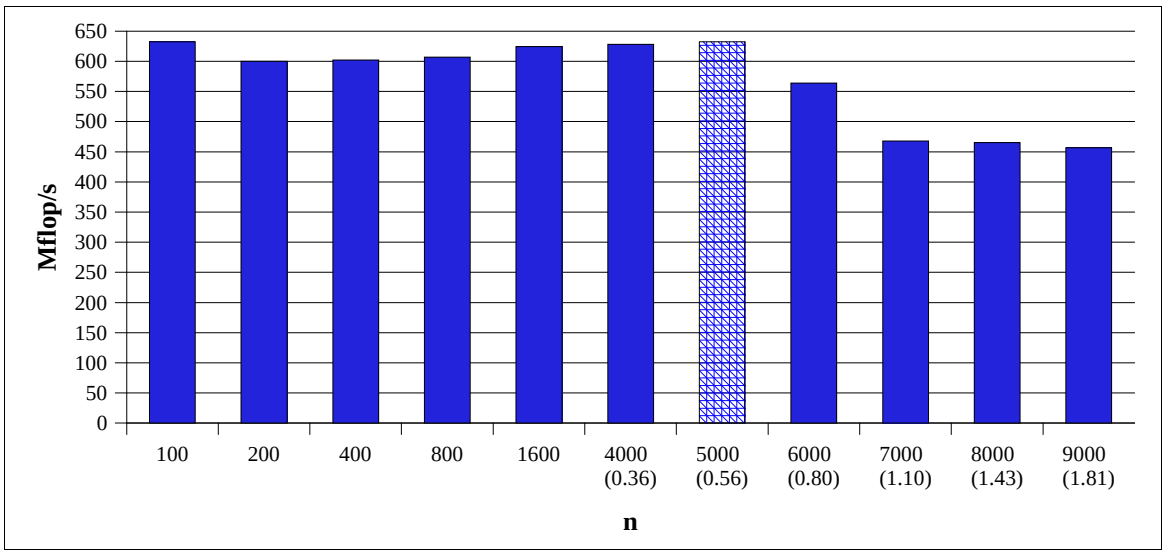


Figura B.14: Mflop/s en **lqt_07**.

En las dos computadoras el rendimiento sigue la misma tendencia: aumenta con el crecimiento de la cantidad de datos del problema mientras la memoria principal es capaz de contener todos los datos a procesar y disminuye a medida que se utiliza mayor proporción de espacio de *swap* (en disco). Sin embargo, las variaciones en **lqt_07** son menores, tanto en lo que se refiere al aumento como a la disminución del rendimiento. Quizás lo más significativo es que el rendimiento no disminuye tanto como en **purmamarca** a partir de la utilización de memoria swap durante los cálculos. Si bien puede ser bastante influyente la velocidad de los discos involucrados, lo más probable es que dos características se combinan para mantener el rendimiento relativamente alto en **lqt_07** aunque se utiliza espacio de memoria swap. Tanto el

- tamaño de las matrices como el
- procesamiento por bloques

se combinan de forma tal que, por un lado la cantidad de operaciones de punto flotante es mucho mayor porque las matrices son mayores y porque los requerimientos de procesamiento son $O(n^3)$, y además el procesamiento por bloques hace que cada dato en memoria principal se utiliza al máximo y por lo tanto se reduce la frecuencia fallos de página que implican el *swapping* de páginas de memoria a disco.

B.5 Rendimiento en las Computadoras del LIDI

La Figura B.15 muestra el rendimiento de las computadoras (homogéneas) del LIDI dependiendo del nivel de optimización de código para la multiplicación de matrices de orden n = 100, 200, 400, 800 y 1600. El rendimiento del código sin ningún tipo de optimización se muestra como "Sin Opt.", el rendimiento del código optimizado por el compilador se muestra como "Compilador" y el rendimiento del programa optimizado a nivel del código fuente para la multiplicación de matrices se muestra como "Completa".

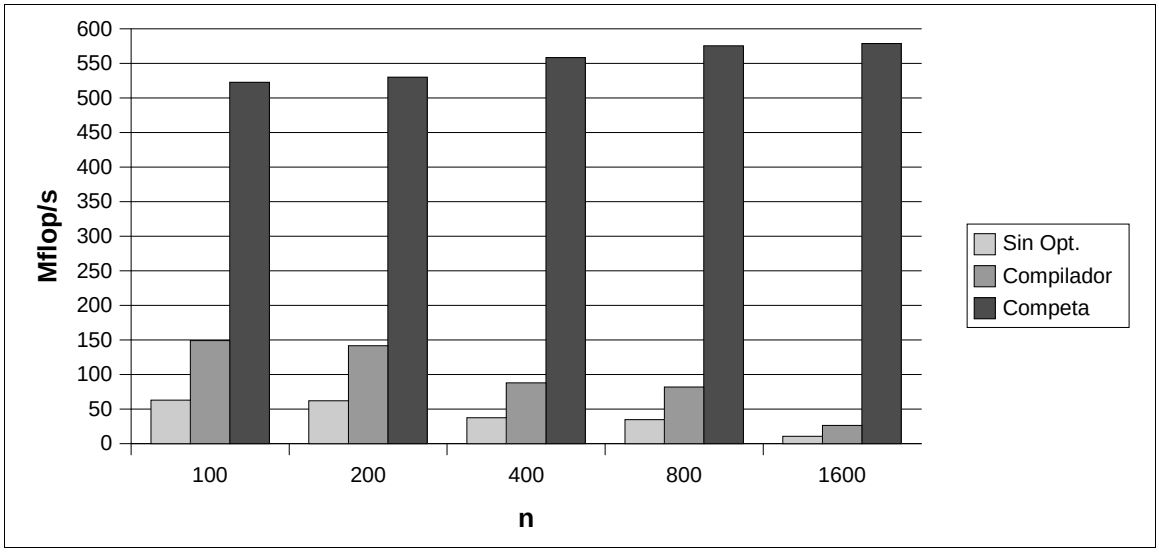


Figura B.15: Rendimiento en Mflop/s Según la Optimización en el LIDI.

De hecho, la Figura B.15 resume lo que sucede en las demás computadoras del CeTAD y del LQT aunque con el valor absoluto de rendimiento propio de las computadoras del LIDI. El peor rendimiento es el que corresponde al código sin ninguna clase de optimización y en el mejor de los casos es de poco más de 50 Mflop/s. Desde el punto de vista del rendimiento el código optimizado por el compilador es aproximadamente tres veces mejor que el no optimizado. Expresado de otra manera, con el código optimizado por el compilador la misma computadora resuelve el mismo problema en un tercio del tiempo que necesita con el código sin optimizar. El rendimiento del código sin optimizar como el del optimizado por el compilador es dependiente del tamaño del problema y se reduce considerablemente a medida que la cantidad de datos a procesar aumenta. El código completamente optimizado es mucho mejor en cuanto a rendimiento que el optimizado por el compilador y es bastante independiente del tamaño del problema, con variaciones que no

superan el 10%. Además, a medida que el tamaño del problema aumenta el rendimiento también aumenta, lo cual es una ventaja considerable con respecto al código sin optimización y al código optimizado solamente por el compilador.

En la Figura B.16 se muestra el rendimiento con código completamente optimizado de las computadoras del LIDI para distintos tamaños de las matrices que se multiplican. Como para las computadoras con mayor capacidad de cálculo del CeTAD y del LQT se muestra el mayor tamaño que se puede contener completamente en memoria principal (n = 2000) y a partir de n = 1600 inclusive la proporción aproximada de memoria principal necesaria para contener todos los datos del problema.

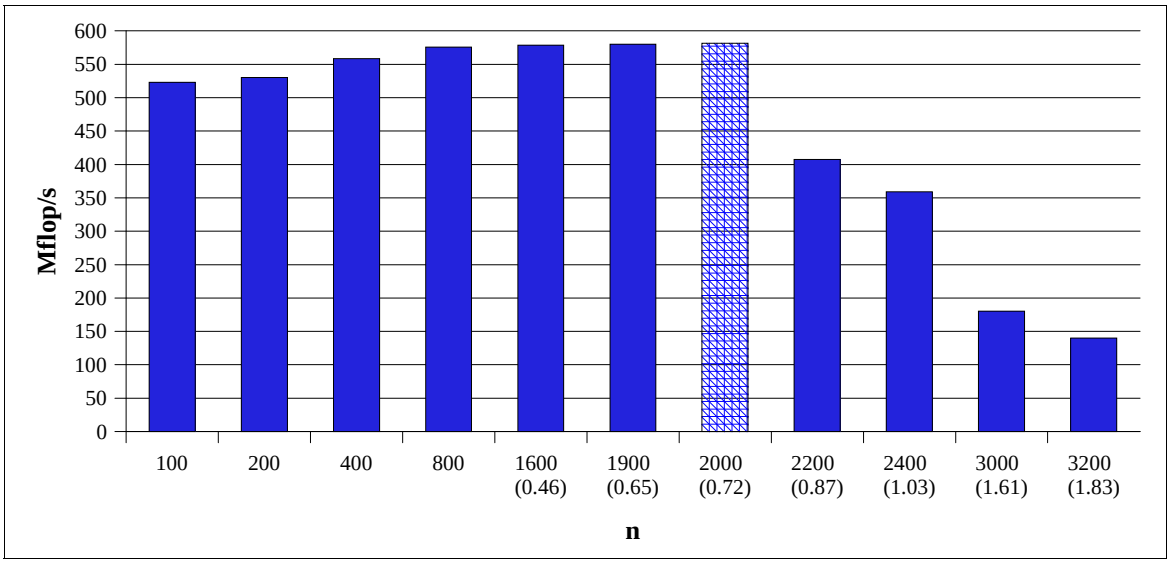


Figura B.16: Mflop/s en las Computadoras del LIDI.

Más allá de las diferencias en valores absolutos, la variación del rendimiento de las computadoras del LIDI es similar a la variación de la máquina con mayor capacidad de cálculo del CeTAD, **purmamarca**, que se muestra en la Figura B.13. De hecho, la gran disminución del rendimiento a medida que se utiliza con mayor frecuencia la memoria swap es similar.

B.6 Conclusiones

A partir de la experimentación realizada, no solamente se tienen valores de rendimiento precisos relacionados con:

- La resolución de la operación elemental de multiplicación de matrices en cada computadora, necesarios para implementar el balance de carga computacional para cómputo paralelo.
- Los valores de rendimiento a utilizar métrica individual de rendimiento de cada una de las mejores computadoras de cada red local, y también para ser utilizado en el cálculo del *speedup* obtenido con la resolución en paralelo.

Además, se llega a que el código a utilizar en cada una de las computadoras debería ser completamente optimizado al menos por dos razones:

1. El rendimiento es varias veces mejor, lo que mejora el tiempo total de ejecución que se requiere de forma significativa (proporcional). El rendimiento secuencial no solamente es crucial para la utilización efectiva de las computadoras (a su máxima capacidad) sino que es sumamente importante para el cálculo correcto de los valores de speedup que se obtienen con procesamiento paralelo.
2. La relación de velocidades de las computadoras es independiente (o con pequeñas variaciones) del tamaño del problema a resolver, lo cual simplifica significativamente la forma en que se resuelve el balance de carga computacional.

Referencias

[1] Alpern B., L. Carter, J. Ferrante, “Space-limited procedures: A methodology for portable high-performance”, International Working Conference on Massively Parallel Programming Models, 1995.

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

[3] Bilmes J., K. Asanovi□, C. Chin, J. Demmel, “Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology”, Proceedings of the International Conference on Supercomputing, Vienna, Austria, July 1997, ACM SIGARC.

[4] Dongarra J., J. Du Croz, I. Duff, S. Hammarling, A set of Level 3 Basic Linear Algebra Subprograms, ACM Trans. Math. Soft., 14 (1), March 1988.

[5] Golub G. H., C. F. Van Loan, Matrix Computation, Second Edition, The John Hopkins University Press, Baltimore, Maryland, 1989.

[6] Henning J. L., SPEC CPU2000: Measuring CPU Performance in the New Millenium, Computer, IEEE Computer Society, July 2000.

[7] Hennessy J. L., D. A. Patterson, Computer Architecture. A Quantitative Approach, Morgan Kaufmann, San Francisco, 1996.

[8] Institute of Electrical and Electronics Engineers, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1984, 1984.

[9] Intel Corporation, Streaming SIMD Extensions - Matrix Multiplication, AP-930, Order Number: 245045-001, June 1999.

[10] McCalpin J. D., M. Smotherman, “Automatic benchmark generation for cache

optimization of matrix algorithms", Proceedings of the 33rd Annual Southeast Conference, R. Geist and S. Junkins editors, Association for Computing Machinery, New York, March 1995.

[11] Saavedra R., W. Mao, D. Park, J. Chame, S. Moon, "The combined effectiveness of unimodular transformations, tiling, and software prefetching", Proceeding of the 10th International Parallel Symposium, IEEE Computer Society, April, 1996.

[12] Whaley R., J. Dongarra, "Automatically Tuned Linear Algebra Software", Proceedings of the SC98 Conference, Orlando, FL, IEEE Publications, November, 1998.

[13] http://developer.intel.com/design/pentiumiii/sml/245045.htm

[14] http://www.netlib.org/atlas

[15] http://www.spec.org

Apéndice C: Comunicaciones en la Red Local del CeTAD

Las redes de interconexión son una parte esencial en la arquitectura de las computadoras paralelas. De hecho, una buena proporción de la bibliografía dedicada al cómputo paralelo está dedicada a este tema. Tiene relevancia desde dos puntos de vista fuertemente relacionados con las aplicaciones paralelas: flexibilidad y rendimiento. Además, se debe tener en cuenta que a mayor flexibilidad y rendimiento de las redes de interconexión se tendrá mayor costo de hardware, y que el crecimiento de este costo suele ser bastante más que lineal con respecto al crecimiento de la cantidad de procesadores de la máquina paralela.

En el caso particular de las multiplicaciones de matrices en paralelo que se ha analizado por experimentación, la red de interconexión es de fundamental importancia. Los índices de rendimiento muestran que la mayoría del tiempo de ejecución de la multiplicación de matrices en paralelo es utilizado para la transmisión de datos. Dado el impacto que el bajo rendimiento de las comunicaciones tiene sobre el rendimiento total de la aplicación paralela, es necesario tener una forma precisa de caracterizar el rendimiento de la red de interconexión para tomar las decisiones necesarias para la optimización del rendimiento de la aplicación completa.

Este Apéndice se dedica principalmente a la caracterización del rendimiento de la red de interconexión de la red local del CeTAD. Se desarrollan y se muestran los resultados de un conjunto de experimentos muy sencillos que permiten evaluar la red con bastante precisión. Si bien la biblioteca de pasaje de mensajes que se utiliza es PVM (Parallel Virtual Machine), se comentan algunas características de rendimiento que son comunes a la mayoría (sino *a todas*) las bibliotecas de pasaje de mensajes implementadas para redes locales de computadoras, tales como las implementaciones de MPI (Message Passing Interface). Es importante notar desde el principio mismo que se intenta conocer el rendimiento a nivel de las aplicaciones de usuario, ya que puede estar muy lejano de los valores definidos por (o *posibles* en) el hardware de comunicaciones.

Si bien se dan un conjunto bastante extenso de resultados de rendimiento de comunicaciones punto a punto (entre dos procesos ejecutándose en diferentes máquinas) en PVM, también se dan los resultados de rendimiento de los mensajes broadcast. De hecho, tal como se ha diseñado el algoritmo paralelo de multiplicaciones de matrices es el rendimiento de los mensajes broadcast el que realmente afecta el rendimiento final del procesamiento paralelo de multiplicación de matrices. Finalmente, se dan las características de rendimiento en las redes locales del LQT y del LIDI, donde se llevaron a cabo experimentos equivalentes, se comentan las diferencias más notables de rendimiento en estas redes y las razones por las cuales se dan estas diferencias.

C.1 Introducción

La red de interconexión de procesadores es elemental en las computadoras paralelas. En el caso de las computadoras paralelas con (o basadas en) memoria física compartida, esta red de interconexión se puede identificar claramente no en cuanto a la interconexión de los procesadores entre sí sino en cuanto a la interconexión de los procesadores con la memoria. Expresado de otra manera, en estas computadoras paralelas quitar la red de interconexión de los procesadores con la (*única*) memoria elimina completamente la posibilidad de ejecutar aplicaciones. Por otro lado, si a una computadora paralela con memoria distribuida se le quita la red de interconexión de procesadores, deja de ser una computadora paralela y se transforma en un conjunto de computadoras separadas o módulos de CPU-Memoria, sin la capacidad de cooperación para la resolución de un problema. Dado que las redes de computadoras utilizadas para cómputo paralelo son claramente pertenecientes a la clase MIMD de memoria distribuida, se continuará considerando a la red de interconexión para la transmisión de datos entre procesadores (o computadoras directamente).

En relación con la flexibilidad de una red de interconexión de procesadores, se busca no solamente que haya una forma de transferir datos entre dos procesadores sino que también se tenga el máximo de comunicaciones al mismo tiempo o simultáneas. Los ejemplos clásicos en este sentido se enfocan en la capacidad o no de que todos los pares posibles de procesadores se puedan comunicar al mismo tiempo, o que sea posible que un solo paso (o en una cantidad de pasos independiente de la cantidad de procesadores que se tengan interconectados) se pueda transferir información desde un procesador hacia todos los demás.

La flexibilidad que tenga una red de interconexión definirá la facilidad (o dificultad) de las aplicaciones de usuario para resolver la comunicación entre sus procesos. La idea subyacente es que nunca se debería perder de vista que cada uno de los procesadores será el encargado de la ejecución de uno o más procesos que se comunicará con otros procesos asignados a otro/s procesador/es.

La visión del rendimiento de una red de interconexión está directamente relacionada con el tiempo de transferencia de los datos entre los procesadores de una computadora paralela. Esta visión del rendimiento no necesariamente es disjunta de la flexibilidad. De hecho, a mayor cantidad de transferencias de datos simultáneas entre pares de procesadores será también mayor la capacidad o la cantidad de datos que una red de interconexión puede transferir por unidad de tiempo.

Si bien es importante esta idea de *ancho de banda* (tasa de transferencia) o cantidad de datos por unidad de tiempo, otro de los índices de rendimiento importantes es el tiempo mínimo de comunicación entre dos procesadores, o tiempo de inicialización de las comunicaciones (*startup*), o también llamado *latencia* de comunicación entre procesadores.

En términos de costo se tiene una relación invariante a través de las distintas posibilidades de redes de interconexión: a mayor flexibilidad y/o rendimiento de la red de interconexión el costo también aumenta. El crecimiento del costo varía según la red de interconexión que se utiliza, pero en muchos de los casos aumentar la cantidad de procesadores de la

computadora paralela implica un crecimiento más que lineal del costo de la red de procesadores. En el caso particular de las redes de computadoras instaladas, el costo es cero (en general, despreciable), dado que ya están interconectadas.

El principal inconveniente de las redes de interconexión de computadoras en cuanto a rendimiento es que no fueron diseñadas para cómputo paralelo. En este sentido, el rendimiento de las redes locales se ubica varios órdenes de magnitud por debajo de las redes de interconexión de las computadoras paralelas *tradicionales*. Es por eso que se torna muy importante evaluar su rendimiento desde el punto de vista de los procesos de usuario que componen una aplicación paralela.

Por otro lado, el rendimiento de la red de interconexión tiene relación directa con el rendimiento y con la granularidad de las aplicaciones paralelas que se pueden ejecutar sobre la computadora. Todo tiempo de comunicación tiende a degradar el tiempo total de ejecución de una aplicación paralela, a menos que se disponga y se aproveche al máximo la capacidad de solapar en el tiempo cómputo con comunicación. Desde el punto de vista de la granularidad, si el tiempo de comunicación para la obtención de un resultado en el procesador P_1 es igual o mayor que el tiempo de cómputo necesario para calcularlo, entonces lo más razonable es obtenerlo localmente (en P_1), ahorrando tiempo y/o complejidad de la aplicación.

C.2 Redes Ethernet

En el contexto particular de las redes de computadoras instaladas, redes locales o LAN (Local Area Networks), la red de interconexión más utilizada es la definida por el protocolo estándar IEEE 802.3. Este estándar es también conocido como red Ethernet de 10 Mb/s por su capacidad de transmisión de 10^6 bits por segundo. Las características de esta red de interconexión es muy bien definida y conocida en términos de hardware y de las características que hacen a su flexibilidad y rendimiento.

Además, la mayoría de las características de la red Ethernet de 10 Mb/s son similares a la red Ethernet de 100 Mb/s, donde se cambian solamente los parámetros/índices referidos a rendimiento o capacidad de comunicación. Esta similitud está ejemplificada en, y también aprovechada por muchas de las empresas de hardware de comunicaciones que se encargan de comercializar placas de interfase de comunicaciones (NIC: Network Interfase Card) con ambas capacidades de comunicación y denominadas de 10/100 Mb/s.

La Figura C.1 muestra esquemáticamente la forma lógica básica en que se conectan las estaciones de trabajo en una red local utilizando Ethernet. Se puede notar fácilmente que es de tipo bus, donde las características principales de cada transferencia de datos son:

- no se manejan prioridades ni es predecible el tiempo de acceso al medio,
- tiene un único emisor,
- ocupa el único canal de comunicaciones,
- puede tener múltiples receptores,
- el modo de acceso al medio es CSMA/CD (Carrier Sense, Multiple Access / Collision

Detect).

Las dos primeras características implican claramente que no puede haber más de una transferencia de datos simultáneamente, porque de hecho hay un único canal de comunicaciones que es compartido por todas las computadoras. La última característica enunciada hace muy natural la implementación de las comunicaciones del tipo *broadcast* y/o *multicast*, donde desde una computadora se emite un mensaje que es recibido en todas las demás o en un subconjunto de las demás de la red respectivamente.

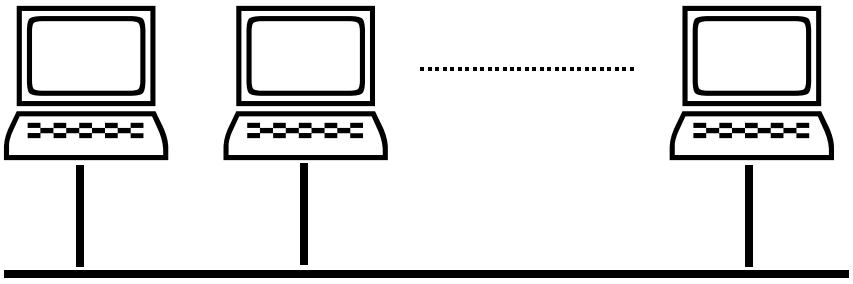


Figura C.1: Red Ethernet.

El hardware de comunicaciones inicialmente adoptado en la mayoría de las instalaciones estuvo basado en cables coaxiales, con lo cual se logra que la topología física sea igual a la topología lógica de la Figura C.1.

Gradualmente, el cableado (*wiring rules*) utilizado en la mayoría de las instalaciones se ha cambiado hacia la utilización del cable de par trenzado con *hubs* (básicamente concentradores y repetidores de comunicaciones) y con *switches* de comunicaciones, que tienen la capacidad de aislar las comunicaciones que son punto a punto. Esta aislación en los *switches* se produce cuando el hardware detecta que hay transmisión de datos punto a punto entre dos de las computadoras que están interconectadas por un *switch*. También son posibles varias combinaciones de *hubs* y *switches* con el objetivo de mantener un cierto rendimiento a medida que el tráfico de datos aumenta y también reducir el costo al evitar la utilización masiva de *switches* de comunicaciones.

Estas redes de comunicación son importantes no solamente por la cantidad de instalaciones actualmente funcionando sino porque son claramente menos costosas que las demás alternativas que se comercializan. Son menos costosas en el hardware necesario (placas, conectores y cables) y en lo referente a instalación: desde mano de obra (técnicos) hasta reconocimiento y puesta en marcha del hardware por parte del sistema operativo. Todo esto necesariamente reduce los costos de instalación y de mantenimiento de las redes Ethernet.

La reducción de costo con respecto a las demás alternativas de interconexión de computadoras implica una gran inercia en el mantenimiento de las redes Ethernet así como también en la instalación de nuevas redes con este hardware. Siempre se debe tener en cuenta que el costo puede incluir aspectos como: placas de red en cada computadora, cableado (que puede incluir hubs y/o switches en el caso de las redes Ethernet), instalación, mantenimiento y personal técnico capacitado.

C.3 Evaluación del Rendimiento

El tiempo de comunicación (utilizado en general para caracterizar el rendimiento de una red de interconexión) entre dos procesadores en general se caracteriza con [7] [11]

$$t(n) = \alpha + \beta n \qquad \text{C.1}$$

donde

- n es la unidad de información que se transfiere y que se desea medir (bit, byte, representación de un número en punto flotante con precisión simple, etc.),
- α es el tiempo necesario para establecer la comunicación entre los dos procesadores, que se suele llamar *tiempo de latencia* de la comunicación. Es básicamente el tiempo mínimo que toda comunicación entre dos procesadores utilizará independientemente de la cantidad de información que se transfiera. Normalmente está dado por el hardware de comunicaciones y puede estimarse con el tiempo que toma transferir una unidad de información o, en el caso en que sea posible, un mensaje sin datos.
- β es el valor inverso del ancho de banda asintótico o tasa de transferencia de datos de la red de comunicaciones, es decir que $1/\beta$ es el ancho de banda asintótico. Normalmente, la tasa de transferencia de datos de la red de comunicaciones está dada por la cantidad de datos (bits, bytes, etc.), por unidad de tiempo que pueden transmitirse entre dos procesadores. Normalmente tiene un límite mínimo dado por el hardware de comunicaciones a lo que se debe agregar el tiempo que consumen los procesos y/o funciones del software de comunicación.

Si bien el hardware de comunicaciones o red de interconexión de procesadores tiene valores bien definidos para los parámetros α y β, en general el usuario suele obtener valores peores desde el punto de vista del rendimiento de la red de interconexión de procesadores que los dados por el hardware. Tanto el tiempo de latencia como el tiempo que lleva transmitir cada unidad de información se ven afectados (y por lo tanto aumentan) cuando en la transferencia intervienen todas las capas de comunicación necesarias para que un mensaje de un proceso de usuario ejecutándose en un procesador llegue a otro proceso de usuario ejecutándose en otro procesador.

Es también muy difícil realizar una estimación *a priori* de la sobrecarga que las bibliotecas de comunicaciones que tienen disponibles los (procesos de) usuarios, la interfase del sistema operativo con el usuario y/o con las bibliotecas mencionadas anteriormente y los protocolos de comunicaciones, por ejemplo, imponen a las comunicaciones que se llevan a cabo físicamente sobre el medio de comunicación que se utilice. Por esta razón son bastante utilizados los métodos experimentales de medición de los parámetros α y β reales que las aplicaciones de usuario encontrarán respecto a la red de interconexión de la computadora paralela.

En el contexto específico de las redes de computadoras, y con la posibilidad de hardware heterogéneo, esta sobrecarga se torna más importante cuando se debe evaluar el rendimiento de la red de interconexión de procesadores. En el caso de las computadoras paralelas tradicionales, el cálculo de α y β reales suele ser mucho más sencillo y con

valores finales (los que los procesos de las aplicaciones paralelas de usuario obtienen) más cercanos a los del hardware porque desde el hardware de comunicaciones hasta la interfase que utilizan las aplicaciones de usuario está *todo* orientado a hacer cómputo paralelo.

Es muy difícil cuantificar de manera exacta la relación de las redes locales con respecto a las redes de interconexión de las computadoras paralelas tradicionales en términos de los índices de rendimiento mencionados. Lo que es generalmente aceptado es que la peor relación está dada con respecto al tiempo de inicialización de los mensajes a comunicar entre dos procesadores. Más aún, en el contexto de las redes locales heterogéneas el tiempo de inicialización de las comunicaciones suele depender de las computadoras utilizadas dado que están involucrados los tiempos de llamadas al sistema operativo y su consiguiente sobrecarga en cuanto al mantenimiento y manejo de los protocolos (o *pila de protocolos*) utilizados. En un nivel más cercano al hardware, también están involucrados los tiempos de

- acceso a memoria,
- inicialización-utilización de canales de DMA (Direct Memory Access) en caso de ser utilizados y
- manejo de interrupciones relacionadas y/o interfase con la placa de red de cada computadora a comunicar.

Resumiendo, el rendimiento de las redes locales es inferior al de las redes de interconexión de procesadores de una máquina paralela tradicional desde varios puntos de vista:

- Latencia y ancho de banda.
- Capacidad de solapamiento.
- *Heterogeneidad* de latencia dependiendo de la heterogeneidad de las máquinas.

C.4 Evaluación con el Método de ping-pong

En el contexto de las computadoras paralelas con arquitectura de base MIMD, el método experimental para evaluar el rendimiento de la red de interconexión de procesadores o, más específicamente, los valores reales de los parámetros α y β, ha sido el de los mensajes *pingpong*. En sí mismo, el método es muy sencillo, dado que para evaluar el tiempo de comunicación entre dos procesadores P_1 y P_2, los pasos a seguir son los que muestra la Figura C.2:

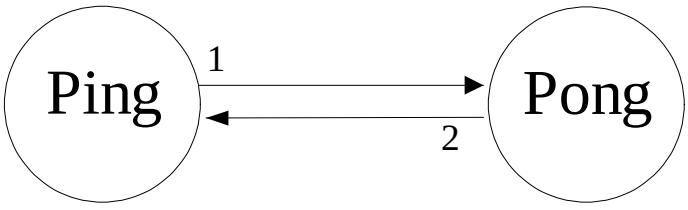


Figura C.2: Procesos *Pingpong*.

1. Enviar un mensaje desde el procesador P_1 al procesador P_2
2. Enviar el mensaje recibido en el procesador P_2 al procesador P_1 nuevamente

3. En el procesador P_1 se conoce el tiempo total utilizado para la comunicación de los dos mensajes y, por lo tanto, se divide ese tiempo por dos llegándose así al tiempo de comunicación del mensaje en una de las direcciones.

Una de las características más atrayentes de este método, además de su sencillez, consiste en que no necesita la sincronización de ninguno de los procesadores que intervienen en la transferencia de información para obtener un tiempo confiable de comunicación de datos. De otra manera, se debería conocer el tiempo de envío del mensaje desde el procesador P_1 al procesador P_2, el tiempo de llegada del mensaje al procesador P_2 y los procesadores deberían estar sincronizados con respecto a su registro de tiempo.

Otra de las ventajas del método consiste en la independencia de la forma de comunicar mensajes entre los procesadores. El tiempo que lleva enviar y recibir el mismo mensaje mensaje en P_1 (comunicándose con P_2) se puede tomar independientemente de la forma en que el mensaje sea transmitido. De hecho, se pueden analizar distintas alternativas disponibles de comunicación entre los procesadores para elegir la más conveniente.

Por otro lado, una de las restricciones respecto a la confiabilidad de este método recae en que el tiempo de comunicación entre los procesadores no debe variar según la *dirección* de la comunicación, es decir que hay cierta *simetría* de comunicaciones entre los procesadores. En la enumeración anterior, se asume que el tiempo para enviar un mensaje desde P_1 a P_2 es el mismo que el de enviar el mismo mensaje de P_2 a P_1. De todas maneras, esta situación es sin lugar a dudas mayoritaria en el campo de las redes de interconexión en general.

Otra de las simplificaciones del método *pingpong* consiste en enmascarar, o mejor expresado, no tener en cuenta la posibilidad de hacer transmisión y recepción de datos simultáneamente (comunicación *full duplex*), o la posibilidad de solapar cómputo con comunicación que las aplicaciones podrían aprovechar. En ambos casos, es decir con el hardware disponible para hacer una o ambas cosas, si se obtienen los tiempos de comunicación punto a punto es posible llegar a los valores de rendimiento de comunicación que las aplicaciones pueden obtener.

En el caso específico de las redes de computadoras, el usuario normalmente no tiene demasiado control (o ningún control) sobre el aprovechamiento o no de las capacidades del hardware. Por lo tanto, los valores de rendimiento que se obtienen por el método del pingpong serán los más cercanos a lo que las aplicaciones (paralelas) de usuario podrán obtener.

C.5 Distintas Formas de Transmisión de Mensajes con PVM

Dado que es necesario caracterizar el tiempo de comunicación entre los procesos de un programa paralelo y que éstos se comunican utilizando las rutinas de comunicación provistas por PVM, es necesario explorar todas las alternativas posibles en cuanto al

rendimiento alcanzable con estas rutinas para las aplicaciones de usuario.

Con el método de pingpong explicado previamente se puede evaluar bastante rápidamente cuál es el rendimiento obtenible en las comunicaciones entre los procesos que se asignan a los distintos procesadores de la máquina (*virtual*) paralela. Además, dado que se tienen los valores relacionados con el rendimiento de la red de comunicaciones (α y β), del hardware de comunicaciones se puede conocer con certeza el grado de sobrecarga de tiempo por utilizar PVM para comunicar tareas.

Cuando se debe evaluar el rendimiento de las comunicaciones teniendo en cuenta no solamente el hardware sino también el software de comunicaciones (básicamente los procesos del sistema operativo involucrados más las rutinas de comunicaciones de PVM) es necesario explorar las distintas alternativas de comunicación para transferir datos entre los procesos de la aplicación paralela.

En general, PVM tiene dos niveles de flexibilidad cuando se deben transferir datos entre los procesos: codificación de datos y "ruteo" (en terminología de PVM) de los mensajes. La codificación de datos tiene relación con la representación de información que se hace en cada procesador (computadora) y el ruteo de los mensajes se relaciona con la forma en que los datos se transfieren entre los procesos de la aplicación paralela utilizando la red de física de comunicaciones y las rutinas/procesos de comunicación de PVM. En las subsecciones que siguen se detallan las alternativas para codificación y ruteo en PVM. En el caso particular de la codificación, se describirá también una alternativa a las formas clásicas que se utilizan tanto en PVM como en MPI, que se denominará *Traducción Directa de Representación de Datos*.

Es necesario aclarar que si bien esta descripción es específica de PVM, tanto en MPI como en cualquier otra biblioteca que se utilice para hacer cómputo paralelo en redes de computadoras será necesario definir tanto la forma en que las distintas representaciones de datos se compatibilizan como la forma en que los datos se transfieren sobre la red de interconexión de computadoras.

C.5.1 Codificación de los Datos de un Mensaje en PVM

La codificación de datos básicamente se debe elegir entre lo que en PVM se denomina:

a) *PvmDataDefault*: se utiliza cuando los procesos que se comunican están asignados a procesadores con distintas arquitecturas o cuando la aplicación no tiene ningún conocimiento de la arquitectura de los procesadores (computadoras) sobre la que se ejecuta. Los datos a transferir entre los procesos son codificados en formato XDR antes de ser enviados, luego se copian a un área de memoria desde donde las rutinas de PVM harán el envío (buffers) de información y luego serán decodificados (del formato XDR) al ser recibidos antes de ser utilizados por el proceso receptor.
b) *PvmDataRaw*: se utiliza cuando la arquitectura de la máquina paralela es homogénea, sea una red de computadoras o sea un multiprocesador. Los datos que se transfieren entre los procesos no se codifican de ninguna manera, solamente se copian a los buffers de PVM antes de hacer el envío por la red de comunicaciones.
c) *PvmDataInPlace*: Esta alternativa es similar a *PvmDataRaw* pero sin copia de los datos

de usuario a los buffers. No hay codificación de los datos, ni gasto extra de memoria por la comunicación de datos, ni tiempo de copia de información, pero el usuario debe asegurar que los datos no son cambiados desde que se efectúa el llamado a la rutina de envío hasta que los datos son enviados efectivamente al proceso destino.

Con la Figura C.3 se pueden mostrar esquemáticamente los tres tipos de codificación y su relación con la codificación y la memoria utilizada para el envío de datos desde el proceso P_1 asignado a un procesador y el proceso P_2 asignado a otro.

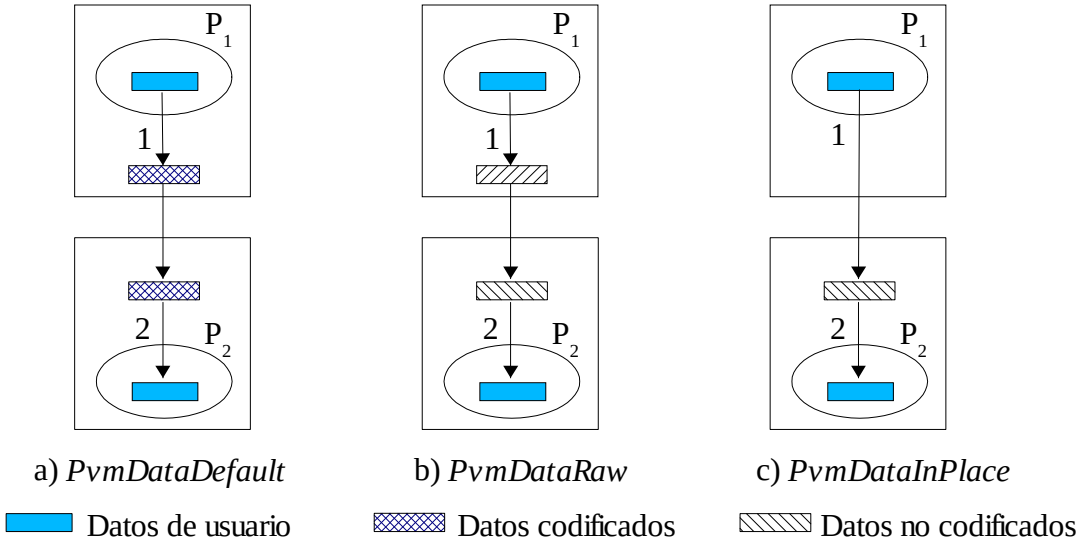


Figura C.3: Alternativas de Codificación en PVM.

Cuando se utiliza codificación a) *PvmDataDefault*, los datos son copiados desde el área de datos de proceso usuario P_1 al área de buffers de PVM y codificados en formato XDR [10] en el paso 1. Luego estos datos codificados son enviados al procesador destino (en el cual está asignado el proceso receptor P_2), utilizando la red de interconexión, donde se reciben sobre otra área de buffers de PVM. En el paso 2 se decodifican y se copian los datos ubicados en los buffers de PVM al área de datos del proceso receptor P_2.

Cuando se utiliza la codificación b) *PvmDataRaw*, el proceso es igual, con la excepción de no utilizar ningún tipo de codificación para los datos que se envían. La secuencia de bytes que estaban en el área de datos del proceso P_1 llegan al proceso P_2. Con este método se se ahorra en memoria de los buffers utilizada por la codificación XDR y también se evita el uso de CPU que tal codificación involucra, pero las copias y los buffers son utilizados de la misma manera que en el caso de utilizar la codificación PvmDataDefault.

Cuando se utiliza la codificación c) *PvmDataInPlace*, se evita no solamente lo relativo a la codificación XDR (procesamiento y memoria asociada) sino también toda la memoria necesaria para almacenar el mensaje en el procesador origen de la comunicación. En el caso de la Figura C.3, en el primer paso se envían los datos desde el proceso de usuario P_1 que envía el mensaje al procesador al cual está asignado el proceso receptor P_2, donde se almacena en los buffers de PVM. Expresado de otra manera, ya no son necesarios los buffers de PVM de salida para el mensaje.

En los ejemplos de evaluación del rendimiento de las comunicaciones que se incluyen en la distribución de PVM, la forma de codificación de datos utilizada es *PvmDataRaw*, con un comentario indicando que en caso de haber heterogeneidad en la máquina paralela se debe cambiar esta forma de codificación a *PvmDataDefault*. En las redes de computadoras en general, no queda más alternativa que la de codificar los datos (con el agregado de las copias a buffers de PVM) con esta forma *PvmDataDefault* para que no pierdan su significado al ser transferidos a otro proceso que se ejecuta sobre otro procesador (computadora). Una vez que se acepta la heterogeneidad de las computadoras utilizadas para cómputo paralelo no es posible asumir que la representación de los datos en todas las computadoras es homogénea.

Desde el punto de vista del rendimiento, el método de codificación más apropiado es el *PvmDataInPlace*. La pérdida de flexibilidad por la restricción en cuanto a la imposibilidad de modificación de los datos que se envían no parece ser un problema en el caso particular de la multiplicación de matrices. Los datos que se envían (submatrices de la matriz B, en la operación $A = B \times C$) son de sólo lectura para todos los procesos que los utilizan. De todas maneras, el problema de la heterogeneidad de la representación de datos persiste.

Todas las aplicaciones numéricas dependen, en cuanto a representación de datos, de una cantidad bastante reducida de tipos de datos que son generalmente predefinidos por los lenguajes que se utilizan (normalmente C o FORTRAN). En general, se pueden identificar dos tipos básicos a partir de los cuales se definen las estructuras de datos con las cuales se opera en las aplicaciones (mayoritariamente vectores y matrices): punto flotante de precisión simple y punto flotante de precisión doble. La representación de números complejos, por ejemplo, necesarios también para una amplia gama de aplicaciones numéricas, se define en función de un par de números de punto flotante en precisión simple o doble, según la aplicación específica.

Como se puede notar a partir de las tres formas de codificación explicadas, la estrategia general para transferir datos de una computadora a otra es, esquemáticamente:

Figura C.4: Estrategia de De/Codificación de PVM/MPI.

Es decir que, antes de transferir datos a de una estación a otra los datos se codifican para conservar la información que representan. En este sentido, tanto PVM como MPI son similares, si hay codificación de datos para su transferencia esta codificación se lleva a cabo antes de hacer la transferencia.

De alguna manera, esta estrategia de codificación-transmisión-decodificación puede ser considerada conservadora o anticipada a la comunicación, porque siempre funciona de

forma independiente de las computadoras que se comunican y se lleva a cabo antes de que los datos se transfieran.

C.5.2 Traducción Directa de Representación de Datos

Una alternativa a esta forma anticipada de PVM (y MPI) de conservar la consistencia de los datos entre distintas computadoras consiste en retardar la de/codificación de forma tal que:

- Los datos son siempre enviados sin ningún tipo de codificación previa (ni a XDR ni a ningún otro formato).
- Los datos son recibidos como una secuencia de bytes en el receptor, junto con un descriptor del tipo de datos que representan más el tipo de arquitectura origen (tipo de representación de origen).
- Si el procesador donde está ubicado el proceso receptor tiene distinta representación de datos que el procesador donde está ubicado el proceso que envía, los datos son "decodificados": se cambia de la representación de datos origen a destino.

Por lo tanto, en realidad no hay de/codificación, sino que en el procesador destino, las rutinas de comunicación se encargarán de "traducir" la representación de datos de una computadora (desde donde se envió el mensaje) a la otra (donde se recibe el mensaje).

En el caso de codificar a una representación conocida se tienen dos traducciones: de la representación origen (desde donde el mensaje es enviado) a la representación conocida y luego de ésta a la representación destino (en donde el mensaje es recibido). En el caso de la traducción directa se tiene a lo sumo una sola, que cambia de la representación de datos origen a la representación de datos destino cuando es necesario.

Se debe notar que la traducción directa se debe hacer en el destino y no en el origen de cada mensaje, dado que de esta forma se evitan problemas con las llamadas comunicaciones colectivas como *multicast* y *broadcast*. En este tipo de comunicaciones desde un mismo proceso en un procesador se pueden enviar datos a múltiples procesos (lo que implica potencialmente procesadores destino), y por lo tanto no habría una única traducción posible (habría tantas como posibles destinos).

La traducción directa se puede hacer transparente para los procesos de usuario tanto como lo es la codificación XDR usada en PVM y en (algunas implementaciones de) MPI. En el caso de PVM no hay ningún inconveniente para su implementación porque se tienen funciones de identificación de cada computadora de la máquina paralela así como también formas de identificar la ubicación de cada tarea (procesador en el que se ejecuta).

Respecto de la codificación de los datos para su transmisión, la traducción tiene la ventaja de minimizar el procesamiento y la memoria utilizada para cada mensaje en el lado (procesador) del proceso que envía. De hecho, los datos se pueden tomar de la memoria del proceso origen, agregar los descriptores necesarios (que de todas maneras están en la codificación) y enviar el mensaje. En el lado (procesador) del proceso que recibe, tanto la memoria como el procesamiento dependen de la complejidad de la traducción de los datos. Como se verá a a continuación, como mínimo en el contexto particular de las redes de computadoras, la traducción de representación es muy sencilla, y por lo tanto también se

reduce la necesidad de memoria y procesamiento.

Avanzando en el análisis de la representación de números en el contexto particular de las computadoras, se encuentra una sorprendente homogeneidad en cuanto a la adhesión al estándar IEEE 754 [6] para la representación de números en punto flotante. En todas las computadoras a las cuales se tiene acceso y sobre las que se llevó a cabo la experimentación, que incluye

- PCs con procesadores Pentium (en alguna de sus múltiples versiones), Celeron, y AMD K6-II,
- Estaciones de trabajo Sun con procesadores MicroSPARC-II,
- Una estación de trabajo IBM RS/6000, con procesador PowerPC,

la representación de punto flotante adoptada (a nivel de operaciones de la unidad de punto flotante del procesador) es la misma: IEEE 754 en ambas versiones (simple y doble).

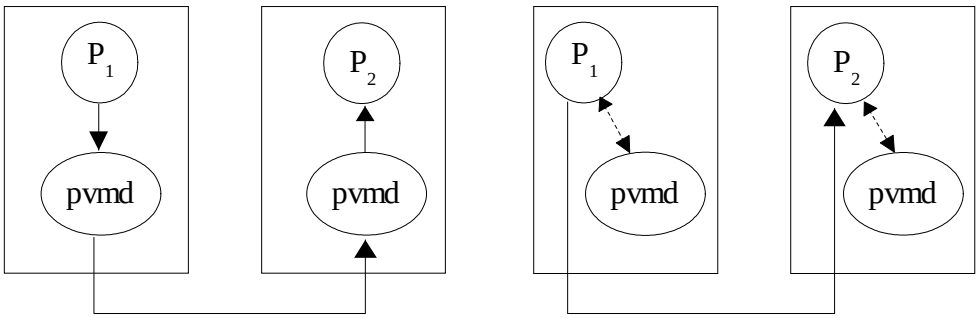
Se debe destacar que la heterogeneidad a nivel de representación de datos puede ser mayor en otros ámbitos, por ejemplo entre distintas máquinas paralelas tradicionales con unidades de punto flotante ampliamente más complejas. Independientemente de esto, se debe recordar que la tendencia (aún en las máquinas paralelas más potentes/costosas) es la utilización de hardware estándar. Las IBM SP2, por ejemplo, están basadas en PowerPCs, y las ASCI Red y Blue están basadas en Pentium Pro y MIPS R1x000 respectivamente.

Aún cuando todas las computadoras adoptan IEEE 754 como su representación de números en punto flotante, esto no significa que al enviar una secuencia de bytes desde una computadora a otra estos bytes significarán (representarán) lo mismo en ambas. Un escollo más a superar es la forma de almacenamiento de los bytes de un tipo particular de datos (en el caso que se viene analizando: números en punto flotante) en la memoria. En este caso las "normas" que se siguen son dos y de hecho no hay muchas más alternativas para explorar: primero el byte más significativo (llamada usualmente *little endian* en la literatura) o primero el byte menos significativo (*big endian* en la literatura). Es claro que la traducción de un formato a otro es inmediata y sin grandes requerimientos de memoria ni de procesamiento.

De los dos párrafos anteriores se deduce que, como mínimo para la representación de números en punto flotante, la traducción directa de las representaciones de datos entre computadoras es ventajosa con respecto a la codificación, tanto en requerimientos de memoria como de procesamiento. No es difícil hacer un análisis similar con los demás tipos básicos de datos (caracteres, número enteros, etc.) y llegar a la misma conclusión. Es por esta razón que la experimentación incluye esta forma de "codificación" de datos para la transmisión de mensajes entre los procesos *pingpong*.

C.5.3 Ruteo de los Datos de un Mensaje en PVM

El ruteo de mensajes en PVM se refiere a la forma en que los datos de un mensaje se transportan entre los procesos de usuario y el proceso mismo de PVM (*pvmd*) en cada computadora. Las dos formas más comunes de ruteo de los mensajes entre las tareas son esquematizadas en la Figura C.5.



a) PvmRouteDefault b) PvmRouteDirect

Figura C.5: Alternativas Básicas de Ruteo en PVM.

En el caso a) *PvmRouteDefault*, los datos se transfieren entre los procesos *pvmd* sobre la red de interconexión de computadoras utilizando el protocolo de comunicaciones UDP (sobre IP).

En el caso b) *PvmRouteDirect*, los datos se transfieren directamente entre los procesos de la aplicación paralela sobre la red de interconexión de computadoras utilizando el protocolo de comunicaciones TCP (también sobre IP). La recomendación general que se hace en la documentación de PVM es que, en términos de rendimiento, la opción b) es la mejor.

C.6 Distintas Formas de Transmisión de Mensajes con MPI

Dado que MPI se propone como una biblioteca estándar de pasaje de mensajes entre procesos de una aplicación paralela, detalles de opciones tales como la codificación y el ruteo (en el sentido de PVM) no están disponibles a nivel del usuario. En este sentido, la creación de MPI es bastante lejana e independiente de las redes de computadoras y por lo tanto la heterogeneidad en la representación de datos o las formas alternativas a nivel de protocolos de comunicación entre procesadores no tienen la relevancia que adquieren desde el principio en PVM que fue creado para redes de computadoras (heterogéneas). En términos de las posibles implementaciones, está ampliamente demostrado que MPI es posible en todo el rango de computadoras paralelas de pasaje de mensajes, sean multiprocesadores, multicomputadoras con redes de interconexión *ad hoc* para cómputo paralelo o redes de computadoras.

Las implementaciones de MPI para redes de computadoras normalmente asumen que la representación de datos es heterogénea (de hecho no les queda mucha alternativa) y conectividad IP. El método de *codificación anticipada* (con XDR por ejemplo), suele utilizarse para resolver la heterogeneidad de la representación de datos. Como siempre en el contexto de MPI debe recordarse que es la implementación la que toma una decisión de este tipo y, por lo tanto, distintas implementaciones de MPI pueden tener distintas formas de implementarlo. De forma similar, lo que en PVM se denomina ruteo (referente a la

forma en que los datos de un mensaje son enviados desde el proceso emisor al proceso receptor), depende de la implementación de MPI, aunque la mayoría coinciden en utilizar conectividad IP (al menos las libres que pueden obtenerse vía Internet).

En todo caso, independientemente de la implementación de MPI, sea para redes de computadoras o para cualquier otro tipo de arquitectura de cómputo paralelo, no se tienen alternativas a nivel de usuario sobre la codificación ni sobre el ruteo de los datos de los mensajes. En MPI se gana en transparencia respecto de la implementación y de la arquitectura paralela y en el caso particular de las redes de computadoras quizás se pierde rendimiento si se tiene homogeneidad de máquinas o si se utiliza la traducción de las representaciones de los tipos de datos explicada previamente.

C.7 Experimentación Inicial con PVM

La experimentación inicial con PVM se llevó a cabo para tener una base de cálculo de α y β utilizando las distintas formas de codificación y ruteo que en PVM se pueden seleccionar a nivel de usuario. Las máquinas con las cuales se llevó a cabo la experimentación se detallan en el Apéndice X: computadoras del CeTAD, interconectadas con una red Ethernet de 10 Mb/s. Dado que las computadoras **cetadfomec1** y **cetadfomec2** son exactamente iguales se muestran los resultados para una de ellas (**cetadfomec1)**, que por razones de espacio en los gráficos se abrevia como **cf1**.

Se utilizó el método de ping-pong ubicando al proceso *ping* siempre en una misma computadora y midiendo los tiempos con el proceso *pong* en cada una de las demás por vez. Si bien lo más importante a estimar (y por lo tanto a medir) son los parámetros α y β, en el caso específico de las redes de computadoras puede ser útil el tiempo necesario para resolver la heterogeneidad de representación de datos (de/codificación o traducción). La computadora elegida para ubicar el proceso ping es **purmamarca** dado que tiene la cantidad de memoria necesaria para todas las longitudes de mensajes (64 MB) y que es la más rápida de todas las computadoras del CeTAD.

Las mediciones se llevaron a cabo con la red libre de interferencias (sin más tráfico en la red del que las computadoras generaban por el pingpong), y de hecho se hicieron en diferentes días bajo las mismas condiciones, obteniendo los mismos resultados.

De acuerdo a lo que se ha explicado, en las redes de computadoras heterogéneas se tienen cuatro alternativas para el manejo de los mensajes entre procesos

1. Ruteo *PvmRouteDefault* con Codificación *PvmDataDefault*, es decir transmitir datos codificados en formato XDR utilizando el proceso pvmd en cada una de las computadoras involucradas.
2. Ruteo *PvmRouteDefault* y sin codificación sino con traducción de la representación de los datos, es decir traduciendo la representación siempre que sea necesario en el proceso destino y utilizando el proceso pvmd en cada una de las computadoras involucradas.
3. Ruteo *PvmRouteDirect* con codificación *PvmDataDefault*, es decir transmitir datos codificados en formato XDR directamente entre las tareas *ping-pong*.

4. Ruteo *PvmRouteDirect* y sin codificación sino con traducción de la representación de los datos, es decir traduciendo la representación siempre que sea necesario en el proceso destino y con los datos transferidos directamente directamente entre las tareas *ping-pong*.

Dado que es difícil encontrar un conjunto de longitudes de mensajes que sea representativo para todas las posibilidades de aplicaciones y paralelización, las longitudes elegidas cubren un amplio rango: desde mensajes de ocho bytes (los necesarios para dos números en punto flotante de precisión simple o uno de precisión doble) hasta mensajes de 10^7 bytes. Las longitudes intermedias elegidas son 10^2, 10^3, $6x10^4$, 10^6 y 10^7. El caso particular de la longitud $6x10^4$ (que no sigue exactamente la "convención logarítmica" de crecimiento de longitud de mensajes) fue elegido para poder luego comparar los resultados con la versión de *pingpong* del sistema operativo (en particular, el de Linux): el comando *ping*, cuya justificación de utilización se describe en la siguiente sección. Dado que la longitud $6x10^4$ es muy cercana a 10^5 como para que ésta proporcione datos significativos, se decidió no incluirla en el reporte de resultados.

Para mayor información y claridad de los datos obtenidos, se presentarán los resultados en dos formatos:
1. Tiempo total de comunicación para el *pingpong*. Esta alternativa se presenta a su vez en formato logarítmico de los tiempos, dadas las longitudes de mensajes elegida.
2. Ancho de banda o bytes/segundo, para una mejor idea de la relación de rendimiento entre lo que se obtiene a nivel de usuario con PVM y el hardware (Ethernet de 10 Mb/s).

C.7.1 Rendimiento con Ruteo entre pvmds y con Codificación

Se supone que esta alternativa es la menos conveniente en cuanto a rendimiento de la red de interconexión de computadoras, y los resultados obtenidos en términos de tiempo de comunicaciones (pingpong) se muestran en la Figura C.6. Se puede ver claramente que el tiempo de latencia de las comunicaciones en PVM entre procesos varía entre 1 y 10 ms dependiendo de la computadora, dado que estos valores inicialmente se repiten a pesar de que las longitudes de mensajes varían entre 8 y 1000 bytes (dos órdenes de magnitud, en términos de crecimiento).

La escala logarítmica de tiempos de la Figura C.6 de alguna manera enmascara las diferencias entre computadoras, aunque se puede ver que para todas las longitudes se tienen diferentes tiempos de comunicación para las diferentes máquinas involucradas.

La Figura C.7 muestra los mismos resultados, pero en función de MB/s (megabytes por segundo), es decir de cantidad de información (2^{20} bytes) por unidad de tiempo. Tanto en esta figura como en todas las siguientes que se expresan en términos de MB/s se pueden visualizar mejor las diferencias relativas entre computadoras, así como también la relación con la capacidad del hardware. En el caso particular de esta figura, para todas las computadoras se puede comprobar lo que era de esperar: a mayor longitud de mensaje se obtiene un rendimiento mayor.

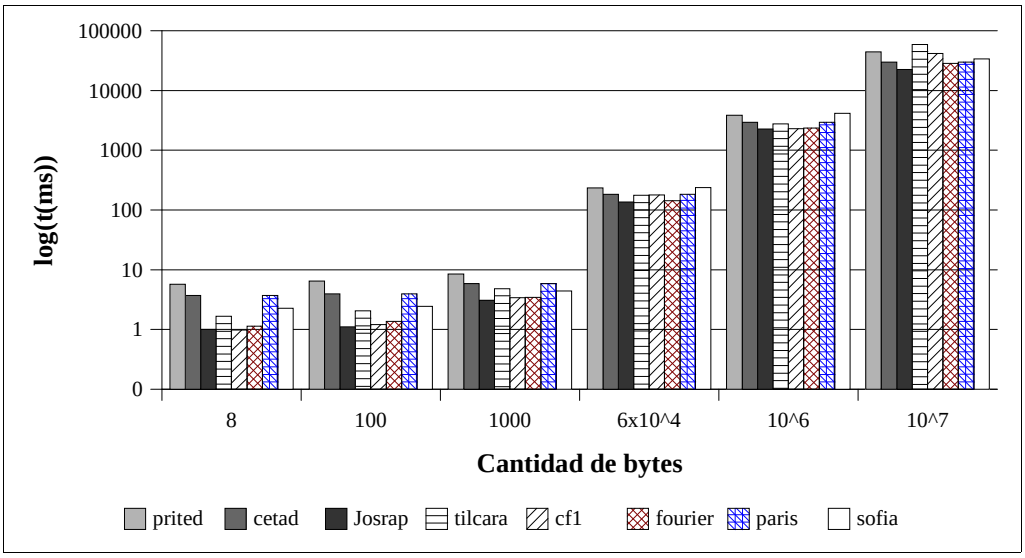


Figura C.6: Tiempos con Ruteo y Codificación *PvmDefault*.

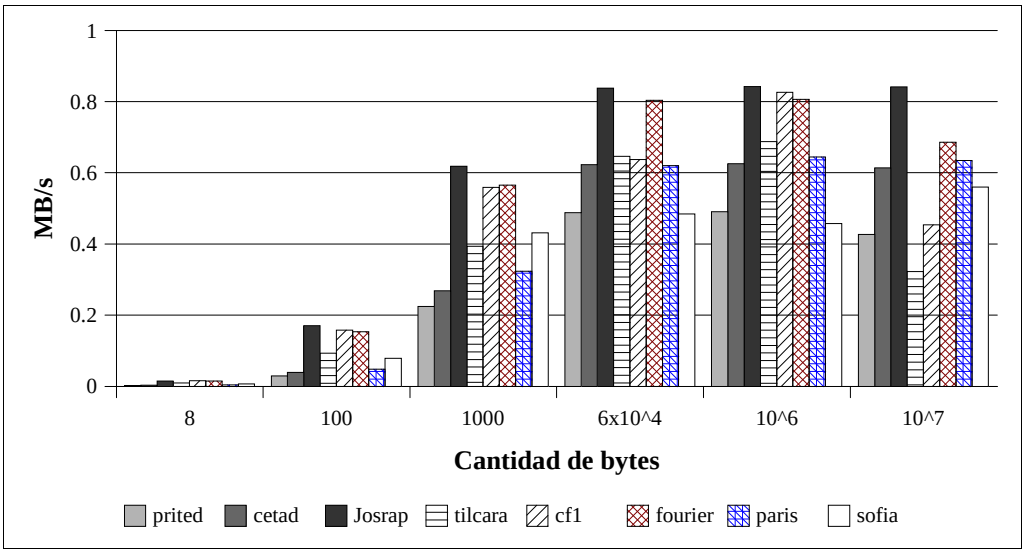


Figura C.7: MB/s con Ruteo y Codificación *PvmDefault*.

En el caso particular de **tilcara**, y **cf1** parece haber un comportamiento anómalo para los mensajes de 10^7 bytes, pero es explicable en función de sus tamaños de memoria principal de 32 MB. Este tamaño es insuficiente para el mensaje, más los buffers de PVM, más los demás procesos (pvmd, sistema operativo, etc.) y sus requerimientos de memoria. En ambos casos se debe recurrir al espacio de *swap* (manejado por el sistema operativo) y en ambos casos también esto produce una degradación notable a nivel de los procesos de usuario del rendimiento. En cierta forma, **prited** también tiene el mismo problema (pérdida de rendimiento por sus 32 MB de memoria principal), pero la pérdida relativa es mucho menor dado que en ningún caso llega a superar los 0.5 MB/s. Las demás computadoras tienen 64 MB de memoria principal o más y por lo tanto no llegan a tener la necesidad de recurrir al espacio de *swap*.

También a partir de la Figura C.7 se puede concluir que:

- existen claras diferencias de rendimiento entre las distintas computadoras, entre poco más de 0.4 MB y poco más de 0.8 MB/s,
- lo mejor que se obtiene es un poco más de 0.8 MB/s para tres de las computadoras (contando **cetadfomec2**, representada con **cf1**),
- el mejor rendimiento obtenido para cada una de las computadoras se logra con longitud de mensajes del orden de 10^4 bytes o más (representado con $6x10^4$ en la figura).

C.7.2 Rendimiento con Ruteo entre pvmds y con Traducción de Representación

La Figura C.8 muestra los resultados obtenidos en términos de tiempos absolutos de comunicación. Comparativamente con los que se muestran en la Figura C.6, los resultados de esta alternativa son similares en términos generales: latencia y tiempos absolutos, lo cual da una idea de la importancia relativamente baja de la forma de codificación de datos (o traducción de la representación, en este caso).

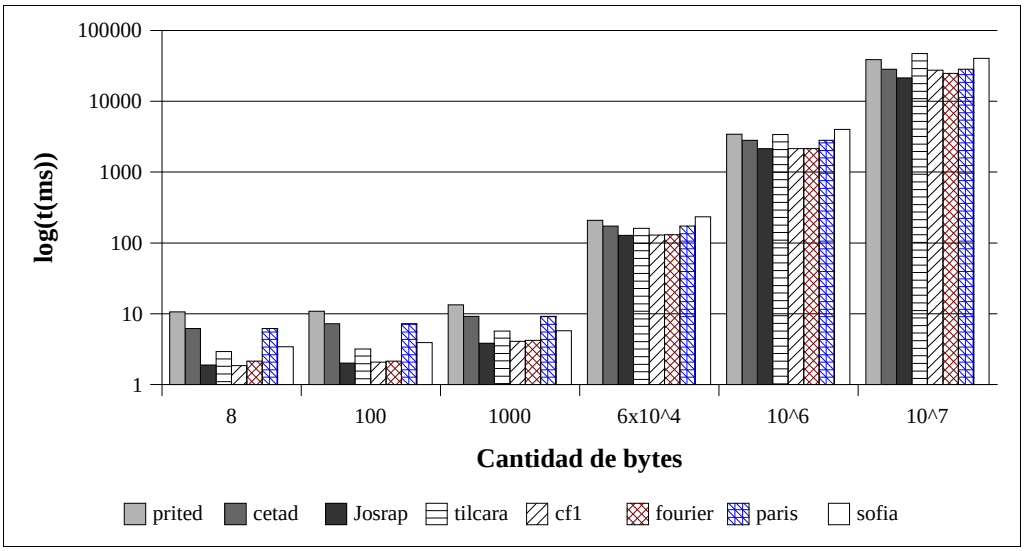


Figura C.8: Tiempos con Ruteo *PvmDefault* y Traducción de Representación.

La Figura C.9 muestra los mismos resultados pero en función de MB/s de acuerdo a los tiempos de comunicación y la cantidad de bytes que se transmiten según la longitud del mensaje. Se puede notar que la disminución en los requerimientos de memoria hacen que las máquinas con 32 MB (**tilcara**, por ejemplo) no reduzcan tan drásticamente su rendimiento cuando los mensajes son de 10^7 bytes, porque hacen un menor uso del espacio de *swap* y por lo tanto el impacto es menor sobre la velocidad de manejo de la memoria.

También se puede ver que como impacto directo de la traducción de la representación se logra que el rendimiento de las comunicaciones con las máquinas de igual representación de datos sea superior. Dado que siempre el proceso *ping* se asigna a una PC con Linux (**purmamarca**), el rendimiento es superior con todas las demás PCs (**tilcara**, **fourier** y

cf1), comparándolo con la alternativa anterior.

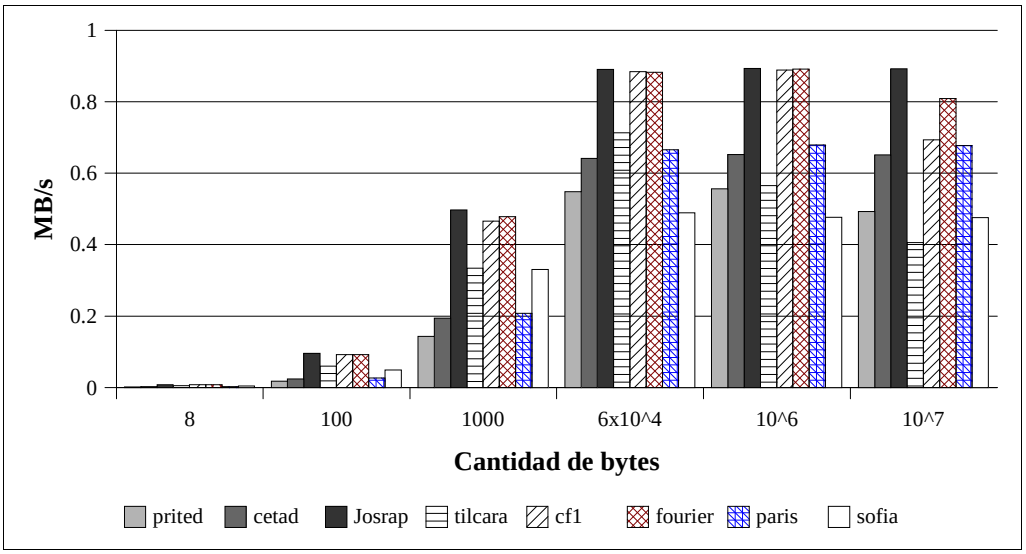


Figura C.9: MB/s con Ruteo *PvmDefault* y Traducción de Representación.

Quizás lo más significativo en ambos casos (codificación XDR y traducción de representación, que es lo único que ha variado hasta ahora) es el muy bajo rendimiento de la computadora **sofia**, que no llega a 0.6 MB/s en ningún caso. Este bajo rendimiento se acentúa cuando se la compara con computadoras varias veces inferiores en velocidad de procesamiento relativa como lo son las computadoras **prited**, **cetad** y **paris**.

C.7.3 Rendimiento con Ruteo entre Tareas de Usuario y con Codificación

La Figura C.10 muestra los tiempos de comunicación obtenidos cuando las tareas de usuario se comunican directamente utilizando el protocolo TCP y con la codificación de datos XDR (PvmDataDefault). Esta alternativa de transmisión de los mensajes es la recomendada por la documentación de PVM cuando las computadoras (y su forma de representar los datos) son heterogéneas.

Comparando los resultados de la Figura C.10 con los anteriores, se pueden observar algunas diferencias menores, como por ejemplo:

- el tiempo de latencia es menor (más cercano a 1 ms que a los 10 ms),
- en escala logarítmica, los tiempos para mensajes pequeños (hasta 1000 bytes) las computadoras no presentan diferencias notables. Nuevamente se debe notar que la escala logarítmica enmascara muchas diferencias que luego se visualizan con claridad en términos de MB/s.

Lo realmente sorprendente que se puede notar en la Figura C.10 es el excesivo tiempo de comunicaciones para mensajes mayores a los 1000 bytes en **prited** y en **sofia**. Aún en la escala logarítmica la diferencia es notable y es muy interesante que, en principio, sucede

con dos computadoras que no están relacionadas (en términos de diseño) de ninguna manera en cuanto a hardware ni a software; **prited** y **sofia**: una Sun SPARCStation 2 de principio de los '90 y una IBM RS/6000 de finales de la misma década.

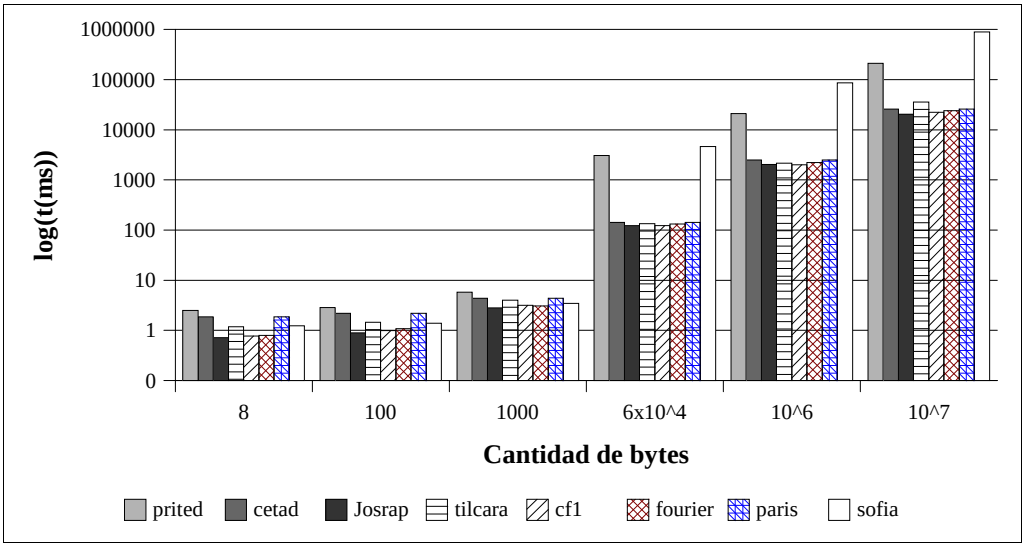


Figura C.10: Tiempos con Ruteo entre Tareas y Codificación *PvmDefault*.

La Figura C.11 muestra los mismos resultados pero en función de MB/s. Ahora las diferencias entre computadoras son más visibles, especialmente con longitud de mensajes mayor a 1000 bytes.

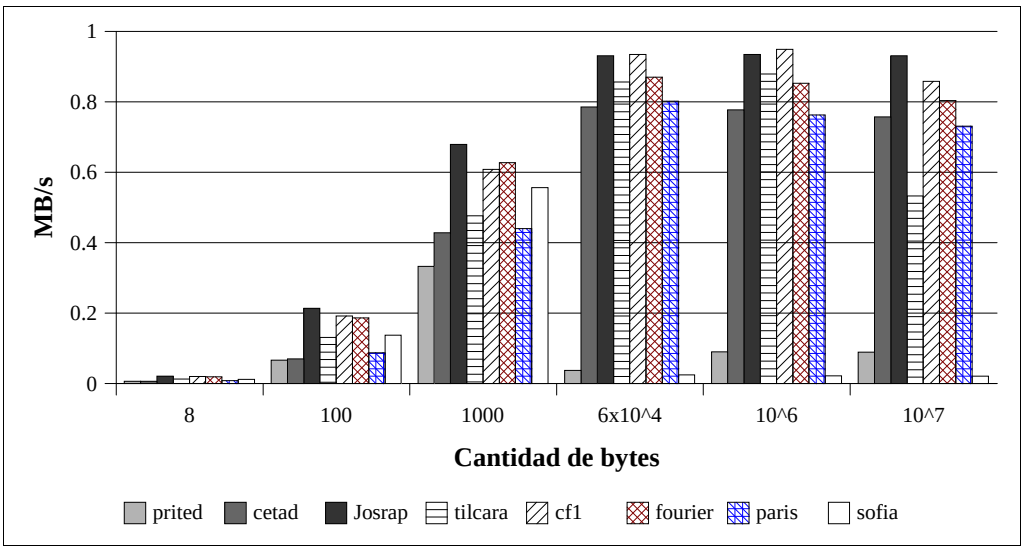


Figura C.11: MB/s con Ruteo entre Tareas y Codificación *PvmDefault*.

La última de las conclusiones enumeradas tiene demasiada importancia dado que se opone totalmente a la documentación y a los reportes al respecto que se han dado de PVM. En este sentido, se verificó la instalación de PVM, los programas *pingpong*, y las posibles razones por las cuales se podría dar esta situación y no se encontró ninguna razón

significativa en relación con la biblioteca PVM.

Pruebas preliminares de conexiones TCP entre las computadoras **purmamarca** y **sofia** mostraron rendimiento similar a nivel de procesos de usuario. Esto implica que las rutinas de comunicaciones de PVM no hacen más que mostrar lo que sucede a nivel de conexión entre las computadoras. La similitud (en términos de rendimiento de las comunicaciones) entres las computadoras **sofia** y **prited** hace asumir el mismo comportamiento a nivel de conexiones TCP.

C.7.4 Rendimiento con Ruteo entre Tareas de Usuario y con Traducción de Representación

La Figura C.12 muestra los tiempos de comunicaciones obtenidos para el ruteo entre tareas (TCP) y con traducción de representación de los datos.

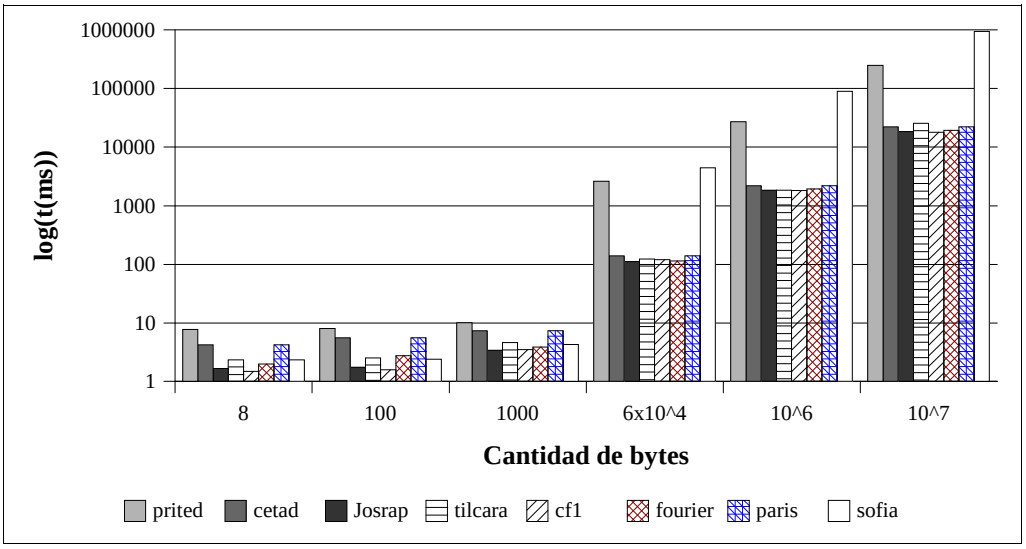


Figura C.12: Tiempos con Ruteo entre Tareas y Traducción de Representación.

La Figura C.13 muestra los mismos resultados en función de MB/s que se obtiene entre tareas de usuario comunicadas con las funciones de PVM, excepto en lo que se refiere a la traducción de la representación de datos, que es propia.

Con esta alternativa se supera por primera vez 1 MB/s de ancho de banda para las comunicaciones entre computadoras con la misma representación de datos. Tanto **purmamarca** donde está ubicada siempre el proceso *ping* como **Josrap**, **tilcara**, **fourier** y **cetadfomec1** (y **cetadfomec2**) son PCs (aunque con distintos procesadores) con sistema operativo Linux. Por lo tanto, los mensajes pueden ser enviados con PvmDataInPlace, lo cual implica que no hay copiado en buffers, ni codificación que pueda incrementar el tamaño final de los datos a transmitir sobre la red de interconexión de computadoras.

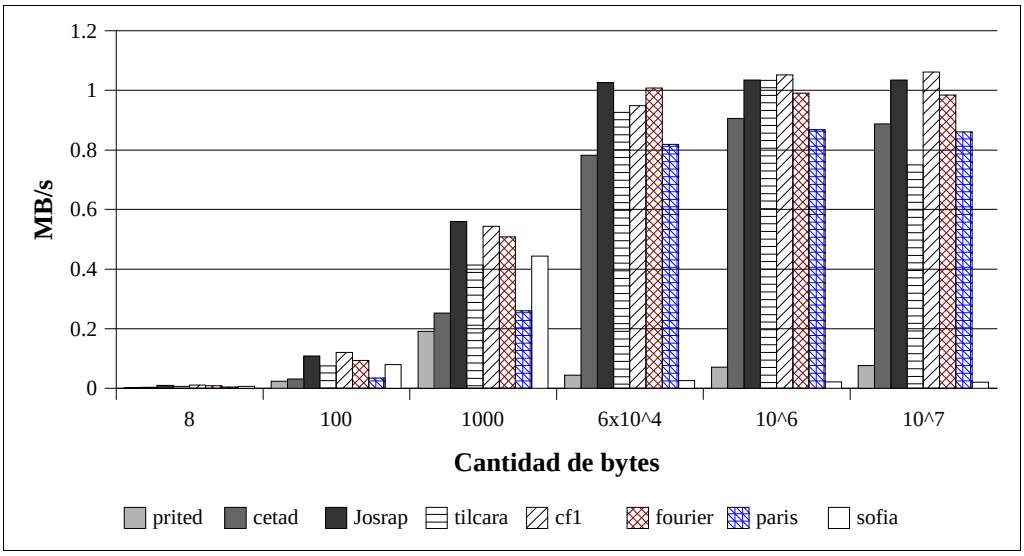


Figura C.13: MB/s con Ruteo entre Tareas y Traducción de Representación.

La información que aportan estas figuras es muy similar a la que aportan las dos anteriores (Figura C.10 y Figura C.11) tanto en valores absolutos como en la relación existente entre las computadoras. También muestra (y en este sentido de alguna manera confirman los resultados anteriores), la gran diferencia en rendimiento que existe entre las computadoras **prited** y **sofia** con respecto a las demás.

El muy bajo rendimiento de **prited** y **sofia** con esta alternativa de comunicaciones (ruteo TCP entre tareas y traducción de representación) entre tareas de PVM hace posible descartar definitivamente que el problema pueda ser por la codificación. Por el contrario, el problema parece ser de las comunicaciones TCP y/o de las comunicaciones de PVM con ruteo TCP.

C.7.5 Conclusiones de la experimentación con PVM

Si se espera utilizar el máximo rendimiento posible de las comunicaciones se debe observar con más cuidado la computadora **sofia**, de la cual es esperable un rendimiento mucho mayor. En el caso particular de los mensajes con ruteo directo, el rendimiento obtenido es realmente muy inferior al esperable y por lo tanto es posible que haya algún error en la forma en que se manejan las comunicaciones TCP en el sistema operativo o en cuanto a la configuración (a nivel de sistema operativo) de las conexiones TCP de algunas computadoras. En este aspecto, la experimentación no hizo más que mostrar que puede haber un (serio) problema en el rendimiento causado por un problema de software.

También en el contexto de rendimiento, es interesante recordar que *todas* las computadoras tienen la misma capacidad en cuanto a hardware de comunicaciones. Esto significa que todas las computadoras, independientemente del fabricante, tienen interfase (NIC) Ethernet 802.3, y por lo tanto todas serían capaces de comunicarse a 10 Mb/s, lo cual implica en teoría 1.25 MB/s (1.25×2^{20} bytes por segundo). Se puede sospechar cierta pérdida de

rendimiento sostenido dada por la sobrecarga (overhead) impuesta por el sistema operativo y sus buffers, procesos, etc. más todo lo que se refiere al mismo PVM, pero no se tiene *a priori* una idea cuantificada de la pérdida de rendimiento que toda esta sobrecarga implica. Por lo tanto, siempre puede quedar pendiente conocer el máximo rendimiento posible de las comunicaciones para las tareas de usuario si solamente se monitoriza el rendimiento con el método de *pingpong* con tareas utilizando PVM.

Volviendo al objetivo inicial de la estimación de α y β, se tienen algunas conclusiones muy importantes: el rango de los valores y la heterogeneidad de los valores.

- Dependiendo de la alternativa de comunicación de mensajes elegida, la latencia varía entre 1 y 10 ms.
- También dependiendo de la alternativa elegida (y excluyendo los casos particulares de **prited** y **sofia**), la tasa de transferencia (o ancho de banda) varía entre algo menos de 0.5 MB/s y algo más de 1 MB/s.
- Sea cual sea la alternativa elegida, tanto la latencia como la tasa de transferencia dependen de la computadora.

Esta última conclusión es un tanto desalentadora en cuanto a que, un subsistema que es en teoría homogéneo como el de la interconexión de computadoras, "se transforma" en heterogéneo cuando se mira desde las tareas de usuario que componen una aplicación paralela.

La consecuencia inmediata de la heterogeneidad de tiempos de comunicación en redes Ethernet con PVM es: un mensaje que debería llegar en la misma cantidad de tiempo a cualquier proceso en otro procesador, ahora llegará en una cantidad de tiempo dependiente de los procesadores origen y destino, más allá de la solución adoptada para resolver las diferencias de representación de datos.

Para intentar corroborar los resultados obtenidos con PVM, se diseña un nuevo conjunto de experimentos que intentan aumentar la precisión en cuanto a:

- α y β para procesos de usuario
- heterogeneidad o no de las comunicaciones
- los casos particulares de bajo rendimiento de **prited** y **sofia**.

C.8 Experimentación con el Comando ping de Linux

El comando *ping* (al menos en su *versión* del sistema operativo Linux) es suficientemente versátil como para:

- Medir tiempos de comunicaciones a nivel de aplicaciones de usuario, y
- Generar "mensajes" de distintas longitudes.

De hecho, cualquier usuario sin permisos especiales puede ejecutar el comando *ping*, lo cual genera un proceso de usuario que produce un *paquete pingpong* por segundo a nivel

de protocolo ICMP (Internet Control Management Protocol). El tiempo de ida y vuelta del paquete es reportado por el mismo comando *ping*, con lo cual no hay que agregar ningún tipo de instrumentación. La longitud del *paquete pingpong* se puede variar utilizando una opción del comando, y esto a su vez permite observar el rendimiento de la red de interconexión en función de la cantidad de datos transferidos, considerando al *paquete pingpong* como un mensaje.

La Figura C.14 muestra los tiempos de comunicación involucrados para distintas cantidades de bytes que contienen los *paquetes pingpong*.

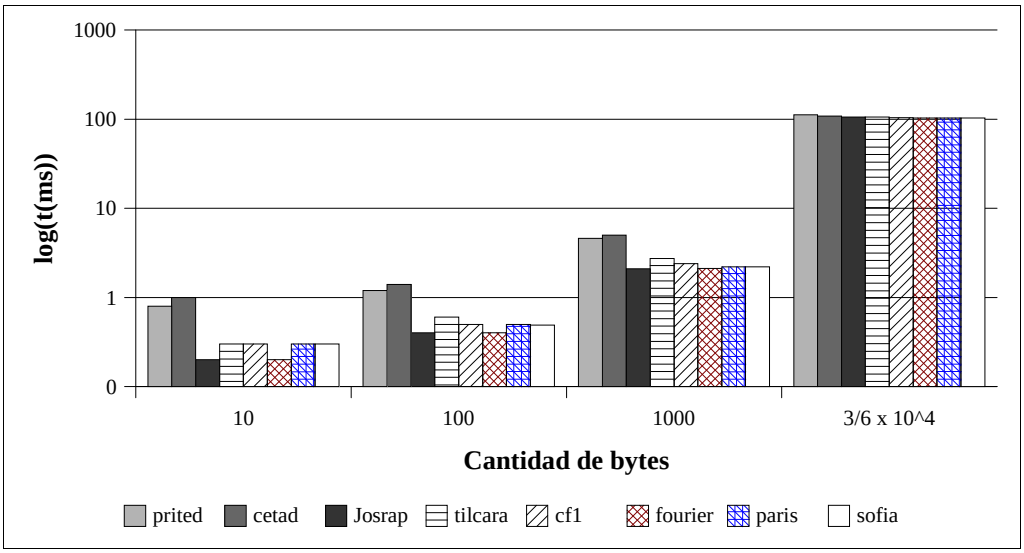


Figura C.14: Tiempos con ping de Linux.

Es necesario hacer algunas aclaraciones con respecto a las longitudes de los *paquetes pingpong* así como también al último identificador que aparece como “3/6 x 10^4”. Todo paquete ICMP debe ser menor a 64 KB tal como los paquetes IP y por lo tanto lo que se puede medir con el comando ping llega a esa cantidad de bytes que se transfieren.

En el caso particular de las computadoras con el sistema operativo Sun 4.1.x (Sun OS basado en BSD), **prited** y **cetad**, por alguna razón no documentada no responden a los paquetes ICMP de más de 32 KB y por lo tanto los pingpong con esas computadoras se llevaron a cabo con tamaño máximo de 30000 bytes (multiplicando luego el tiempo por dos para ecualizarlos con los tiempos de las demás computadoras.

A partir de la información de la Figura C.14 se puede concluir que:

- El tiempo de latencia (α) de mensajes para tareas de usuario (que se comunican con protocolo ICMP) es del orden de 1 ms. Como en PVM, el tiempo de latencia depende de la computadora, aunque el rango de variación es bastante menor.
- Recién a partir de mensajes de 1000 bytes o mayores el tiempo comienza a ser proporcional a la cantidad de datos que se transfieren. Esto significa, como siempre en este contexto, que hasta esa longitud de “mensajes” (en este caso, *paquetes de pingpong*) el tiempo de latencia es suficientemente grande como para ser mayor que el tiempo de transmisión de datos.

La Figura C.15 muestra los mismos resultados pero en términos de MB/s de la transferencia de datos de los paquetes de *pingpong*. Se pueden notar las diferencias entre las computadoras enmascaradas en parte por la escala logarítmica de los tiempos de la figura anterior.

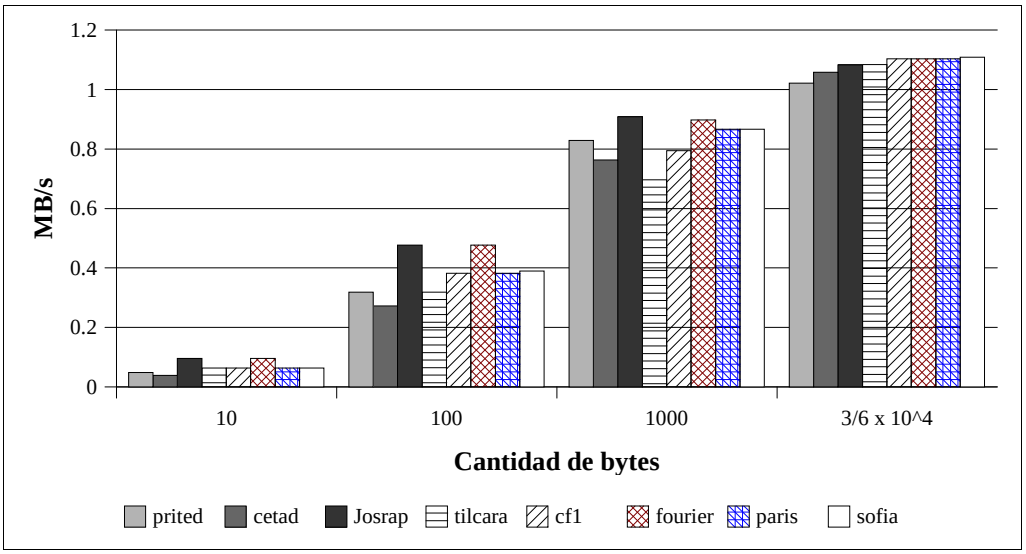


Figura C.15: MB/s con ping de Linux.

Como sucede con la latencia, la tasa de transferencia de información no es exactamente igual para todas las máquinas, pero el rango de variación es mucho menor que el que se tiene cuando los procesos se comunican en PVM. También en términos de tasa de transferencia se confirma que:

- El máximo obtenible con ICMP es casi el físico, con lo cual la diferencia entre este máximo (con el ping de Linux) y el que se obtiene en PVM es debido a la sobrecarga de PVM.
- El caso particular de bajo rendimiento de las computadoras **prited** y **sofia** no se debe a hardware ni a los protocolos más cercanos al hardware (como IP e ICMP).

Como con la experimentación con PVM, recién para tamaños de mensajes del orden de 10^4 bytes (30000 bytes para **prited** y **cetad** y 60000 para el resto) se logra el mejor rendimiento de las comunicaciones. Sin embargo, con estos experimentos se agrega información importante que complementa lo que aportó la experimentación inicial llevada a cabo con PVM.

C.9 Conclusiones de la Experimentación de PVM y ping de Linux

A partir de los resultados obtenidos en PVM y con el comando ping de Linux cabe intentar un intermedio. Es muy probable que no se llegue a tener un rendimiento exactamente igual

al observado con el comando ping porque:
- Con el comando ping no se tienen en cuenta ni se resuelven las diferencias de representación de datos.
- Necesariamente se deben implementar rutinas de comunicación que puedan superar los 64 KB de longitud de mensajes. Expresado de otra manera, lamentablemente no siempre los mensajes a transferir entre procesos se pueden encapsular en un solo paquete de un protocolo de comunicaciones muy cercano al hardware tal como ICMP.

Sin embargo, no parece que estas dos restricciones, aunque fuertes, sean las que hacen que PVM tenga:
- Tan bajo rendimiento de comunicaciones, como en el caso de **prited** y **sofia** con ruteo directo.
- Rendimiento tan heterogéneo dependiente de las computadoras entre las cuales se transfieren datos.

De alguna manera, por un lado el comando ping parece dar una versión muy optimista del rendimiento de la red de comunicaciones y por el otro, PVM parece tener una sobrecarga muy alta. Esta gran sobrecarga de PVM produce una gran reducción del rendimiento de la red de interconexión y esta reducción del rendimiento es proporcional a la velocidad relativa de la computadora. Por lo tanto, el rendimiento de las comunicaciones pasa a ser dependiente de las computadoras que se comunican y tan heterogéneo como la heterogeneidad de las computadoras.

Debido a esto, parece natural buscar una solución intermedia en cuanto a rendimiento de la red de interconexión de computadoras. Esto implica tener e nivel de los procesos de una aplicación paralela de usuario mejor rendimiento y más homogéneo (de acuerdo al hardware de comunicaciones) que el que se obtiene con PVM aunque quizás no sea el obtenido con el comando ping.

Además, es necesario recordar que, si bien toda la experimentación (y por lo tanto las conclusiones que a las que se llegaron) son basadas en el método de *pingpong*, es decir por comunicaciones punto a punto entre dos y sólo dos procesos:
- el problema original a resolver es el de la multiplicación de matrices,
- el algoritmo desarrollado para multiplicar matrices en redes de computadoras está basado en broadcasts,
- es importante tener una implementación eficiente de los broadcasts dada su amplia utilización en los programas paralelos [11].

Como se puede concluir a partir de la experimentación que se mostró en este capítulo, el rendimiento de las comunicaciones punto a punto con PVM no es del todo satisfactorio. Por lo tanto, no hay razón para suponer que el rendimiento de las comunicaciones colectivas, que son las más importantes para la aplicación que se está analizando y para muchas otras, sí sea satisfactorio.

Por las razones expuestas, en el capítulo siguiente se explicará el desarrollo y el rendimiento obtenido en principio para una rutina de comunicaciones colectivas broadcast que puede ser extendida a una biblioteca de comunicaciones colectivas.

C.10 Broadcasts Basados en UDP

Los principales objetivos de rendimiento para el desarrollo de una rutina broadcast distinta de la que proveen bibliotecas de pasaje de mensajes como PVM y las implementaciones de MPI son:

- Mejorar el rendimiento obtenido desde los procesos de usuario en relación al obtenido con las bibliotecas de "propósito general" como PVM.
- Obtener homogeneidad de rendimiento de acuerdo a la homogeneidad del hardware, que en caso de PVM no se verifica en la experimentación.

Además, utilizando PVM, las formas de enviar un mismo mensaje a más de un proceso destino son dos:

- La rutina de multicast, **pvm_mcast().**
- la rutina de broadcast en un grupo, **pvm_bcast().**

Y la implementación de ambas rutinas se basa en múltiples mensajes punto a punto. Es decir que tanto pvm_mcast() como pvm_bcast() implican que, como mínimo, el mismo mensaje es enviado *m* veces desde la computadora origen (donde está ejecutándose el proceso que envía el mensaje) hacia las *m* máquinas donde hay al menos un proceso destino del mensaje. Si, por ejemplo, un mensaje broadcast o multicast tiene cinco receptores y cada uno de estos procesos receptores está ejecutándose en una máquina distinta (y distinta de la máquina en donde se ejecuta el proceso que envía el mensaje), el tiempo total del mensaje será aproximadamente igual a cinco veces el tiempo del mismo mensaje si se envía a otro proceso que se ejecuta en otra máquina. Para estas comunicaciones punto a punto entre máquinas se utilizan las mismas rutinas con las que se ha realizado la experimentación.

En principio, se requiere una rutina para mensajes broadcast con rendimiento *aceptable* en redes Ethernet, donde aceptable se puede definir de acuerdo con:

- Valores absolutos de tiempo de comunicaciones más cercanos a los que provee el hardware de los que se vieron en la experimentación con PVM.
- Escalabilidad en cuanto a cantidad de máquinas, dado que al aprovechar las capacidades de broadcast de las redes Ethernet el mismo mensaje puede ser enviado y recibido hacia/en tantas computadoras como haya conectadas. Evidentemente habrá una penalización dependiente de la cantidad de computadoras que reciben un mismo mensaje por razones de sincronización y mantenimiento de la confiabilidad de los datos que se transfieren, pero esta penalización debería ser muy lejana a la repetición del mismo mensaje tantas veces como computadoras distintas deben recibirlo.

El rendimiento de pvm_mcast() como pvm_bcast() no es aceptable y por lo tanto la biblioteca PVM ya no sería útil para los mensajes broadcast que requiere el algoritmo de multiplicación de matrices en paralelo. En este punto se tienen varias alternativas, siendo las dos más importantes:

1. Utilizar otra biblioteca de pasaje de mensajes, tal como alguna implementación de MPI, que es lo que se suele recomendar para este tipo de arquitecturas paralelas.
2. Implementar una rutina de mensajes broadcast (y eventualmente toda una biblioteca de

comunicaciones colectivas) para utilizar explícitamente la capacidad de broadcast de las redes Ethernet.

La utilización de otra biblioteca de pasaje de mensajes tiene, en principio un inconveniente fundamental desde el punto de vista de rendimiento, o de "predicción" de buen rendimiento de los mensajes broadcast. En el caso específico de MPI, es claro que el rendimiento es dependiente de la implementación. Más específicamente, la implementación será la que determina el grado de aprovechamiento de la/s característica/s de las redes Ethernet para los mensajes broadcast. En este sentido, MPI y en particular todas sus implementaciones comparten *cierto grado de incertidumbre* respecto del rendimiento de los mensajes broadcast con todas las demás bibliotecas de pasaje de mensajes, incluida PVM. La diferencia en este caso son los experimentos específicos se llevaron a cabo y que determinaron las características de rendimiento de los mensajes broadcast (y multicast) para PVM y no para las demás bibliotecas. De hecho, es bastante difícil que las bibliotecas de pasaje de mensajes sean optimizadas para las características de las redes Ethernet dado que:

- En general, las bibliotecas son propuestas de una forma u otra como estándares para las máquinas paralelas de pasaje de mensajes y por lo tanto no tiene sentido orientarlas hacia un tipo específico de redes de interconexión. De hecho, tanto PVM como MPI se han implementado para distintos tipos de máquinas paralelas y por lo tanto no tiene sentido orientarlas *a priori* a las redes de interconexión Ethernet.
- En general, las bibliotecas proveen una gran cantidad de rutinas de comunicación. Aunque en teoría se puede afirmar que con las primitivas **send** - **receive** para la comunicación de procesos punto a punto es suficiente, también se ha concluido que existe una gran variedad de rutinas de comunicación que se consideran útiles y hasta necesarias en algunos casos. Quizás el ejemplo más claro al respecto sea la propia definición del estándar MPI. En este contexto, es muy difícil orientar u optimizar una o una clase de rutinas de comunicaciones para una o una clase de redes de interconexión sin producir una biblioteca excesivamente costosa (en términos de desarrollo, mantenimiento, etc.) y/o *demasiado* específica.

Por estas razones, se eligió implementar una rutina de mensajes broadcast entre procesos de usuario con un conjunto de premisas de diseño e implementación, de forma tal que:

- Aproveche el *propio* broadcast de las redes Ethernet, y de esta manera se optimiza en cuanto a rendimiento. Dado que el algoritmo depende exclusivamente de los mensajes broadcast, al aprovechar el broadcast de las redes Ethernet automáticamente se tiene una buena expectativa en cuanto a escalabilidad ya que se espera que el tiempo de comunicaciones se mantenga constante y no aumente de manera proporcional a la cantidad de computadoras que se utilizan.
- Sea suficientemente simple para no imponer una carga demasiado pesada en cuanto a implementación y mantenimiento. Además, es claro que simplicidad *per se* hace un gran aporte para el rendimiento óptimo. Por otro lado, la propuesta es suficientemente específica como para que la implementación sea simple.
- Con el máximo de portabilidad posible, para ser utilizada si es posible incluso en el contexto de las redes de interconexión que no sean Ethernet.
- Implementada e instalada desde modo usuario, sin cambiar el sistema operativo (el *kernel*) y sin que sea necesario obtener permisos especiales (*superuser*). Es normalmente aceptado que los mejores resultados en cuanto a rendimiento se obtienen

adaptando el kernel y/o con la posibilidad de manejar las prioridades de los procesos, tal como en [5] [4] [12]. Estas posibilidades se descartan dado que:
    - En general, las bibliotecas de uso libre no utilizan estas características y por lo tanto sería como cambiar el contexto de desarrollo de software paralelo. Básicamente, un usuario que haya utilizado siempre PVM nunca tuvo ni tiene por qué obtener prioridades especiales y aún cambiar el mismo sistema operativo.
    - La propuesta original se dirige a redes de computadoras instaladas y por lo tanto cada computadora no necesariamente tiene como objetivo único y/o principal el cómputo paralelo. De hecho, se puede tener el caso de distintos administradores para cada una de las computadoras a utilizar en paralelo y esto produce, por lo menos, un múltiple trabajo de administración que en general no es fácil de resolver.
- Eventualmente pueda ser extendida a toda una biblioteca de comunicaciones colectivas, tal como otras propuestas [2] [1] [3] pero orientada específicamente a las redes de interconexión Ethernet.

La mayoría (sino *todas*) las premisas anteriores se cumplen al basar todo el diseño y la implementación de la rutina de broadcast en el protocolo estándar UDP (User Datagram Protocol) [8] sobre IP (Internet Protocol) [9] ya que:
- UDP permite enviar un mismo dato o conjunto (paquete) de datos a múltiples destinos a nivel de aplicaciones de usuario.
- Tal como se verificó en todas las máquinas utilizadas, la implementación del protocolo UDP aprovecha directamente la capacidad de broadcast de las redes Ethernet.
- En principio, parece razonable que el broadcast implementado directamente como parte del protocolo UDP tenga mejor rendimiento que el implementado por un usuario. Si en una red ATM, por ejemplo, se tiene la posibilidad de utilizar UDP es muy probable que el broadcast de UDP sea *mejor* (en términos de rendimiento) que el que se pueda implementar desde los procesos de usuario. Aunque el rendimiento no se tenga en cuanta, siempre que exista una implementación del protocolo UDP sea podrá utilizar el broadcast propuesto, independientemente de que la red de interconexión sea Ethernet o no.
- La interfase de usuario provista por los sockets es suficientemente simple y ampliamente extendida a todas las versiones de UNIX, como para simplificar la implementación de la rutina de broadcast, aún cuando se deben resolver problemas relacionados con *sincronización* de procesos (en una misma y en diferentes computadoras) y *confiabilidad* de las comunicaciones.
- Los protocolos UDP, TCP e IP son fácilmente utilizables desde los procesos de usuario, al menos en las computadoras estándares de las redes locales instaladas.

En resumen, se propone una *nueva* rutina de mensajes broadcast basada en UDP y portable como mínimo a todas las versiones de UNIX utilizadas en todas las redes locales en las cuales se lleva a cabo la experimentación. Con esta rutina de mensajes broadcast se realizaron los mismos experimentos que para las rutinas de comunicación de PVM. Inicialmente se muestran los resultados de las comunicaciones punto a punto (mensaje “broadcast” o “multicast” utilizando un solo proceso receptor) y por último también se muestran los resultados de mensajes broadcast con pvm_mcast() y pvm_bcast() de PVM y con la rutina propuesta basada en UDP.

C.10.1 Un Unico Receptor (Mensajes Punto a Punto)

Con el fin de comparar los resultados con los que ya se obtuvieron por la experimentación con las rutinas de comunicaciones punto a punto de PVM y con el comando ping de Linux se utilizó la rutina de mensajes broadcast con un único proceso receptor. Los resultados referentes al tiempo de comunicaciones se muestran en la Figura C.16. Se debe destacar que estos tiempos de comunicaciones entre dos máquinas pueden no ser óptimos dado que la rutina de comunicaciones está diseñada para mensajes broadcast que involucran más de una computadora que debe recibir mensajes. De todas maneras se puede utilizar para comparar con los resultados que se muestran en

- La Figura C.14 con el rendimiento de las comunicaciones de acuerdo con el comando ping (protocolo ICMP).
- La Figura C.12 con el rendimiento de las comunicaciones de PVM con ruteo entre tareas (protocolo TCP) y traducción de la representación de los datos.
- La Figura C.8 con el rendimiento de las comunicaciones de PVM con ruteo entre los procesos de PVM (pvmd, protocolo UDP) y traducción de la representación de los datos.

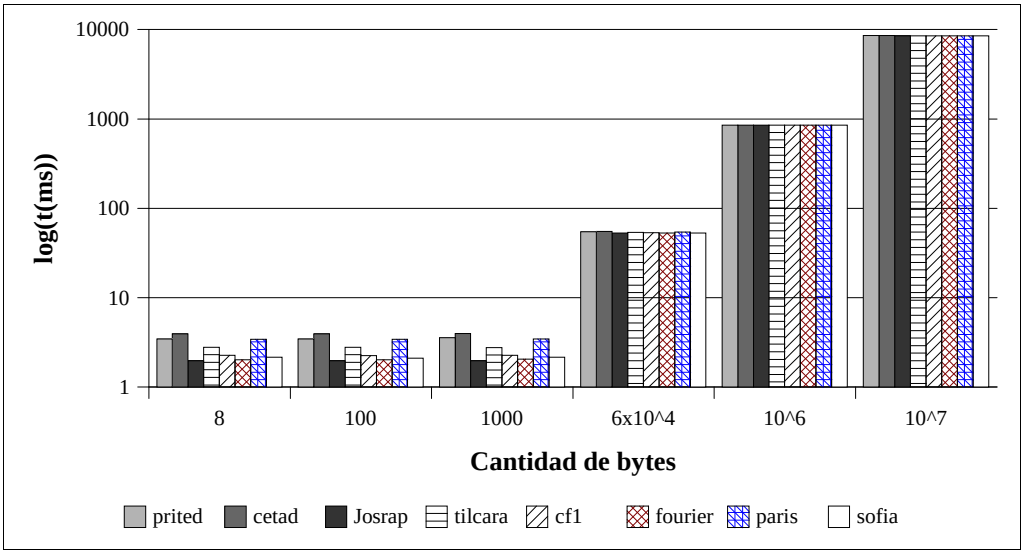


Figura C.16: Tiempos “Punto a Punto” con Broadcast Basado en UDP.

Comparando estos resultados con los del comando ping de Linux (Figura C.14) se tiene que:

- La latencia es bastante mayor y se verifica que al menos hasta la transferencia de mil bytes el tiempo de latencia es el que domina el tiempo total. Por otro lado, se confirma que la latencia de las comunicaciones es dependiente de las máquinas involucradas en la transferencia de datos.
- El tiempo de comunicaciones es similar al que se obtiene con el comando ping de Linux para 30000 (estaciones de trabajo **cetad** y **prited**) y 60000 bytes (las demás computadoras).

Comparando estos resultados con los de las rutinas de comunicaciones punto a punto de PVM (Figura C.8 y Figura C.12) se tiene que:

- La latencia es similar a la que se tiene con PVM, y en muchos casos casi idéntica.
- Las computadoras **prited** y **sofia** no muestran ninguna anomalía de rendimiento para ningún tamaño de mensajes. Esto confirma que el problema se debe a la configuración y/o a la implementación del protocolo TCP en esas computadoras.
- Al menos a partir de la longitud de mensajes de 60000 bytes, el rendimiento de las comunicaciones es homogéneo, tal como es de esperar desde el punto de vista del hardware de comunicaciones.
- El rendimiento con el broadcast basado en UDP es bastante superior al que se tiene con las rutinas de comunicaciones punto a punto de PVM. Aún a pesar de la escala logarítmica, se puede notar que el tiempo de comunicaciones es muy cercano al óptimo, tal como si no hubiera sobrecarga (overhead) de las distintas capas de software involucrado (sistema operativo, la rutina de broadcast, etc.). Por ejemplo, el tiempo de comunicaciones de los mensajes de 10^6 bytes es muy cercano a un segundo (o 1000 milisegundos, tal como se muestra en la Figura C.16).

La Figura C.17 muestra los mismos resultados en términos de ancho de banda asintótico, donde se puede verificar con claridad que los resultados son altamente satisfactorios en términos de rendimiento.

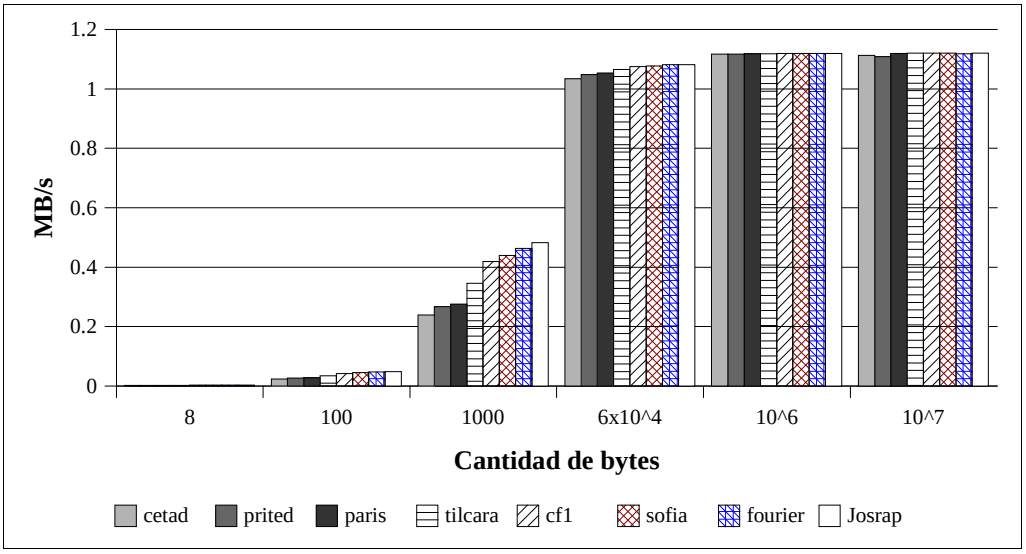


Figura C.17: MB/s “Punto a Punto” con Broadcast Basado en UDP.

Con estos resultados de comunicaciones punto a punto se tiene, como mínimo:

- Casi todo el rendimiento del hardware de comunicaciones a nivel de los procesos de las aplicaciones paralelas. La sobrecarga de todas las capas intermedias de software casi no afecta el rendimiento final entre los procesos.
- En ausencia de colisiones, el rendimiento es independiente de la red de comunicaciones es independiente de las computadoras involucradas. La heterogeneidad de las computadoras con sus diferencias relativas en cuanto a capacidad de cálculo no se traduce, como en PVM, a “rendimiento heterogéneo”.

C.10.2 Mensajes Broadcasts

El objetivo final de la rutina de comunicaciones broadcast no es la transferencia de datos de un proceso a otro, sino la transferencia de un proceso a una cantidad determinada de procesos que se ejecutan en diferentes computadoras. Es por esta razón que se debe verificar al menos por experimentación que se pueden llevar a cabo mensajes broadcast entre procesos con rendimiento cercano al óptimo y relativamente independiente de la cantidad de procesos receptores, o computadoras involucradas en los mensajes broadcast.

La Figura C.18 muestra los tiempos de comunicaciones involucrados para distintas longitudes de mensajes broadcast y diferentes cantidades de receptores asignados en diferentes máquinas. Con el objetivo de comparar mejor las diferentes longitudes de mensajes se elige mostrarlas a todas en mismo gráfico en vez de mostrar un gráfico por cada longitud de mensajes. Separadas por las líneas de puntos verticales, sobre el eje **x** se muestran los tiempos de mensajes broadcast para distinta cantidad de receptores (máquinas). Los resultados que se muestran de PVM son independientes de la utilización de la rutina pvm_mcast() o pvm_bcast(), porque tienen rendimiento similar.

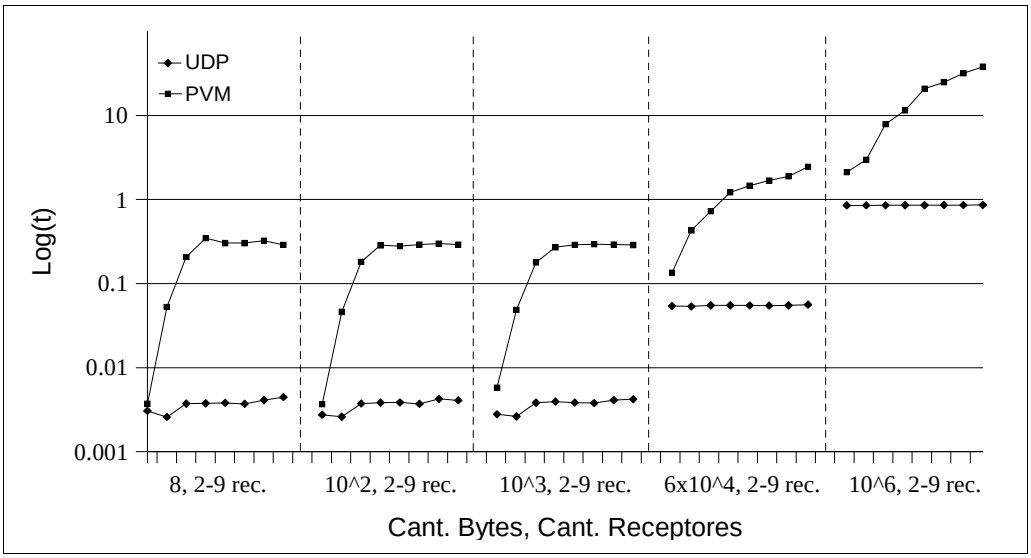


Figura C.18: Tiempos de Broadcast con PVM y Basado en UDP.

Se puede notar que el tiempo con las rutinas de PVM se mantiene bastante independiente de la longitud de mensajes de 8, 100 y 1000 bytes respectivamente. De hecho, depende más de las máquinas involucradas. Para estas longitudes de mensajes la rutina propuesta que se basa en UDP utiliza tiempos relativamente constantes y también independientes de la cantidad de computadoras involucradas. Para los mensajes broadcast de 60000 y 1000000 de bytes el tiempo es directamente proporcional a la cantidad de receptores cuando se utilizan las rutinas de PVM. Una vez más, cuando se utiliza la rutina basada directamente en UDP el tiempo (además de ser un poco mejor que el mejor de PVM) es relativamente constante e independiente de las computadoras involucradas.

La Figura C.19 muestra los mismos resultados pero en términos de MB/s y por lo tanto se pueden notar algunos detalles enmascarados por la escala logarítmica de tiempos del gráfico anterior. Para el cálculo del ancho de banda asintótico o tasa de transferencia o MB/s de un mensaje broadcast se debe recordar que el mensaje es único, los mismos datos deben llegar a múltiples destinos independientemente de que la implementación se haga con múltiples mensajes punto a punto o de otra manera.

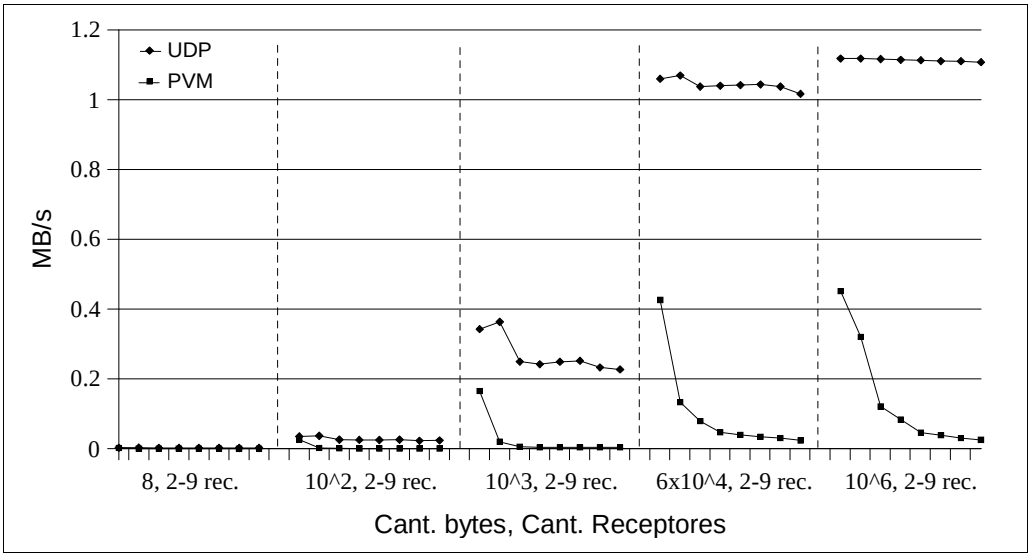


Figura C.19: MB/s de Broadcast con PVM y Basado en UDP.

Como se explicó anteriormente, el tiempo de comunicaciones es dominado por la latencia al menos para los mensajes de hasta 1000 bytes. Por lo tanto, el rendimiento en términos de ancho de banda o tasa de transferencia es bastante pobre utilizando PVM o la rutina de broadcasts basada en UDP. Para los mensajes de 60000 bytes se puede notar fácilmente que el rendimiento al utilizar las rutinas provistas por PVM es dependiente de la cantidad de receptores y es relativamente constante cuando se utiliza la rutina basada en UDP. Las variaciones en este último caso se deben básicamente a que para esta longitud de mensajes la latencia de cada máquina se hace notar en el tiempo total de las comunicaciones. Para los mensajes de 10^6 bytes la latencia de cada máquina ya es mucho menor al tiempo de transferencia de los datos y por lo tanto el rendimiento tiene mucha menor variación en ancho de banda asintótico. En el caso de PVM, se verifica una vez más que el rendimiento disminuye a medida que la cantidad de computadoras involucradas aumenta y por lo tanto se puede afirmar que la implementación del broadcast utiliza múltiples mensajes punto a punto.

C.11 Broadcasts en la Red Local del LQT

La experimentación en la red local del LQT es similar en cuanto a los mensajes broadcast. Se dan las mismas características de rendimiento dependiente de la cantidad de

computadoras involucradas para las rutinas de PVM y rendimiento casi constante para la rutina propuesta basada directamente en UDP. A diferencia de lo que sucede en el CeTAD la heterogeneidad es menor ya que, por ejemplo, no hay diferencias en cuanto a la representación de datos, dado que todas las computadoras disponibles son PC. También a diferencia de lo que sucede en el CeTAD la cantidad de computadoras es menor y por lo tanto la máxima degradación de rendimiento (cuando se utilizan todas las computadoras) es también menor.

Una de las diferencias más notables que arroja la experimentación del LQT con respecto a la detallada del CeTAD se refiere a la latencia de los mensajes punto a punto. Dado que las máquinas son bastante más similares y las diferencias de rendimiento en cuanto a capacidad de cómputo son más reducidas, la latencia de los mensajes está mucho más acotada en cuanto al rango de valores absolutos. De hecho, esto independiza aún más el rendimiento de las comunicaciones entre procesos de usuario dado que ahora no solamente el ancho de banda asintótico es cercano al del hardware de la red de interconexión (utilizando la rutina de mensajes broadcast basada en UDP) sino que el tiempo completo de comunicaciones es mucho más cercano al que se puede estimar con los valores de hardware.

C.12 Broadcasts en la Red Local del LIDI

Como es de esperar en la red local del LIDI, el rendimiento de las comunicaciones es mejor dado que la red de comunicaciones es Ethernet de 100 Mb/s en vez de 10 Mb/s como las del CeTAD y del LQT. Aún así se confirma con la experimentación que para PVM el tiempo de los mensajes broadcast es dependiente de la cantidad de computadoras involucradas. También se confirma que el tiempo de los mensajes broadcast es relativamente independiente de la cantidad de máquinas involucradas cuando se utiliza la rutina basada directamente en UDP.

Es muy interesante el impacto de la latencia en el tiempo total de comunicaciones cuanto el ancho de banda del hardware es diez veces mayor. Dado que gran parte de la latencia se debe a la sobrecarga de factores externos a la propia red de interconexión Ethernet (sistema operativo, protocolos, rutinas de comunicaciones entre procesos, etc.) el tiempo absoluto de la latencia es bastante independiente de la capacidad en ancho de banda. Al multiplicar por diez el ancho de banda de la red Ethernet, implícitamente se multiplica (aunque no necesariamente por diez) el “peso” del tiempo de la latencia en el tiempo total de las comunicaciones. La cuantificación de estos factores debería calcularse al menos con el diseño de experimentos específicos que están fuera del alcance de esta Apéndice .

El impacto de la latencia en el tiempo final de las comunicaciones es definitivamente importante teniendo en cuenta la tendencia a utilizar redes con mayor capacidad de ancho de banda. De todas maneras, es de esperar que la latencia a nivel de hardware de interconexión se reduzca proporcionalmente con el aumento de ancho de banda y también que el overhead impuesto para los procesos de usuario se reduzca.

Referencias

[1] Bala V., J. Bruck, R. Cypher, P. Elustondo, A. Ho, C. Ho, S. Kipnis, M. Snir, “CCL: A Portable and Tunable Collective Communication Library for Scalable Parallel Computing”, Proc. of the 8th International Conference on Parallel Processing, IEEE, April 1994.

[2] Banikazemi M., V. Moorthy, D. Panda, “Efficient Collective Communication on Heterogeneous Networks of Workstations”, Proc. International Conference on Parallel Processing, pp. 460-467, 1998.

[3] Barnett M., S. Gupta, D. Payne, L. Shuler, R. van de Geijn, J. Watts, “Interprocessor Collective Communication Library (InterCom)”, Proc. of the Scalable High-Performance Computing Conference '94, Knoxville, TN, USA, IEEE Computer Society Press, pp. 357-364, May 1994.

[4] Chiola G., G. Ciaccio, “Lightweigth Messaging Systems”, in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 246-269, 1999.

[5] Ciaccio G., “Optimal Communication Performance on Fast Ethernet with GAMMA”, Proceedings Workshop PC-NOW, IPPS/SPDP'98, Orlando, FL, LNCS No. 1388, Springer, pp. 534-548, April 1998.

[6] Institute of Electrical and Electronics Engineers, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1984, 1984.

[7] Pacheco P., Parallel Programming with MPI, Morgan Kaufmann, San Francisco, California, 1997.

[8] Postel J., “User Datagram Protocol”, RFC 768, USC/Information Sciences Institute, Aug. 1980.

[9] Postel J., “Internet Protocol”, RFC 791, USC/Information Sciences Institute, Sep. 1981.

[10] Sun Microsystems, Inc. XDR: External Data Representation Standard. RFC 1014, Sun Microsystems, Inc., June 1987.

[11] Wilkinson B., Allen M., Parallel Programming: Techniques and Applications Using Networking Workstations, Prentice-Hall, Inc., 1999.

[12] GAMMA Home Page http://www.disi.unige.it/project/gamma/

Indice Alfabético