Cómputo Paralelo en Redes Locales de Computadoras

Departament d'Informàtica
Unitat d'Arquitectura d'Ordinadors
i Sistemes Operatius

Memoria presentada por Fernando G. Tinetti para optar al grado de Doctor por la Universitat Autònoma de Barcelona.

Barcelona, 9 de Diciembre de 2003

Cómputo Paralelo en Redes Locales de Computadoras

Memoria presentada por Fernando G. Tinetti para optar al grado de Doctor por la Universitat Autonòma de Barcelona. Este trabajo se ha realizado en el Departamento de Informática de la Universitat Autònoma de Barcelona bajo la dirección del Dr. Emilio Luque Fadón.

El Director de la Tesis

Emilio Luque Fadón

Barcelona, 9 de Diciembre de 2003

Agradecimientos

A mi director, Emilio Luque, por su soporte durante el tiempo de desarrollo de este trabajo.

A los laboratorios involucrados y sus integrantes, a quienes de alguna manera u otra he afectado con este trabajo (en orden cronológico):

- Unidad de Arquitectura de Ordenadores y Sistemas Operativos, Facultad de Ciencias, Universidad Autónoma de Barcelona, Barcelona, España.
- Centro de Técnicas Analógico-Digitales, Facultad de Ingeniería, Universidad Nacional de La Plata, La Plata, Argentina.
- Laboratorio de Investigación y Desarrollo en Informática, Facultad de Informática, Universidad Nacional de La Plata, La Plata, Argentina.
- Laboratorio de Química Teórica, Facultad de Ciencias Exactas, Universidad Nacional de La Plata, La Plata, Argentina.

A los profesores Alicia Jubert y Reinaldo Pis Diez, por su aporte desinteresado a este trabajo.

A los profesores Antonio Quijano y Armando De Giusti, por sus múltiples aportes, en múltiples áreas de este trabajo.

Tabla de Contenido

Capítulo 1: Introducción

En este capítulo se da el contexto bajo el cual se desarrolla todo el resto de esta tesis. Inicialmente, se hace una breve descripción de las aplicaciones numéricas y de la relación existente entre la multiplicación de matrices y las demás operaciones y métodos numéricos para las operaciones provenientes del álgebra lineal en general.

Desde el punto de vista del hardware de procesamiento, se describen las redes locales de computadoras como una alternativa más de cómputo paralelo y se detalla brevemente su relación con las demás plataformas de procesamiento paralelo que se utilizan actualmente. Avanzando sobre este punto, se identifican con mayor nivel de detalle los costos asociados con el cómputo paralelo en las redes de computadoras instaladas, comparándolos con los de las demás arquitecturas de cómputo paralelo.

Finalmente, se hace un resumen de los objetivos, aportes y método utilizado en el desarrollo de esta tesis así como se explica la organización del contenido.

1.1 Aplicaciones y Arquitecturas Paralelas

Las arquitecturas de procesamiento paralelo han sido extensiva e intensivamente utilizadas en numerosas aplicaciones, y el área de los problemas numéricos ha sido normalmente el punto de partida donde se ha estudiado y aprovechado la posibilidad de cómputo paralelo. Se han desarrollado numerosos métodos de solución para los problemas numéricos, la mayoría de ellos con una clara orientación hacia al menos dos sentidos [1] [59] [101] [102] [84]:

- Estabilidad numérica cuando se resuelven con aritmética que involucra algún tipo de error. La aritmética que se suele asumir es la que normalmente se denomina “de punto flotante”.
- Implementación directa o con costo mínimo en algún lenguaje de programación para ser resuelto en una computadora.

Desde el punto de vista de quien tiene un problema numérico a resolver, la utilización de una computadora no es nada más (y nada menos) que una forma de obtener lo que necesita. Más aún, si la computadora es paralela o no, o la forma en que una computadora obtiene los resultados correctos tampoco es de importancia excepto por el tiempo que debe esperar para utilizar el resultado. De hecho, el término *supercomputadora* (*supercomputer*) o la denominación *computadora de alto rendimiento* (*high-performance computer*) son utilizados desde hace mucho tiempo y refleja esta realidad: lo importante es la velocidad, si eso se logra con procesamiento paralelo entonces se lo utiliza [113].

El área de problemas provenientes del álgebra lineal tradicionalmente ha aprovechado el rendimiento que proporcionan las arquitecturas de cómputo (paralelo) disponibles. Quizás unos de los esfuerzos más significativos de los investigadores en esta área ha sido enfocado hacia el desarrollo de una biblioteca de rutinas que se consideran de vital importancia. Este esfuerzo ha dado como resultado la biblioteca denominada LAPACK (Linear Algebra PACKage) [7] [8] [LAPACK]. Más específicamente relacionado con el cómputo paralelo se ha desarrollado ScaLAPACK (Scalable Linear Algebra PACKage o simplemente Scalable LAPACK) [21] [27] [29] [ScaLAPACK].

Dentro de las aplicaciones del álgebra lineal se han identificado un conjunto de operaciones o directamente rutinas de cómputo que se consideran y a partir de las cuales, por ejemplo, se puede definir todo LAPACK. 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.

A partir de la definición misma de las rutinas del nivel 3 de BLAS y más específicamente a partir de [77], la multiplicación de matrices es considerada como el pilar o la rutina a partir de la cual todas las demás incluidas en este nivel de BLAS 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 y existen numerosas propuestas y publicaciones al respecto [20] [144] [57] [30] [142] [26]. 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. En términos de problema a resolver o de procesamiento a llevar a cabo en paralelo, esta tesis se orienta específicamente hacia la multiplicación de matrices con algunos comentarios hacia la generalización dada su importancia y representatividad en el área de álgebra lineal.

La Figura 1.1 resume esquemáticamente la relación de la multiplicación de matrices con los problemas numéricos en general. Dentro de los problemas numéricos que se resuelven computacionalmente existe un área relativamente bien definida que es la que corresponde al álgebra lineal. Como se ha explicado anteriormente, en esta área se ha definido una biblioteca estándar *de facto* que se ha denominado LAPACK. Un conjunto básico de rutinas de cómputo bien definidas puede ser utilizado para construir todo LAPACK: BLAS. Este conjunto básico de rutinas se ha dividido en tres niveles con el objetivo de identificar más claramente cuáles de ellas son las más apropiadas para la obtención del mejor rendimiento posible en una arquitectura de cómputo dada. Las rutinas del nivel 3 de BLAS o L3 BLAS son las que se deben analizar e implementar con mayor énfasis en la optimización y todo este nivel de BLAS puede ser definido en función de la multiplicación de matrices.

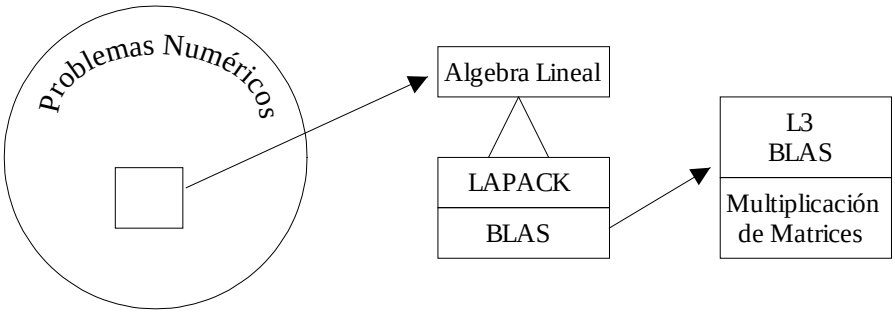


Figura 1.1: La Multiplicación de Matrices dentro de los Problemas Numéricos.

Las computadoras secuenciales tienen varios límites respecto de la capacidad máxima de procesamiento por lo que se recurre al procesamiento paralelo en general [2]. En realidad, las dos formas clásicas de obtener mayor rendimiento de cómputo se han explotado simultáneamente y de forma complementaria:

- Aumento de la velocidad de cómputo secuencial de un procesador
- Aumento de las unidades funcionales y/o procesadores que se pueden utilizar de forma

simultánea.
De hecho, se puede afirmar que la principal razón de ser del procesamiento paralelo es el rendimiento.

El área de cómputo paralelo ha evolucionado en muchos sentidos, desde el hardware de cómputo hasta los algoritmos y bibliotecas especializados en determinadas tareas. Desde el punto de vista del hardware de procesamiento ha habido una gran variación en cuanto a los objetivos de diseño y costos asociados. De hecho, la gran mayoría de las empresas creadoras de diseños y máquinas paralelas consideradas como muy destacables en su época han dejado de existir o han sido absorbidas por otras corporaciones. Solamente a modo de ejemplo se pueden mencionar empresas como Cray o Thinking Machines (creadora de la Connection Machine). La que puede considerarse como una "gran lección" al respecto es: los costos asociados a una computadora (paralela) específica no siempre se pueden afrontar y existen alternativas menos costosas con resultados similares o por lo menos apropiados para los problemas que se necesitan resolver. Los costos de computadoras (paralelas) específicas están relacionados con [113]:

- Diseño. La computadora paralela se desarrolla en términos de hardware casi desde cero. Por lo tanto se debe involucrar a especialistas con una alta capacidad y experiencia.
- Tecnología de construcción. Tanto en términos de materiales como de mecanismos de fabricación, los costos son mucho más altos que los involucrados en cualquier computadora de fabricación y venta masiva. La diferencia se cuantifica en varios órdenes de magnitud.
- Instalación. Las instalaciones tanto del lugar físico como del ensamblado de la computadora misma involucra muchas características específicas que tienen un muy alto costo. Esto involucra desde el personal que realiza la instalación hasta las condiciones de temperatura, por ejemplo.
- Mantenimiento del hardware. Siempre ha sido y es un porcentaje del costo total de una máquina y por lo tanto está en el mismo o similar orden de magnitud.
- Software. Tanto el software de base como el de desarrollo y ejecución de programas que es tanto o más importante, son específicos. En este caso, la especificidad del hardware se combina con la propia complejidad de un sistema operativo o de un ambiente de desarrollo y depuración de aplicaciones paralelas. En este caso el costo no solamente involucra tiempo y dinero sino que además hace más probables los errores en el software desarrollado para estas computadoras.
- Operación. Tanto el personal de producción de software como el de monitoreo y sintonización del sistema debe ser específicamente capacitado.

Por otro lado, el hardware básico de procesamiento que se utiliza masivamente en las computadoras de escritorio ha incrementado su capacidad también en órdenes de magnitud a la vez que ha reducido sus costos a los usuarios finales. Tanto el alto costo de las computadoras paralelas como la reducción de costo y el incremento de capacidad del hardware de procesamiento que se puede denominar *estándar* y de uso masivo ha llevado a que las computadoras paralelas actuales tengan una fuerte tendencia a incorporar este hardware estándar como básico. Los beneficios tienen una fuerte relación con la reducción de todos los costos mencionados y además se ha probado que es factible.

A medida que la cantidad de computadoras instaladas en una misma oficina y también a nivel de las instituciones, se ha incrementado, también se incrementaron las propuestas y el

estudio de los problemas y soluciones en lo que se ha denominado redes locales. De manera simultánea y desde distintos puntos de vista, la existencia de una cantidad relativamente grande de computadoras interconectadas en red ha dado lugar a los sistemas operativos distribuidos y a la utilización masiva de las redes locales como formas útiles de resolver problemas en ambientes acotados respecto de usuarios y aplicaciones. Sin embargo, a medida que estas redes locales se han aumentado pronto se identificaron varias alternativas de procesamiento que son posibles y de muy bajo costo:

- Utilización de los períodos de inactividad de las máquinas interconectadas en redes locales [90].
- Utilización de más de una computadora interconectada en red para resolver un problema, haciendo uso de los conceptos de cómputo paralelo [9].

A partir de la interconexión masiva de las redes locales a Internet también se han introducido las ideas de “Internet Computing” y actualmente se está impulsando con fuerza la idea de “Grid Computing” como una forma más amplia de compartir recursos en general [54] [75] [53] [55] o “Metacomputing” [24] [13].

La idea de cómputo paralelo en las redes locales de alguna manera es posible a partir de la interconexión misma de las computadoras. Sin embargo, es evidente que se deben resolver varios problemas para que esta forma de cómputo paralelo se utilice de manera confiable y con rendimiento aceptable. Si bien es bastante difícil definir el término *aceptable*, se tienen dos ideas subyacentes al respecto que son muy importantes:

- Aprovechamiento máximo de los recursos disponibles, especialmente de los recursos de cómputo (procesadores).
- Tiempo de procesamiento necesario para resolver un problema.

Puede ser posible que el tiempo de procesamiento para resolver un problema sea aceptable sin que se utilicen al máximo los recursos disponibles, pero dado que usualmente las computadoras interconectadas en una red local son las de menor rendimiento del mercado, la probabilidad de que esto suceda es bastante baja en general.

Una alternativa de cómputo paralelo con computadoras estándares de escritorio que ha probado ser muy satisfactoria en algunas áreas es la que se asocia con las instalaciones o con el proyecto Beowulf [19] [103] [99] [143] [111] [BEOWULF]. Aunque casi cualquier red local usada para cómputo paralelo se puede considerar como una instalación Beowulf [37], existe cierto consenso con respecto a que:

- Las computadoras de una instalación Beowulf son PCs homogéneas, con a lo sumo una computadora dedicada a la administración del sistema completo. De hecho, estas instalaciones son creadas para cómputo paralelo y en principio no tiene sentido instalar múltiples tipos de computadoras (heterogéneas).
- La red de interconexión básica es Ethernet [73] de 100 Mb/s y el cableado debería incluir la utilización de *switches* de interconexión que son capaces de aislar las comunicaciones entre pares de computadoras. De todas maneras, mejores redes de interconexión nunca son descartadas aunque siempre se tiende al menor costo.

Con el objetivo de estudiar y distinguir diferentes características se han propuesto múltiples clasificaciones de las arquitecturas de procesamiento [50] [51] [33]. Desde el punto de vista de capacidad y costo, las plataformas de cómputo paralelo actuales se pueden organizar de mayor a menor en una pirámide tal como lo muestra la Figura 1.2.

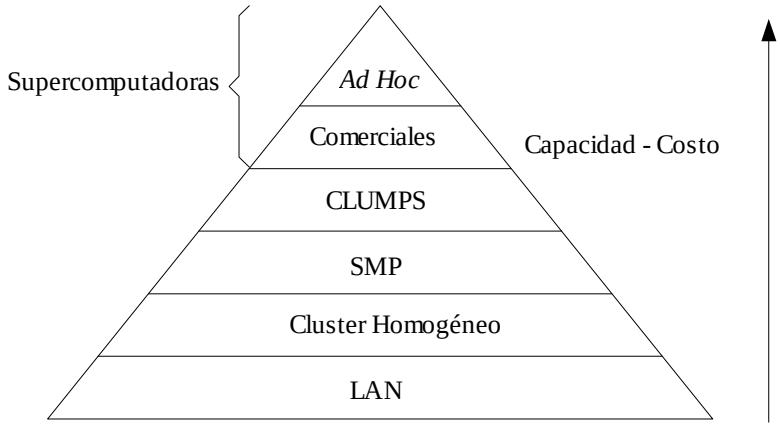


Figura 1.2 Una Clasificación de Plataformas de Cómputo Paralelo.

Ad Hoc. Computadoras paralelas planificadas/construidas específicamente bajo programas o aplicaciones definidas con anticipación. Dos de los más conocidos sobre el final de la década 1990-2000 son los denominados Accelerated Strategic Computing Initiative (ASCI) del Departamento de Energía de Estados Unidos de América [61] y EA (Earth Simulator) de varios departamentos y/o centros de investigación de Japón [EA] [139] [140]. Están entre las de mayor potencia de cálculo absoluta, tal como se reporta en [86] [TOP500]. Son las clásicas computadoras paralelas construidas *ad hoc* y se les asignan nombres como [TOP500]: Earth Simulator, ASCI Q, ASCI White, ASCI Red, ASCI Blue-Pacific, ASCI Blue Mountain. Llegan a varios Tflop/s (*Tera*flop/s, o 10^{12} operaciones entre números de punto flotante por segundo) y son proporcionalmente costosas (en realidad, tan costosas como construidas a medida). De hecho, pueden considerarse como las *únicas* computadoras paralelas construidas a medida de una aplicación o conjunto de aplicaciones en la actualidad.

Comerciales. Son casi específicamente dedicadas a cómputo científico en general y la mayoría de los centros con grandes requerimientos de cálculo tienen alguna/s. Suelen construirse con hardware básico que es promocionado como "escalable" dado que se pueden agregar procesadores, memoria, discos, etc. según las necesidades y siempre dentro de un rango definido. Pueden considerarse como ejemplos de esta clase: IBM SP, Compaq AlphaServer, Hitachi SR, SGI Origin, Cray T3E y las computadoras con procesadores vectoriales (Fujitsu).

CLUMPs: Clusters de SMP (Symmetric MultiProcessing) [12] o SMPs conectadas en red, no parece estar muy explorado en general y en/para cómputo paralelo en particular porque es bastante nuevo, al menos no hay muchas publicaciones al respecto. Desde el punto de vista de la paralelización es muy importante la combinación de modelos:

- De memoria compartida en un SMP que puede ser considerado como MIMD fuertemente acoplado.
- De pasaje de mensajes o al menos MIMD de memoria distribuida dada por las utilización de más de una máquina SMP interconectada por una red.

SMP: Symmetric MultiProcessing, suelen estar directamente orientadas a almacenamiento y procesamiento de grandes volúmenes de datos en disco y/o "servidores web". No hay

muchos reportes de su utilización en aplicaciones científicas y sí hay una gran cantidad de publicaciones de su utilización/utilidad en el campo de recuperación de información almacenada en disco. Muchas empresas directamente las promocionan como “servidores”. Su utilidad en las aplicaciones paralelas en general y particularmente en aplicaciones científicas es inmediata. Aunque tienen límites en cuanto a escalabilidad y más precisamente en cuanto a la cantidad máxima de procesadores que pueden tener, ponen a disposición del usuario una cantidad relativamente grande de procesamiento sobre una única memoria compartida, lo cual hace inmediata la utilización de los algoritmos paralelos orientados a los multiprocesadores.

Clusters Homogéneos: Conjunto de computadoras dedicadas a cómputo paralelo. Estaciones de trabajo “clásicas” (Sun, SGI), o PCs homogéneas. Desde el punto de vista técnico pueden considerarse como las instalaciones Beowulf que nacieron como un conjunto de PCs conectadas en una red local (normalmente Fast Ethenet hasta ahora). Desde el inicio mismo se hace una gran inversión en la red de comunicaciones (básicamente a través de switches en el cableado de la red Ethernet. Se han desarrollado múltiples herramientas tanto para administración de todo el sistema paralelo como para la producción de software paralelo. Los demás clusters homogéneos (con estaciones de trabajo “clásicas” no tienen muchas diferencias con las instalaciones Beowulf, pero en [99] [111] se las excluye casi sin discusión por no tener Linux como sistema operativo en cada nodo básico de cómputo. El costo de hardware es bastante mayor que el de de los sistemas Beowulf por la relación de costos entre una PC y una estación de trabajo Sun o SGI por ejemplo. En términos de rendimiento y confiabilidad la diferencia no es tan clara.

Por la relación de costo de hardware existente entre una PC y una estación de trabajo “clásica” es muy difícil establecer cuál configuración es más potente. En general se cumple que a igual costo, la capacidad de cálculo (al menos la suma de capacidad de los procesadores) es mayor en los sistemas Beowulf. Sin embargo, la diferencia es aún más difícil de cuantificar y establecer *a priori* cuando se tiene en cuenta el costo de las herramientas, confiabilidad, etc. de cada uno de los sistemas.

LAN: redes locales de computadoras instaladas y que se pueden aprovechar para cómputo paralelo. Cada computadora tiene uno o un conjunto de usuarios que la deja/n disponible para cómputo paralelo por determinados períodos de tiempo. Otra de las alternativas en este contexto es la de ceder un porcentaje o fracción de cómputo de cada máquina. Es muy difícil diferenciar los términos usados en este contexto tales como “clusters”, “NOW” (Networks of Workstations), “COW” (Cluster of Workstations) y “Workstation Clusters”. En [12], por ejemplo, se afirma directamente que son sinónimos y se diferencia las distintas posibilidades según las características de procesamiento, objetivos y hardware de base entre otros índices. De hecho, a lo largo de toda esta tesis se hará referencia a esta clase de plataforma de cómputo paralelo como “redes locales” y también como “clusters”, que aparentemente es una de las denominaciones más utilizadas en la bibliografía. Sin embargo, es importante notar que las redes locales que no se han construido con el objetivo específico de cómputo paralelo como en los casos anteriores de los clusters homogéneos y tienen características propias de costo y de procesamiento que no tienen las demás. Por lo tanto, es muy importante identificar las formas de paralelizar aplicaciones o las características propias de las aplicaciones paralelas para utilizar al máximo y/o de manera optimizada la capacidad de las redes locales instaladas y que se pueden aprovechar para

cómputo paralelo.

Las definiciones y el alcance de cada término o expresión no está aún muy bien especificado en general. Solamente como ejemplo, se puede notar que en [48] [49] toda red de computadoras interconectadas en red se denomina *cluster* y se dividen en dos clases generales:

- NOW (Network of Workstations): cada computadora tiene uno o más usuarios que permiten la utilización de la computadora en los períodos de inactividad.
- PMMPP ("Poor Man's" MPP): cluster dedicado a ejecución de aplicaciones paralelas con requerimientos de alto rendimiento.

Sin embargo, en [12] por ejemplo, se mantiene que todas las computadoras en red que se usan para cómputo paralelo son clusters y se los clasifica según

- Objetivo (alto rendimiento o disponibilidad).
- Relación de pertenencia de los usuarios (cada computadora se dedica exclusivamente a cómputo paralelo o no).
- Hardware de cada nodo o computadora (PC, SMP, etc.).
- Sistema operativo de cada nodo o computadora.
- Configuración de cada computadora (Homogéneo o Heterogéneo).
- Niveles de "clustering".

Por lo tanto, de alguna manera se combinan términos de hardware, software y configuración de la red completa para cada caracterización. Más allá de que se utilizarán los términos "redes locales" y "clusters" de aquí en más, las redes locales instaladas y que se aprovechan para cómputo paralelo en esta tesis se consideran como:

- Clusters, dado que son un conjunto de computadoras interconectadas en red que se utilizan para cómputo paralelo.
- NOW, en el sentido de que cada computadora tiene un usuario o conjunto de usuarios que son quienes la ceden para cómputo paralelo.
- PMMPP, dado que por el tiempo que se dispone una computadora se tiene de manera completa. Aún en el caso de tener una fracción de cada computadora, esa fracción se tiene siempre disponible.
- Heterogénea, dado que es muy difícil que en las redes locales instaladas todas las computadoras tengan el mismo hardware y el mismo software de base. Esto se debe principalmente a
    - Objetivo de cada computadora o aplicaciones usuales que resuelve. La existencia de cada máquina se debe a un usuario o a un conjunto de aplicaciones que deben ser resueltas de manera más o menos periódica. Ni los usuarios ni las aplicaciones necesariamente tienen los mismos requerimientos de cómputo, almacenamiento, etc. Por lo tanto, en general la existencia de cada computadora no tiene relación o tiene muy poca relación con las demás (al menos en cuanto a características físicas).
    - Tiempo de instalación de la red. El ingreso de nuevas máquinas a la red local así como la reparación y actualización de las máquinas existentes, que se puede denominar evolución de la red hace muy difícil mantener la homogeneidad. La evolución de las redes locales es innegable, y la relación entre la evolución y la heterogeneidad tampoco. En general, a mayor tiempo de instalación de una red local mayor será la heterogeneidad. Solamente como ejemplo, se puede mencionar que la disponibilidad de un tipo de computadora es bastante limitada.

> En el caso de las PCs, este tiempo de disponibilidad suele contabilizarse en meses y a lo sumo quizás en algo más de dos años.

Es muy interesante estimar la evolución de los clusters homogéneos que actualmente cuentan con un período relativamente corto de instalados y en producción. Aunque con aplicaciones y usuarios controlados y bien administrados, la probabilidad de fallas en el hardware sigue presente. Y a mayor cantidad de computadoras y hardware en general, mayor será la probabilidad de que algo falle. Por lo tanto, la reparación y/o reposición no es algo *fuera de lo común* ni lo será en los clusters homogéneos. Cuando no sea posible mantener homogéneo un cluster por la disponibilidad de hardware a reparar/reponer se tienen, en general, dos alternativas (descartando la fabricación *ad hoc*, cuyo costo es sumamente alto):

- Mantener la homogeneidad y no reparar ni reponer. Es válida siempre y cuando las aplicaciones se sigan resolviendo. Teniendo en cuenta que en general las aplicaciones tienden a aumentar sus requerimientos esta alternativa parece poco útil en general.
- Reparar y/o reponer con hardware disponible. Automáticamente el cluster se transformará en heterogéneo. La heterogeneidad se traduce directamente en diferentes valores para las métricas de rendimiento cuando se trata de procesadores y/o memoria en el caso de las tareas de cómputo intensivo.

Otra de las razones por las cuales un cluster no necesariamente se puede mantener homogéneo es la necesidad de resolver aplicaciones con mayores requerimientos de cómputo y/o almacenamiento. En este sentido, a medida que transcurre el tiempo, la disponibilidad del hardware es menor y por lo tanto, si se decide mantener la homogeneidad o resolver las aplicaciones con hardware homogéneo se debería instalar un *nuevo* cluster. Esta decisión tiene el costo inmediato de la adquisición e instalación de *todo* un cluster, además de la necesidad de tomar alguna decisión respecto del cluster homogéneo que ya no tiene la capacidad suficiente para resolver la/s aplicación/es. La alternativa que parece ser más lógica, es decir la instalación del hardware necesario para *aumentar* la capacidad total del cluster, en general implica heterogeneidad.

Desde la perspectiva de que es muy difícil el mantenimiento homogéneo aún de los clusters dedicados a cómputo paralelo, todo aporte que se haga en el contexto de las redes de computadoras heterogéneas tiene un amplio espectro de utilización. De hecho, cada vez que, por la razón que sea, un cluster se “transforma” en heterogéneo las aplicaciones y los usuarios se tendrán que adaptar de una manera u otra al hardware para obtener rendimiento óptimo (que, como se afirma antes, es la principal razón para la existencia misma del cómputo paralelo).

1.2 Costos de Cómputo Paralelo en las Redes Locales Instaladas

Tal como se ha adelantado, las redes locales instaladas son la plataforma de cómputo paralelo de “costo cero” en cuanto a hardware. De hecho, los costos de

- Instalación

- Administración y monitoreo
- Mantenimiento

de cada computadora no tienen relación con el cómputo paralelo y por lo tanto no debería ser asumido por quienes las aprovechan para cómputo paralelo, dado que

- Las redes locales ya están instaladas y funcionando, y entre los objetivos no ha estado el de cómputo paralelo (al menos en la *gran* mayoría de las redes locales instaladas). El hecho de aprovecharlas para cómputo paralelo no cambia esta realidad, aunque de alguna manera se agrega como una de las utilizaciones de las redes locales.
- Cada computadora y la red local misma tiene al menos un administrador que se encarga de la configuración, interconexión y una mínima verificación de funcionamiento.
- Cada computadora tiene al menos un usuario o un conjunto de usuarios que se encarga de una manera u otra del mantenimiento dado que la utilizan. También es común que estos usuarios se encarguen de gestionar y/o llevar a cabo la actualización del hardware, con lo que se tendría costo cero también de actualización del hardware.

Pero el costo de hardware no es todo el costo relacionado con el procesamiento de las redes locales instaladas que se pueden aprovechar para cómputo paralelo. Los costos que se deben asumir en este contexto son los relacionados con:

- Las herramientas de administración de la red local para cómputo paralelo y para desarrollo y ejecución de programas paralelos.
- La disponibilidad de las computadoras de la red local y de la misma red local.
- La paralelización de aplicaciones.

Las herramientas de desarrollo y ejecución de programas paralelos son imprescindibles para aprovechar las redes locales para cómputo paralelo. Lo primero que ha sido desarrollado en este sentido son bibliotecas de pasaje de mensajes tales como PVM (Parallel Virtual Machine) [44] que se estableció como un estándar *de facto* en esta área. Posteriormente se propuso el estándar MPI (Message Passing Interface) [88] [107] [92] y actualmente se utilizan ambos, con la tendencia de desarrollo de las nuevas aplicaciones paralelas en las implementaciones de MPI disponibles para la arquitectura paralela que se utiliza. Específicamente para las redes de computadoras ambas bibliotecas están implementadas y su utilización es libre [PVM] [LAM/MPI] [MPICH] y se obtienen vía Internet. En este sentido, el único costo de estas herramientas es el de instalación. Aunque de manera no tan centralizada o estandarizada como PVM y MPI, también se han desarrollado múltiples herramientas de administración de redes de computadoras para cómputo paralelo y también están disponibles en Internet. Las instalaciones Beowulf son una de las principales fuentes de herramientas para administración de redes que se utilizan para cómputo paralelo y que se pueden aprovechar. Sin embargo, existe cierto consenso en que el costo de administración de una red de computadoras que se utiliza para cómputo paralelo es bastante mayor que en el caso de las computadoras paralelas clásicas [17].

La disponibilidad de las computadoras de una red local instalada para cómputo paralelo no es completa. Si bien este costo es muy difícil de cuantificar, la utilización de las redes locales tiene varias alternativas, dos de las cuales son:

- Períodos de casi total inactividad tales como las noches o los días no laborables. Si bien es dependiente de características específicas en cada red local, incluso en horarios considerados como de mucha utilización, las computadoras no necesariamente se aprovechan al máximo [90].

- Determinar *a priori* un porcentaje de cada computadora para ser utilizado por procesos de aplicaciones paralelas. En este caso es como tener la misma cantidad de computadoras pero cada una de ellas con menor capacidad.

Desde el punto de vista de las aplicaciones paralelas, ambas alternativas son similares: se tiene un conjunto de recursos disponibles. Este es el contexto de "disponibilidad" de las computadoras que se utilizarán. Más específicamente, la experimentación se llevará a cabo durante los períodos en los cuales las redes locales no se utilizan para ninguna otra cosa y por lo tanto la disponibilidad será total (durante la ejecución de los programas paralelos).

Quizás el costo más considerable o al menos más desconocido es el de la paralelización y/o desarrollo de programas paralelos para las redes locales instaladas y cuyos recursos se pueden aprovechar. El problema mayor en este contexto es el de la paralelización misma. Siempre ha sido considerado uno de los grandes problemas (y con su costo asociado), dado que no hay métodos generales. Una de las grandes razones para que esto suceda es justamente la razón de ser del cómputo paralelo: el rendimiento. Si bien se pueden diseñar algoritmos paralelos sintáctica, semántica y estilísticamente muy buenos, no son útiles cuando no obtienen rendimiento aceptable. En general, es muy difícil cuantificar el término *aceptable*, pero se pueden mencionar dos posibles acepciones:

- Utilizan al máximo los recursos disponibles. En general se acentúa la importancia en términos de utilización de los procesadores disponibles.
- Tiempo de respuesta mínimo. En este caso es dependiente de la aplicación. En el caso de paralelizar la tarea de predicción de las condiciones meteorológicas (estado del tiempo) para un día determinado, es claramente inadecuado obtener la respuesta de predicción en uno o varios días posteriores.

Varias métricas de rendimiento de sistemas paralelos se concentran en la utilización de los recursos disponibles dado que:

- Es independiente de las aplicaciones y en este sentido las métricas son generales.
- Si se utilizan al máximo los recursos disponibles se asume que no se puede obtener nada mejor, como mínimo en la máquina paralela con el algoritmo paralelo que se utiliza.

Sin embargo, como se ha afirmado antes, muchos de los problemas de álgebra lineal han sido resueltos satisfactoriamente con máquinas paralelas. Una de las primeras tareas que se deben llevar a cabo entonces es la revisión de los algoritmos paralelos ya desarrollados para aprovecharlos en el contexto de las redes de computadoras. En este punto no hay que perder de vista que las redes de computadoras y más específicamente cada computadora que se puede conectar en una red (con una interfase de red) no ha sido ni en principio es diseñada para cómputo paralelo distribuido en múltiples máquinas. Por lo tanto, se tiene que

- Como mínimo se deben analizar los algoritmos paralelos propuestos y su efectividad en cuanto a rendimiento en las redes de computadoras.
- Si no hay ningún algoritmo paralelo que se pueda considerar apropiado para cómputo paralelo asociado a un problema en una red se debería proponer al menos otro para considerar la efectividad de este tipo de arquitecturas paralelas.

Si bien el análisis de los algoritmos paralelos propuestos hasta el momento se debe hacer de manera exhaustiva y quizás "caso por caso" (o al menos por área de aplicaciones o características de procesamiento) se tiene un inconveniente desde el principio: las máquinas paralelas han sido y son diseñadas para cómputo paralelo y las redes de computadoras no. Es esperable, por lo tanto, que los algoritmos paralelos no sean

directamente utilizables en las redes locales que han sido instaladas y son utilizadas con múltiples propósitos y el de cómputo paralelo no es uno de ellos, o no es uno de los más importantes.

La paralelización de aplicaciones para este tipo de arquitectura paralela tiene varios inconvenientes o al menos varias características que no se conocen en las máquinas paralelas tradicionales. Los dos inconvenientes que pueden considerarse como más importantes son:

- Heterogeneidad de los nodos de cómputo.
- Características de la red de interconexión.

Tanto las máquinas paralelas tradicionales como los clusters homogéneos tal como se los ha caracterizado antes (construidos o instalados para cómputo paralelo) tienen elementos de procesamiento (procesadores) homogéneos. Esto simplifica de manera notable la distribución de la carga de cómputo, dado que para lograr balance de carga de procesamiento *simplemente* es necesario distribuir la misma cantidad de operaciones que cada elemento de cómputo debe ejecutar. En general, la definición del término *simplemente* que se ha utilizado no es trivial, pero en el contexto de las aplicaciones de álgebra lineal normalmente implica la distribución de la misma cantidad de datos (o porciones iguales de datos de matrices) entre los procesadores. Por lo tanto, la paralelización de las aplicaciones provenientes del área de álgebra lineal no ha tenido muchos inconvenientes asociados al balance de carga en las computadoras paralelas homogéneas. En las redes locales que se aprovechan para cómputo paralelo evidentemente también se tendrá que resolver el problema causado por las diferencias entre las capacidades de cómputo de las distintas máquinas utilizadas.

La red de interconexión más extendida en cuanto a redes locales de computadoras instaladas es sin lugar a dudas la definida por el estándar Ethernet [73] de 10 Mb/s y de 100 Mb/s. De hecho, se reconoce que la red Ethernet de 10 Mb/s no es adecuada en general para cómputo paralelo [91]. Más allá de las características de rendimiento máximo, las redes Ethernet tienen varios inconvenientes por su propia definición y dependencia del protocolo CSMA/CD (Carrier Sense-Multiple Access/Collision Detect), que implica rendimiento en general muy dependiente del tráfico individual de cada una de las máquinas interconectadas. Sin embargo, la enorme potencia de cálculo instalada en las redes locales hace valioso cualquier aporte al respecto. Las aplicaciones provenientes del álgebra lineal, por otro lado, tiene una cantidad relativamente grande de usuarios potenciales y de redes locales *disponibles* para cómputo paralelo. Desde el punto de vista del costo, las redes Ethernet no solamente son las más difundidas y por lo tanto *aprovechables* en cuanto a instalaciones actuales, sino que además tiene una gran *inercia* en cuanto a instalaciones nuevas dado que

- Existe una gran cantidad de personal técnico capacitado y con experiencia que puede seguir instalando estas redes. Cambiar de tecnología en general implica mayor costo por lo menos de capacitación de personal técnico.
- Existe una gran cantidad de software desarrollado y que en general tiende a ser estable y utilizado.
- La norma Ethernet se ha definido con mayores capacidades de transferencia de datos, tales como 1 Gb/s (10^6 bits por segundo) [76] y 10 Gb/s (10^7 bits por segundo) [110].

Desde otro punto de vista, hay otros costos asociados que no son tan relacionados con los estrictamente técnicos como los que se han mencionado. Tal como se puntualiza en [18] (en el contexto de proporcionar *high throughput*), la utilización de redes locales es un problema tecnológico y sociológico. En este sentido, existe de un "evangelista" (tal como se lo denomina en [18]) que desarrolla y crea con sus propios recursos y con los de "aliados" un *high throughput computing cluster* y con él "genera demanda" hacia un conjunto más amplio de usuarios potenciales que a su vez aportarán sus propios recursos individuales. Pero además, toda red local o conjunto de redes locales debe tener apoyo explícito de una organización para posibilitar su aprovechamiento para cómputo paralelo. Sin embargo, en esta tesis este/estos costos no se abordarán.

1.3 Resumen de Objetivos y Aportes de la Tesis y Organización del Contenido

El principal objetivo de esta tesis es la evaluación de los problemas y soluciones para el cómputo paralelo en las redes de computadoras instaladas, con sus características de cómputo y comunicaciones. El problema a través del cual se realiza la evaluación es el de la multiplicación de matrices por varias razones:

- Representatividad del tipo de procesamiento con respecto al resto de las operaciones (y aplicaciones) provenientes del álgebra lineal.
- Representatividad en cuanto a requerimientos de cómputo y de almacenamiento, específicamente con respecto a las rutinas incluidas en BLAS de nivel 3.
- Cantidad de publicaciones identificando tanto algoritmos paralelos propuestos como rendimiento obtenido.

Toda la tesis está orientada a la paralelización de aplicaciones para la obtención del máximo rendimiento posible, por lo tanto aunque inicialmente se hará uso de herramientas estándares y ampliamente utilizadas en este contexto como PVM la tendencia será mejorar y/o reemplazar todo lo que imponga una excesiva penalización en el rendimiento. Esto abarca desde la forma de llevar a cabo cómputo local hasta la monitorización y evaluación de las rutinas de comunicaciones que se utilizan.

El método con el que se hará la evaluación tiene, en principio dos partes:

- Análisis de algoritmos, métodos y herramientas ya propuestos para llevar a cabo procesamiento paralelo. Si este análisis implica *a priori* una clara penalización en cuanto a rendimiento, se propondrá al menos una alternativa considerada apropiada.
- Experimentación exhaustiva. Se definirán un conjunto de experimentos a llevar a cabo y se analizará la validez o no del rendimiento obtenido.
  En el caso de que el rendimiento obtenido no sea satisfactorio, se identificará la fuente de la penalización de rendimiento y se propondrá al menos una alternativa de solución (en términos de algoritmos o implementaciones de rutinas específicas) con la cual se realizarán nuevamente los experimentos propuestos.

Los aportes se pueden resumir en:

- Identificación del nivel de aprovechamiento de los algoritmos paralelos propuestos

específicamente para la multiplicación de matrices en el contexto de cómputo paralelo en redes de computadoras instaladas.
- Identificación de las características de procesamiento paralelo con las cuales se debe resolver específicamente la multiplicación de matrices.
- Identificación de las características de procesamiento paralelo con las cuales se deben resolver las aplicaciones provenientes del álgebra lineal en general.
- Identificación y propuestas de solución para los posibles problemas de rendimiento ocasionados tanto por el rendimiento local de cada computadora como de las rutinas de comunicaciones que se utilizan.

La utilización de estos aportes tiende a:
- Aprovechar la capacidad de cómputo disponible en las redes locales instaladas, aunque no tengan como objetivo el procesamiento paralelo.
- Aprovechar las actuales instalaciones de clusters homogéneos que se utilizan para procesamiento paralelo cuando, por la evolución en en hardware y/o en las aplicaciones, sean heterogéneos.

Más específicamente, a lo largo de esta tesis se tendrá:
1. Del análisis de los algoritmos paralelos de multiplicación de matrices surgirá claramente que no son directamente utilizables en la plataforma de cómputo paralelo que representan las redes locales de computadoras.
2. Aún cuando se proponga un algoritmo específico de multiplicación de matrices para aprovechar al máximo las características de la arquitectura de cómputo paralelo que proveen las redes locales, el rendimiento optimizado no está asegurado. Se mostrará por experimentación y monitorización (vía instrumentación) del rendimiento del algoritmo que la biblioteca de pasaje de mensajes PVM impone penalización de rendimiento inaceptable.
3. Se analiza y explica brevemente las razones por las cuales las bibliotecas de pasaje de mensajes de propósito general (entre las cuales se incluyen la propia biblioteca PVM y las implementaciones de MPI, por ejemplo) no pueden asegurar rendimiento optimizado en las redes locales de computadoras utilizadas para cómputo paralelo.
4. Se propone e implementa una única operación de pasaje de mensajes directamente orientada al aprovechamiento de las características de las redes Ethernet utilizadas para la de interconexión de computadoras en las redes locales.
5. Se evalúa, vía experimentación, el rendimiento de los algoritmos propuestos incluyendo la operación de pasaje de mensajes también propuesta, mostrando que el rendimiento puede ser considerado como aceptable u optimizado para las redes locales de computadoras utilizadas para cómputo paralelo.
6. Se muestra también cómo uno de los algoritmos propuestos para multiplicación de matrices puede ser utilizado para evaluar la capacidad de las máquinas de una red local de computadoras de llevar a cabo comunicaciones en *background* (*solapadamente*) mientras realiza cómputo local.
7. Finalmente, también se compara la propuesta algorítmica de esta tesis con los algoritmos específicos de la biblioteca ScaLAPACK, que es considerada como la que implementa los mejores algoritmos (en cuanto a rendimiento y a escalabilidad) de cómputo paralelo en arquitecturas paralelas de memoria distribuida. En este caso específico de comparación solamente se tienen en cuenta redes de computadoras homogéneas, dado que ScaLAPACK no tiene ninguna previsión para el caso de las

arquitecturas con elementos de cómputo heterogéneos. Esta comparación también da resultados favorables para la propuesta de esta tesis, dado que se tienen mejores resultados de rendimiento y escalabilidad, al menos hasta donde fue posible evaluar en cuanto a cantidad de máquinas utilizadas para cómputo paralelo.

El siguiente capítulo, **Capítulo 2: Multiplicación de Matrices**, se dedicará al análisis de la multiplicación de matrices en general, en el contexto de las operaciones de álgebra lineal y también se analizarán los algoritmos paralelos que se han propuesto.

En el **Capítulo 3: Clusters Heterogéneos** se analiza en detalle las características de las redes locales instaladas desde el punto de vista de cómputo paralelo. De acuerdo con este análisis se identifican las características principales de cómputo paralelo en esta plataforma de procesamiento y también se proponen dos algoritmos específicos para multiplicar matrices en paralelo.

El **Capítulo 4: Experimentación**, detalla los experimentos realizados y analiza los resultados obtenidos con la biblioteca de pasaje de mensajes PVM. Muestra con detalle los problemas de rendimiento que genera esta biblioteca en particular y también se comentan en general las expectativas con respecto a las demás bibliotecas de pasaje de mensajes para las redes de computadoras. Se propone una rutina de comunicaciones propia que, al rehacer los experimentos muestra cómo se logra utilizar al máximo o de manera optimizada tanto los recursos disponibles de cómputo como los de comunicaciones y esta combinación proporciona rendimiento satisfactorio.

En el **Capítulo 5: Comparación con ScaLAPACK** 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.

En el **Capítulo 6: Conclusiones y Trabajo Futuro**, se resumen las principales conclusiones a partir de la tarea de análisis y experimentación realizada. También se adelantan de manera estimada las principales líneas de investigación que siguen a partir del trabajo realizado en esta tesis.

Posteriormente se dan los detalles de la **Bibliografía** a la cual se hace referencia a lo largo de esta tesis y se incluyen los **Apéndices**. En general, la idea de los apéndices es que sean autocontenidos y es por eso que tienen su propia lista de referencias bibliográficas.

El **Apéndice A: Características de las Redes Locales** muestra todo el detalle del hardware las computadoras y del cableado de las redes locales que se utilizaron para la experimentación.

El **Apéndice B: Rendimiento de Procesamiento Secuencial de las Computadoras**

muestra el impacto de los niveles de optimización de código de procesamiento en cada una de las computadoras, su rendimiento máximo (que se aprovecha en el procesamiento paralelo) así como algunos comentarios respecto del código que se utiliza y que se debería utilizar en general en la experimentación con y en la producción de programas paralelos.

El **Apéndice C: Comunicaciones en la Red Local del CeTAD** muestra el rendimiento punto a punto de las comunicaciones en una de las redes locales utilizadas y también de las comunicaciones con la rutina *broadcast* provista por la biblioteca PVM entre procesos de una aplicación paralela. También se hacen comentarios respecto del rendimiento de las comunicaciones en las demás redes locales utilizadas y con algunas referencias a otras bibliotecas de pasaje de mensajes disponibles para cómputo paralelo en redes de computadoras.

Capítulo 2: Multiplicación de Matrices

Casi desde el principio de la aplicación del procesamiento paralelo a los problemas numéricos se han estudiado, diseñado, implementado y experimentado distintas formas de paralelizar la multiplicación de matrices.

Desde el punto de vista del problema mismo, es muy útil tener una optimización de esta operación matricial, ya que siempre es posible encontrarla en las distintas aplicaciones que se deben resolver en el ámbito numérico, y por lo tanto, tener optimizada esta operación implica optimizar una parte de muchas aplicaciones en las cuales sea necesario multiplicar matrices.

Desde un punto de vista más cercano a la investigación, este problema tiene muchas características que lo hacen adecuado para su estudio extensivo e intensivo. Las dos más importantes son su simplicidad y la posibilidad de extender sus resultados a otras operaciones similares.

En este capítulo se incluirán algunos aspectos considerados importantes de esta operación para su paralelización, así como las características sobresalientes de los algoritmos de paralelización ya desarrollados.

2.1 Definición de la Multiplicación de Matrices

La definición de la operación de multiplicación de matrices es muy sencilla, lo que también simplifica su comprensión. Dada una matriz $A^{(m \times r)}$ de m filas y r columnas, donde cada uno de sus elementos se denota a_{ij} con $1 \leq i \leq m$, y $1 \leq j \leq r$; y una matriz $B^{(r \times n)}$ de r filas y n columnas, donde cada uno de sus elementos se denota b_{ij} con $1 \leq i \leq r$, y $1 \leq j \leq n$; la matriz C resultante de la operación de multiplicación de las matrices A y B, $C = A \times B$, es tal que cada uno de sus elementos que se denota como c_{ij} con $1 \leq i \leq m$, y $1 \leq j \leq n$, y se calcula de la siguiente manera

$$c_{ij} = \sum_{k=1}^{r} a_{ik} \times b_{kj} \tag{2.1}$$

Como la mayoría de las operaciones básicas de álgebra lineal, la cantidad de operaciones entre escalares, (o "flops" en [59], como contracción de "*floating point operations*") necesaria para el cálculo de la matriz resultante es conocida (calculable) de manera exacta. Tal como ha sido definida previamente la multiplicación, son necesarias exactamente

$$cant\_op = m \times n \times (2r\text{-}1) \tag{2.2}$$

operaciones (multiplicaciones y sumas entre escalares). Por razones de simplicidad, normalmente se hace todo el análisis en función de matrices cuadradas de orden n y de esta manera se llega a que la cantidad de operaciones básicas entre escalares es exactamente

$$cant\_op = 2n^3\text{-}n^2 \tag{2.3}$$

Esta cantidad de operaciones es la que usualmente se denomina como *complejidad* de la multiplicación de matrices, y es la que determina el tiempo de ejecución necesario para ser resuelto por una computadora. En este contexto, también lo usual es encontrar que la multiplicación de matrices es $O(n^3)$ ("de orden n^3"), enfatizando que el término dominante en la Ecuación (2.3) es de grado cúbico y dejando de lado las constantes de multiplicación y todos los términos de menor grado.

Es muy importante hacer notar que esta cantidad de operaciones es independiente del algoritmo y/o la computadora que se utilicen para resolver el problema. En este sentido, también es importante (aunque no necesariamente "esencial" en este caso) diferenciar esta cantidad de operaciones de, por ejemplo, la cantidad de operaciones realizadas durante la ejecución de un programa secuencial basado normalmente en la asignación

$$c_{ij} = c_{ij} + a_{ik} \times b_{kj}$$

que implica la ejecución de $2n^3$ operaciones aritméticas entre escalares. Este es un ejemplo inmediato de que los programas no necesariamente resuelven los problemas con la cantidad mínima de operaciones. Como se aclaró antes, este ejemplo no presenta demasiados problemas en cuanto al tiempo de ejecución total, pero sí es útil para mostrar que la

cantidad de operaciones resueltas por una computadora no necesariamente es la mínima. Siempre se debería tener en cuenta este hecho a la hora de evaluar el rendimiento de las computadoras para la resolución de un problema en particular.

En el caso de las computadoras paralelas se debe ser más cuidadoso aún, dado que muchas veces conviene replicar cómputo (y por lo tanto aumentar la cantidad de operaciones efectivamente resueltas por los procesadores) para evitar comunicación o sincronización que llevaría más tiempo de ejecución que el cómputo que se replica. En todo el análisis de rendimiento a realizarse en la experimentación, se utilizará la cantidad de operaciones de la Ecuación (2.3) como valor de referencia para evitar al máximo conclusiones equivocadas por la cantidad de operaciones que se ejecutan en el/los procesadores.

2.2 Operaciones de Algebra Lineal

Casi desde el comienzo mismo de la utilización de computadoras se ha buscado que el software desarrollado tenga la mayor calidad en términos de índices considerados claves, tales como: rendimiento, reusabilidad y portabilidad. En este sentido, el área encargada de resolver problemas numéricos en general y los problemas de álgebra lineal en particular no han sido una excepción.

En el contexto de las operaciones de álgebra lineal, desde hace mucho tiempo se han definido, propuesto y desarrollado varias bibliotecas destinadas al establecimiento de un conjunto lo más reducido y también lo más general posible de rutinas u operaciones básicas que se utilicen en la mayoría (sino en todas) las aplicaciones de álgebra lineal. Uno de los primeros ejemplos es EISPACK [46], basado en un conjunto de rutinas enumeradas en [145].

La biblioteca que se ha convertido en el estándar de facto en el área de álgebra lineal es LAPACK (Linear Algebra PACKage), desarrollada a finales de la década de 1980 [36] [7] [8]. Además de haber aprovechado la experiencia de bibliotecas anteriores como EISPACK y LINPACK, junto con LAPACK (o al menos como una evolución lógica relacionada con esta biblioteca) se agregan al menos dos conceptos fundamentales en cuanto a claridad de la especificación misma de la biblioteca y también en cuanto a la posibilidad de la máxima optimización local (de acuerdo a la arquitectura de cómputo). Estos dos conceptos son:

- Operaciones básicas en niveles.
- Algoritmos de bloque.

En realidad ambos conceptos están muy relacionados, pero la división de las operaciones básicas por niveles se hace desde el punto de vista mismo de LAPACK como biblioteca de resolución de problemas de álgebra lineal. Por otro lado, los algoritmos de bloque son una consecuencia de reconocer que la arquitectura de la mayoría de las computadoras (independientemente de que sea paralela o no) tienen una estructura de memoria jerárquica, donde los niveles más cercanos al procesador mismo (niveles 1 y 2 de cache) deberían ser explotados al máximo para obtener la máxima capacidad de procesamiento.

2.2.1 BLAS: Basic Linear Algebra Subprograms y Rendimiento

A partir de las subrutinas incluidas y definidas en LAPACK, un conjunto de subprogramas ha sido reconocido como básico y de hecho a estas subrutinas se las ha denominado Basic Linear Algebra Subprograms (BLAS) [80] [81] [43]. BLAS normalmente se divide en tres clases (conocidas como niveles), en función de la cantidad datos sobre los que operan y en función de la cantidad de operaciones que requiere cada una de ellas. Se tienen tres niveles de BLAS [46]:

- Nivel 1 (o L1 BLAS): para subrutinas que operan entre vectores, tal como $y = \alpha x + y$
- Nivel 2 (o L2 BLAS): para subrutinas que realizan operaciones con matrices y vectores, tal como $y = \alpha Ax + \beta y$
- Nivel 3 (o L3 BLAS): para subrutinas que operan con matrices, tal como $C = \alpha AB + \beta C$

donde A, B, y C representan matrices, x e y representan vectores y α y β representan escalares.

Más allá de la utilidad de esta clasificación para la caracterización e identificación de las operaciones, se estableció teniendo en cuenta que:

- La cantidad de datos sobre los que operan las subrutinas de nivel 1 es de $O(n)$, donde n representa la longitud de un vector, y la cantidad de operaciones básicas entre escalares también es de $O(n)$.
- La cantidad de datos sobre los que operan las subrutinas de nivel 2 es de $O(n^2)$, donde n representa el orden de matrices cuadradas (cantidad de filas y columnas), y la cantidad de operaciones básicas entre escalares también es de $O(n^2)$.
- La cantidad de datos sobre los que operan las subrutinas de nivel 3 es de $O(n^2)$, donde n representa el orden de matrices cuadradas (cantidad de filas y columnas), y la cantidad de operaciones básicas entre escalares es de $O(n^3)$.

Esto implica que, por un lado, las subrutinas incluidas en BLAS de nivel 3 son las que tienen mayor requerimiento en cuanto a capacidad de procesamiento. De hecho, la diferencia con BLAS de nivel 2 es tan grande, $O(n^3)$ vs. $O(n^2)$, que la mayoría de las veces (sino todas), optimizando BLAS de nivel 3 se tiene toda la optimización necesaria. Por otro lado, es claro que en cuanto a la optimización de las subrutinas, las de BLAS de nivel 3 son las más apropiadas, dado que tienen mayores requerimientos de cómputo que los otros dos niveles. Normalmente se considera que [46]

- Las subrutinas de L1 BLAS no pueden lograr alto rendimiento en la mayoría de las supercomputadoras. Aún así, son útiles en términos de portabilidad.
- Las subrutinas de L2 BLAS son especialmente apropiadas para algunas computadoras vectoriales (en términos de rendimiento) aunque no en todas, dado el movimiento de datos que imponen entre los distintos niveles de la jerarquía de memoria.
- Las subrutinas de L3 BLAS son las más apropiadas para lograr el máximo rendimiento en las supercomputadoras actuales, donde la jerarquía de memoria juega un rol muy importante en el rendimiento de todo el acceso a los datos que se procesan en la/s CPU/s.

De hecho, aunque LAPACK esté implementado (o pueda ser implementado directamente) en términos de L1 BLAS [46], actualmente se reconoce que está diseñado para explotar al

máximo L3 BLAS [LAPACK] porque actualmente estas subrutinas son casi las únicas con las que se puede lograr un alto rendimiento (cercano al máximo rendimiento de cada procesador utilizado), en las supercomputadoras. Por lo tanto, no cabe ninguna duda que en términos de rendimiento es esencial poner especial atención en las subrutinas definidas como BLAS nivel 3.

¿Qué es lo que hace que las subrutinas de BLAS nivel 3 sean especialmente apropiadas para su optimización? La respuesta tiene que ver con la forma en que se implementan los algoritmos, que se han denominado algoritmos de bloques [42] [5] [85] [104] [20] [144]. Estos algoritmos optimizan los accesos a memoria dado que maximizan la cantidad de operaciones que se llevan a cabo por cada dato que se referencia. En general, organizan el cómputo de manera tal que un bloque de datos es accedido y por lo tanto se asigna implícitamente en memoria cache/s, cambiando lo que sea necesario (orden de las iteraciones, niveles de "loop unroll", etc.) para que todas (o la mayoría) de las operaciones en las que interviene ese bloque de datos se ejecute inmediatamente y por lo tanto se aproveche al máximo. De esta manera, se logra reducir el tiempo efectivo a memoria dado que de hecho se aumenta al máximo el "cache hit" o la cantidad de veces que un dato al que se hace referencia desde el procesador se encuentra inmediatamente en memoria cache.

Los algoritmos de bloque se pueden adaptar a cada arquitectura (o, más específicamente, a cada jerarquía de niveles de cache/memoria) y por esta razón se suele utilizar el término *transportable* en vez de *portable*. Se tiende a que las rutinas con mayores posibilidades de optimización o con mayor potencial para obtener el máximo rendimiento posible se "adapten" a la arquitectura subyacente [46]. La mayoría de las empresas que diseñan y comercializan procesadores también proveen todas las subrutinas definidas como BLAS (y aún otras similares) de manera tal que aprovechen al máximo la capacidad de procesamiento [CXML] [SML] [SCSL1] [SCSL2]. Se suele indicar que estas subrutinas son optimizadas para los procesadores aún a nivel del lenguaje del procesador (assembly language) y por lo tanto el esfuerzo y el costo puesto en su implementación es también un indicador de la importancia de estas subrutinas.

2.2.2 L3 BLAS y Multiplicación de Matrices

La especificación de L3 BLAS es hecha originalmente para el lenguaje FORTRAN y las las subrutinas definidas/incluidas son [42] [BLAS]:

a. Productos de matrices "generales" (subrutinas con nombres que terminan con GEMM):

   $C \leftarrow \alpha\, op(A)\, op(B) + \beta C$

   donde op(X) puede ser X, X^T o X^H

b. Productos de matrices donde una de ellas es real o compleja simétrica o compleja hermítica (subrutinas con nombres que terminan con SYMM o HEMM):

   $C \leftarrow \alpha AB + \beta C$ o $C \leftarrow \alpha BA + \beta C$

donde A es simétrica para SYMM o hermítica para HEMM y está a izquierda o derecha de la multiplicación dependiendo de un parámetro (SIDE) de la subrutina.

c. Productos de matrices donde una de ellas triangular (subrutinas con nombres que terminan con TRMM):

$B \leftarrow \alpha\ op(A)\ B$ o $B \leftarrow \alpha\ B\ op(A)$

donde A es una matriz triangular, está a izquierda o derecha de la multiplicación dependiendo de un parámetro (SIDE) de la subrutina, y op(A) puede ser A, A^T o A^H.

d. Actualizaciones de rango *k* (rank-*k* update) de una matriz simétrica (subrutinas con nombres que terminan con SYRK):

$C \leftarrow \alpha AA^T + \beta C$ o $C \leftarrow \alpha A^T A + \beta C$

donde C es simétrica y A está a derecha o a izquierda de la multiplicación por A^T dependiendo de un parámetro de la subrutina (TRANS).

e. Actualizaciones de rango *k* (rank-*k* update) de una matriz hermítica (subrutinas con nombres que terminan con HERK):

$C \leftarrow \alpha AA^H + \beta C$ o $C \leftarrow \alpha A^H A + \beta C$

donde C es hermítica y A está a derecha o a izquierda de la multiplicación por A^H dependiendo de un parámetro de la subrutina (TRANS).

f. Actualizaciones de rango 2*k* (rank-2*k* update) de una matriz simétrica (subrutinas con nombres que terminan con SYR2K):

$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$ o $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$

donde C es simétrica y A está a derecha o a izquierda de la multiplicación por B^T dependiendo de un parámetro de la subrutina (TRANS).

g. Actualizaciones de rango 2*k* (rank-2*k* update) de una matriz hermítica (subrutinas con nombres que terminan con HER2K):

$C \leftarrow \alpha AB^H + \overline{\alpha}\ BA^H + \beta C$ o $C \leftarrow \alpha A^H B + \overline{\alpha}\ B^H A + \beta C$

donde C es hermítica y A está a derecha o a izquierda de la multiplicación por B^H dependiendo de un parámetro de la subrutina (TRANS).

h. Soluciones a sistemas de ecuaciones triangulares (subrutinas con nombres que terminan con TRSM):

$B \leftarrow \alpha\ op(A)\ B$ o $B \leftarrow \alpha\ B\ op(A)$

donde A es una matriz triangular, está a izquierda o derecha de la multiplicación dependiendo de un parámetro (SIDE) de la subrutina, y op(A) puede ser A^{-1}, A^{-T} o A^{-H}.

Dejando de lado las operaciones de $O(n^2)$ tales como al cálculo de $op(A) = A^T$, se puede notar intuitivamente que todas las subrutinas de L3 BLAS tienen como operación predominante (en cuanto a cantidad de operaciones aritméticas) la multiplicación de matrices. Además, en [77] se muestra cómo todo el nivel 3 de BLAS puede ser implementado en términos de la operación de multiplicación de matrices manteniendo el rendimiento cercano al óptimo posible de cada computadora. En el contexto comercial, se puede citar un ejemplo de Intel que, junto con la comercialización del Pentium III puso a disposición en Internet el documento [74] donde explica cómo aprovechar al máximo la capacidad de cálculo del procesador en términos de multiplicación de matrices, además de poner también a disposición de los usuarios del procesador una biblioteca de funciones de cálculo matricial.

2.3 La Multiplicación de Matrices como Benchmark

La caracterización del rendimiento de las computadoras se ha utilizado con varios propósitos, entre los cuales se pueden mencionar [63]:

- 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 y/o justificación 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 posibles de 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 [64] [SPEC]. 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 que puede existir 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 representar a *toda* la gama de posibles aplicaciones 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 específico de aplicaciones sin utilizar los benchmarks más generales. Este es el caso de las aplicaciones definidas en términos de las multiplicaciones de matrices, 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 toma como el *benchmark* de referencia en cuanto a rendimiento.

Utilizar un *benchmark* tan específico y tan cercano a la aplicación que se debe resolver tiene, en el contexto de los programas paralelos que se ejecutan sobre hardware heterogéneo, 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 (al menos de manera estática).

2.3.1 “Benchmark” del Nivel 3 de BLAS

Es claro que si se elige implementar todo el nivel 3 de BLAS directamente en términos de la multiplicación de matrices [77] el rendimiento obtenido será casi directamente el de la multiplicación de matrices misma. Pero si se elige implementar cada subrutina (L3 BLAS) aprovechando sus particularidades de cálculo de manera óptima e independiente de la multiplicación de matrices, aún es de esperar que el rendimiento sea muy similar al de la multiplicación de matrices misma. De esta manera, la multiplicación de matrices es un buen “representante” (y con esto se constituye en un *benchmark*) en cuanto a rendimiento de todas las rutinas del nivel 3 de BLAS.

Un argumento quizás más sólido para considerar a la multiplicación de matrices como representativa con respecto a rendimiento de todo el nivel 3 de BLAS es que al menos en el ámbito secuencial está demostrado experimentalmente que el rendimiento obtenible con cada una de las subrutinas del nivel 3 de BLAS es similar al que se puede obtener con la multiplicación de matrices [144]. En este sentido, conociendo el rendimiento que se obtiene con la multiplicación de matrices se puede tener una idea bastante concreta del rendimiento obtenible con todas las subrutinas del nivel 3 de BLAS.

2.3.2 Como Benchmark "General"

En el campo de los benchmarks en general, es decir de los programas que se intentan utilizar para identificar la capacidad de cálculo de las computadoras (sean paralelas o no), la representatividad de la multiplicación de matrices es mucho más discutible. De hecho, existe una gran cantidad de investigadores y empresas que consideran que lo *único* que puede ser un benchmark es una aplicación real [64]. Sin embargo, en algunas distribuciones de benchmarks de uso libre para máquinas paralelas [68] [94] se lo sigue incluyendo al menos como "benchmark de bajo nivel".

De una manera u otra, se siguen reportando los resultados en cuanto a rendimiento de la multiplicación de matrices en computadoras paralelas y en computadoras secuenciales. Una de las razones más importantes consiste en que con la multiplicación de matrices se puede lograr rendimiento cercano al óptimo teórico de la computadora utilizada. En este sentido, la multiplicación de matrices se ha transformado de una manera o de otra en una métrica de calidad de la implementación de algoritmos numéricos o al menos de los algoritmos relacionados con las operaciones de álgebra lineal. Un ejemplo en principio académico lo constituye ATLAS [144] [ATLAS] y, sólo por dar un ejemplo comercial, en [SCSL2] se intenta mostrar cuán *buena* es la biblioteca de cómputo científico provista asegurando que para máquinas monoprocesador el rendimiento excede el 95% teórico y para máquinas paralelas de 64 procesadores el rendimiento relativo global excede el 85% teórico. La idea en este sentido es: "existe código que resuelve al menos un problema de álgebra lineal con rendimiento cercano al óptimo teórico de cada procesador", con la intención de que esta idea se extrapola al menos a un subconjunto de los problemas que se quieren resolver en la/s computadora/s.

2.4 Paralelización de la Multiplicación de Matrices

Quizás por razones académicas (relacionadas con la simplicidad), uno de los primeros algoritmos paralelos que se explican en los libros dedicados a explicar procesamiento paralelo es el de la multiplicación de matrices [56] [82] [79] [146] [10] [58] [52] [3]. Pero más allá de su importancia académica, la investigación se ha mantenido activa a través del tiempo por su importancia como problema a resolver, y esto se demuestra por las numerosas publicaciones al respecto, algunas de las cuales son las mencionadas previamente y otras (en una lista resumida) son [23] [35] [30] [142] [26] [83].

La multiplicación de matrices tiene características muy específicas en lo que se refiere al diseño e implementación de un algoritmo paralelo en el contexto de los algoritmos paralelos en general:

- Cada elemento que se calcula de la matriz resultado C, c_{ij}, es, en principio, independiente de todos los demás elementos. Esta independencia es sumamente útil dado que permite un amplio grado de flexibilidad en lo referente a la paralelización.
- La cantidad y el tipo de operaciones a realizar es independiente de los datos mismos. La excepción en este caso la constituyen los algoritmos de multiplicación de matrices denominadas *ralas* (*sparse*), donde se intenta aprovechar el hecho de que la mayoría de

los elementos de las matrices a multiplicar (y por lo tanto de la matriz resultado) son iguales a cero.
- Regularidad de la organización de los datos y de las operaciones que se realizan sobre los datos. Los datos están organizados en estructuras bidimensionales (las matrices mismas) y las operaciones son básicas, de multiplicación y suma.

La primera característica hace que la multiplicación de matrices sea especialmente apropiada para las máquinas paralelas denominadas multiprocesadores, donde un conjunto de procesadores, o elementos de procesamiento, comparten una misma memoria. Los algoritmos paralelos para multiprocesadores suelen seguir las ideas básicas de descomposición o división de los datos a calcular y/o de *Divide-y-Conquista* (*Divide-and-Conquer*) recursivamente. En general en todos ellos se establece un período previo estático o dinámico de particionamiento o división de la cantidad de datos (o partes de la matriz resultado de la multiplicación) a ser calculados en cada procesador y eventualmente un período posterior de utilización de cálculos intermedios para calcular el resultado final.

Las últimas dos características hacen que los algoritmos propuestos para la multiplicación de matrices en paralelos sigan en general el modelo de cómputo paralelo SPMD (Single Program - Multiple Data) [52] [135]. De esta manera, un mismo programa se ejecuta asincrónicamente en cada procesador de la máquina paralela y eventualmente se sincroniza y/o comunica con los demás procesadores. Es de destacar que el modelo de cómputo SPMD es independiente de que la implementación se haga sobre una máquina paralela multiprocesador o multicomputadora o sobre una máquina paralela con arquitectura de procesamiento tan distribuida como una red de computadoras.

En general, es muy difícil encontrar en las publicaciones (sobre todo en los libros dedicados a explicar cómo llevar a cabo procesamiento paralelo), algoritmos paralelos presentados *para* un determinado tipo de arquitectura de cómputo paralelo. Si bien es cierto que los algoritmos se pueden adaptar de una manera más o menos compleja a cada una de las arquitecturas de procesamiento paralelo disponibles, también es cierto que hay un costo de adaptación a nivel algorítmico y de implementación y, quizás más importante en el contexto de las aplicaciones con grandes requerimientos de cómputo, el costo en términos de rendimiento obtenido puede ser demasiado alto. Es así que en realidad, en la mayoría de los casos, se puede encontrar una estrecha relación entre cada algoritmo y una arquitectura de cómputo paralelo en particular. Es por eso que en el resto de esta sección se asociará directamente cada uno de los algoritmos mencionados con la arquitectura subyacente de cómputo paralelo con la cual se obtendrán los mejores resultados de acuerdo al rendimiento obtenido o posible de obtener.

2.4.1 Algoritmos Paralelos *para* Multiprocesadores

Como se mencionó antes, los algoritmos que siguen los principios de división o descomposición de los cálculos y de “Divide-y-Conquista” recursivamente se consideran como los más apropiados para las máquinas paralelas de memoria compartida o multiprocesadores. De hecho, en el cálculo de $C = A \times B$ lo primero que se puede intentar es directamente dividir el cálculo de C en tanta cantidad de partes como procesadores se puedan utilizar. En este sentido, el hecho de las matrices A y B se acceden solamente para

lectura de sus elementos y la matriz C es la única que se accede para *escritura* (lo que se explicó antes en cuanto a que cada elemento de la matriz resultado se calcula independientemente de los demás) de sus elementos es ampliamente favorable.

Particionamiento Directo. Como lo muestra la Figura 2.1 con una determinada trama para cada procesador P_1, ..., P_4, cada uno de ellos puede acceder a los datos de las matrices A y B sin necesidad de sincronización con los demás (excepto a nivel físico, dependiendo de la organización de la memoria compartida) dado que los datos de ambas matrices se acceden solamente para lectura. De la misma manera, cada procesador puede acceder a la matriz C independientemente de los demás para almacenar cada uno de los elementos que debe calcular en función de los elementos de A y de B.

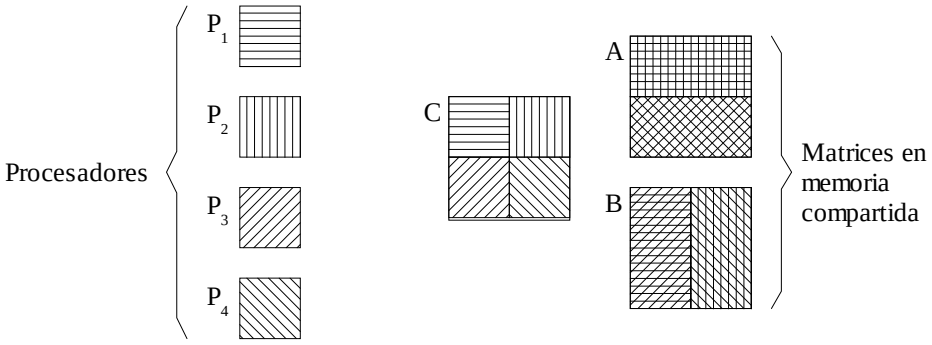


Figura 2.1: División del Cálculo de la Multiplicación en Multiprocesadores.

Esta forma de realizar los cálculos a lo sumo necesitaría una fase inicial para determinar la parte que cada procesador debe procesar y a lo sumo una sincronización final para determinar cuándo todos los procesadores han terminado los cálculos y por lo tanto el resultado está completamente calculado. Por otro lado, no se agrega ningún tipo de replicación de los datos a pesar de que algunas partes de las matrices A y B son utilizadas por más de un procesador dado que todos los datos se almacenan en la memoria compartida.

El hecho de que más de un procesador accede a la misma parte de una matriz (A o B) puede generar inconvenientes referidos a simultaneidad de accesos a memoria. En este sentido, y dependiendo del diseño de la memoria compartida, pueden llegar a ser secuencializados y por lo tanto penalizados en cuanto a rendimiento. Sin embargo, estos problemas son de solución muy simple dado que:

- Se pueden intercalar en el acceso a distintas partes de una misma matriz. En el ejemplo de la Figura 2.1, por ejemplo, el procesador P_1 podría comenzar el acceso a la matriz A desde la primera fila en adelante y el procesador P_2 desde la última fila que le corresponde acceder hacia la primera.
- Los distintos niveles de memoria cache intermedias (entre cada procesador y la memoria principal compartida) más los algoritmos de cálculo de bloques reducen de forma significativa la cantidad de accesos a la memoria compartida.

Es interesante notar la simplicidad de la división, aprovechando las características de la

propia operación de multiplicación más la homogeneidad de los multiprocesadores en cuanto a la capacidad de cálculo de cada elemento de cálculo (procesadores).

Divide-y-Conquista Recursivo. La idea de llevar a cabo la multiplicación por partes o submatrices es aprovechada en este tipo de algoritmos [70] [62] para la paralelización del procesamiento. El algoritmo en pseudo-código puede ser expresado como lo muestra la Figura 2.2 [146]:

```
mat_mul(A, B, C, s)
/* A, B: matrices a multiplicar */
/* C: matriz resultado */
/* s: tamaño de las matrices */
{
   if (multiplicación secuencial)
   {
      C = A×B;
   }
   else
   {
      mat_mul(A00, B00, C00, s/2);   /* (1) */
      mat_mul(A01, B10, C100, s/2);   /* (2) */
      mat_mul(A00, B01, C001, s/2);   /* (3) */
      mat_mul(A01, B11, C101, s/2);   /* (4) */
      mat_mul(A10, B00, C010, s/2);   /* (5) */
      mat_mul(A11, B10, C110, s/2);   /* (6) */
      mat_mul(A10, B01, C011, s/2);   /* (7) */
      mat_mul(A11, B11, C111, s/2);   /* (8) */
   }
   C00 = C000 + C100;
   C01 = C001 + C101;
   C10 = C010 + C110;
   C11 = C011 + C111;
}
```

Figura 2.2: Pseudo-Código de Divide-y-Conquista Recursivo.

donde:

- Cada una de las matrices A, B, y C se divide en cuatro partes iguales, tal como lo muestra la Figura 2.3. Esta cantidad de partes de cada matriz tiene relación directa con la cantidad de llamadas recursivas que se deben llevar a cabo para la obtención de los cálculo intermedios.
- La mayor cantidad de operaciones se lleva a cabo en las llamadas recursivas a `mat_mul` y las últimas cuatro operaciones de suma entre las matrices de cálculos intermedios $C0_{ij}$ y $C1_{ij}$ ($0 \leq i, j \leq 1$) se deben realizar para obtener el resultado correcto de cada una de las submatrices de la matriz C, tal como se puede identificar en la Figura 2.3. Estas últimas operaciones se pueden llevar a cabo en un subconjunto de los procesadores utilizados para resolver las multiplicaciones de las llamadas recursivas.
- Cada una de las llamadas recursivas a `mat_mul` numeradas de (1) a (8) puede ser ejecutada en un procesador diferente, dependiendo de la cantidad de procesadores disponibles y del rendimiento obtenido de acuerdo a la cantidad de datos de las matrices

a multiplicar.

- La condición (`multiplicación secuencial`) puede estar dada en función del tamaño de las matrices a multiplicar (s = 1 en el caso extremo), o la cantidad de llamadas recursivas, que determina a su vez la cantidad de procesadores a utilizar simultáneamente para el cálculo de resultados parciales.

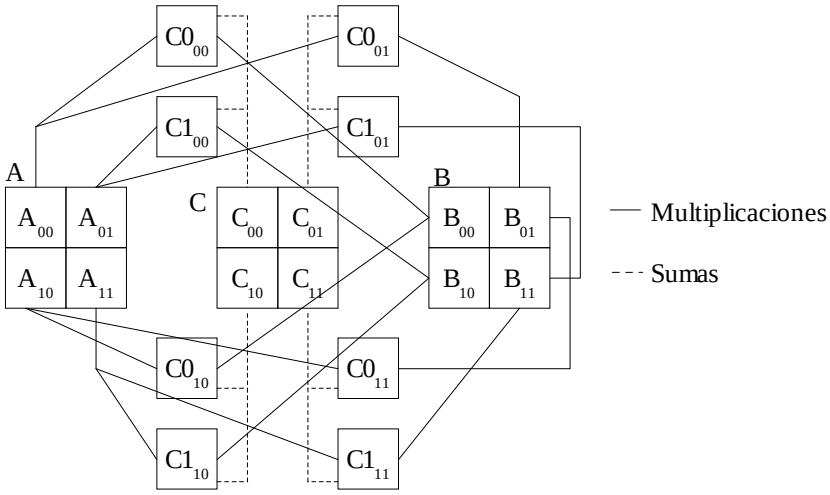


Figura 2.3: Submatrices y Cálculos de Divide-y-Conquista.

Para este algoritmo, tal como para el anterior de particionamiento directo, también es interesante notar que la división de los cálculos (y la consiguiente paralelización) se ve ampliamente favorecida por la homogeneidad en los elementos de procesamiento de los multiprocesadores.

Se debe notar también que, tal como está enunciado, el espacio requerido para los datos aumenta considerablemente teniendo en cuenta que por cada bloque de la matriz resultado C, C_{ij} se tienen dos bloques de datos intermedios $C0_{ij}$ y $C1_{ij}$. Sin embargo, con algunas modificaciones, tal como la reducción del paralelismo en la cantidad de llamadas recursivas o aumentando la dependencia entre los cálculos intermedios con bloques de datos, este requerimiento extra de memoria se puede evitar.

La Figura 2.4 muestra la modificación al pseudo-código de la Figura 2.2 para evitar que la cantidad de memoria requerida sea mayor que la cantidad de memoria requerida para el algoritmo secuencial. De esta manera, se modifica `mat_mul` para que realice una multiplicación y una suma (al estilo de BLAS _GEMM) en vez de una multiplicación solamente, transformándose en `mat_mul_sum`. Todo el procesamiento por bloques se mantiene, aunque ahora se tienen pares de llamadas recursivas a `mat_mul_sum` que utilizan un mismo bloque de C. Estas serían las llamadas numeradas, con (1) y (2), (3) y (4), (5) y (6) y (7) y (8) respectivamente. Esta utilización de un mismo bloque de la matriz C en pares de llamadas recursivas tiene como consecuencias:

- Ya no son necesarios (con respecto a `mat_mul`) dos bloques de datos intermedios para un único bloque de la matriz resultado.
- Las llamadas recursivas numeradas de (1) a (8) ya no son totalmente independientes entre sí, sino que entre pares de llamadas hay dependencias de datos y por lo tanto no

podrían ejecutarse simultáneamente. En este sentido, se reduce de 8 a 4 la cantidad de multiplicaciones que se pueden llevar a cabo simultáneamente (en distintos procesadores).

- Ya no son necesarias las sumas finales que aparecen en la Figura 2.3 porque se resuelven directamente en la misma subrutina `mat_mul_sum`.

```
mat_mul_sum(A, B, C, s) /* C = A×B + C */
/* A, B: matrices a multiplicar */
/* C: matriz resultado */
/* s: tamaño de las matrices */
{
   if (multiplicación secuencial)
   {
      C = A×B + C;
   }
   else
   {
      mat_mul(A00, B00, C00, s/2);   /* (1) */
      mat_mul(A01, B10, C00, s/2);   /* (2) */
      mat_mul(A00, B01, C01, s/2);   /* (3) */
      mat_mul(A01, B11, C01, s/2);   /* (4) */
      mat_mul(A10, B00, C10, s/2);   /* (5) */
      mat_mul(A11, B10, C10, s/2);   /* (6) */
      mat_mul(A10, B01, C11, s/2);   /* (7) */
      mat_mul(A11, B11, C11, s/2);   /* (8) */
   }
}
```

Figura 2.4: Modificación de Divide-y-Conquista Recursivo.

Lo que en el algoritmo de particionamiento directo es la fase inicial de división de las matrices, en este algoritmo serían las llamadas recursivas que serían resueltas por diferentes procesadores.

Paralelización del Método de Strassen. El método de Strassen es uno de los más innovadores en cuanto a la multiplicación de matrices resueltos de manera secuencial [114] y en [59] se lo denomina también como algoritmo de Divide-y-Conquista y se lo presenta además como un algoritmo recursivo. La Figura 2.5-a) muestra los cálculos intermedios asumiendo que las matrices a multiplicar se dividen en cuatro partes o submatrices o bloques iguales como los mostrados en la Figura 2.5-b).

Aunque la cantidad de operaciones aritméticas para este método es un poco más difícil de calcular de manera exacta que para la multiplicación de matrices clásica, se suele calcular (o estimar en términos de órdenes de magnitud), la cantidad de operaciones de multiplicaciones asumiendo que la cantidad de sumas es aproximadamente igual [59]. Con esta consideración, el método de Strassen tiene a su favor que reduce la complejidad o cantidad de cálculos entre números de punto flotante a $O(n^{\log_2 7})$ considerando como referencia el método convencional de multiplicación, que es de $O(n^3)$. También se podría mencionar como una ventaja el hecho de que puede ser implementado utilizando recursión.

$P_0 = (A_{00} + A_{11}) \times (B_{00} + B_{11})$
$P_1 = (A_{10} + A_{11}) \times B_{00}$
$P_2 = A_{00} \times (B_{01} - B_{11})$
$P_3 = A_{11} \times (B_{10} - B_{00})$
$P_4 = (A_{00} + A_{01}) \times B_{11}$
$P_5 = (A_{10} - A_{00}) \times (B_{00} + B_{01})$
$P_6 = (A_{01} - A_{11}) \times (B_{10} + B_{11})$
$C_{00} = P_0 + P_3 - P_4 + P_6$
$C_{01} = P_2 + P_4$
$C_{10} = P_1 + P_3$
$C_{11} = P_0 + P_3 - P_1 + P_5$

a) Cálculos con submatrices

A

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

B

B_{00}	B_{01}
B_{10}	B_{11}

C

C_{00}	C_{01}
C_{10}	C_{11}

b) Partición de las matrices

Figura 2.5: Método de Strassen.

Desde el punto de vista de la implementación secuencial del método de Strassen:

- Normalmente se puntualiza que las operaciones entre los elementos de las matrices son distintas de las definidas por el método convencional y por lo tanto los efectos de redondeo y estabilidad numérica pueden ser diferentes dependiendo de los valores de los elementos de las matrices a multiplicar [59].
- De forma similar a lo que sucede con el algoritmo mencionado antes como Divide-y-Conquista recursivo, son necesarios datos intermedios para llegar a los valores definitivos a calcular: los bloques P_0, ..., P_6 de la Figura 2.5-a). A diferencia del algoritmo presentado como Divide-y-Conquista recursivo la eliminación de esos bloques de datos intermedios es muy difícil o imposible y de hecho no se considera, por lo que los requerimientos de memoria del método de Strassen son bastante mayores a los del método tradicional.

Desde el punto de vista de la paralelización del método de Strassen:

- Considerando los multiprocesadores de memoria compartida es inmediata, ya que, tal como el método de Divide-y-Conquista recursivo, existen varias multiplicaciones que se pueden llevar a cabo simultáneamente. Específicamente en el caso del método de Strassen son siete: el cálculo de cada P_k, con $0 \leq k \leq 6$.
- Existe cierto desbalance de carga computacional tanto en el cálculo de los bloques intermedios P_i como en el cálculo de los bloques definitivos C_{ij} a partir de los P_k. Para calcular P_0 por ejemplo, son necesarias dos sumas de bloques y una multiplicación, pero para el cálculo de P_1 son necesarias una suma y una multiplicación. Esto afecta tanto a la cantidad de datos que se acceden como a la cantidad de operaciones entre escalares que se deben llevar a cabo. De todas maneras se debe puntualizar que estas diferencias son a nivel de operaciones de $O(n^2)$ (sumas o restas de matrices) frente a operaciones de multiplicación que tienen complejidad de $O(n^3)$ u $O(n^{\log_2 7})$.

Es importante recordar que la idea de dividir las matrices a utilizar/calcular en bloques es intensiva y extensivamente utilizada en todos los algoritmos matriciales secuenciales y también paralelos dado que, como se puntualizó antes, permite la organización del

cómputo por bloques y de esta manera aumenta notablemente la utilización de la/s memoria/s cache/s. Dado que las matrices a procesar normalmente son muy grandes (en general, se tiende a ocupar toda la memoria principal disponible y en algunos casos también el espacio de memoria *swap*) esta organización de los cálculos es imprescindible para obtener rendimiento aceptable. Expresado de otra manera, sin procesamiento por bloques el tiempo de acceso a los datos es varios órdenes de magnitud más grande de lo que el procesador requiere para operar a su máxima velocidad o al menos a una fracción importante de la máxima posible.

2.4.2 Algoritmos Paralelos *para* Multicomputadoras

La mayoría de los reportes de investigación de algoritmos paralelos para la multiplicación de matrices (y problemas similares) corresponden a los diseñados teniendo en cuenta que la arquitectura de cómputo subyacente será la de una multicomputadora [59] [136]. Por un lado, las multicomputadoras siempre han sido consideradas más escalables que los multiprocesadores y por otro lado, el diseño y desarrollo de las multicomputadoras se ha mantenido constante a través del tiempo, lo que las ha hecho más atractivas también para el desarrollo de algoritmos paralelos.

Arreglo Sistólico o Malla de Procesadores. Aunque esta forma de multiplicar matrices es pensada inicialmente para computadoras del tipo SIMD o simplemente para ser implementada directamente en hardware [82] [146], puede ser aplicada en general considerando bloques de matrices tal como en los algoritmos anteriores. La Figura 2.6 muestra la disposición inicial de los datos y los elementos de procesamiento para multiplicar dos matrices de 3×3 elementos.

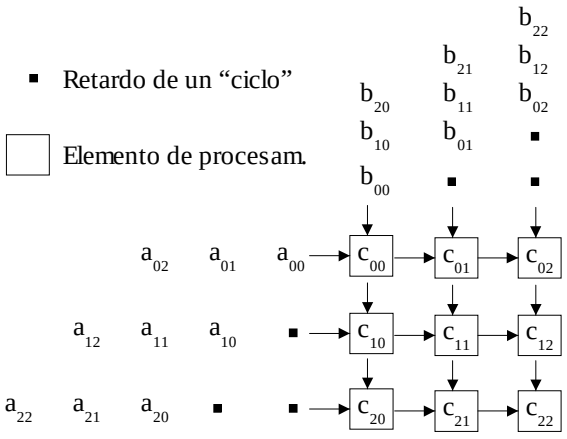


Figura 2.6: Multiplicación de 3×3 en un Arreglo Malla.

Claramente, los elementos de procesamiento o procesadores están interconectados en forma de malla o arreglo bidimensional. Normalmente se cuenta como un “ciclo” o “paso” del procesamiento las operaciones de comunicación que indican las flechas y todas las operaciones de multiplicación y suma que cada elemento de procesamiento pueda realizar

dependiendo de los datos que tenga disponibles.

En la Figura 2.7-a) se muestra el primer paso del procesamiento donde:
- Todos los elementos de las matrices A y B "avanzan" en la dirección de las flechas correspondientes y los elementos a_{00} y b_{00} llegan al procesador dedicado a calcular c_{00}.
- Se lleva a cabo la primera operación para el cálculo de c_{00}, es decir $c_{00}^{(1)}=a_{00}\times b_{00}$ (el superíndice indica la iteración que corresponde).

En la Figura 2.7-b) se muestra el segundo paso del procesamiento donde:
- Una vez más todos los elementos de las matrices A y B "avanzan" en la dirección de las flechas correspondientes, los elementos
  - a_{01} y b_{10} llegan al procesador dedicado a calcular c_{00},
  - a_{00} b_{01} llegan al procesador dedicado a calcular c_{01},
  - a_{10} y b_{00} llegan al procesador dedicado a calcular c_{10}.
- Se llevan a cabo las operaciones posibles en este paso para el cálculo de c_{00}, c_{01} y c_{10}, es decir $c_{00}^{(2)}=c_{00}^{(1)}+a_{01}\times b_{10}; c_{01}^{(2)}=a_{00}\times b_{01}; c_{10}^{(2)}=a_{10}\times b_{00}$.

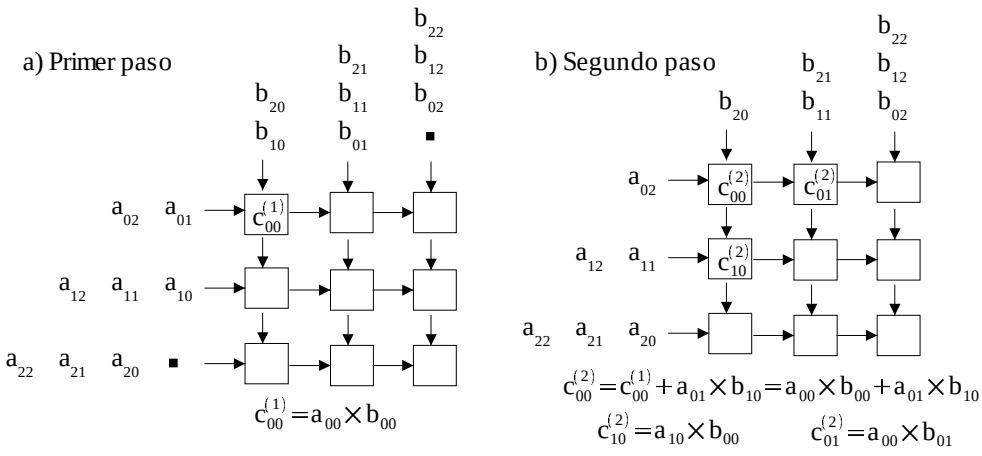


Figura 2.7: Primeros Dos Pasos de la Multiplicación en una Malla.

Para que este tipo de procesamiento llegue a tener rendimiento óptimo se debería tener:
- Todas las comunicaciones se llevan a cabo simultáneamente. En el caso de la Figura 2.6 y la Figura 2.7, esto implica que todos los elementos de procesamiento pueden realizar simultáneamente (solapadamente en el tiempo) hasta:
  - dos comunicaciones para recepción de datos
  - dos comunicaciones para envío de datos
- Cómputo solapado con las comunicaciones, es decir que se realizan operaciones aritméticas a la vez que se envían y reciben datos. En este caso, en los pasos explicados previamente se solapa (realiza simultáneamente) el cómputo de $c_{00}^{(1)}=a_{00}\times b_{00}$ con las comunicaciones correspondientes al segundo paso del procesamiento.
- Operaciones simultáneas de I/O, considerando que las comunicaciones en dirección a la primera fila y primera columna del arreglo bidimensional de la Figura 2.6 y la Figura 2.7 implican I/O. En caso de no contar con I/O de datos en paralelo se deberían tener todos los datos de las matrices inicialmente en la primera fila y primera columna de procesadores, lo que llevaría a su vez a distintos requerimientos de memoria para los

distintos procesadores dependiendo de la posición que ocupan en la malla.

Esta forma de multiplicar matrices es sumamente sencilla por varias razones, siendo las más destacables:

- Simplicidad y naturalidad de la distribución de los cálculos: básicamente un procesador o elemento de procesamiento se dedica exclusivamente al cálculo de una parte de la matriz resultado, sea un elemento o una submatriz o bloque. En este punto una vez más se asume que los elementos de procesamiento son homogéneos y por lo tanto la distribución es trivial.
- Patrón de comunicaciones "fijo" y conocido a priori: todas las transmisiones de datos son punto a punto (entre dos procesadores) y conocidas de antemano en cuanto a cantidad y tipo de transmisiones (cuántos datos y entre cuáles procesadores).
- Requerimientos de memoria iguales para todos los procesadores: no hay ningún procesador que deba tener más o menos datos que los demás, aún si se cuentan los *buffers* necesarios para las comunicaciones, dado que son las mismas en todos los procesadores.

Las primeras dos características hacen que el balance de carga sobre los procesadores (en cuanto a cálculos que se deben llevar a cabo) y sobre la red de interconexión (en cuanto a las transferencias de datos que se deben realizar entre los procesadores) sea muy sencilla.

Algoritmo de Cannon. También está propuesto para un arreglo bidimensional de elementos de procesamiento [23] [79], interconectados como una malla y además con los extremos de cada fila y columna interconectados entre sí, es decir formando la estructura denominada toro tal como lo muestra la Figura 2.8 para 3×3 elementos de procesamiento.

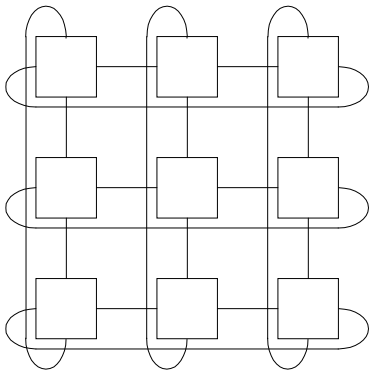
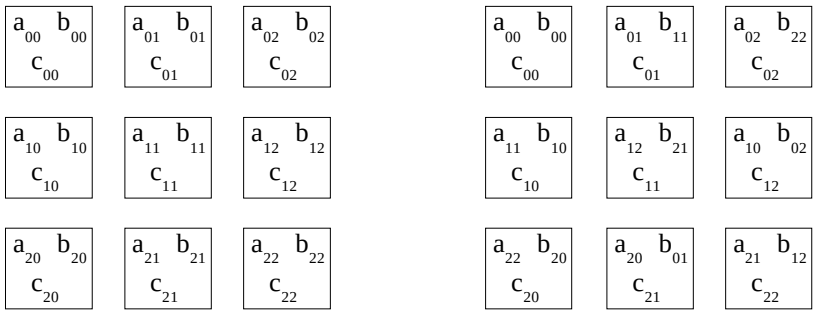


Figura 2.8: Toro de 3×3 Elementos de Procesamiento.

Inicialmente, la distribución de los datos de las matrices A, B, y C es similar a la definida previamente en la malla, es decir que si se numeran los procesadores de acuerdo a su posición en el arreglo bidimensional, el procesador P_{ij} ($0 \leq i, j \leq P-1$), contiene los elementos o bloques de la posición *ij* ($0 \leq i, j \leq P-1$), de las matrices A, B y C. Para simplificar la explicación, se utilizarán elementos de matrices en vez de bloques. A partir de esta distribución de datos, se "realinean" o reubican los datos de las matrices A y B de forma tal que si se tiene un arreglo bidimensional de P×P procesadores, el procesador P_{ij}

tendrá asignado el elemento o submatriz de A en la fila i y columna $(j+i)$ mod P, $a_{i,(j+i)\text{mod}P}$, y también el elemento o submatriz de B en la fila $(i+j)$ mod P y columna j, $b_{(i+j)\text{mod}P,i}$. Puesto de otra manera, cada dato de la fila i ($0 \le i \le$ P-1) de elementos o submatrices de A se traslada o rota (*shift*) hacia los procesadores de la izquierda i veces y cada dato de la columna j ($0 \le j \le$ P-1) de elementos o submatrices de B se traslada o rota (*shift*) hacia los procesadores de arriba j veces. En la Figura 2.9-a) se puede ver la asignación inicial, y en la Figura 2.9-b) la reubicación inicial, impuesta por el algoritmo de Cannon para matrices de 3×3 elementos en un toro de 3×3 procesadores (para simplificar no se muestran en la figura las interconexiones).



a) Distribución Inicial. b) Alineación Inicial.

Figura 2.9: Ubicación de 3×3 Datos para el Algoritmo de Cannon.

A partir de la reubicación inicial, se realizan iterativamente los pasos de:
- Multiplicación local de los datos asignados en cada procesador para el cálculo de un resultado parcial.
- Rotación a izquierda de los elementos o submatrices de A.
- Rotación hacia arriba de los elementos o submatrices de B.

y después de P de estos pasos se tienen los valores de la matriz C totalmente calculados.

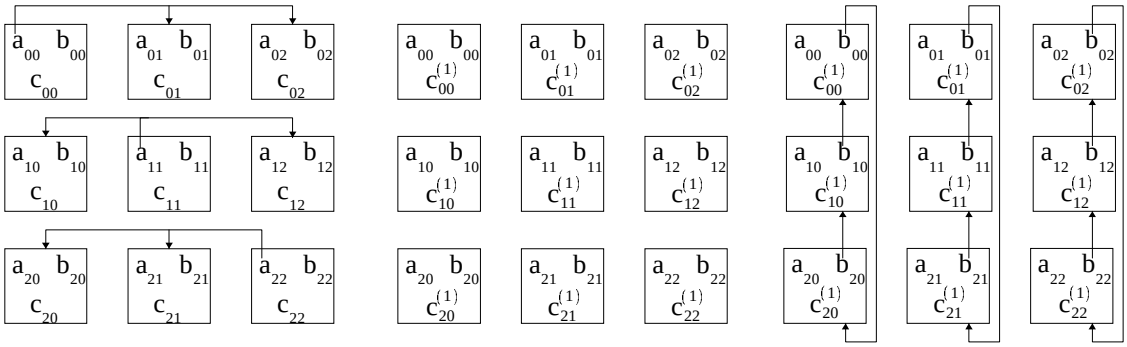
Resumiendo, las características sobresalientes de esta forma de realizar la multiplicación de matrices son:
- Por la forma en que se comunican los datos de las matrices A y B se puede decir que este es un algoritmo de “alineación inicial y rotaciones”.
- El balance de carga tanto en lo referente a cómputo como a comunicaciones está asegurado siempre que los elementos de procesamiento sean homogéneos.
- Como en el procesamiento definido para la grilla de procesadores, el tiempo de ejecución se minimiza si se tiene la capacidad de solapar en el tiempo cómputo con comunicaciones.
- La distribución de los datos de las matrices A y B no es la inicial al terminar el cálculo de la multiplicación de matrices.

Es de destacar que la última características de las enumeradas previamente se torna bastante importante dado que, como se ha mencionado anteriormente y como con todas las rutinas de L3 BLAS, se supone que son parte de alguna aplicación y no necesariamente la aplicación misma. Por lo tanto, cualquier procesamiento siguiente deberá tener en cuenta esta nueva distribución de las matrices, o después del algoritmo de Cannon se deberán realinear las matrices para recuperar la asignación inicial de los datos.

Algoritmo de Fox. También está propuesto para un arreglo bidimensional de elementos de procesamiento [57] [56] [58] [79], interconectados como una malla y además con los extremos de cada fila y columna interconectados entre sí, es decir formando la estructura denominada toro. Una vez más, la distribución de los datos de las matrices A, B, y C es similar a la definida antes, es decir que si se numeran los procesadores de acuerdo a su posición en el arreglo bidimensional, el procesador P_{ij} ($0 \le i, j \le P-1$), contiene los elementos o bloques de la posición *ij* ($0 \le i, j \le P-1$), de las matrices A, B y C, tal como se muestra en la Figura 2.9-a) y en la Figura 2.10-a). En este caso, no se define ningún paso inicial de reubicación de los datos, sino que a partir de aquí se define iterativamente el algoritmo. En la iteración o paso *k*:

- El bloque o dato de la matriz A almacenado en el procesador de la posición *i*, *i+k* mod P (fila *i*, columna *i+k* mod P) se envía a todos los procesadores de la fila *i* (su misma fila) en un *broadcast por fila* del arreglo bidimensional.
- Se multiplican en cada procesador los datos de A recibidos con los datos de B almacenados localmente, obteniendo un resultado parcial de la matriz C.
- Se rotan los datos de B hacia arriba tal como en el algoritmo de Cannon.

La Figura 2.10 muestra el procesamiento correspondiente al primer paso del algoritmo de Fox en un arreglo bidimensional de 3×3 procesadores, y la Figura 2.11 muestra el procesamiento correspondiente al segundo paso del algoritmo de Fox en un arreglo bidimensional de 3×3 procesadores.



$$c_{00}^{(1)} = a_{00} \times b_{00};\ c_{01}^{(1)} = a_{00} \times b_{01};\ c_{02}^{(1)} = a_{00} \times b_{02}$$
$$c_{10}^{(1)} = a_{11} \times b_{10};\ c_{11}^{(1)} = a_{11} \times b_{11};\ c_{12}^{(1)} = a_{11} \times b_{12}$$
$$c_{20}^{(1)} = a_{22} \times b_{20};\ c_{21}^{(1)} = a_{22} \times b_{21};\ c_{22}^{(1)} = a_{22} \times b_{22}$$

a) Broadcast. b) Cómputo local. c) Rotación.

Figura 2.10: Primer paso del Algoritmo de Fox, Matrices de 3×3.

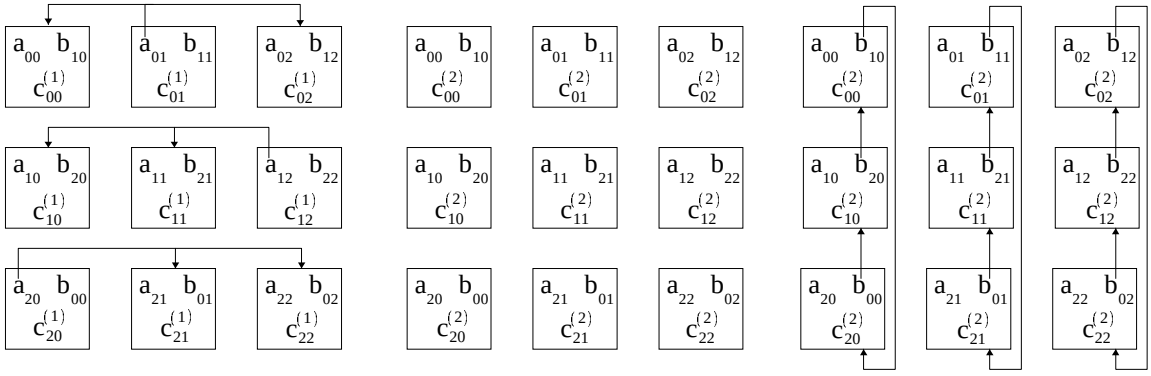
Las principales características del algoritmo de Fox que se pueden enumerar son:

- Como en los casos anteriores, el balance de carga en cuanto a cómputo que debe realizar cada procesador es muy sencillo asumiendo que los procesadores son homogéneos.
- El balance de carga en cuanto a la red de intercomunicaciones no es tan sencillo de analizar como en el algoritmo de Cannon, ya que se deben tener en cuenta no solamente comunicaciones punto a punto sino también comunicaciones colectivas: broadcasts por

filas. De todas maneras, se tendría balance de carga de la red de comunicaciones en cuanto a comunicaciones por columnas, dado que son todas punto a punto; y por filas, dado que son todas comunicaciones broadcast. Desde el punto de vista de las comunicaciones que se generan en cada procesador, todos generan la misma carga sobre la red de interconexión:

- un envío de datos de la matriz B hacia arriba
- una recepción de datos de la matriz B desde abajo
- un envío o una recepción de broadcast de datos de la matriz A hacia los demás procesadores de fila o desde un procesador de la misma fila respectivamente.

- Dado que todos los procesadores tienen la misma cantidad y tipo de comunicaciones, en caso de ser necesarios buffers, éstos serán los mismos en todos los procesadores.
- Los requerimientos de memoria en cada procesador son mayores en comparación con el algoritmo de Cannon, ya que todos los procesadores deberían tener:
  - Un bloque o submatriz de la matriz A local
  - Un bloque o submatriz de la matriz B local
  - Un bloque o submatriz de la matriz C local
  - Un bloque más, para recibir el broadcast por filas del bloque de A que en cada paso se comunica.
- A diferencia del algoritmo de Cannon, al finalizar la multiplicación de matrices, los datos de cada una de las matrices queda como en la asignación inicial.



$$c_{00}^{(2)} = c_{00}^{(1)} + a_{01} \times b_{10};\ c_{01}^{(2)} = c_{01}^{(1)} + a_{01} \times b_{11};\ c_{02}^{(2)} = c_{02}^{(1)} + a_{01} \times b_{12}$$
$$c_{10}^{(2)} = c_{10}^{(1)} + a_{12} \times b_{20};\ c_{11}^{(2)} = c_{11}^{(1)} + a_{12} \times b_{21};\ c_{12}^{(2)} = c_{12}^{(1)} + a_{12} \times b_{22}$$
$$c_{20}^{(2)} = c_{20}^{(1)} + a_{20} \times b_{00};\ c_{21}^{(2)} = c_{21}^{(1)} + a_{20} \times b_{01};\ c_{22}^{(2)} = c_{22}^{(1)} + a_{20} \times b_{02}$$

a) Broadcast. b) Cómputo local. c) Rotación.

Figura 2.11: Segundo paso del Algoritmo de Fox, Matrices de 3×3.

En lo que se refiere a las comunicaciones broadcast por filas, se debe tener en cuenta que tanto en una malla como en un toro las únicas comunicaciones que se podrían denominar *predefinidas* son las que se llevan a cabo entre procesadores *vecinos*. En este sentido, se requiere algún esfuerzo adicional posiblemente penalizado en cuanto a rendimiento, para realizar cualquiera de las comunicaciones colectivas, incluyendo el broadcast por filas, que también puede ser definido como un multicast desde el punto de vista de la red completa de comunicaciones. En este sentido, existen muchas publicaciones que se dedican a

implementar o de alguna manera adaptar el algoritmo para la red de interconexión estática toro [30][142][26]. La idea básica consiste en implementar un broadcast vía múltiples comunicaciones punto a punto solapadas, es decir que mientras un broadcast se está llevando a cabo con comunicaciones punto a punto pueden comenzar las transmisiones punto a punto para el broadcast siguiente. Este tipo de adaptaciones al algoritmo de Fox es la que finalmente se implementa en la biblioteca ScaLAPACK [21].

También es posible reducir el requerimiento de memoria, que en principio puede ser bastante mayor que el definido en el algoritmo de Cannon (más específicamente, 33.3...% mayor). La clave de la reducción está dada en la comunicación por subbloques de los datos de la matriz A asignada en cada procesador. En vez de hacer un broadcast de todos los datos de la matriz A se podrían, por ejemplo, hacer dos broadcasts cada uno con la mitad de los datos de la matriz A asignada localmente. Esto automáticamente reduce el requerimiento extra (tomando como referencia el algoritmo de Cannon) de memoria a la mitad de los datos de la matriz A que cada procesador tiene localmente. De la misma manera, se pueden hacer diez broadcasts, cada uno con la décima parte de la matriz A asignada localmente y esto reduce a la décima parte la cantidad *extra* de memoria necesaria.

Es de destacar que al finalizar el procesamiento, los datos de las matrices quedan asignados en cada procesador exactamente igual que antes de comenzar la multiplicación de las matrices. En este sentido, recordando lo que se mencionó antes en el algoritmo de Cannon, que la multiplicación de matrices se debe hacer en el contexto de otras operaciones, es una ventaja considerable.

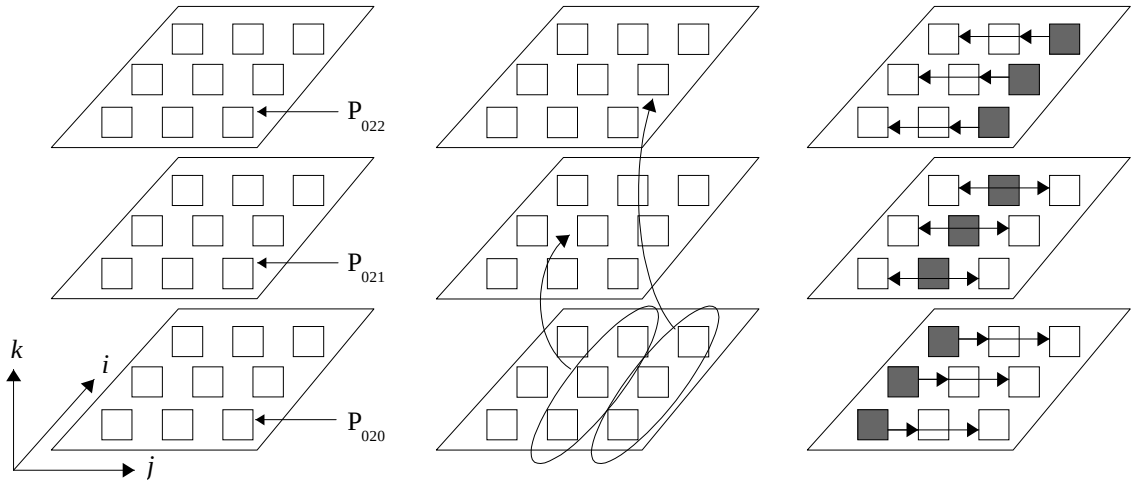
De hecho, tanto para resolver el problema de hacer un broadcast en un toro como para reducir la cantidad de memoria necesaria para almacenar los datos localmente se utiliza el mismo principio que para acelerar el acceso a los datos en la jerarquía de memoria asumiendo la existencia de uno o más niveles de memoria cache: procesamiento por bloques. En vez de manejar todos los datos de las matrices A, B y C asignados localmente, cada una de las submatrices se considera por bloques, y son estos bloques los que se procesan localmente (haciendo las multiplicaciones de matrices parciales), y también son los que se comunican a través de las conexiones punto a punto en una misma fila de procesadores formando un pipeline de comunicaciones con el que finalmente se resuelve el broadcast. Por lo tanto, dado que son subbloques o partes de la matriz A asignada localmente, la cantidad de memoria extra que se necesita en cada procesador se reduce a un subbloque o conjunto de subbloques que se transmite en una misma comunicación punto a punto entre procesadores de una misma fila.

Método DNS y Mallas de Arboles. El algoritmo DNS (por las letras iniciales de los apellidos de sus autores: Dekel, Nassimi, Sahni) [35] [100] [79] propone multiplicar matrices cuadradas de orden *n* en una multicomputadora con sus procesadores interconectados en un hipercubo de tres dimensiones. Una vez más, la descripción básica del procesamiento se hará para elementos de matrices, pero su extensión/aplicación a bloques o submatrices es inmediata.

Como punto de partida para el algoritmo DNS, se focaliza la atención directamente en cada

producto de elementos de matrices, $a_{ik} \times b_{kj}$, que son exactamente n^3 operaciones entre escalares en el caso de multiplicar matrices cuadradas de orden n. Teniendo n^3 procesadores, cada procesador se hace cargo de exactamente una multiplicación y luego se deben sumar los valores obtenidos de cada multiplicación de la manera correcta para obtener cada uno de los n^2 elementos de la matriz resultado. Más específicamente, numerando los procesadores como si estuvieran organizados en un arreglo de tres dimensiones (es decir $n \times n \times n$ procesadores), el procesador P_{ijk}, $0 \leq i, j, k \leq (n\text{-}1)$, es el encargado de llevar a cabo la multiplicación $a_{ik} \times b_{kj}$. Luego, acumulando (sumando) los valores $P_{i,j,0} \ldots P_{i,j,n-1}$ se obtiene el valor c_{ij} de la matriz resultado.

Si se consideran los datos (elementos o submatrices) de las matrices distribuidos en los procesadores P_{ij0} (es decir que a_{ik}, b_{kj} y c_{ij} se asignan el procesador P_{ij0}), antes de realizar las multiplicaciones se deben distribuir/replicar los datos. Para este paso también conviene visualizar los procesadores como si estuvieran organizados en un arreglo de tridimensional, tal como lo muestra la Figura 2.12-a) para veintisiete procesadores, es decir como sucesivos planos de 3×3 procesadores (correspondientes a los índices ij) superpuestos para distintos valores de k.



a) Identificación de Procesadores. b) Distribución de Columnas de A. c) Broadcast por Planos.

Figura 2.12: Datos en DNS sobre un Arreglo de 3×3×3 Procesadores.

A partir de la asignación inicial de elementos de matrices a los procesadores P_{ij0}, en el plano inferior en la Figura 2.12-a), la columna j-ésima de la matriz A se envía a los procesadores correspondientes (columna j-ésima de procesadores) en el plano j-ésimo, es decir desde los procesadores P_{ij0} a los procesadores P_{ijj} ($0 \leq i \leq n\text{-}1$, $1 \leq j \leq n\text{-}1$), como lo muestra la Figura 2.12-b). Como último paso en la distribución de los datos de la matriz A, los procesadores en cada plano con los datos de A, P_{ijj} ($0 \leq i, j \leq n\text{-}1$), hacen un broadcast por filas de procesadores tal como lo muestra la Figura 2.12-c) y de esta manera se logra que en cada P_{ijk} se tenga el elemento a_{ik}. La forma de distribuir los datos de la matriz B es análoga pero en vez de distribuir por columnas se hace por filas. Una vez que todos los procesadores tienen los datos, pueden ejecutar la multiplicación y luego se suman los resultados en los procesadores $P_{i,j,0} \ldots P_{i,j,n-1}$ tal como se mencionó previamente.

Las características sobresalientes de esta forma de calcular la multiplicación de matrices se pueden resumir en:

- Puede multiplicar matrices de orden n en $O(\log(n))$ pasos.
- Utiliza (hasta) n^3 procesadores, aunque está explicado para elementos de matrices se puede aplicar a submatrices o bloques.
- Un mismo dato de cada matriz A se utiliza y por lo tanto se copia en toda una fila de procesadores, tal como lo muestra la Figura 2.12-c), y de forma análoga un mismo dato de la matriz B se copia en toda una columna de procesadores, y por lo tanto se tienen numerosos datos replicados.
- Está claramente orientado a computadoras paralelas con procesadores interconectados como si estuvieran en una estructura tridimensional, y de hecho esto produce la reducción en la cantidad de cálculos a $O(\log(n))$ y a su vez la replicación de datos en varios procesadores.

Estas características son similares a las que tiene el método de multiplicación de matrices sobre una malla de árboles de procesadores [82] [98]. De hecho, este método también sigue la secuencia:

- Asignación inicial de los n^2 elementos de cada matriz a n^2 procesadores o elementos de procesamiento.
- Comunicación-Replicación de los elementos de las matrices para que cada procesador tenga los datos necesarios para hacer las multiplicaciones $a_{ik} \times b_{kj}$.
- Suma de los resultados de las multiplicaciones para obtener los elementos de la matriz C.

2.5 Resumen del Capítulo

Los temas más importantes que se han introducido y/o explicado en este capítulo, además de la definición misma de la multiplicación de matrices y la cantidad de operaciones necesarias para su resolución, son:

- La identificación del contexto de las aplicaciones de álgebra lineal, dada por la biblioteca LAPACK, las operaciones básicas incluidas en BLAS y, aún más específicamente, la relación entre la multiplicación de matrices. No solamente la multiplicación de matrices es similar al resto de las operaciones de L3 BLAS en cuanto a requerimientos de cómputo y memoria, sino que está demostrado que todas las operaciones de L3 BLAS se puede implementar en utilizando la multiplicación de matrices.
- La utilización de la multiplicación de matrices como benchmark. Si bien es muy acotada su validez en general, sí se puede afirmar que es un buen benchmark de todo L3 BLAS. Además, es utilizada para mostrar la calidad de optimización que se puede lograr cuando se relaciona el rendimiento obtenido con el máximo teórico del hardware utilizado.
- Se describieron los algoritmos paralelos para llevar a acabo la multiplicación de matrices en multiprocesadores. En general son sencillos en cuanto a descripción como en cuanto a su análisis teórico de rendimiento.

- Se describieron también los algoritmos paralelos para multiplicación de matrices en multicomputadoras. Normalmente se puede identificar claramente la relación de cada algoritmo con la interconexión de procesadores *subyacente* (en la cual el algoritmo obtiene su mejor rendimiento). También se describieron las adaptaciones del algoritmo de Fox en particular a las redes de interconexión de procesadores estáticas y bidimensionales (grillas y toros), que es el que se implementa en la biblioteca ScaLAPACK.
- Todos los algoritmos paralelos comparten características comunes relativamente importantes a nivel conceptual:
    - Adoptan el modelo de procesamiento SPMD.
    - Asumen que los nodos de procesamiento son homogéneos y *aprovechan* esta homogeneidad para obtener balance de carga.
    - Utilizan el procesamiento por bloques para optimizar la jerarquía de memoria de las computadoras, específicamente orientados a la optimización en el acceso a los datos en memoria/s cache/s.

Se necesita conocer cada uno de estos puntos para, una vez que se haga la descripción del hardware de procesamiento paralelo que proveen las redes locales, como mínimo se pueda analizar si los algoritmos de cómputo paralelo para multiplicar matrices son apropiados o no.

Capítulo 3: Clusters Heterogéneos

Dado que se ha hecho una descripción de los principales métodos para multiplicar matrices, ya es posible poner toda la atención sobre las redes de computadoras instaladas que se pueden utilizar para cómputo paralelo. En este sentido, y tal como se puntualiza en el capítulo anterior, es necesario conocer con bastante nivel de detalle la arquitectura de cómputo subyacente para, como mínimo, analizar los métodos que han sido propuestos.

Inicialmente se describirán las características principales de las redes de computadoras instaladas para ser utilizadas como máquinas paralelas. Dado que las redes de computadoras no han sido concebidas inicialmente para realizar cómputo paralelo (al menos en el contexto de las aplicaciones científico-numéricas), se deben identificar con la mayor claridad posible las capacidades en cuanto a: cómputo, interconexión de los procesadores, sincronización y escalabilidad.

Con las características de las redes de computadoras bien definidas, es posible analizar los métodos paralelos de multiplicación de matrices y de acuerdo a este análisis decidir si es posible que su implementación en clusters obtenga rendimiento aceptable. En este análisis se encontrará que los métodos de cálculo paralelo de la multiplicación de matrices que han sido propuestos no son adecuados para los clusters heterogéneos.

A partir del análisis anterior, se propone un nuevo algoritmo de multiplicación de matrices en paralelo, mostrando sus características principales en cuanto a balance de carga de los cálculos que debe realizar cada estación de trabajo, balance de carga de la red de interconexión y requerimientos de memoria que impone el algoritmo. Una vez definido este algoritmo *inicial*, se proponen cambios en la forma de llevar a cabo cómputo y comunicaciones a fin de aprovechar la capacidad (si la hay) de solapar (o hacer en *background*) cómputo local con comunicaciones. También se analizan las formas de aplicar los mismos conceptos de paralelización de la multiplicación de matrices en clusters heterogéneos a otros problemas, aunque es claro que es muy difícil hacerlo en general.

3.1 Características de los Clusters

Como se puntualiza en el primer capítulo, las redes de computadoras constituyen la plataforma de cómputo paralelo más ventajosa en cuanto a la relación costo/rendimiento. Y esta relación costo/rendimiento es aún mejor en el caso de utilizar las redes de computadoras que ya están instaladas. En este sentido, y para la descripción/discusión que sigue, se consideran redes de computadoras instaladas a todas las redes locales, independientemente de que las computadoras interconectadas sean PCs, estaciones de trabajo o computadoras con procesamiento simétrico (SMP: Symmetric MultiProcessing) [71]. De hecho, en todo este capítulo

- NOW (Network of Workstations),
- redes locales,
- redes de computadoras,
- redes de estaciones de trabajo,
- redes de PCs,
- clusters,

son utilizados como sinónimos.

Es importante identificar las características propias de las redes locales de computadoras que son utilizadas para cómputo paralelo ya que en general los algoritmos propuestos para resolver la mayoría (sino todas) las aplicaciones en paralelo, incluyendo la operación de multiplicación de matrices, deben ser evaluados en este contexto. Si bien es cierto que la mayoría (sino todos) los algoritmos paralelos propuestos pueden ser implementados sobre redes locales de computadoras con mayor o menor grado de dificultad, también es cierto que el rendimiento puede ser *muy* diferente. Dado que en general la paralelización de las operaciones que se han propuesto es *orientada* a un tipo de arquitectura, la implementación de estos algoritmos paralelos debe ser analizada teniendo en cuenta las características propias de las redes locales de computadoras a utilizar como plataforma de cálculo paralelo.

En las subsecciones que siguen, se identifican las características según se consideren como propias de las redes de computadoras homogéneas o propias de las redes de computadoras heterogéneas. Inicialmente, también se describe como una característica de la red local de computadoras utilizada para cómputo paralelo la propia red física de interconexión de máquinas, que de hecho es la que se utiliza para el pasaje de mensajes entre los distintos procesadores.

Toda la caracterización de las redes locales que se hace tiene como referencia las computadoras paralelas tradicionales, más específicamente las multicomputadoras. Este punto de referencia tiene importancia desde dos puntos de vista:

- Todo el conocimiento y la experiencia adquirida en cuanto a diseño de hardware de procesamiento y de interconexión. En este sentido, se tiene un punto de referencia sólido con respecto al cual es más sencillo describir características y es posible comparar e identificar similitudes y diferencias.
- Paralelización de aplicaciones y rendimiento obtenido. Como se puntualiza anteriormente, los algoritmos que tienen rendimiento razonable en una máquina paralela normalmente están orientados a la arquitectura de la misma máquina paralela. Por lo

tanto, al identificar similitudes y diferencias con respecto a las máquinas paralelas tradicionales se tiende a simplificar el análisis de los algoritmos paralelos propuestos en general y en particular las formas de procesar en paralelo el trabajo necesario para multiplicar matrices.

3.1.1 Características de la Red de Interconexión de Procesadores

Gran parte del esfuerzo de las máquinas paralelas tradicionales ha sido dedicado a las redes de interconexión de procesadores. De hecho, se considera que la red de interconexión de procesadores y en particular su rendimiento es elemental en las computadoras paralelas [69] [71] [87].

Para las computadoras paralelas con (o basadas en) memoria física distribuida, la red de interconexión se puede identificar claramente como la que provee comunicación entre los procesadores. Expresado de otra manera, 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. En el caso de las computadoras paralelas con memoria físicamente compartida, quitar la red de interconexión de los procesadores con la (*única*) memoria elimina completamente la posibilidad de ejecutar aplicaciones sobre el hardware que queda. Dado que las redes de computadoras utilizadas para cómputo paralelo claramente son computadoras 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. 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.

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 utilice, 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.

Como se mencionó antes, la red de interconexión de los procesadores de una computadora paralela construida a partir de una red local es la misma red. 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 [73] [109] [108]. Este estándar es conocido inicialmente 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 son muy bien definidas y conocidas en términos de hardware y de lo que tiene relación directa con 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, también llamada *Ethernet Rápida* (*Fast Ethernet*), donde se cambian solamente los parámetros/índices referidos a rendimiento de las comunicaciones. Esta similitud está ejemplificada en, y también aprovechada por, muchas de las empresas de hardware de comunicaciones que se encargan de diseñar y construir placas de interfase de comunicaciones (NIC: Network Interface Card) con ambas capacidades de transmisión de datos y denominadas placas de red de 10/100 Mb/s. Además, y siempre dentro de la norma 802.3 se ha definido también Gigabit Ethernet con capacidad de transmisión de 10^9 bits por segundo [105] [76] y se está considerando también la definición del estándar para 10^{10} bits por segundo o 10-Gbps [110].

La Figura 3.1 muestra esquemáticamente la forma lógica básica en que se conectan las computadoras 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).

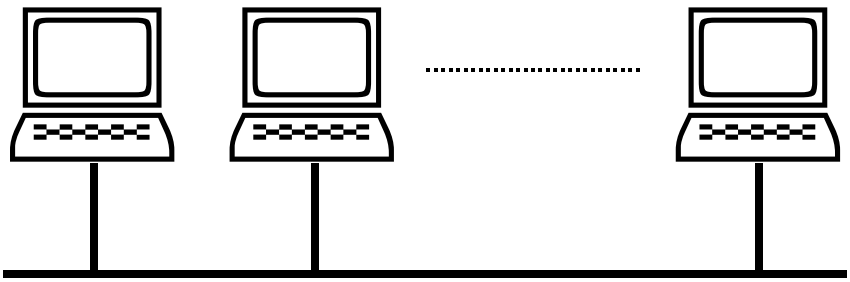


Figura 3.1: Red Ethernet.

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 anteúltima 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. 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 3.1.

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* que son básicamente concentradores y repetidores de comunicaciones. La Figura 3.2 muestra que desde el punto de vista del cableado la red tiene topología estrella, pero dado que los hubs distribuyen una misma señal en todos los cables la interconexión lógica sigue siendo la de un bus.

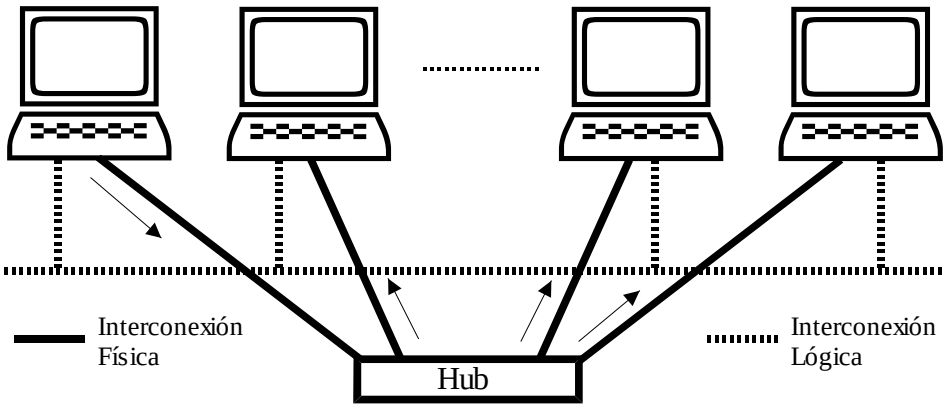


Figura 3.2: Red Ethernet con Hub.

En la Figura 3.2 también se muestra que la computadora ubicada a la izquierda envía un dato y todas las demás están en condiciones de recibirlo simultáneamente, a menos que se produzca una colisión. En este sentido, cada hub no solamente es un repetidor de las señales que llegan por cada cable sino que además es un *repetidor broadcast*, es decir que la señal que llega por un cable es repetida hacia *todos* los demás.

La utilización de *switches* en vez de hubs en las redes Ethernet se considera un gran avance, dado que además de tener todas las características de los hubs, los switches tienen la capacidad de aislar las comunicaciones que son punto a punto [109] [108] [138]. 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 y por lo tanto varias transmisiones de datos punto a punto se pueden realizar de manera simultánea. De todas maneras, por estar definido en el estándar Ethernet, se conservan todas las operaciones definidas por Ethernet, como el broadcast a nivel físico y en ese caso el switch se comporta como un hub.

La Figura 3.3 muestra una red Ethernet donde se llevan a cabo dos comunicaciones punto a punto simultáneas que son posibles por la actividad del switch. Las ventajas que proporcionan los switches sobre los hubs son evidentes, pero también se debe tener en cuenta que el costo de los switches es sustancialmente mayor que el de los hubs a pesar de que se están extendiendo rápidamente en la instalación de las nuevas redes locales. Por otro lado, la utilización de switches se considera esencial en la instalación de redes de computadoras destinadas directamente a cómputo paralelo, del tipo Beowulf [19] [103] [111].

Las redes Ethernet con cableado orientado al uso de hubs 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. Todo esto necesariamente reduce los costos de instalación y de mantenimiento de las redes Ethernet.

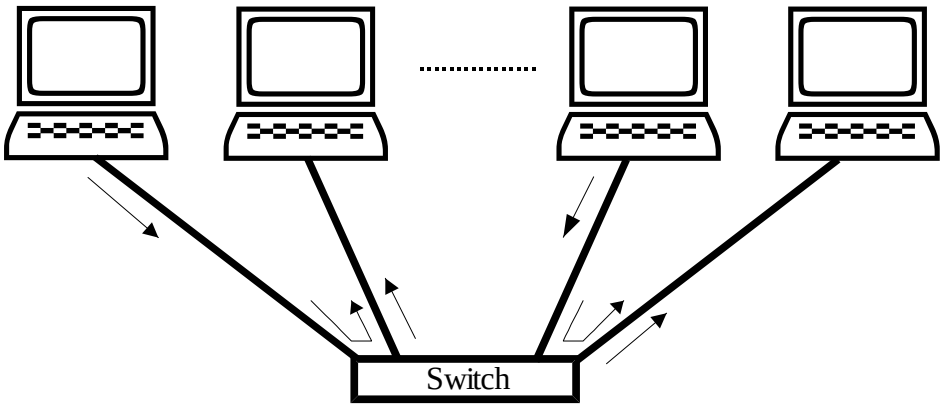


Figura 3.3: Red Ethernet con Switch.

La reducción de costo que representan las redes basadas en hubs con respecto a las demás alternativas de interconexión de computadoras implica una gran inercia en el mantenimiento de las redes Ethernet instaladas así como también en la instalación de nuevas redes con este hardware.

Desde el punto de vista de las aplicaciones que hacen uso de la red de interconexión, de forma tanto o más homogénea que la utilización de Ethernet a nivel físico es la utilización de IP (Internet Protocol) y los que lo utilizan, como TCP (Transmission Control Protocol) o UDP (User Datagram Protocol) [89] [97] [112] [32]. Es así que tanto a nivel físico como en los protocolos de utilización de la red de interconexión, la gran mayoría de las redes locales son homogéneas y con características comunes y bien conocidas.

3.1.2 Cluster Homogéneo como Máquina Paralela

En realidad es difícil encontrar una red local con todas las computadoras interconectadas iguales, a menos que la red local se haya instalado *recientemente*. En la mayoría de los casos recientemente significa no más de algunos meses, aunque depende de la organización en la que se utilice y la finalidad de la misma red.

Aún así es útil considerar el caso homogéneo a los efectos de identificar/describir las características de los clusters desde el punto de vista de cómputo paralelo ya que:

- Permite una separación más clara de las características, mejorando por lo tanto su análisis.
- Todas (o la *mayoría de*) las características que se identifiquen para el caso homogéneo serán compartidas por los clusters heterogéneos. Puesto de otra forma, las redes de computadoras tienen características que en sí mismas se deben tener en cuenta a la hora

de la paralelización de aplicaciones independientemente de que las máquinas sean homogéneas o no. Desde este punto de vista, las redes de computadoras heterogéneas agregan otras características que también deben ser tenidas en cuenta para la paralelización de aplicaciones.
- Sistemas como los denominados Beowulf, o simplemente las redes locales instaladas para cómputo paralelo son usualmente homogéneas. En este sentido, las características que se identifican como importantes en un ambiente homogéneo se deben tener en cuenta no solamente en las redes ya instaladas que son homogéneas sino también en todos estos sistemas de cómputo paralelo basados en redes locales de computadoras.

Resumiendo, las principales características técnicas de las redes de computadoras homogéneas que se deben tener en cuenta para el procesamiento paralelo son:
- Acoplamiento débil.
- Bajo rendimiento de la red de interconexión de procesadores.

Acoplamiento. Las redes locales de computadoras se construyen a partir de computadoras completas interconectadas con el fin de compartir algún tipo de recurso. Los recursos clásicos que se han compartido son espacio de almacenamiento, datos almacenados e impresora/s. Por lo tanto, el hardware de las distintas computadoras en general no tiene ninguna relación. De hecho, la única condición para que una computadora esté conectada en una red local es la instalación de una placa de interfase de comunicaciones (NIC) y no mucho más que las rutinas de software dedicadas a manejarla. Por lo tanto, desde el punto de vista de la arquitectura de cómputo de cada computadora, implica agregar un dispositivo más de entrada/salida de datos. Ahora volviendo a la visión de la red local como una máquina paralela esto implica que:
- La memoria física de las computadoras en la red local está totalmente distribuida y no hay ningún medio de hardware que facilite compartirla.
- El procesamiento en una red local es totalmente asincrónico, y no hay ningún medio de hardware que facilite la sincronización.

La única forma de hacer que un procesador lea o escriba una posición de memoria en otra máquina es utilizando apropiadamente la red de comunicaciones. De la misma manera, la única forma de sincronización entre los procesadores de cada computadora de la red local es utilizando apropiadamente la red de comunicaciones. Es claro que ambas cosas son posibles, pero también es claro que las redes locales no han sido diseñadas con ninguno de estos dos propósitos y por lo tanto la penalización en rendimiento para lograr la sincronización entre procesadores y/o memoria compartida puede ser muy grande en términos de rendimiento.

Descartar el uso de memoria compartida por la penalización en rendimiento implica necesariamente descartar los algoritmos diseñados para los multiprocesadores, que son computadoras paralelas con memoria compartida. En el caso de los algoritmos numéricos en general y de los algoritmos de multiplicación de matrices en particular, implica en muchos casos descartar algoritmos que han probado ser muy efectivos en cuanto a obtención de rendimiento satisfactorio en multiprocesadores.

El problema de sincronización de los procesadores no parece ser tan problemático como el de la memoria distribuida. En principio, la necesidad de sincronización no es tan fuerte en

la mayoría de los algoritmos, al menos no se proponen algoritmos en función de que se utilicen máquinas con procesadores sincronizados o no. Por otro lado, el hecho de que los procesadores no estén sincronizados por hardware no implica que sea imposible sincronizarlos y por lo tanto si la sincronización es poco frecuente el impacto en el rendimiento no será relevante. Normalmente la frecuencia de sincronización entre los procesadores que no implica pérdida de rendimiento está dada de manera directa por los índices de rendimiento de la red de interconexión, específicamente por el tiempo de inicialización (latencia o *startup*) de comunicaciones.

Rendimiento de la red de interconexión de procesadores. El rendimiento de una red de interconexión está directamente relacionado 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 la idea de *ancho de banda* (tasa de transferencia) o ancho de banda asintótico, dado por la cantidad de datos por unidad de tiempo que se pueden transferir, otro de los índices de rendimiento importantes es el tiempo mínimo de comunicación entre dos procesadores. Este tiempo mínimo es básicamente el tiempo de inicialización de las comunicaciones (*startup*), o también denominado por algunos autores [146] [25] como *latencia* de comunicación entre procesadores. La forma clásica de cálculo del tiempo comunicaciones de una red de interconexión tiene en cuenta tanto la latencia como el ancho de banda asintótico de acuerdo con la siguiente ecuación [69] [68]

$$t(n) = \alpha + \beta n \tag{3.1}$$

donde

- n es la unidad de información que se transfiere (bit, byte, representación de un número en punto flotante con precisión simple, etc.).
- α es el tiempo de latencia (*startup*) de la comunicación.
- β es el valor inverso del ancho de banda asintótico de la red de comunicaciones, es decir que $1/\beta$ es el ancho de banda asintótico.

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, se ubican varios órdenes de magnitud por debajo de las demás 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.

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 general, utilizando 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 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.

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. Esta capacidad de solapamiento de cómputo con comunicación es otra característica que se ha intentado aprovechar en el diseño de las redes de interconexión de las máquinas paralelas tradicionales.

Desde el punto de vista del rendimiento, 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 llevar a cabo el procesamiento de forma local (en P_1), ahorrando tiempo y/o complejidad de la aplicación.

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.

3.1.3 Cluster Heterogéneo como Máquina Paralela

Las diferencias de velocidad de cómputo relativa de las computadoras en los clusters heterogéneos son lo más importante que se agrega a lo que ya se ha identificado en cuanto a la red de interconexión y a las redes locales con computadoras homogéneas. Por otro lado, la posibilidad de red de interconexión heterogénea es realmente muy poco probable en las redes locales instaladas y por lo tanto se descarta.

En general, las redes locales instaladas pueden ser ampliamente heterogéneas en cuanto al hardware de procesamiento o las computadoras interconectadas en la red local. En las redes locales con un mínimo tiempo de existencia (y *evolución*), se pueden encontrar múltiples modelos de computadoras, quizás algunas computadoras paralelas y/o computadoras con multiprocesamiento simétrico (SMP: Symmetric MultiProcessing) y múltiples modelos de PCs. En el caso de las PCs, también es posible que máquinas con el mismo procesador y operando con la misma frecuencia de reloj tengan distinta capacidad de procesamiento dependiendo de las diferencias que haya, por ejemplo, en velocidad de acceso a memoria, capacidad de memoria cache externa y frecuencia de operación del bus del sistema. Las

diferencias entre las computadoras conectadas en una red local normalmente están relacionadas con:

- Tiempo de existencia de la red local, con su consiguiente impacto en la reposición y/o actualización de las computadoras. En el caso de la reposición, se tiende a mantener un mínimo de computadoras que cada usuario dispone y a medida que transcurre el tiempo el hardware deja de ser fabricado y es reemplazado por otro (normalmente más veloz). En el caso de la actualización, se tiende a incorporar el mejor hardware de procesamiento a un mismo costo, como suele ser el caso de los componentes básicos como procesadores, memoria principal y discos.
- Evolución en cuanto a requerimientos. Las razones y condiciones por las cuales se produjo la instalación de una red local no necesariamente se mantienen invariantes. El cambio de las tareas o la incorporación de nuevas tareas que se llevan a cabo en una red local generalmente implican la incorporación de nuevas computadoras, quizás específicas para las tareas a desarrollar en el ambiente en el que está instalada la red local.

Para obtener el máximo rendimiento posible en una red heterogénea necesariamente se debe llevar a cabo un balance de carga de procesamiento adecuado. Si se parte de asumir que todas las computadoras tendrán que realizar la misma cantidad de procesamiento, se llegará a que la utilización de un mayor número de computadoras implica peor rendimiento. De hecho, la máquina paralela completa funciona en términos del tiempo de procesamiento de la computadora más lenta que se utiliza porque será la última en completar los cálculos asignados y por lo tanto el procesamiento completo no se terminará sino hasta que la *peor* (en términos de rendimiento de procesamiento) termine.

La idea básica para balancear la carga de procesamiento de las computadoras en una red local es simple en cuanto a definición aunque no necesariamente simple en general para su implementación: si una computadora, ws_1, es *n* veces más veloz para llevar a cabo el mismo procesamiento que otra, ws_2, entonces ws_1 debe recibir una tarea *n* veces más compleja en términos de cómputo que ws_2. Esta idea básica entonces, significa *nada más* que tener en cuenta las velocidades relativas entre las computadoras para la asignación de tarea/s de cómputo.

Se podría agregar que otra de las características a mencionar en las redes heterogéneas es el de la diferencia de capacidad de almacenamiento en memoria principal. Sin embargo, se debe aclarar que en la mayoría de las redes locales instaladas esta diferencia no es muy amplia ni es proporcional a la diferencia entre velocidades relativas. Haciendo referencia al ejemplo anterior, es muy difícil encontrar que si una computadora ws_1 es *n* veces más rápida que otra ws_2 el tamaño de memoria de ws_1 es *n* veces mayor que ws_2. De hecho, es bastante frecuente encontrar tamaños de memoria bastante similares en computadoras con velocidades relativas muy diferentes. En el caso extremo de encontrar que la diferencia entre los tamaños de memoria es muy grande, se puede recurrir a calcular las velocidades relativas entre las computadoras en función de la utilización de memoria swap de forma tal que se tenga en cuenta que las computadoras con menor cantidad de memoria principal disponible llevan a cabo los cálculos haciendo uso de memoria swap y por lo tanto esto se verá directamente reflejado en su velocidad de cómputo.

En un contexto más general de procesamiento heterogéneo las variaciones son también

mayores. Reportes como [22] por un lado identifican los problemas que pueden causar a nivel de estabilidad numérica o convergencia de algoritmos la gran variación en cuanto a representación de los datos numéricos (básicamente de punto flotante). Otros, como [41] por ser más cercanos al aporte teórico llegan a niveles de complejidad mayores aún en cuanto a estimación de eficiencia. Desde el punto de vista de las redes locales de computadoras se puede considerar que estos inconvenientes no son muy probables. En el caso específico de la representación de datos, la gran mayoría de (sino *todas*) las computadoras interconectadas en redes locales utilizan procesadores estándares que a su vez se adhieren explícitamente a los estándares de representación definidos por IEEE, dado por el ANSI/IEEE 754-1984 [72] para números en punto flotante.

3.2 Cómputo Paralelo en Clusters Heterogéneos

Las características enumeradas de las redes locales necesariamente deben ser tenidas en cuenta para la paralelización de las tareas de cómputo que se resuelvan sobre esta plataforma de hardware de procesamiento. Resumiendo de la sección anterior, las características de una red local vista como una máquina paralela son:

- Computadora de memoria distribuida, multicomputadora débilmente acoplada.
- Red de interconexión de procesadores Ethernet.
- Bajo rendimiento de la red de interconexión.
- Heterogeneidad de procesamiento, traducida en distintas velocidades relativas.

El impacto de cada una de estas características en la paralelización de tareas a resolver en redes locales de computadoras instaladas se verá en cada una de las subsecciones que siguen. Es conveniente recordar que la paralelización que se haga sobre las redes locales o, lo que es equivalente, los algoritmos paralelos que se ejecuten sobre las redes locales, tendrá/n impacto directo sobre el rendimiento obtenido. Por lo tanto, se deben favorecer (tal como se ha hecho tradicionalmente en el ámbito de cómputo de alto rendimiento) los algoritmos paralelos que tengan en cuenta la arquitectura de cómputo subyacente. Los algoritmos numéricos en general no escapan a este principio, ni los pertenecientes al álgebra lineal, ni la multiplicación de matrices en particular.

3.2.1 Multicomputadora Débilmente Acoplada

El modelo de programación que normalmente se impone en las multicomputadoras débilmente acopladas y aún en las multicomputadoras en general es el de pasaje de mensajes [3] [52] que a su vez se deriva de CSP (Communicating Sequential Processes) [66]. Este modelo de pasaje de mensajes implica que el programa paralelo será un conjunto de procesos secuenciales que se comunican y/o sincronizan, donde normalmente la sincronización en este contexto implica alguna forma directa o indirecta de comunicación entre procesos.

Puede considerarse que la característica de hardware mencionada anteriormente en lo referente a memoria físicamente distribuida tiene relación directa con el modelo de pasaje

de mensajes. Dado que la memoria está físicamente distribuida, es muy difícil compartir memoria entre procesos asignados a (ejecutándose en) diferentes procesadores sin una penalización relativamente alta en cuanto a rendimiento.

Tradicionalmente se ha asumido y en muchos casos experimentado que por un lado es cierto que en última instancia cualquier algoritmo puede ser implementado sobre una máquina paralela de memoria distribuida débilmente acoplada con suficiente esfuerzo (normalmente a nivel de capas intermedias de software). Pero por otro lado, también se ha experimentado que también es cierto que la probabilidad de obtener rendimiento aceptable u óptimo en cuanto a rendimiento es mayor cuando se adopta el modelo de programación de pasaje de mensajes en este ambiente de hardware. De hecho, el caso específico de la gran cantidad de algoritmos propuestos que coexisten para la multiplicación de matrices no hace más que corroborar esta realidad. Puesto de otra manera, dado un algoritmo paralelo basado en memoria compartida por ejemplo, es altamente probable que se pueda diseñar e implementar un algoritmo (que resuelve la misma tarea) basado en pasaje de mensajes que obtenga igual o mejor rendimiento. A priori, es aún más probable que dado un algoritmo basado en pasaje de mensajes no pueda ser superado en rendimiento por otro algoritmo basado en memoria compartida que resuelve la misma tarea.

El área específica de las operaciones matriciales en general, las provenientes del álgebra lineal en particular y específicamente la multiplicación de matrices no escapa a la *regla* mencionada anteriormente. Por lo tanto, en el ambiente específico de procesamiento paralelo que proporcionan las redes locales de computadoras los algoritmos a considerar (como *mejores*) serán los que puedan expresarse en términos de pasaje de mensajes sin ningún tipo de adaptación o traducción intermedia que implique sobrecarga de procesamiento sobre la básica de las mismas operaciones a resolver.

3.2.2 Red Ethernet para Interconexión de Procesadores

Como se explica previamente, las redes Ethernet tienen una gran variación en cuanto a rendimiento (10 o 100 Mb/s, por ejemplo) y cableado, o lo que podría considerarse como *topología* de hardware (utilización de hubs y/o switches, por ejemplo). Sin embargo, por la definición misma de estándar definido para la interconexión de máquinas [73] en todos los casos se mantiene la capacidad de broadcast físico de los mensajes. Es decir que, independientemente del rendimiento y del cableado de las redes locales Ethernet, siempre se podrá enviar desde una computadora y recibir el mismo mensaje en todas las demás computadoras de la misma red.

A nivel de los protocolos (o *pila* o *serie* de protocolos [112] [32]) que se utilizan sobre las redes Ethernet, la capacidad de comunicaciones del tipo broadcast se mantiene al menos en los protocolos más cercanos al hardware como IP, IGMP y UDP [112] [34] [96] [95]. La capacidad de llevar a cabo broadcast *físico* es muy relevante en las redes Ethernet, al menos por dos razones:

- Desde el punto de vista de escalabilidad: en relación con la utilización del medio físico de comunicación (cableado), el broadcast físico es *independiente* de la cantidad de receptores. Es decir que el tiempo de comunicaciones tiende a ser el mismo independientemente de la cantidad de computadoras que intervienen en una

transferencia de datos (básicamente *reciben* datos). No se puede asegurar que el tiempo total de un broadcast sea *exactamente* el mismo para cualquier cantidad de receptores porque el tiempo total depende de otros factores como:

- Método de sincronización de receptores, para que un mismo dato sea simultáneamente recibido por todos.
- Método de reconocimiento de recepción (*acknowledge*) de datos o método/s para asegurar que los datos enviados han sido recibidos en todas las computadoras.
- Tasa de pérdida en cuanto a recepción de datos de cada computadora, que depende a su vez de la calidad de la placa de red que cada computadora utiliza y del cableado mismo.
- Disponibilidad de *buffers* de recepción y envío de datos.

Sin embargo, se considera que cada uno de estos factores son relativamente menos importantes que la transmisión de los datos misma (implican la suma de tiempos uno o varios órdenes de magnitud inferiores respecto del tiempo de los datos transferidos).

- Desde el punto de vista del rendimiento: tal como es generalmente aceptado, la utilización de broadcast en los programas paralelos es muy extendida [146]. Si los mensajes broadcast entre procesos no se implementan utilizando el broadcast físico de las redes Ethernet, se transforman en múltiples transferencias punto a punto (que comunican *siempre* los *mismos* datos). Además, es claro que toda comunicación de datos entre computadoras implica la utilización del cableado. Si en el cableado no se utilizan switches, se tiene que cada comunicación implica la utilización de todo el medio de comunicaciones (Figura 3.1 y Figura 3.2) y por lo tanto este medio de comunicaciones debe ser multiplexado entre las distintas transferencias de datos. Expresado de otra manera, en todas las redes locales en las que no se utilizan switches de comunicación, todas las transferencias de datos se secuencializan y de hecho cada comunicación punto a punto entre procesos ocupa todo el medio de comunicaciones. De esta manera se llega a que el tiempo de comunicaciones de los mensajes broadcast se multiplica por la cantidad de receptores. Por otro lado, aún en el caso en que se utilizan switches en el cableado de las redes locales, la posibilidad de asignar más de un proceso a una computadora se debe tener en cuenta para no desaprovechar la capacidad de realizar todas las comunicaciones punto a punto de manera simultánea. Desde otro punto de vista, dado que Ethernet por definición tiene broadcast independientemente del cableado, siempre que la implementación de los mensajes broadcast lo utilice no se tendrá *ninguna* penalización en cuanto al tiempo de transferencia de datos entre computadoras.

Por otro lado, y también desde el punto de vista del rendimiento, la utilización de mensajes punto a punto entre procesos tiende a que se tenga en total mayor tiempo de comunicaciones. Dado que, como se ha mencionado, cuando se tiene cableado sin switches necesariamente se tendrá que el tiempo de transferencia de datos aumenta de manera proporcional a la cantidad de mensajes punto a punto que se llevan a cabo. Esta situación se mantiene en el caso en que se utilizan switches en el cablead pero no se maneja adecuadamente la situación en la que más de un proceso es asignado a una computadora y los mensajes se multiplexan en el canal de comunicaciones.

Por lo tanto, en el ambiente específico de procesamiento paralelo que proporcionan las redes locales de computadoras los algoritmos a considerar (como *mejores*) serán los que puedan expresarse en términos de mensajes del tipo broadcast. Estos mensajes pueden ser

implementados de manera directa en las redes Ethernet y de esa manera:
- Se evitan los costos de algoritmos de broadcast utilizando mensajes punto a punto.
- Se tiene mejor escalabilidad al menos en cuanto a comunicaciones entre procesos.
- Se evita la penalización por el uso secuencial del medio de comunicaciones en las redes locales que no tienen cableado con switches.

Tomando como referencia la construcción de máquinas paralelas de bajo costo a partir de computadoras interconectadas con Ethernet, en el caso extremo se podría reducir aún más el costo, dado que los switches que se utilizan actualmente (y que son considerados *esenciales*) podrían ser reemplazados por hubs sin penalización en cuanto a rendimiento. Sin embargo, este extremo implica que *todos* los programas a ejecutar están expresados en términos de mensajes broadcast. Sin embargo, esta puede no ser una buena decisión en el contexto de las redes locales instaladas, donde ya está/n instalado/s los switches en particular.

3.2.3 Bajo Rendimiento de la Red de Interconexión

El rendimiento de las redes de locales de computadoras estándares y específicamente de las redes Ethernet instaladas es muchas veces considerado inaceptable para llevar a cabo cómputo paralelo [12]. De hecho, existen muchas propuestas de interconexión de computadoras de bajo costo (PCs o las estaciones de trabajo menos costosas) estándares que:
- Se interconectan con interfases de red de mayor rendimiento y más costosas [12] [91].
- Se interconectan con más de una interfase de red de bajo costo [65] [48] [78].
- Se interconectan con interfases de red específicamente diseñadas para cómputo paralelo. Unp de los primeros proyectos reportados en este sentido es el TTL_PAPERS: Purdue's Adapter for Parallel Execution and Rapid Synchronization [39] [40] [67] [38] [PAPERS].

Todas estas posibilidades y varias combinaciones posibles se pueden encontrar en [37] en el contexto de procesamiento paralelo utilizando PCs con sistema operativo Linux, pero en la mayoría de los casos pueden ser aplicadas a redes de computadoras en general.

Sin embargo, en todos estos casos se tiene como contrapartida el considerable aumento de costo dado que:
- Se utiliza hardware de interconexión más costoso o más hardware (más interfases de red). Siempre el hardware de interconexión con mayor rendimiento será más costoso y en muchos casos este costo se multiplica cuando el hardware no es de uso masivo como las redes Ethernet. En el caso de utilizar más placas de red, es claro que el costo de hardware de interconexión se multiplica por la cantidad placas que se agregan en cada computadora.
- Tanto el software de base como las herramientas de utilización se vuelven más complejas. En muchos casos el mismo kernel del sistema operativo debe ser modificado y en la mayoría de los casos se deben agregar capas intermedias de software para que las aplicaciones paralelas no sean afectadas o sean afectadas lo menos posible en cuanto a la interfase de utilización de las capacidades de comunicación. Todos estos agregados implican costo de diseño e implementación (que suele reducirse mucho por la utilización de bibliotecas de uso libre), y de mantenimiento, dado que cualquier cambio

en el hardware de red o en el kernel del sistema operativo o en las bibliotecas puede tener impacto directo en las aplicaciones paralelas que lo utilizan.
- Se descartan automáticamente todas las redes locales ya instaladas, que en general son de muy bajo costo y rendimiento relativo: Ethernet de 10 Mb/s.

Teniendo en cuenta estas referencias, es muy probable que la decisión de utilizar las redes de computadoras instaladas implica la revisión de los patrones y frecuencia de las comunicaciones de los programas paralelos a ejecutar. No hacerlo implica asumir que los programas paralelos son lo suficientemente "buenos" como para que la reducción de varios órdenes de magnitud en cuanto al rendimiento de las comunicaciones no implique *ningún* cambio o cambios menores en el rendimiento total.

El impacto del bajo rendimiento de las comunicaciones de las redes locales instaladas sobre el rendimiento total de los programas paralelos a ejecutar siempre depende del patrón de comunicaciones y especialmente de la frecuencia de las comunicaciones entre los procesos del programa paralelo. Es aquí donde se torna muy importante la granularidad de las aplicaciones y del mismo programa paralelo a ejecutar. Teniendo en cuenta el modelo de tiempo de comunicación de datos de cualquier red de interconexión, Ecuación (3.1), los dos aspectos a considerar son: latencia y ancho de banda asintótico.

La latencia de las redes locales de comunicaciones es quizás el aspecto que debe ser tomado con mayor atención desde el punto de vista del rendimiento por dos razones:
- Como se dijo previamente, este tiempo es varios órdenes de magnitud mayor que en las redes de interconexión de las computadoras paralelas *tradicionales*. Por lo tanto se debe tener a priori la tendencia de que la cantidad de datos de las transferencias entre los procesos de un programa paralelo debería ser varios órdenes de magnitud mayores. Esto evidentemente impone una restricción muy fuerte (tan *fuerte* como órdenes de magnitud mayores) sobre la paralelización que se realice o los programas paralelos que se ejecuten sobre las redes de computadoras. Este impacto es o debe ser directamente visible en la granularidad, ya que asumiendo que la cantidad de datos que se transfiere es la misma, aumentar el tamaño de los mensajes tiende a producir menor cantidad de mensajes con menor frecuencia y esto de hecho implica programas con granularidad mayor. Puesto de otra manera, la granularidad mínima de los programas se restringe dado que la latencia es muy grande y por lo tanto reducir la granularidad implica reducir el rendimiento de manera significativa.
- También como se puntualizó antes, la latencia de las comunicaciones suele depender o puede depender de las computadoras que intervienen en la transferencia de datos. Esto significa que no necesariamente se cumple que el tiempo de latencia para transferir datos de una computadora ws_i a otra ws_j es igual al tiempo de latencia de la transferencia de datos de ws_i a otra ws_k, $\forall$ *i, j, k*. Puesto de otra forma, es bastante probable que el tiempo de latencia a tener en cuenta en las redes de computadoras esté relacionado con el *mayor* tiempo de latencia entre todos los pares de computadoras interconectadas. Esta restricción no es independiente de la anterior sino que es complementaria porque ahora la latencia a tener en cuenta es la mayor de toda la red, que a su vez será igual o mayor de la definida por el estándar Ethernet.

En lo referente específicamente al bajo rendimiento del ancho de banda asintótico de la red, también afecta el rendimiento de los programas paralelos de manera significativa.

Nuevamente el impacto más fuerte en rendimiento lo tendrán los programas paralelos con granularidad más fina.

Suponiendo, para ejemplificar, que se utiliza una red Ethernet de 10 Mb/s y asumiendo que:
- el tiempo de latencia es cero,
- se transfieren los datos a la máxima capacidad de la red sin que haya ninguna sobrecarga (overhead) de empaquetado, protocolos, capas de software, etc.,

se pueden transferir datos a una tasa de 1.25 MB/s. Siguiendo con el ejemplo, y considerando el contexto mismo de la operación de multiplicación de matrices, son necesarias 1990000 operaciones de punto flotante (utilizando la cantidad de operaciones dada en la tercera ecuación del capítulo anterior), para resolver una multiplicación de matrices de 100×100 elementos. Dado que es hoy bastante común encontrar computadoras en una red local con capacidad de procesamiento de 300 Mflop/s o más, el tiempo aproximado para resolver la multiplicación de matrices de 100×100 elementos es poco menos de 0.0067 segundos (6.7 ms). ¡Y el tiempo para transferir una matriz de 100×100 elementos representados en punto flotante de precisión simple sobre la red es de 0.032 segundos o 32 ms! Por lo tanto, calcular el resultado localmente es 32/6.7 ≅ 4.7 veces mejor que recibir la respuesta desde otra computadora conectada por una red Ethernet de 10 Mb/s.

El ejemplo anterior tiene varias simplificaciones y varias alternativas de *solución* (como usar una red de 100 Mb/s), pero también da una idea de los órdenes de magnitud de tiempos en los cuales se opera y los peligros de asumir que los programas paralelos diseñados para máquinas paralelas no tendrán penalización en rendimiento. Muchas veces, este tipo de análisis ha llevado a asumir (prematuramente) que no es útil resolver en paralelo los problemas numéricos en redes de locales computadoras. Una vez más, para evitar el impacto del ancho de banda de la red de comunicaciones sobre el rendimiento se tiende a aumentar la granularidad. Pero muchas veces no es posible (o no alcanza con) disminuir la frecuencia y aumentar el tamaño de los mensajes como en el ejemplo comentado para resolver el problema del tiempo de latencia de los mensajes. De hecho, en el ejemplo se considera que el tiempo de latencia es cero.

El aumento de granularidad que tiene relación con mejorar el rendimiento de las aplicaciones paralelas utiliza una de las características que es bastante frecuente en los problemas numéricos. Normalmente los problemas numéricos se resuelven con una o más operaciones sobre datos estructurados en arreglos (multidimensionales, en general). También es común que el procesamiento involucrado es uno o más órdenes de magnitud mayor que la cantidad de datos a procesar. En el caso de la multiplicación de matrices ya se ha explicado que la cantidad de datos sobre la que se opera es $O(n^2)$ y la cantidad de operaciones necesarias es $O(n^3)$. Por lo tanto, al aumentar la cantidad de datos es relativamente sencillo aumentar la granularidad, dado que se deben realizar mayor cantidad de operaciones. Esto es equivalente a considerar en el ejemplo anterior matrices de 1000×1000 elementos con todo lo demás invariante:
- El tiempo de cálculo ahora es de aproximadamente 6.7 s.
- El tiempo de comunicaciones para recibir la matriz resultado es de 3.32 s.

De todas maneras, se debe recordar que aumentar el tamaño de los mensajes

necesariamente implica procesar mayor cantidad de datos o, lo que es lo mismo, aumentar el tamaño mínimo de los problemas u operaciones a resolver.

Resumiendo, los programas paralelos que tengan mejor rendimiento en las redes locales de computadoras serán los de mayor granularidad. En este sentido, las redes de computadoras no difieren de la mayoría (o *todas*) las demás plataformas de cómputo paralelo. Pero las redes de interconexión Ethernet implican un esfuerzo mucho mayor en el aumento de granularidad mínima de los programas paralelos y este esfuerzo es proporcional a la diferencia de latencia y ancho de banda que tienen las redes Ethernet con respecto a las redes de interconexión de procesadores en las computadoras paralelas tradicionales.

3.2.4 Heterogeneidad de procesamiento

Tal como se ha explicado previamente, la heterogeneidad en capacidad de procesamiento en una red local se traduce directamente en las distintas velocidades relativas de las computadoras interconectadas en redes locales. Avanzando en el nivel de detalle del problema de balance de carga o balance de carga proporcional a las velocidades relativas se deben resolver dos aspectos:

- Identificar con la mayor precisión posible las velocidades relativas de las computadoras a utilizar.
- Distribución de la carga de procesamiento.

Velocidad Relativa. Identificar con precisión las velocidades relativas de las computadoras no es sencillo en general. De hecho, es uno de los grandes problemas que intentan resolver los programas de *benchmark* y sobre los cuales sigue habiendo discusión. Afortunadamente, a medida que se avanza en la especificación del área de los problemas numéricos a resolver la situación suele encontrarse más *estable* o más posible de predecir en cuanto a velocidad de cálculo relativa. De hecho, en las aplicaciones numéricas y las del álgebra lineal en particular el procesamiento es sumamente regular y por lo tanto el cálculo de velocidades relativas entre diferentes computadoras también es más sencillo. En este ámbito, dos de los caminos a tomar son:

- Utilizar un tamaño reducido del problema a resolver para ejecutarlo en todas las computadoras y calcular las diferencias de velocidades en base al tiempo de procesamiento en cada computadora. Suponiendo que, por ejemplo, se debe calcular una FFT (Fast Fourier Transform), por ejemplo de 10^7, 10^8 o más datos, se puede hacer el cálculo de FFT con 10^5 o 10^6 datos en todas las computadoras y tomar las diferencias relativas directamente proporcionales al tiempo de cómputo de esta FFT “reducida”. Se debe ser bastante cuidadoso en cuanto al tamaño del problema dado que se pueden producir resultados muy erróneos [11] si los datos del problema “reducido” se pueden almacenar totalmente en memoria cache en algunas computadoras y en otras no. Por otro lado, si se toman conjuntos de datos demasiado grandes el tiempo de ejecución de este “mini-benchmark” sería muy alto.
- Utilizar la velocidad de cada computadora para la multiplicación de matrices como referencia en cuanto a velocidad de cálculo. Sería el caso anterior en particular para la multiplicación de matrices. Aunque como se aclaró en el capítulo anterior la multiplicación de matrices es particularmente apropiada para aprovechar todas las optimizaciones de código, es de esperar que las diferencias en el tipo de procesamiento

que cada operación o problema numérico impone tenga consecuencias similares de rendimiento en los procesadores de las computadoras que se utilizan. En el caso del problema anterior, implicaría tomar como referencia en cuanto a velocidad de procesamiento el tiempo de ejecución de una multiplicación de matrices de, por ejemplo, 1000×1000 elementos para hacer la distribución del procesamiento necesario para calcular la FFT de 10^7, 10^8 ó más datos. De esta manera se asume que si una computadora ws_i es dos veces más rápida que otra ws_j para hacer una multiplicación de matrices también esta diferencia de capacidad de procesamiento se mantiene para realizar el cálculo necesario para una FFT. De esta manera se ahorra el costo de ejecutar un problema "reducido" para cada aplicación aunque se agrega algo de imprecisión derivada de la diferencia de procesamiento que puede haber entre una multiplicación de matrices y cualquier otro problema numérico que se resuelve en una red de computadoras.

En ambos casos, es claro que existe un costo asociado en lo que se refiere al cálculo mismo de las velocidades relativas que no existe en las máquinas paralelas homogéneas en cuanto a la capacidad de cálculo de los elementos de procesamiento. Para el caso específico de las multiplicaciones de matrices no hay diferencias entre los dos caminos anteriores. Ampliando el espectro de aplicaciones a L3 BLAS, la segunda opción es sumamente apropiada dado que, de hecho, la operación básica en todas las definidas en L3 BLAS es la misma multiplicación de matrices. En el caso de ampliar aún más el espectro a las operaciones provenientes del álgebra lineal (a las operaciones definidas en LAPACK, por ejemplo), el tipo de cálculo de la multiplicación de matrices sigue siendo muy similar y, además, las operaciones que son importantes desde el punto de vista del rendimiento (y tiempo de cómputo necesario), siguen siendo las de L3 BLAS.

En el ámbito de las computadoras con elementos de procesamiento heterogéneos, la capacidad de cálculo de cada computadora o procesador suele utilizarse para el cálculo de la heterogeneidad de la máquina paralela [147]. De hecho, en general se recurre a alguna forma de "potencia de cálculo relativa de cada computadora ws_i" o $pw(ws_i)$. En este contexto, suele ser muy útil recurrir a la capacidad de cómputo de ws_i expresada en millones de operaciones de punto flotante por segundo o Mflop/s(ws_i) y utilizarla para el cálculo de $pw(ws_i)$

$$pw(ws_i) = \frac{Mflop/s(ws_i)}{\underset{j=0..P-1}{máx}\ (Mflop/s(ws_j))} \tag{3.2}$$

donde se tienen en cuenta P computadoras identificadas como ws_0, ..., ws_{P-1} y $pw(ws_i)$ se calcula en función de la más veloz.

Distribución de Carga. Una vez establecido el método con el cual calcular las velocidades relativas y las velocidades relativas específicas entre las computadoras de la red local a utilizar, se debe distribuir la carga de procesamiento de cada computadora.

Sobre la idea de cálculo de velocidad relativa de cada computadora dada en le Ecuación (3.2) se avanza para llegar a la "potencia de cálculo relativa normalizada de ws_i" o directamente pw_i

$$pw_i = \frac{Mflop/s(ws_i)}{\sum_{j=0}^{P-1}(Mflop/s(ws_j))} \tag{3.3}$$

de forma tal que se tiene

$$\sum_{j=0}^{P-1}(pw_j) = 1 \tag{3.4}$$

Y con esta métrica o índice se debe cumplir que cada computadora ws_i debe llevar a cabo pw_i (o, en porcentaje, $pw_i \times 100$ %), del total del trabajo a realizar.

En el caso específico de la multiplicación de matrices, esto es equivalente a establecer que la computadora ws_i realice todos los cálculos necesarios para $pw_i \times n^2$ de los elementos de la matriz resultado C, lo cual es equivalente a establecer que ws_i ejecute $pw_i \times (2n^3 - n^2)$ operaciones. En el caso más general de las aplicaciones de álgebra lineal y de los problemas numéricos, la identificación de la tarea a resolver en cada computadora ws_i quizás no sea tan claramente definible. Sin embargo, como mínimo en todos los problemas para los cuales se han diseñado e implementado métodos paralelos evidentemente ya se ha hecho algún tipo de división del trabajo total en partes (aunque normalmente estas partes han sido *iguales*).

3.3 Principios de Paralelización de Aplicaciones en Clusters

Habiendo hecho la descripción de los clusters en general, y de los clusters heterogéneos en particular desde el punto de vista de procesamiento paralelo, los dos principios de paralelización que se deberían seguir para la obtención de rendimiento paralelo optimizado son:

- 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. Esta distribución de los datos, además, facilita la utilización de mensajes broadcast dado que, por ejemplo, no hay definidas *filas* y *columnas* de procesadores como en las redes estáticas bidimensionales de interconexión de procesadores.

Se podría afirmar que estos principios son específicamente asociados a los clusters heterogéneos. En el caso específico de la multiplicación de matrices, ninguno de los algoritmos que se describieron en el capítulo anterior *sigue* o *respeta* estos tres principios de paralelización. De hecho, el algoritmo de Fox específicamente se ha modificado para

que los mensajes broadcast se puedan llevar a cabo de manera optimizada en redes bidimensionales de interconexión de procesadores.

Sin embargo, también se pueden seguir otros principios de paralelización derivados del área de aplicaciones de álgebra lineal y procesamiento numérico en general, tales como

- La adopción del modelo de procesamiento SPMD (Single Program - Muliple Data), que simplifica notablemente la programación, el depurado y la optimización de aplicaciones de cómputo paralelo.
- El aprovechamiento de la capacidad (si existe) de solapamiento de las comunicaciones con respecto al cómputo local de cada máquina. Aunque en el caso específicos de las redes locales no está asegurada esta capacidad, es posible intentar, al menos, reducir el tiempo de latencia de las comunicaciones llevando a cabo las comunicaciones en *background* con respecto al cómputo local en cada máquina.

3.4 Multiplicación de Matrices en Paralelo

Aunque ya se han dado algunas ideas en cuanto a la multiplicación de matrices sobre redes de computadoras heterogéneas, es necesario avanzar en cuanto al nivel de detalle y precisión del método por el cual se calcula en paralelo C = A×B. Este método ya ha sido explicado de forma resumida en [136] [131] [137].

De manera similar a la presentación tradicional de los algoritmos numéricos orientados a computadoras paralelas de memoria distribuida, se explica por un lado la distribución de los datos y por otro la secuencia de pasos a ejecutar en cada computadora (procesador) para llegar al resultado esperado. Como en todos los algoritmos orientados a computadoras paralelas de memoria distribuida, se descarta casi desde el inicio la posibilidad de replicación de los datos en todos los procesadores (en este caso, computadoras de la red local) dado que esto impone un requerimiento muy elevado de memoria en cada una de ellas.

3.4.1 Distribución de Datos

Asumiendo que cada una de las P computadoras a utilizar ws_i ($0 \leq i \leq P$-1), tiene calculada su "potencia de cálculo relativa normalizada", pw_i, que cumple con la Ecuación (3.3) y con la Ecuación (3.4), una de las formas de balancear la carga de procesamiento es establecer que cada estación de trabajo ws_i calcule pw_i del total de datos de la matriz C. En el contexto de la paralelización de cómputo sobre matrices, definir que una determinada parte de una matriz X, o submatriz $X^{(i)}$, sea calculada en una computadora ws_i es equivalente a asignar $X^{(i)}$ a ws_i. Es decir que los datos de la submatriz que se calcula en una computadora residen localmente en esa computadora.

Las formas más simples en que se determina la cantidad de datos a calcular de la matriz C en una computadora se muestran de manera esquemática en la Figura 3.4, donde:

- $C^{(i)}$ es la submatriz de C asignada a la computadora ws_i, o lo que es igual la parte de C

que reside y se calcula en ws_i.

- La Figura 3.4-a) muestra la asignación por filas ($pw_i \times n$ filas), es decir que ws_i calcula una parte de C, $C^{(i)}$, de $fC_i = pw_i \times n$ filas por n columnas. Esta forma de asignación de los datos de una matriz es denominada particionamiento por bloques de filas (*block-striped rowwise partitioning*) en [79].
- La Figura 3.4-b) muestra la asignación por columnas ($pw_i \times n$ columnas), es decir que ws_i calcula una parte de C, $C^{(i)}$, de n filas por $cC_i = pw_i \times n$ columnas. Esta forma de asignación de los datos de una matriz es denominada particionamiento por bloques de columnas (*block-striped columnwise partitioning*) en [79].
- La Figura 3.4-c) muestra la asignación por "bloques en general", es decir que ws_i calcula una parte de C, $C^{(i)}$, de $fC_i = q$ filas por $cC_i = r$ columnas tal que $qr = pw_i n^2$. Esta forma de asignación de los datos de una matriz puede considerarse directamente relacionada con la que se denomina particionamiento en tablero por bloques (*block-checkerboard partitioning*) en [79].

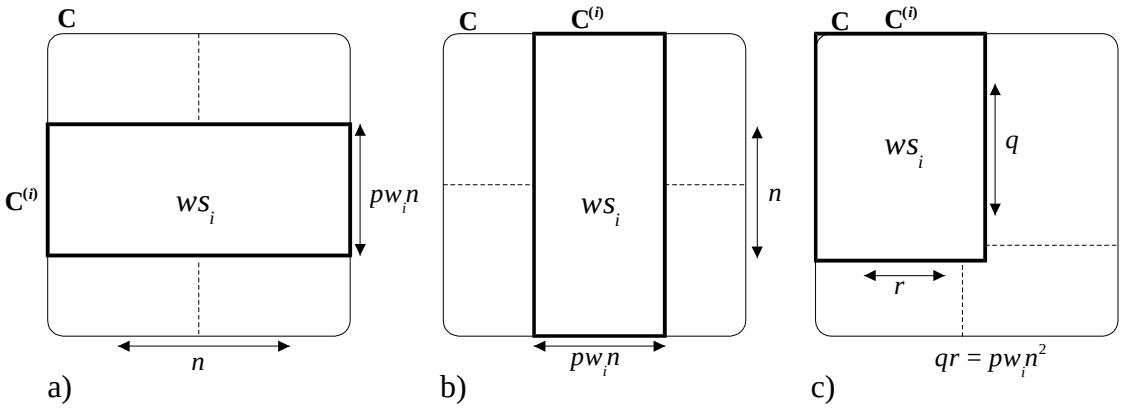


Figura 3.4: Formas Simples de Asignación de Datos.

De las formas de asignación de datos que se muestran en la Figura 3.4, se puede notar que:

- La asignación por filas es totalmente equivalente a la asignación por columnas.
- La asignación por "bloques en general" es bastante más compleja que las anteriores y no parece aportar ningún beneficio sobre las otras. Por otro lado, tiende a ser una distribución bidimensional y por lo tanto no sigue uno de los principio de paralelización enumerados en la sección anterior, por lo tanto se puede, en principio, descartar.

A pesar de que existen formas más complejas y consideradas más apropiadas para la distribución de los datos para cómputo paralelo, se elige distribuir la matriz resultado C por filas para llevar a cabo la multiplicación de matrices en paralelo sobre clusters. La relación y las implicaciones de llevar a cabo este particionamiento comparándolo con los más complejos se explicará después de completada la presentación del algoritmo, para mejorar la claridad en la presentación del algoritmo propuesto. En resumen, la computadora ws_i "tiene a su cargo" (contiene los datos de y calcula) la submatriz $C^{(i)}$ que es de $fC_i = pw_i \times n$ filas y n columnas, es decir $C^{(i)}_{fC_i \times n}$.

Una vez definida la distribución de la matriz resultado C en las computadoras de la red, la forma de distribuir los datos de las matrices A y B se define en función de los cálculos

locales a realizar en cada computadora. Dado que se ha definido que la computadora ws_i tiene que computar $C^{(i)}_{fC_i \times n}$, debe realizar la multiplicación

$$C^{(i)}_{fC_i \times n} = A^{(i)}_{fA_i \times n} \times B \quad (3.5)$$

En la Figura 3.5 se muestran en cada matriz (sombreados) los datos involucrados en el cálculo de $C^{(i)}$ de acuerdo con la Ecuación (3.5).

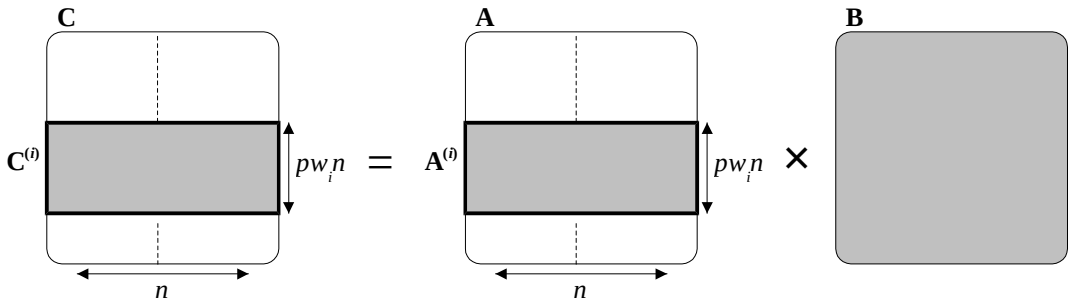


Figura 3.5: Cálculo de una Submatriz.

Por lo tanto, queda bastante claro que la computadora ws_i debería tener localmente los datos de la submatriz de A, $A^{(i)}$, de la Figura 3.5, y esto significa que la distribución de la matriz A entre las *P* computadoras ws_0, ..., ws_{P-1} es igual a la de la matriz C. En resumen, la computadora ws_i "tiene a su cargo" (o contiene los datos de) la submatriz de A, $A^{(i)}$, que es de $fA_i = fC_i = pw_i \times n$ filas y *n* columnas, es decir $A^{(i)}_{fA_i \times n}$.

De acuerdo con la Ecuación (3.5) y la Figura 3.5, lo óptimo en cuanto a disponibilidad de datos en ws_i para el cálculo de $C^{(i)}$ sería tener disponible (en memoria local) toda la matriz B. Con esto, se podría llevar a cabo el cálculo simultáneo de todos los $C^{(i)}$ porque todas las computadoras tendrían todo lo necesario para hacerlo. Como se aclaró anteriormente, todos los datos de las matrices se deberían distribuir entre las distintas computadoras para no establecer requerimientos de memoria demasiado grandes y por lo tanto la matriz B también debería ser distribuida entre las computadoras. De acuerdo con la Ecuación (3.5) y la Figura 3.5, todas las computadoras requieren toda la matriz B, por lo tanto esta matriz se distribuye en partes iguales entre todas las computadoras ws_0, ..., ws_{P-1}. Durante la ejecución del programa paralelo, las computadoras se deben comunicar para intercambiar-recibir las partes de la matriz B que necesitan para el cálculo de la porción de la matriz resultado. Una vez más, esta distribución puede hacerse de forma equivalente por filas o por columnas. Teniendo en cuenta que todas las computadoras harán el cálculo de $C^{(i)}$ en función de esta distribución, debería analizarse (al menos de manera aproximada), si hay alguna de ellas que tenga alguna ventaja sobre la otra.

Distribución de B por filas. Si la matriz B se distribuye en partes iguales por filas entre las computadoras, cada submatriz $B^{(i)}$ sería de *n*/*P* filas por *n* columnas, y se tendrían todos los datos de las matrices distribuidos tal como lo muestra la Figura 3.6 para la computadora ws_i. De esta manera, el cálculo parcial de la submatriz $C^{(i)}$ (y que se denotará $C^{(i_i)}$), que cada computadora puede llevar a cabo con lo datos que dispone localmente está dado por la multiplicación de matrices de la forma que lo muestra la Figura 3.7.

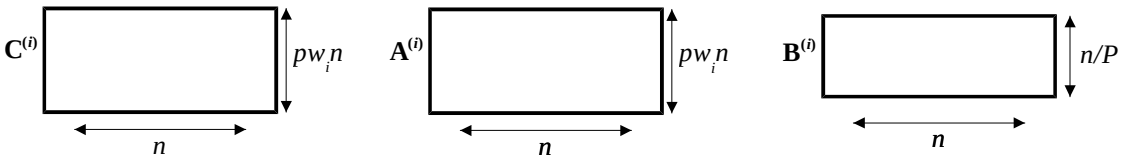


Figura 3.6: Submatrices en ws_i con B Distribuida por Filas.

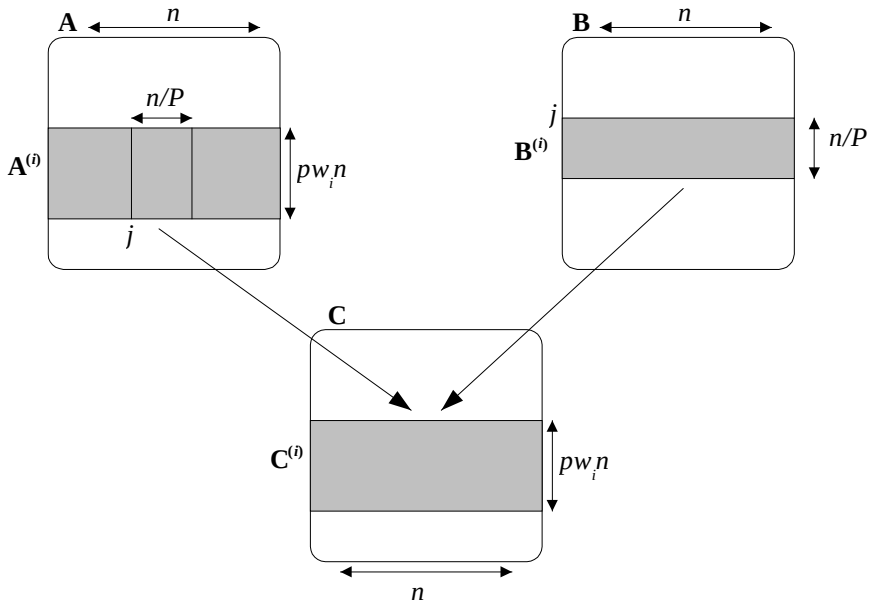


Figura 3.7: Cálculo Parcial de $C^{(i)}$ con B distribuida por Filas.

De manera análoga, con los datos de $B^{(k)}$ de cada computadora ws_k se podrá agregar el cálculo parcial de una parte de $A^{(i)}$ (de pw_in filas y n/P columnas) con $B^{(k)}$ (de n/P filas y n columnas), a $C^{(i_j)}$, tal como el cálculo parcial que se muestra en la Figura 3.7. De esta manera, con sucesivos cálculos parciales de $C^{(i)}$ se llega al resultado final. Esto significa que cada cálculo parcial de $C^{(i)}$ involucra una parte de $A^{(i)}$, todo un bloque $B^{(k)}$, y todo $C^{(i)}$, o sea la multiplicación de dos matrices: una de $pw_in \times n/P$ y otra de $n/P \times n$.

Distribución de B por columnas. Si la matriz B se distribuye en partes iguales por columnas entre las computadoras, cada submatriz $B^{(i)}$ sería de n filas por n/P columnas, y se tendrían todos los datos de las matrices distribuidos tal como lo muestra la Figura 3.8 para la computadora ws_i.

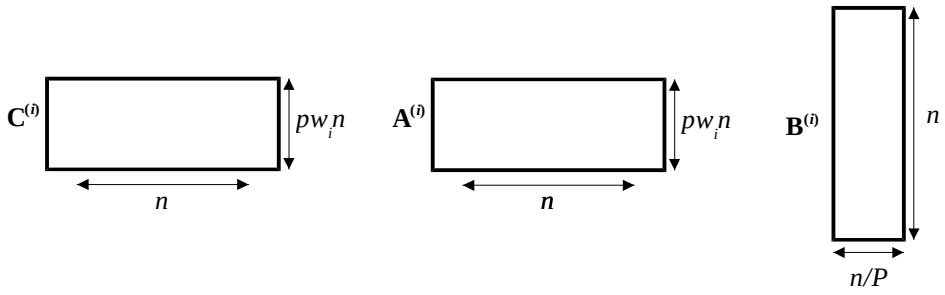


Figura 3.8: Submatrices en ws_i con B Distribuida por Columnas.

De esta manera, el cálculo parcial de la submatriz $C^{(i)}$ (y que se denotará también $C^{(i_i)}$), que cada computadora puede llevar a cabo con lo datos que dispone localmente está dado por la multiplicación de matrices de la forma que lo muestra la Figura 3.9. En ese caso, el cálculo parcial de $C^{(i)}$ que se puede llevar a cabo con los datos $B^{(k)}$ de cada computadora ws_k es el de un bloque de columnas de $C^{(i)}$, y que se denotará $C^{(i_k)}$. En este caso, la multiplicación a realizar es la que corresponde a toda la matriz A, que es de $pw_i n \times n$, con todo un bloque de B, que es de $n \times n/P$, y se calcula una parte de $C^{(i)}$ de $pw_i n \times n/P$.

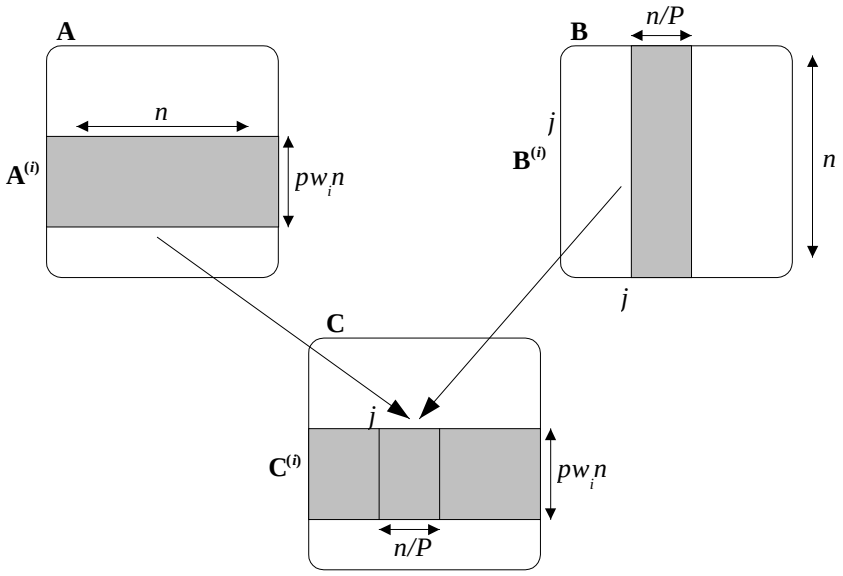


Figura 3.9: Cálculo Parcial de $C^{(i)}$ con B distribuida por Columnas.

Es claro que ambas alternativas de distribución de los datos de B no altera la cantidad de operaciones que se deben realizar, aunque en principio parece más sencilla (o un poco más intuitiva respecto de la definición de la multiplicación de matrices misma), la distribución por columnas, y por lo tanto será la elegida para el algoritmo.

Resumen y Comentarios Acerca de la Distribución de Datos. Las principales características de la distribución de datos entre las computadoras son:

- La matriz A y la matriz C se distribuyen por filas, con la cantidad de filas dada por la potencia de cálculo relativa de cada máquina. Es decir que la computadora ws_i con potencia de cálculo relativa normalizada pw_i tendrá una submatriz de A, $A^{(i)}$, y una submatriz de C, $C^{(i)}$, de $fA_i = pw_i n$ filas por n columnas.
- La matriz B se distribuye de manera uniforme por columnas, lo cual implica que todas las computadoras tendrán la *misma* cantidad de columnas (y por lo tanto datos) de B. Es decir que la computadora ws_i tendrá una submatriz de B, $B^{(i)}$, de n filas por $cB_i = n/P$ columnas, donde P es la cantidad total de computadoras.

Aunque se cumple que

$$\sum_{j=0}^{P-1}(pw_j) = 1 \quad \text{y} \quad \sum_{j=0}^{P-1}(n/P) = n$$

no necesariamente se tendrá que $fA_i = pw_i n$ sea un número entero, dado que $0 < pw_i < 1$, y por lo tanto, operando con números enteros, lo más probable es que

$$df = \lfloor fA_1 \rfloor + ... + \lfloor fA_P \rfloor < n$$

donde $\lfloor fA_i \rfloor$ es el mayor número entero tal que $\lfloor fA_i \rfloor < fA_i$.

Las filas que "quedan" sin repartir, *n-df*, se reparten uniformemente entre las computadoras ws_0, ..., ws_{n-df-1}. Dado que normalmente $P << n$ esta fila "agregada" puede considerarse irrelevante respecto a la posibilidad de generar desbalance de carga de cómputo. El principio a seguir con respecto a la cantidad de columnas de B (n/P no necesariamente es un número entero), es similar.

3.4.2 Cómputo

Como es de esperar para la mayoría (o todos) los algoritmos numéricos que se resuelven en paralelo, se diseña bajo el modelo SPMD (Single Program - Multiple Data) y está dado directamente en pseudocódigo en la Figura 3.10 para la computadora ws_i.

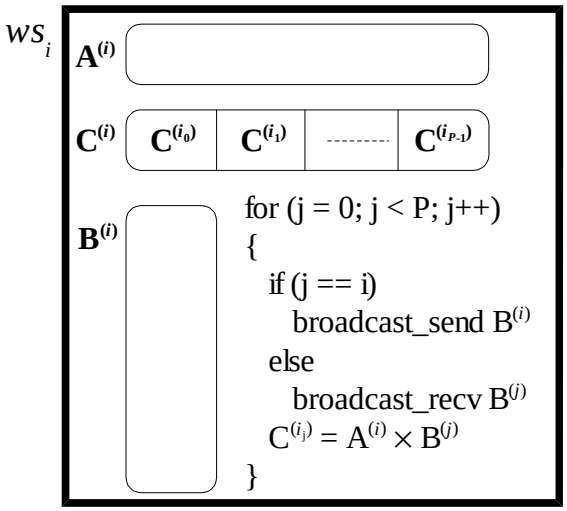


Figura 3.10: Pseudocódigo de la Multiplicación en ws_i.

donde

- P es la cantidad total de computadoras.
- Las matrices A y C están distribuidas por filas, y $A^{(i)}$ y $C^{(i)}$ son las submatrices que ws_i contiene localmente.
- La matriz B está distribuidas por columnas, y $B^{(i)}$ es la submatriz que ws_i contiene localmente.

La Figura 3.11 muestra esquemáticamente la secuencia de pasos de ejecución del algoritmo para cuatro computadoras. Tanto del pseudocódigo de la Figura 3.10 como de la secuencia de pasos que se muestra en la Figura 3.11, se puede ver que las características más importantes del algoritmo son:

- Tiene P pasos de comunicación broadcast (mensajes broadcast), dado que cada una de las computadoras envía un mensaje broadcast y recibe P-1 mensajes broadcast.
  - Tiene P pasos de cómputo local, dado que todas las computadoras realizan P cálculos parciales ($C^{(i_k)}$, $\forall\ k = 0, 1, ..., P\text{-}1$) de la parte de la matriz C que almacenan localmente.
- La cantidad total de pasos Secuenciales de ejecución es $2P$, y el tiempo de cómputo total se calcula aproximadamente como lo muestra la Ecuación (3.6),

$$t_{par} = P\ (t_{bcast} + t_{cómp}) \qquad (3.6)$$

donde

- t_{bcast} es el tiempo necesario para llevar a cabo una operación de broadcast de una submatriz $B^{(i)}$, recordando que $|B^{(i)}| \cong |B^{(j)}|$ $0 \le i, j \le P\text{-}1$, donde $|X|$ indica la cantidad de datos (elementos) de la matriz X.
- $t_{cómp}$ es el tiempo necesario para ejecutar el cómputo local en todas las computadoras, que es el mismo dado que la carga está balanceada en función de los datos de A y C que cada computadora almacena localmente.

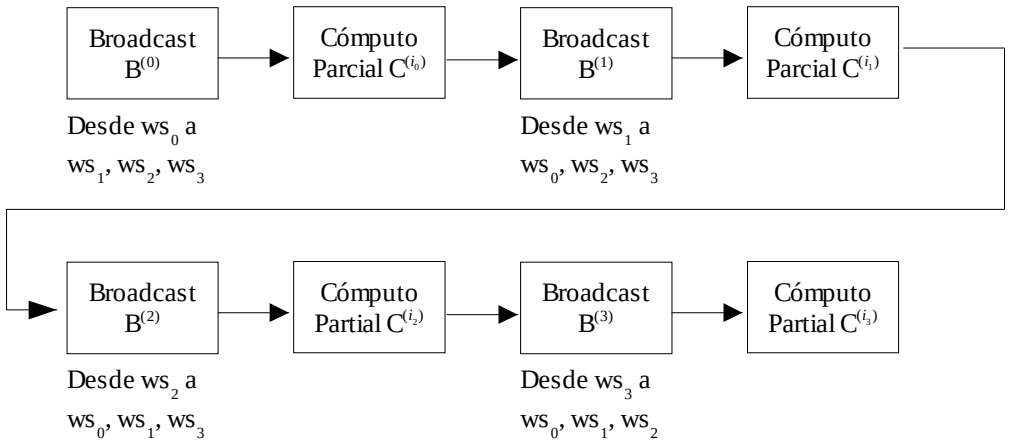


Figura 3.11: Secuencia de Pasos de Ejecución para Cuatro Computadoras.

En el contexto de las redes locales con interconectadas por redes Ethernet, el tiempo de comunicaciones a utilizar en la Ecuación (3.6), t_{bcast}, se puede calcular directamente utilizando la Ecuación (3.1), para lo cual es necesario conocer el tiempo de latencia y de ancho de banda asintótico de la red de comunicaciones. Sin embargo, esto no es posible en general para cualquier computadora paralela y en muchos casos aún en las redes locales interconectadas con Ethernet el tiempo de broadcast puede depender de la forma en que las bobliotecas de comunicaciones utilizadas para el pasaje de mensajes han sido implementadas. De hecho, es bastante usual que el tiempo de un mensaje broadcast con k receptores sea k veces mayor al de una comunicación punto a punto, dado que los mensajes broadcast se implementan como múltiples (k) mensajes punto a punto. En cualquier caso, se debe recalcar que tanto el tiempo de latencia como el ancho de banda asintótico de la red

de comunicaciones debería ser calculado desde los procesos de usuario que componen un programa paralelo [115] [136].

Por lo tanto, considerando que
- Se implementan los mensajes del tipo broadcast aprovechando la capacidad de broadcast de las redes Ethernet.
- Se conoce el tiempo de latencia de los mensajes entre procesos de un programa paralelo y se denota como α.
- Se conoce el ancho de banda asintótico de los mensajes entre procesos de un programa paralelo y se denota como $1/\beta$ (expresado en términos del tipo de elementos que se procesan, tal como números en punto flotante de precisión doble: 8 bytes).
- La cantidad de datos de B que cada computadora tiene, $|B^{(i)}|$, está dada por n filas por $cB_i = n/P$ columnas, es decir n^2/P elementos.

$$t_{bcast} = \alpha + \beta\, n^2/P \tag{3.7}$$

Para calcular el tiempo de cómputo local a utilizar en la Ecuación (3.6), $t_{cómp}$, se pueden reutilizar los valores de capacidad de procesamiento de cada computadora utilizados para calcular las velocidades relativas normalizadas. De acuerdo con la Ecuación (3.3), para el cálculo de pw_i ya se debería tener la capacidad de cómputo de ws_i en Mflop/s, es decir Mflop/s(ws_i). Estos valores de Mflop/s se pueden utilizar de dos maneras para el cálculo de tiempo local:
1. Teniendo en cuenta la capacidad de cómputo de una sola computadora: *Mflop/s*(ws_i).
2. Teniendo en cuenta la capacidad de cómputo de la máquina paralela completa (con la suma de todas las computadoras): $\Sigma_{i=0}^{P-1}(Mflop/s(ws_i))$.

En el primer caso, teniendo en cuenta la capacidad de cómputo de una sola computadora, se utiliza el hecho de que todas las computadoras tienen que ejecutar la misma cantidad de operaciones de punto flotante y por lo tanto el tiempo de ejecución en todas ellas es el mismo. En el segundo caso, teniendo en cuenta la capacidad de cómputo de la máquina paralela completa, se utiliza que:
- Se conoce la capacidad de cálculo de la máquina paralela completa.
- Se conoce la cantidad total de operaciones que se deben realizar ($2n^3$-n^2).
- La cantidad total de operaciones se divide en P pasos, por lo tanto en cada paso se llevan a cabo $1/P$ del total.

Por lo tanto, el tiempo de ejecución $t_{cómp}$ de cada uno de los pasos de procesamiento se puede estimar independientemente de la heterogeneidad de las computadoras con

$$t_{cómp} = \frac{2n^3 - n^2}{P\ pw} \tag{3.8}$$

donde pw es la potencia de cálculo de la máquina paralela

$$pw = \sum_{i=0}^{P-1} Mflop/s(ws_i) \tag{3.9}$$

De esta manera, utilizando la Ecuación (3.7) y la Ecuación (3.8) en la Ecuación (3.6) se llega a que el tiempo de cómputo total del algoritmo está dado por

$$t_{par} = P\alpha + \beta n^2 + \frac{2n^3 - n^2}{pw} \qquad (3.10)$$

Es decir que, en el tiempo total se contabilizan *P* tiempos de latencia correspondientes a los *P* mensajes broadcast, más todo el tiempo de comunicaciones de los datos de la matriz B a la máxima velocidad de transferencia más el tiempo de cómputo de la multiplicación de matrices considerando la potencia de cálculo de la máquina paralela que es la suma de las potencias de cálculo de las computadoras utilizadas.

3.4.3 Cómputo Solapado con Comunicación

El algoritmo presentado previamente no tiene en cuenta la posibilidad de solapar cómputo con comunicaciones, dado que en la especificación misma (pseudocódigo de la Figura 3.10), no está contemplado. Como se ha mencionado antes, la posibilidad de solapar cómputo con comunicaciones en las computadoras paralelas tradicionales ha sido una constante y muchas veces asociada al diseño mismo de las redes de interconexión. En el caso de las computadoras estándares encontradas en las redes locales, esta capacidad no necesariamente se puede asegurar *a priori*. Sin embargo, adaptar el algoritmo anterior para que tenga la capacidad de solapar las comunicaciones con el cómputo local en cada computadora tiene varias ventajas:

- Aunque depende de la adaptación que se haga (sobrecarga agregada), en general no se pierde rendimiento con respecto al algoritmo presentado, ya que lo *peor* que puede suceder es que las comunicaciones se hagan secuenciales con respecto al cómputo.
- Se tiende a aprovechar la posibilidad de hacer cómputo solapado con comunicaciones en caso de que ésta exista en una o más computadoras. En general, se puede mejorar el rendimiento incluso aunque algunas computadoras puedan solamente recibir o enviar datos de forma solapada con el cómputo local.
- En el mejor de los casos, si la granularidad de la aplicación es suficientemente grande y además todas las computadoras tienen la capacidad de solapar las comunicaciones con cómputo local se puede llegar a un valor de rendimiento cercano al óptimo.
- El mismo algoritmo de cómputo solapado con comunicaciones puede identificar la capacidad o no que tiene cada máquina de hacer el solapamiento y de esa manera evaluar cada máquina por separado y la máquina completa (como una clase de *benchmark* para la evaluación de la capacidad de solapamiento de cómputo con comunicaciones).

La adaptación del algoritmo dado en la Figura 3.10 para aprovechar cómputo solapado con comunicaciones está dado en la Figura 3.12, donde

- P es la cantidad total de computadoras.
- Se mantiene la distribución de los datos de las matrices invariante con respecto al algoritmo anterior.
- Las funciones “broadcast_recv_b” y “broadcast_send_b” reciben y envían mensajes

broadcast de forma solapada si es posible (o en *background* desde el punto de vista del sistema operativo), mientras se computan los resultados parciales $C^{(i_j)}$.

- Se puede notar que el primer mensaje broadcast no se lleva a cabo de forma solapada dado que inicialmente solamente ws_0 tiene los datos de $B^{(0)}$ y para el primer paso de cómputo todas las computadoras necesitan $B^{(0)}$.
- El pseudocódigo de cada computadora se vuelve un poco más complejo pero no demasiado con respecto al de la Figura 3.10.

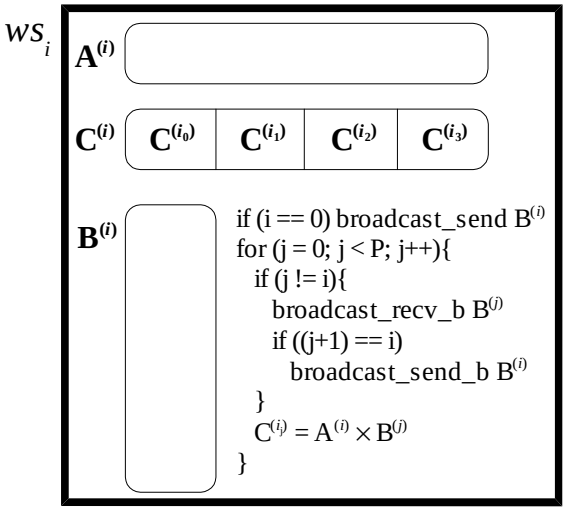


Figura 3.12: Pseudocódigo de Cómputo Solapado con Multiplicación en ws_i.

La Figura 3.13 muestra esquemáticamente la secuencia de pasos de ejecución del algoritmo para cuatro computadoras.

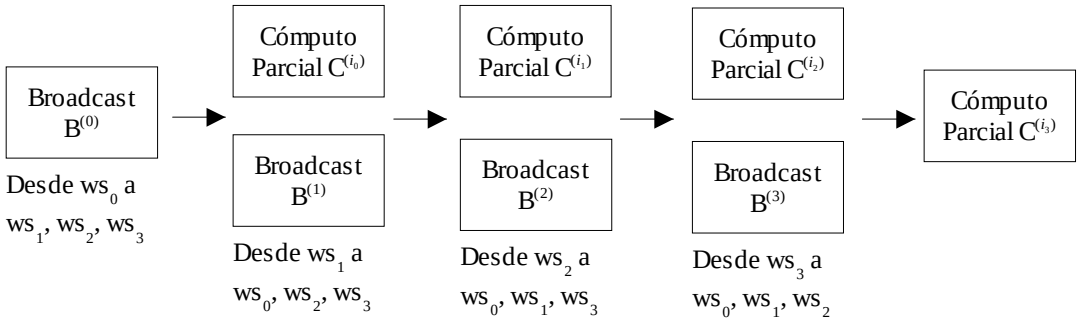


Figura 3.13: Cómputo Solapado con Comunicación en Cuatro Computadoras.

Tanto del pseudocódigo de la Figura 3.12 como de la secuencia de pasos que se muestra en la Figura 3.13, se puede notar que las características más importantes del algoritmo son:

- Tiene *P* pasos de comunicación broadcast (mensajes broadcast), dado que cada una de las computadoras envía un mensaje broadcast y recibe *P*-1 mensajes broadcast. Excepto el primer broadcast, los *P*-1 restantes se pueden llevar a cabo (dependiendo de las capacidades de las computadoras) de forma solapada con el cómputo local.
- Tiene *P* pasos de cómputo local, dado que todas las computadoras realizan *P* cálculos parciales ($C^{(i_k)}$, $\forall\ k = 0, 1, ..., P\text{-}1$) de la parte de la matriz C que almacenan localmente.
- La cantidad total de pasos secuenciales de ejecución es *P*+1.

El tiempo de cómputo total depende ahora de varios factores, tales como:

- La capacidad de solapar cómputo con comunicaciones de cada una de las computadoras. Esto determinará si las comunicaciones se llevan a cabo de forma "simultánea" con el cómputo como lo muestra la Figura 3.13 o no. Si una computadora no es capaz de solapar cómputo con comunicaciones, la tarea de cómputo y de comunicaciones se deberán hacer secuenciales.
- La influencia de la tarea de comunicación "en background" sobre el cómputo. Dado que se están transfiriendo datos, habrá uno o más procesos que controlen esta transferencia y por lo tanto utilizarán CPU y recursos como la jerarquía de memoria, por lo tanto el o los procesos de cómputo ahora competirán por estos recursos.
- La granularidad del problema, que determinará qué porcentaje de las comunicaciones se puede solapar con el cómputo. Expresado de otra manera, aunque una computadora pueda solapar cómputo con comunicaciones, si el tiempo de comunicar $B^{(i)}$ es mayor que el de cómputo local, necesariamente el tiempo de comunicaciones determinará el tiempo total de ejecución.
- Asumiendo el mejor de los casos, es decir que:
    - todas las computadoras son capaces de solapar cómputo con comunicaciones
    - y no hay penalización en la capacidad de cómputo por llevar a cabo transferencias de datos solapadas,

  el tiempo total de ejecución del algoritmo Figura 3.12 se puede estimar con la Ecuación (3.10),

$$t_{par} = t_{bcast} + (P\text{-}1)\ máx(t_{bcast}, t_{cómp}) + t_{cómp} \qquad (3.11)$$

donde t_{bcast} y $t_{cómp}$ se calculan de acuerdo con la Ecuación (3.7) y con la Ecuación (3.8) respectivamente.

3.4.4 Reducción de Requerimientos de Memoria para Mensajes

Tanto el algoritmo presentado inicialmente en la Figura 3.10 con los períodos de cómputo y comunicación ejecutados de manera secuencial como el de la Figura 3.12, organizado para aprovechar la posibilidad de solapamiento de cómputo local con comunicaciones, tienen una característica común en cuanto a memoria para mensajes: ambos comunican toda una submatriz de B. Esto significa que, eventualmente, se necesitan buffers o memoria en cada computadora para ws_i, al menos, recibir los datos de la matriz B, $B^{(j)}$, de la computadora ws_j sin destruir o sobreescribir los datos locales.

En el caso específico del algoritmo con cómputo y comunicaciones solapados esto no solamente genera requerimientos de memoria mayores sino que, además, hace cada mayor el tiempo inicial del envío de los primeros datos a todas las computadoras (Figura 3.12, primer paso de comunicación: envío de $B^{(0)}$) a medida que la matriz B es mayor y por lo tanto es mayor la submatriz involucrada en las comunicaciones.

La reducción de requerimientos de memoria para los mensajes es relativamente sencilla y también basada en la idea de procesamiento por bloques: en vez de enviar (y recibir) toda

la submatriz de B local de cada computadora se envía (recibe) por partes. Si, por ejemplo, cada submatriz de B, $B^{(i)}$, se envía en diez partes, cada una de estas partes será de $|B^{(i)}/10|$ elementos, y por lo tanto se reducen a 1/10 los requerimientos de memoria para las comunicaciones de manera *inmediata*. De hecho, el "bloque de comunicaciones" básico definido para los algoritmos presentados es, justamente $n/10$, donde n es el orden de las matrices cuadradas a multiplicar.

Se debe recordar que la matriz $B^{(n \times n)}$ se distribuye por bloques de columnas entre las computadoras, es decir que habiendo P computadoras:
- Cada una de ellas tendrá una submatriz de B con n filas y n/P columnas.
- Las submatrices se envían-reciben en "bloques" de $n/10$ filas (para reducir a 1/10 los requerimientos de memoria para comunicaciones) y n/P columnas (todas las columnas).

En el caso específico del algoritmo con cómputo y comunicaciones solapados esto implica, además, reducir el tiempo inicial del envío de los primeros datos a todas las computadoras (Figura 3.12, primer paso de comunicación: envío de $B^{(0)}$) porque ahora solamente hay que comunicar 1/10 del total de los datos de la submatriz para comenzar a hacer los cálculos parciales. Más específicamente, la propia trasnmisión de $B^{(0)}$ se solapa con los cálculos parciales en los que está involucrada la submatriz $B^{(0)}$.

3.4.5 Características Generales

Más allá de las diferencias en cuanto a la posibilidad de solapar cómputo con comunicaciones, las características más importantes de ambas formas presentadas de calcular en paralelo una multiplicación de matrices $C = A \times B$ en P computadoras son:
- Siguen el modelo SPMD (Single Program - Multiple Data).
- No hay replicación de datos, ya que todos los datos se distribuyen y un dato reside localmente en una única computadora.
- Las comunicaciones entre computadoras son únicamente de tipo broadcast, y se llevan a cabo de forma tal que no generan interferencias en la red de comunicaciones de manera independiente del cableado que se utilice (hubs, switches, etc.).
- Se tiende a la máxima granularidad, todos los datos que se tienen que comunicar de una computadora a otra se transfieren de una sola vez. Esto tiende a reducir la penalización dada por el tiempo de latencia (startup) de los mensajes.
- El balance de carga en cuanto a cómputo está dado por la cantidad de datos de la matriz resultado que debe calcular cada computadora, la cual a su vez está definida proporcionalmente a su capacidad de cálculo relativa con respecto a las demás computadoras que calculan en paralelo. Por lo tanto, con la distribución de los datos de la matriz resultado, C, todas las computadoras realizan el procesamiento en aproximadamente el mismo tiempo.
- El balance de comunicaciones está dado por la asignación de los datos de la matriz B, que son los únicos que se transfieren entre las computadoras. Dado que todas tienen aproximadamente la misma cantidad de datos de la matriz B y que todas las computadoras envían un mensaje broadcast y reciben P-1 mensajes broadcast, todas las computadoras envían y reciben aproximadamente la misma cantidad de datos.

3.4.6 Otras Formas de Distribuir Datos

Tal como se adelantó en la explicación de la distribución de los datos para el algoritmo, existen formas más complejas que la presentada y utilizada en esta capítulo para distribuir los datos de las matrices entre múltiples procesadores de una máquina paralela. De hecho, han surgido algunas discusiones al respecto a nivel de reportes técnicos tales como [47] [106]. Una de las formas consideradas más apropiadas es la que se denomina como particionamiento cíclico en tablero por bloques en [79], aunque quizás sea más conocida como descomposición cíclica de bloques bidimensional (*two-dimensional block cyclic decomposition*), tal como se la menciona en [26] [21] [45] y en el contexto de la biblioteca denominada ScaLAPACK (Scalable LAPACK) [27] [28] [ScaLAPACK]. La Figura 3.14 muestra esta forma de distribuir los datos de una matriz A de 7×8 elementos, considerando bloques de 1×1 elementos, en una malla de 3×2 procesadores.

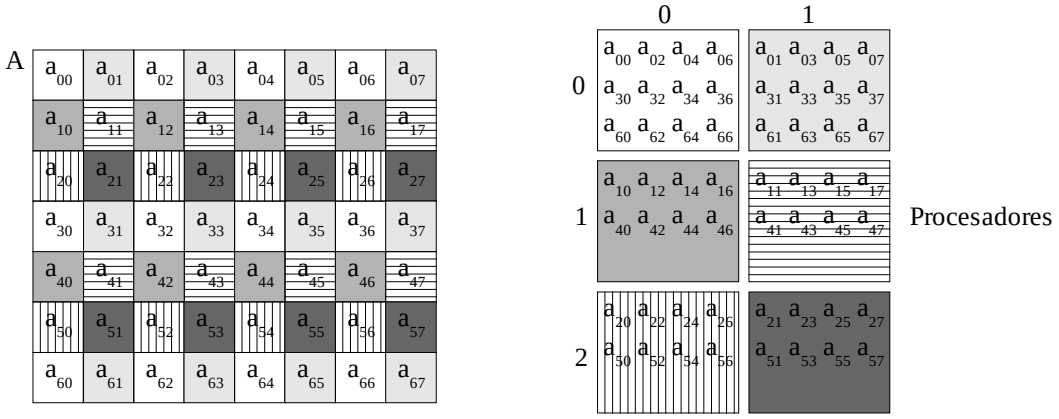


Figura 3.14: Descomposición Cíclica de Bloques Bidimensional.

Como lo muestra la Figura 3.14, la descomposición cíclica de bloques bidimensional está claramente orientada a las redes de interconexión de procesadores con topología de grilla o toro bidimensional. Además, es aplicable para cualquier tamaño de bloques de matrices, cualquier tamaño de matrices y cualquier malla o toro bidimensional de procesadores.

Toda la biblioteca ScaLAPACK asume que los datos (matrices y vectores) están distribuidos según esta descomposición. De hecho, dado que existen muchas alternativas para esta distribución, la representan con un arreglo descriptor de la distribución en el cual se identifica, entre otras cosas:

- Cantidad de filas de la matriz.
- Cantidad de columnas de la matriz.
- Cantidad de filas del bloque.
- Cantidad de columnas del bloque.

y este arreglo descriptor es utilizado (como parámetro) para resolver las rutinas básicas de cómputo tal como la multiplicación de matrices. La principal ventaja que se identifica explícitamente con esta distribución es la de balance de carga.

En el contexto de cómputo paralelo en redes de computadoras se deben hacer al menos tres consideraciones con respecto a la comparación de la Descomposición Cíclica de Bloques

Bidimensional con la presentada en este capítulo:
- Tamaño de los bloques a distribuir.
- Distribución de datos y su relación con la interconexión de los procesadores.
- Balance de carga del algoritmo.

Tamaño de los Bloques a Distribuir. Este es un serio inconveniente para las bibliotecas como ScaLAPACK y también PLAPACK [4] [141] [PLAPACK], que realizan las tareas de cómputo paralelo con un mismo tamaño de bloque (dado en ScaLAPACK por el arreglo descriptor de la distribución de datos). El tamaño de los bloques determina de manera directa dos aspectos en el procesamiento paralelo:
- Cómputo local en cada procesador.
- Comunicación entre procesadores.

Respecto del cómputo local, el tamaño de bloque está definido para obtener el óptimo rendimiento de los procesadores involucrados. Esto es relativamente *sencillo* en el caso de las máquinas paralelas con procesadores homogéneos, pero es inevitablemente un problema para las computadoras con procesadores heterogéneos. En este contexto, hasta es posible que dos procesadores puedan tener la misma capacidad de procesamiento (velocidad relativa o Mflop/s), pero con distintos valores de bloques, dado que esto depende fuertemente de las jerarquías de memoria (niveles y tamaños de memoria cache) que pueden ser muy diferentes.

Por lo tanto, tener un mismo tamaño de bloque es un problema más a resolver, dado que habría que determinar un valor que sea el *mejor*, lo cual no es sencillo en general dada la gran variedad de computadoras en las redes locales. Cualquiera sea el tamaño de bloque elegido, siempre se tendrá que algunas computadoras se aprovecharán al máximo en cuanto a rendimiento, mientras que en otras se procesará a una fracción del máximo rendimiento posible.

Por otro lado, en la distribución elegida no hay ningún tamaño de bloque predefinido, y esto tiene al menos dos consecuencias ventajosas:
- Se elimina un parámetro en las rutinas de procesamiento paralelo. La distribución de los datos es invariante, no hay más de una posibilidad para distribuir los datos de las matrices.
- El tamaño de bloques que optimiza el rendimiento de cada procesador se puede elegir localmente y no tiene efectos colaterales, no afecta el rendimiento ni el procesamiento en las demás computadoras.

Desde el punto de vista de ScaLAPACK esto puede ser considerado como una optimización que se debe hacer en el contexto de la heterogeneidad de las redes locales instaladas, pero que ScaLAPACK no resuelve (aún).

Distribución de Datos e Interconexión de Procesadores. Tanto en los los algoritmos orientados a computadoras paralelas de memoria distribuida presentados en el capítulo anterior como en los presentados en este capítulo, se puede notar que la distribución de los datos está íntimamente relacionada con el algoritmo de procesamiento y con la interconexión de los procesadores mismos. De hecho, esta es una característica distintiva de la mayoría (o *todos*) los algoritmos numéricos orientados a computadoras paralelas de memoria distribuida. Por lo tanto, una distribución bidimensional de datos está

directamente relacionada con una arquitectura de cómputo paralelo subyacente que tiene al menos dos características:
- Memoria distribuida.
- Procesadores interconectados con topología de malla o toro bidimensional.

Como ya se ha explicado, la interconexión de las redes locales de computadoras basadas en Ethernet (la *gran* mayoría) puede variar dependiendo principalmente del cableado, y más específicamente de la utilización de hubs y/o switches. Las redes en las que se utilizan switches se puede considerar la interconexión física como una malla o toro bidimensional (y de hecho, de varias maneras más, dada la flexibilidad de las redes con switches), pero es claro que esto no es válido para *todas* las redes, en donde se pueden encontrar hubs. Lo que sí es válido en general es que en todas las redes locales basadas en Ethernet es inmediato ("por hardware", dada la definición de Ethernet) considerar que la interconexión de las computadoras está proporcionada de manera lógica por un bus de comunicaciones, tal como el de la Figura 3.1 o el de la Figura 3.2.

Por lo tanto:
- No siempre se puede tener de manera inmediata una interconexión de las computadoras de las redes locales como una malla o toro bidimensional. Esto implica que todas las distribuciones de datos bidimensionales con sus algoritmos de cómputo orientados a redes de interconexión bidimensionales tendrán la tendencia de ser penalizados. Esta penalización será producto de la *serialización* de los mensajes sobre el bus de comunicaciones definido por Ethernet, sobre el cual se producirán las colisiones en las transferencias de datos.
- Al considerar una distribución de datos como la presentada en este capítulo se está considerando la interconexión de los procesadores en "topologías unidimensionales", tales como la del bus definido por Ethernet o los anillos con conexiones punto a punto. Más específicamente, la distribución de los datos presentada en este capítulo está orientada al aprovechamiento directo de las redes locales que en su mayoría están basadas en Ethernet y además de manera independiente de la variedad posible en el cableado de las mismas.

Balance de Carga de los Algoritmos. Como se afirma en general, el balance de carga es trivial en el caso de la Descomposición Cíclica de Bloques Bidimensional. Esta característica no necesariamente se mantiene en el contexto de los procesadores heterogéneos de las redes locales instaladas.

Pero desde el punto de vista de la distribución presentada en este capítulo, en términos generales no se pierde balance de carga ni generalidad de utilización en diferentes circunstancias, ya que:
- Se conserva el balance de carga de procesamiento en el caso de computadoras paralelas con procesadores homogéneos.
- Se conserva el balance de carga de procesamiento en el caso de computadoras paralelas con procesadores heterogéneos.
- El balance de carga es trivial, aunque en el caso de procesadores heterogéneos se debe conocer con anterioridad la velocidad relativa de los procesadores.

3.5 Resumen del Capítulo

A lo largo de este capítulo se han presentado:

- Las características de los clusters interconectados por redes Ethernet, tanto homogéneos como heterogéneos cuando se utilizan como plataformas de cómputo paralelo.
- Los principios de paralelización de aplicaciones a resolverse específicamente sobre clusters para obtener rendimiento optimizado. Estos principios se orientan a la utilización optimizada de todas las características de cómputo y procesamiento local en clusters.
- Dos algoritmos paralelos para multiplicación de matrices específicamente orientados a la obtención de rendimiento optimizado en clusters.
- Comentarios necesarios para la caracterización de cada algoritmo y también las comparaciones mínimas de estos algoritmos con los que se utilizan en ScaLAPACK, por ejemplo, básicamente en lo referente a la distribución de los datos.

No solamente quedan claras las diferencias a nivel de distribución de datos de los algoritmos propuestos en esta tesis, sino también en cuanto a la utilización (y en cierta forma *dependencia*) de los mensajes broadcast como la única forma de comunicar datos entre los procesos de una aplicación paralela.

Capítulo 4: Experimentación

En este capítulo se presentará inicialmente todo el contexto de experimentación con los algoritmos de cómputo paralelo para el cálculo de multiplicaciones de matrices. Se explica detalladamente el hardware (tres redes locales) y los tamaños de matrices con los cuales se llevaron a cabo los experimentos. También se dan los valores de velocidades relativas de las computadoras y con estos valores más una estimación (optimista) *a piori* del rendimiento de las redes de interconexión se estima el máximo valor de speedup posible en cada una de las redes locales con las máquinas disponibles.

La primera aproximación a la implementación de los algoritmos se basa en la biblioteca PVM y se muestran todos los resultados de la experimentación. El rendimiento que se obtiene es muy distante del óptimo y no es aceptable. Por esta razón se muestran algunos detalles de la ejecución paralela que dan a conocer que el problema no son los algoritmos sino específicamente la implementación de los mensajes broadcast de la biblioteca PVM.

Finalmente, se propone e implementa un mensaje broadcast basado directamente en el protocolo de comunicaciones UDP, se vuelven a realizar los experimentos y se muestran los resultados obtenidos con esta implementación de los mensajes broadcast. En este contexto se llega a que: o bien los algoritmos obtienen un rendimiento aceptable o se puede detectar de manera automática las computadoras que generan problemas de rendimiento y por lo tanto se las puede aislar para obtener rendimiento optimizado.

4.1 Características de las Redes Locales Utilizadas

Cada una de las redes locales utilizadas en la experimentación son preexistentes al presente trabajo y no se cambiaron ni se adaptaron para obtener mejores resultados en cuanto a rendimiento. En las subsecciones que siguen se describen sintéticamente cada una de las redes, identificando las características más importantes en cuanto a las computadoras que la componen y la topología de la red de interconexión. Básicamente, las redes locales son:

- CeTAD: perteneciente al Centro de Técnicas Analógico-Digitales, Departamento de Electrotecnia, Facultad de Ingeniería, Universidad Nacional de La Plata, La Plata, Argentina. Es la que está instalada desde hace más tiempo y las computadoras que la componen son utilizadas con múltiples propósitos.
- LQT: perteneciente al Laboratorio de Química Teórica, CEQUINOR, Departamento de Química, Facultad de Ciencias Exactas, Universidad Nacional de La Plata, La Plata, Argentina. 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, La Plata, Argentina. 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.

En términos generales, el software paralelo se desarrolló utilizando PVM (Parallel Virtual Machine). En el caso particular de las PCs utilizadas, el sistema operativo elegido para la ejecución fue Linux [44] [PVM].

Las razones para la elección de PVM (además de su libre disponibilidad), son básicamente dos:

- Existe un único grupo de desarrollo y fuente del software. Esto puede ser una desventaja, pero simplifica el análisis de los resultados obtenidos en cuanto a rendimiento dado que no hay posibilidad de distintas implementaciones. Esto no es posible de asegurar en el caso de MPI (Message Passing Interface) [88] [92] [107] que tiene múltiples implementaciones y potencialmente distintas características en cuanto a rendimiento.
- Es ampliamente utilizado, tiene varios años de evolución y sus características son muy bien conocidas, lo que simplifica también la interpretación de los resultados obtenidos en cuanto a rendimiento de las aplicaciones paralelas que lo utilizan.

En la mayoría de las redes locales, las PCs ya contaban con Linux instalado, principalmente de la distribución RedHat [LinuxRH]. Siempre que fue posible, se intentó la instalación de Linux en una partición de disco separada y la distribución utilizada en tal caso es la de RedHat.

4.1.1 Red Local del CeTAD

Tal como se mencionó anteriormente, la red local del CeTAD tiene dos características

generales que implican una gran variedad en cuanto a la heterogeneidad de las máquinas que la integran:
1. Tiene más de diez años de instalación, con los cambios, agregados y actualizaciones que esto implica.
2. Las máquinas tienen múltiples propósitos, que abarcan trabajo administrativo, prototipación de algoritmos de procesamiento de señales y diseño de circuitos integrados de propósito específico.

La Tabla 4.1 muestra las características más importantes de las computadoras que integran la red local y que se utilizan en la experimentación. En el Apéndice A se dan mayores detalles de cada una de las máquinas. Excepto las computadoras **cetadfomec1** y **cetadfomec2** identificadas con el "tipo" IBM PC, todas las PCs son construidas por partes, lo cual suele ser el caso general de las PCs.

	Nombre	**Tipo**	**CPU**	**Frec. Reloj**	**Mem.**
1)	purmamarca	PC	Pentium II	400 MHz	64 MB
2)	cetadfomec1	IBM PC	Celeron	300 MHz	32 MB
3)	cetadfomec2	IBM PC	Celeron	300 MHz	32 MB
4)	sofia	IBM RS6000	IBM PPC604e	200 MHz	64 MB
5)	fourier	PC	Pentium MMX	200 MHz	32 MB
6)	Josrap	PC	AMD K6-2	450 MHz	62 MB
7)	tilcara	PC	Pentium	133 MHz	32 MB
8)	paris	SPARCstation 4	MicroSPARC-II	110 MHz	96 MB
9)	cetad	SPARCstation 5	MicroSPARC-II	85 MHz	96 MB
10)	prited	SPARCstation 2	CY7C601	40 MHz	32 MB

Tabla 4.1: Computadoras del CeTAD.

Una vez más se debe aclarar que no se ha cambiado nada de lo que se tenía ya instalado, solamente se agregaron las herramientas de software necesarias para el desarrollo, implementación y ejecución de programas paralelos en el caso de las computadoras que no lo tenían antes de la experimentación. El mayor costo de instalación al respecto se dio con las PCs que no tenían instalado Linux ni las herramientas de software necesarias para cómputo paralelo (bibliotecas tales como PVM [44] [PVM]):
- **purmamarca**, se particionó el disco rígido y se instaló Linux (RedHat) y PVM.
- **fourier** y **Josrap**, donde se instaló la distribución Winlinux [WinLinux] dado que era muy difícil (o riesgoso) particionar el disco rígido y se suponía que la distribución Winlinux era la más sencilla de instalar en tales condiciones.

La red de interconexión de las máquinas es Ethernet de 10 Mb/s y el cableado se muestra en la Figura 4.1, donde
1. Se utilizan Hubs en cascada, y la interconexión lógica de las computadoras sigue siendo la de un bus.
2. "Trans." indica *Transceiver*, necesario porque la placa de red de la computadora con nombre **prited** tiene salida con conexión BNC (para cable coaxial) únicamente.

3. "**cf1**" y "**cf2**" se utilizan como abreviaciones de los nombres **cetadfomec1** y **cetadfomec2** respectivamente (y tales abreviaciones se seguirán utilizando por razones de espacio).

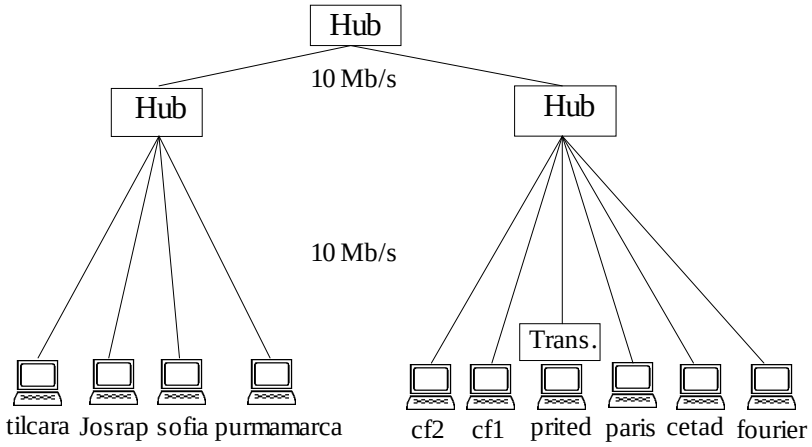


Figura 4.1: Cableado de la Red Local del CeTAD.

4.1.2 Red Local del LQT

A diferencia de la red local del CeTAD, la del LQT está dedicada a resolver problemas numéricos. 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 Tabla 4.2 muestra las características más importantes de las computadoras que integran la red local y que se utilizan en la experimentación. En el Apéndice A se dan mayores detalles de cada una de las máquinas.

	Nombre	**Tipo**	**CPU**	**Frec. Reloj**	**Mem.**
1)	lqt_07	PC	Pentium III	1 GHz	512 MB
2)	lqt_06	PC	Pentium III	1 GHz	512 MB
3)	lqt_02	PC	Celeron	700 MHz	512 MB
4)	lqt_01	PC	Pentium III	550 MHz	512 MB
5)	lqt_03	PC	Pentium II	400 MHz	512 MB
6)	lqt_04	PC	Pentium II	400 MHz	512 MB

Tabla 4.2: Computadoras del LQT.

A diferencia de la red local del CeTAD, todas las computadoras disponibles en esta red local son PCs construidas por partes y tienen una capacidad bastante mayor en cuanto a cálculo (procesadores y frecuencias de reloj de operación) y almacenamiento (memoria

principal instalada).

La red de interconexión de las máquinas es Ethernet de 10 Mb/s y el cableado se muestra en la Figura 4.2, donde se puede notar que la interconexión es de las más sencillas posibles.

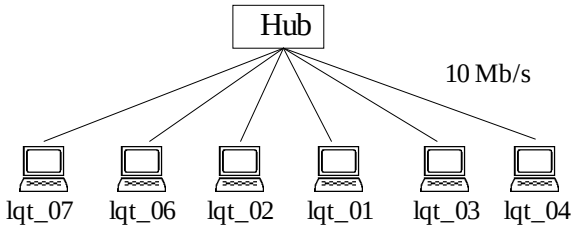


Figura 4.2: Cableado de la Red Local del LQT.

4.1.3 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 por lo tanto no ha tenido ningún cambio desde su instalación, manteniéndose homogénea.

La Tabla 4.3 muestra las características más importantes de las computadoras que integran la red local y que se utilizan en la experimentación. En el Apéndice A se dan mayores detalles de cada una de las máquinas. Se muestran las computadoras en una tabla solamente para utilizar el mismo formato que en el caso de las redes anteriores, pero dado que las computadoras son iguales alcanza con la descripción de una de ellas.

	Nombre	**Tipo**	**CPU**	**Frec. Reloj**	**Mem.**
1)	lidipar14	PC	Pentium III	700 MHz	64 MB
2)	lidipar13	PC	Pentium III	700 MHz	64 MB
3)	lidipar12	PC	Pentium III	700 MHz	64 MB
4)	lidipar9	PC	Pentium III	700 MHz	64 MB
5)	lidipar8	PC	Pentium III	700 MHz	64 MB
6)	lidipar7	PC	Pentium III	700 MHz	64 MB
7)	lidipar6	PC	Pentium III	700 MHz	64 MB
8)	lidipar5	PC	Pentium III	700 MHz	64 MB

Tabla 4.3: Computadoras del LIDI.

Como lo muestra la Figura 4.3, el cableado de la red del LIDI coincide en cuanto a simplicidad con el de la red del LQT, pero no en cuanto a costo y rendimiento ya que
- Las placas de red instaladas son Ethernet de 10/100 Mb/s, lo cual implica que la velocidad de comunicación depende del hub o switch que las interconecta.
- En vez de utilizar un hub se utiliza un switch que al igual que las placas de red de las

computadoras también es de 10/100 Mb/s.
- Toda la red entonces tiene la capacidad de llevar a cabo varias (hasta cuatro) comunicaciones punto a punto simultáneas de 100 Mb/s.

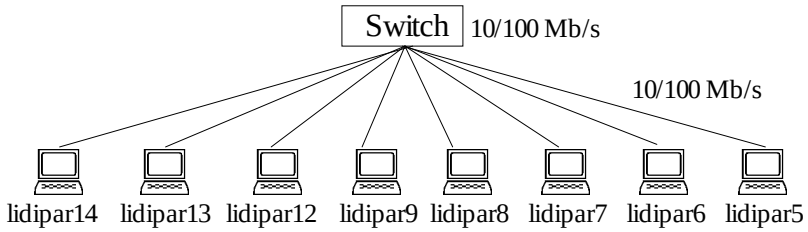


Figura 4.3: Cableado de la Red Local del LIDI.

4.2 Rendimiento Secuencial de las Computadoras

El cálculo del rendimiento secuencial de las computadoras tiene principalmente dos razones:
1. Cálculo del speedup obtenido al utilizar procesamiento paralelo. Para conocer la ganancia de utilizar procesamiento paralelo necesariamente se debe conocer como mínimo el rendimiento de la computadora más rápida disponible en cada red local.
2. Tal como se adelantó en el capítulo anterior, se calcula la velocidad relativa de las computadoras de acuerdo con la capacidad de cálculo secuencial de cada una de ellas para resolver una multiplicación de matrices.

Por estas dos razones se incluyen en esta sección los valores de rendimiento obtenidos para cada una de las máquinas en cada una de las redes locales. Además, en el Apéndice B se explica detalladamente cómo se obtuvieron estos valores de acuerdo a los distintos experimentos realizados.

Dada la naturaleza casi exclusivamente numérica de procesamiento de datos, los valores de rendimiento se expresan en Mflop/s (Millones de operaciones de punto flotante por segundo). La representación de los datos numéricos es la de punto flotante de precisión simple (norma IEEE 754 [72]) en todas las computadoras. La opción de punto flotante de precisión doble, aunque recomendada en general [11] se descarta por dos razones:
- Las computadoras utilizadas tienen la misma o similar capacidad de procesamiento de números en punto flotante de precisión simple que de precisión doble, dadas las características de sus unidades de punto flotante.
- Al utilizar números representados en precisión simple se pueden tener mayores tamaños de matrices en memoria y por lo tanto mayores requerimientos en cuanto a cantidad de operaciones de punto flotante necesarias para resolver una multiplicación de matrices.

4.2.1 Tamaños de Matrices Utilizados

Es indudable que el rendimiento de las computadoras en cuanto a procesamiento de datos en general y de números en punto flotante en particular depende de las relaciones entre:

- Cantidad de datos.
- Jerarquía de memoria (niveles y tamaños de memoria cache).
- Patrón de acceso a los datos.

Y es por esto que los valores de rendimiento se muestran en función de tamaños de matrices considerados significativos. Estas relaciones y los detalles más significativos de los tamaños elegidos en particular se explican con mayor precisión en el Apéndice B.

Por un lado, es importante tener una referencia de la capacidad de procesamiento de las computadoras cuando todos o una buena proporción de los datos a procesar se pueden contener en los niveles de memoria cache más cercanos al procesador (niveles 1 y 2, por ejemplo). En este contexto, se tomaron como referencia varios tamaños de matrices que son relativamente pequeños con respecto al tamaño de memoria principal: matrices de orden n = 100, 200, 400. Teniendo en cuenta que los datos numéricos se representan con números en punto flotante de precisión simple, 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.

La utilización de las computadoras al límite de su capacidad (al menos en cuanto a memoria principal) ha sido una constante y, de alguna manera, es lo que se refleja en la "revisión" a la Ley de Amdhal [6] [60]. Dos valores se tomaron como representativos de los tamaños de matrices que se pueden manejar en memoria principal de 32 MB: n = 800 y n = 1600. Con matrices de 800×800 datos, la cantidad de memoria necesaria para contener las tres matrices que intervienen en una multiplicación ($C = A \times B$) es de aproximadamente 7.3 MB de datos (22.8% del total de 32 MB de memoria principal aproximadamente). En el caso de matrices de 1600×1600 datos, la cantidad de memoria requerida es de aproximadamente 29.3 MB, lo que representa el 91.6% del total de 32 MB de 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.

Si bien cuando los datos ocupan toda o la mayor parte de la memoria principal se puede considerar que se está utilizando *al máximo* una computadora (al menos en cuanto a datos en memoria principal), el caso extremo se da cuando se consideran los tamaños que exceden la capacidad de memoria principal y se recurre al espacio de memoria swap. 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: a) memoria principal instalada, b) espacio de swap configurado y c) 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 requerimientos de memoria principal 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.

4.2.2 Red Local del CeTAD

Los valores de rendimiento obtenidos para cada una de las computadoras de la red local del CeTAD se muestran en la Figura 4.4.

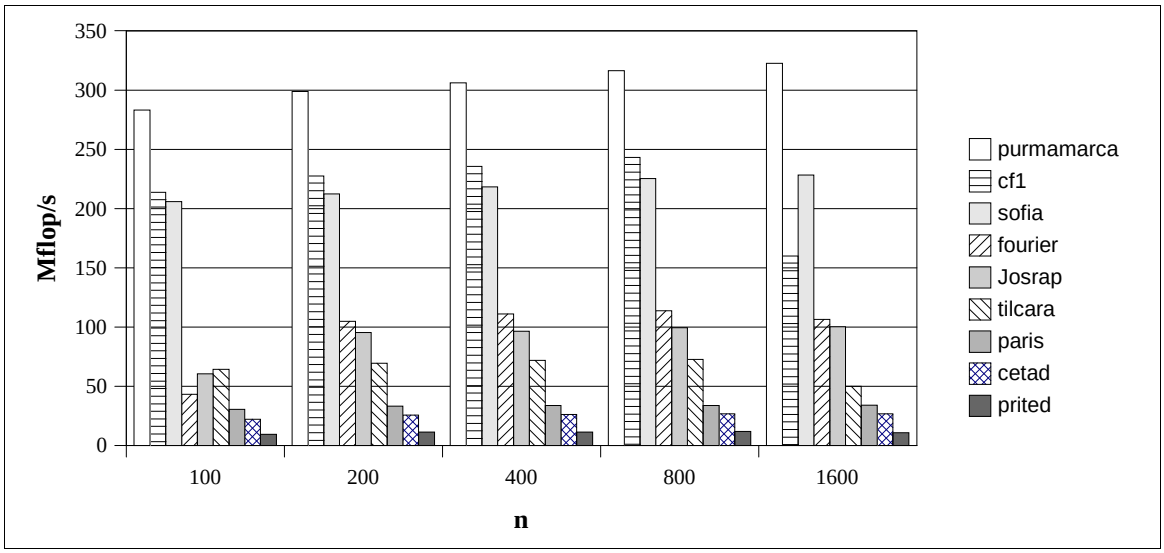


Figura 4.4: Rendimiento de las Computadoras del CeTAD.

En la Figura 4.4 se puede notar que:

- **purmamarca** es la computadora con mayor velocidad relativa y es aproximadamente 30 veces más rápida en procesamiento que **prited**, que es la que tiene menor capacidad.
- Solamente aparece **cf1**, en referencia a **cetadfomec1,** dado que el rendimiento de **cetadfomec2** es el mismo.
- Se puede comprobar con los valores de la Tabla 4.1 que la frecuencia de reloj a las cuales operan las computadoras no necesariamente determina la capacidad de procesamiento (al menos en operaciones de punto flotante).

Las características de rendimiento y/o los valores obtenidos se explican con mayor nivel de detalle en el Apéndice B.

En la computadora con mayor capacidad de procesamiento, **purmamarca**, se llevaron a cabo experimentos con matrices mayores y los resultados obtenidos se muestran en la Figura 4.5.

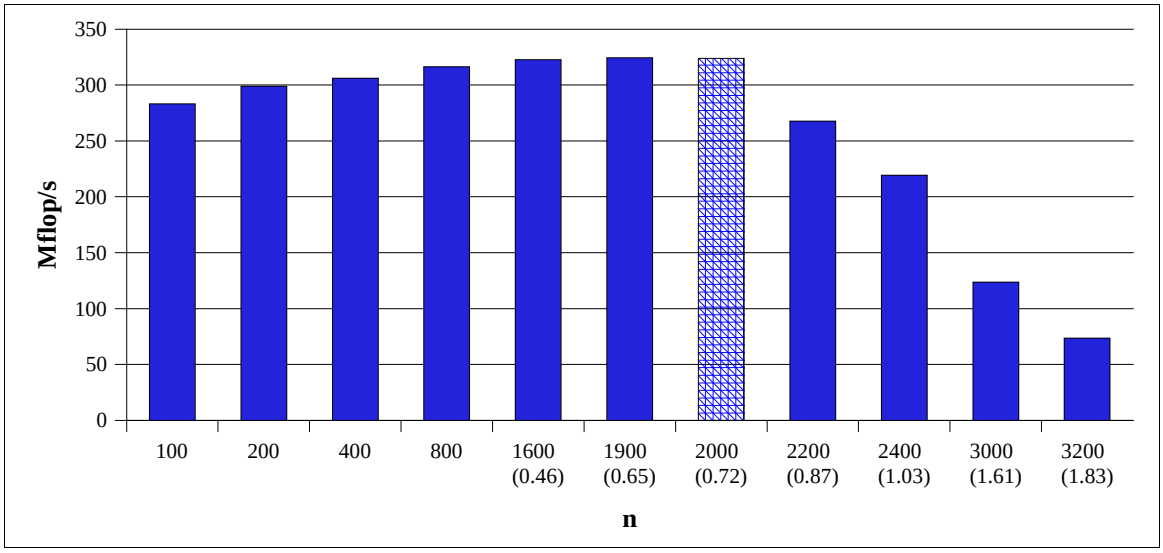


Figura 4.5: Rendimiento de **purmamarca** para Multiplicación de Matrices.

donde:

- Se indica entre paréntesis la proporción de memoria necesaria para contener las tres matrices. Para n = 1900, por ejemplo, se necesita el 65% de la memoria para almacenamiento de datos.
- Se muestra resaltado (con el relleno distinto de la barra correspondiente) el mayor tamaño de matrices para el cual no se utiliza espacio de swap. Se debe notar que cuando el espacio para contener los datos de las matrices excede el 72% de la memoria ya se comienza a utilizar el espacio de swap y por lo tanto el rendimiento disminuye.

4.2.3 Red Local del LQT

Los valores de rendimiento obtenidos para las computadoras de la red local del LQT se muestran en la Figura 4.6, donde

- Las computadoras **lqt_07** y **lqt_06** son las más veloces y las diferencias relativas no son

tan grandes como en el caso de las computadoras del CeTAD.

- Dado que no hay problema de espacio para la representación de las barras, se incluyen todas las máquinas aunque **lqt_06** es igual a **lqt_07** y **lqt_03** es igual a **lqt_04**. Debe notarse que en el contexto de las PCs construidas por partes, aunque las computadoras sean "iguales" en cuanto a procesador, frecuencia de reloj del sistema y cantidad de memoria instalada, aún es posible que tengan distinto rendimiento porque, por ejemplo, tienen distinta velocidad de acceso a memoria.
- Una vez más la frecuencia de reloj de operación no necesariamente determina la velocidad relativa de las máquinas (ver Tabla 4.2).

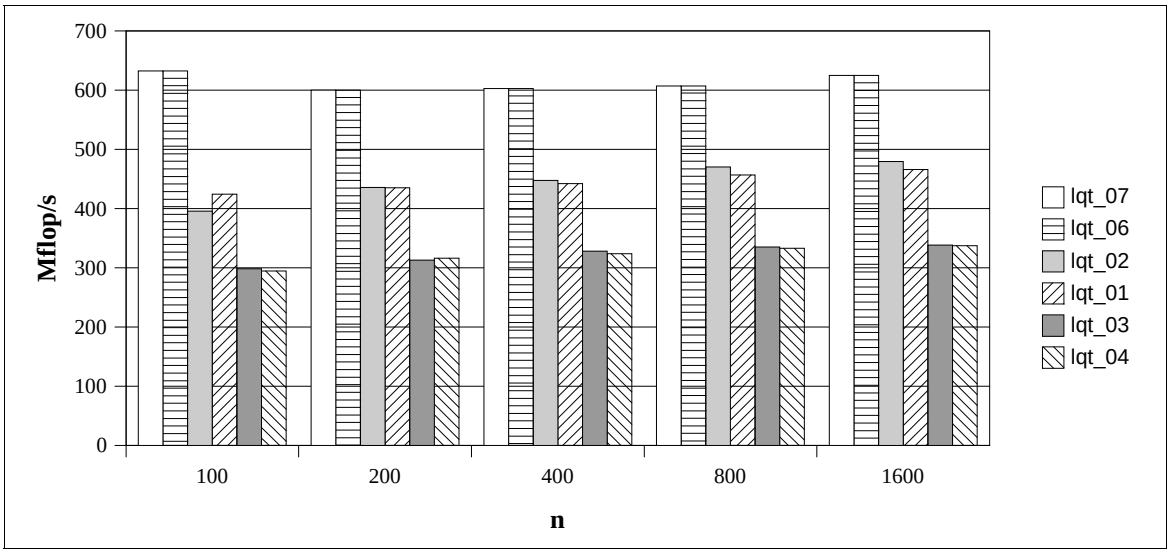


Figura 4.6: Rendimiento de las Computadoras del LQT.

En la computadora con mayor capacidad de procesamiento, **lqt_07**, se llevaron a cabo experimentos con matrices mayores y los resultados obtenidos se muestran en la Figura 4.7, donde se puede notar que:

- El mayor tamaño de matrices en el cual no se utiliza espacio de swap durante el procesamiento es n = 5000. La cantidad de memoria que se debe destinar al almacenamiento de los datos de las matrices de 5000×5000 elementos representa aproximadamente el 56% del total de memoria.
- Nuevamente se reduce el rendimiento a medida que es necesario utilizar mayor espacio de memoria swap aunque esta caída no es tan abrupta en relación a la que se produce en **purmamarca** (Figura 4.5).
- Muy probablemente, para matrices de 100x100 elementos la mayoría de los datos puedan ser alojados en la/s memoria/s cache/s del procesador y por lo tanto se tiene mejor rendimiento que para las matrices de, por ejemplo 200x200 y 400x400 elementos. Este efecto se *diluye* a medida que las matrices son mayores y por lo tanto siempre se logra optimizar el acceso a los datos en cache (reutilizarlos tantas veces como es posible) por el procesamiento por bloques de datos que utiliza el código optimizado.

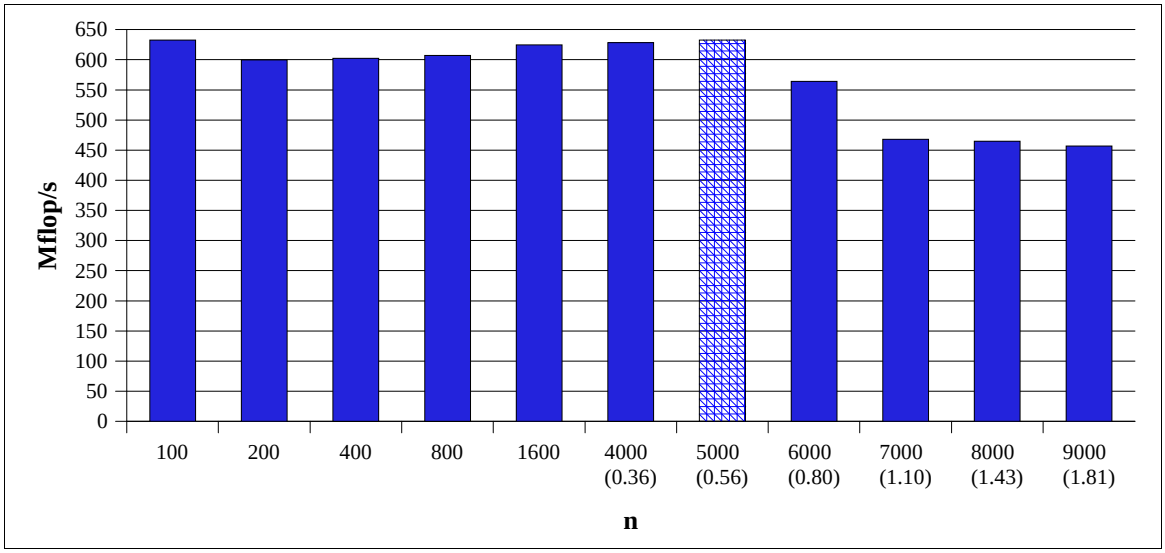


Figura 4.7: Rendimiento de **lqt_07** para Multiplicación de Matrices.

4.2.4 Red Local del LIDI

Dada la homogeneidad de las máquinas del LIDI, se muestran los resultados obtenidos en una de las máquinas, **lidipar_14**, para todos los tamaños de matrices en la Figura 4.8.

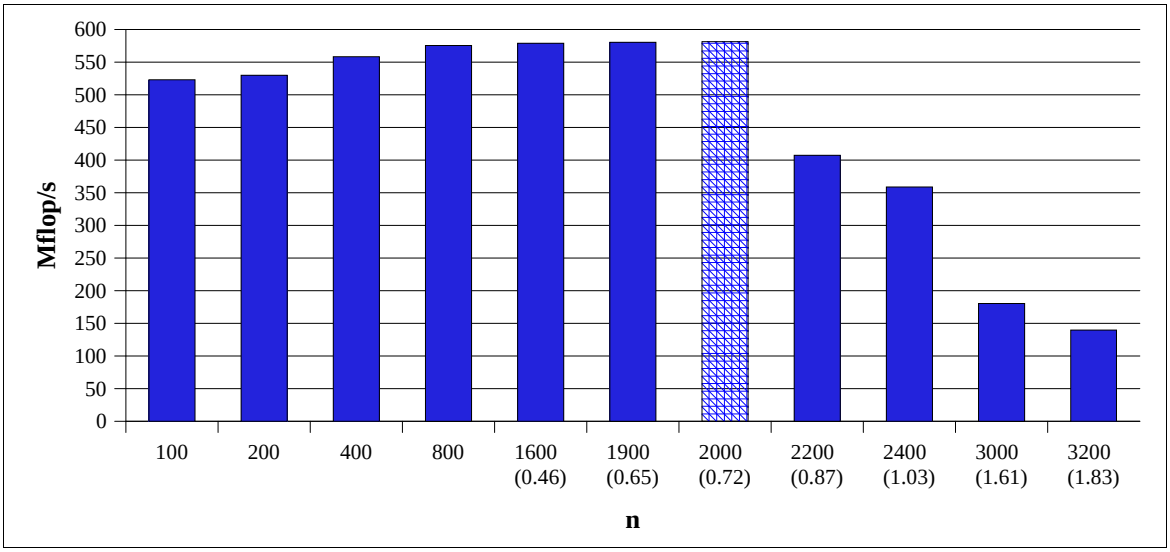


Figura 4.8: Rendimiento de **lidipar14** para Multiplicación de Matrices.

El *comportamiento* del rendimiento de las computadoras de la red del LIDI es similar al de **purmamarca** de la red local del CeTAD dado que:

- El mayor tamaño de matrices para el cual no se utiliza espacio de memoria swap durante el procesamiento de la multiplicación de matrices es n = 2000, lo cual es consistente dado que tienen la misma memoria principal instalada: 64 MB.

- A partir de que se comienza a utilizar espacio de memoria swap durante el procesamiento de la multiplicación de matrices el rendimiento cae abruptamente.

A diferencia de **purmamarca**, las computadoras de la red local del LIDI son bastante más veloces, dado que llegan a procesar a razón de más de 580 Mflop/s mientras que **purmamarca** no llega a 350 Mflop/s.

Dado que las computadoras de la red local del LIDI son todas iguales, se las puede utilizar, y de hecho se las utiliza como referencia del algoritmo de multiplicación de matrices en redes homogéneas. También esta red es la más apropiada para cómputo paralelo ya que la la red de interconexión es de 100 Mb/s (las otras dos son de 10 Mb/s) y además en el cableado se utiliza un switch en vez de uno o más hubs.

4.3 Análisis de Rendimiento Paralelo de las Redes Locales

Siempre se ha buscado definir y calcular analíticamente el rendimiento óptimo o posible de las computadoras paralelas y también el rendimiento que se puede obtener con un algoritmo paralelo sobre una computadora paralela en particular. Las razones más importantes para calcular analíticamente el mejor rendimiento posible de una computadora paralela son:

- Estimar de antemano si la computadora paralela es capaz de proporcionar un resultado en un tiempo determinado. No sería útil, por ejemplo, tener en dos semanas la predicción del estado meteorológico de un día de la semana siguiente.
- Evaluar y comparar máquinas paralelas para, por ejemplo, calcular la relación entre el costo y el beneficio (o costo/rendimiento) de cada una de ellas.

Por otro lado, estimar analíticamente el rendimiento de un algoritmo paralelo es útil para determinar si el algoritmo diseñado puede obtener el máximo rendimiento de la computadora para la cual se implementa y donde se ejecuta para resolver el problema un dado.

Dado que ya se dispone de

- Las características de rendimiento de cada una de las máquinas de todas las redes locales.
- Las características de rendimiento de la red de interconexión de cada una de las redes locales.
- Las principales características de los algoritmos de cómputo paralelo para el cálculo de la multiplicación de matrices (con y sin mensajes solapados, por ejemplo).

Se puede calcular analíticamente el mejor valor posible para el índice de speedup y utilizarlo luego como referencia en la evaluación de los resultados de experimentación. En cierta forma, el cálculo analítico del mejor valor de speedup (o speedup óptimo) trata de predecir el rendimiento de las redes de computadoras utilizadas como máquinas paralelas.

4.3.1 Cálculo del Speedup *Real*

La idea básica del índice de speedup de una computadora paralela es determinar cuántas veces más capacidad tiene una computadora paralela con respecto a un procesador, o a una computadora secuencial. La definición *clásica* del speedup es

Tiempo de ejecución del mejor algoritmo secuencial / Tiempo de ejecución paralelo

Por lo tanto, habría que determinar el *Tiempo de ejecución del mejor algoritmo secuencial* y también el *Tiempo de ejecución paralelo*. En el ambiente heterogéneo de las redes de computadoras instaladas el

Tiempo de ejecución del mejor algoritmo secuencial

se "transforma" en [147]

Tiempo de ejecución del mejor algoritmo secuencial en la computadora más rápida

o, de forma resumida,

Tiempo de ejecución en la computadora más rápida

asumiendo directamente que siempre se utilizará el mejor algoritmo. Básicamente se trata del mejor tiempo de ejecución secuencial posible en la red de estaciones de computadoras, es decir utilizando

- la computadora con mayor capacidad de cálculo y
- el mejor algoritmo secuencial.

En el caso de las tres redes locales que se presentaron esta tarea ya está resuelta, dado que:

- En la red local del CeTAD, **purmamarca** es la de mayor velocidad relativa y la experimentación ya ha determinado su capacidad en Mflop/s que a su vez determina unívocamente el tiempo de cómputo.
- En la red local del LQT, **lqt_07** es la de mayor velocidad relativa y la experimentación ya ha determinado su capacidad en Mflop/s que a su vez determina unívocamente el tiempo de cómputo.
- En la red local del LIDI son todas las computadoras iguales y la experimentación ya ha determinado la capacidad en Mflop/s de **lidipar14** que a su vez determina unívocamente el tiempo de cómputo.

De la misma manera, el *tiempo de ejecución paralelo* se determina vía experimentación, utilizando las máquinas disponibles de cada una de las redes locales.

4.3.2 Cálculo del Speedup *Optimo*

Desde el punto de vista teórico, lo mejor que puede suceder en una máquina paralela es que todos los procesadores se utilicen todo el tiempo o que todos los procesadores se utilicen a

su máxima capacidad de cómputo. Esto induce a asumir que la capacidad de cálculo de la computadora paralela es igual a la suma de las capacidades de cálculo de cada uno de los procesadores que son parte de la misma. En el contexto de las máquinas paralelas con procesadores homogéneos esto significa que utilizar un procesador más indica reducir proporcionalmente el tiempo de ejecución paralelo. Es decir que si se utilizan *P* procesadores, el mejor tiempo de ejecución paralelo está dado por

Tiempo de ejecución paralelo = Tiempo de ejecución del mejor algoritmo secuencial / P

Y de hecho, no es más que asumir que la potencia de cálculo de la máquina paralela con *P* procesadores es *P* veces mayor que la potencia de cálculo de la máquina secuencial (un procesador). Puesto de otra forma, el speedup consiste en identificar la relación entre la potencia de una máquina con un procesador y una máquina paralela con *P* procesadores. De esta manera se llega a que el speedup óptimo en las computadoras paralelas *clásicas* homogéneas es igual a la cantidad de procesadores que se utilizan. Esto es equivalente a definir la "potencia de cálculo relativa de la máquina paralela con respecto a un procesador" o directamente el valor del speedup óptimo como

$$SpeedupOptimo = \sum_{i=0}^{P-1} rpw(proc_i) \tag{4.1}$$

donde $proc_0$, $proc_1$, ..., $proc_{P-1}$, son los *P* procesadores de la máquina paralela y $pw(proc_i)$ es la potencia de cálculo relativa de $proc_i$ con respecto a los demás o a cualquiera de los demás procesadores. En el contexto de las aplicaciones numéricas se puede calcular utilizando la capacidad de cálculo en Mflop/s como en el capítulo anterior, pero asumiendo que los procesadores son iguales se cumple que

$$rpw(proc_i) = 1; \quad \forall\ i = 0, ..., P\text{-}1 \tag{4.2}$$

y por lo tanto

$$SpeedupOptimo = P \tag{4.3}$$

En el contexto de las aplicaciones numéricas, esto es equivalente al cálculo del speedup óptimo utilizando directamente las potencias de cálculo de la máquina secuencial y la máquina paralela en Mflop/s, es decir como

$$SpeedupOptimo = \frac{\sum_{i=0}^{P-1} Mflop/s(proc_i)}{Mflop/s(proc_0)} \tag{4.4}$$

En el contexto de las máquinas paralelas con procesadores heterogéneos no es posible afirmar lo expresado en la Ecuación (4.2) ya que los procesadores tienen o pueden tener distinta velocidad relativa. Por lo tanto, se debe calcular *SpeedupOptimo* de acuerdo con la

Ecuación (4.1), es decir utilizando el cálculo de cada $rpw(proc_i)$ o, como se denomina de manera equivalente en el capítulo anterior, $pw(ws_i)$, dada por

$$pw(ws_i) = \frac{Mflop/s(ws_i)}{\underset{j=0..P-1}{máx}\ (Mflop/s(ws_j))} \tag{4.5}$$

Y de manera análoga, la Ecuación (4.3) se debe adaptar al ambiente heterogéneo como lo muestra la Ecuación (4.6), que determina como referencia al procesador con mayor capacidad de cálculo entre todos los utilizados.

$$SpeedupOptimo = \frac{\sum_{i=0}^{P-1} Mflop/s(ws_i)}{\underset{j=0..P-1}{máx}\ (Mflop/s(ws_j))} \tag{4.6}$$

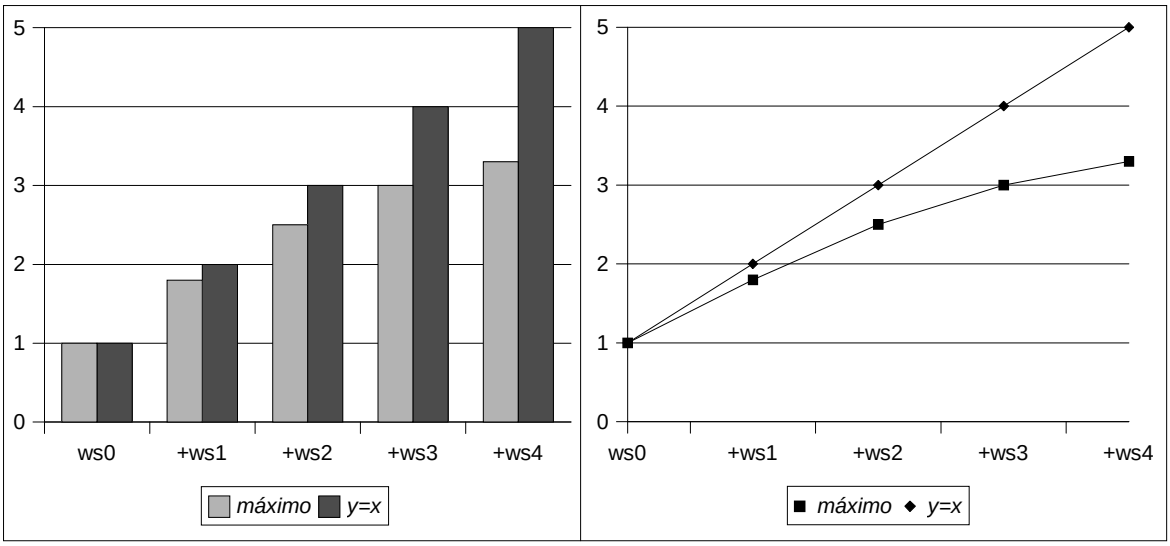
De esta manera, también se *pierden* dos ideas subyacentes en la interpretación de los gráficos de speedup que provienen de las máquinas paralelas homogéneas:

- Ya no se puede afirmar que el máximo teórico está dado por la recta $y = x$, o que para x cantidad de procesadores el máximo teórico del speedup está dado por x. En el ambiente heterogéneo ya no es posible relacionar la cantidad de procesadores con la potencia de cálculo de la máquina paralela completa. Más específicamente, agregar un procesador implica agregar potencia de cálculo pero no necesariamente relacionada con la cantidad total de procesadores sino con la suma de las potencias de cálculo de los procesadores.
- Ya no se puede afirmar que como mínimo debería lograrse speedup lineal, o que aunque el speedup no sea exactamente igual a la cantidad de procesadores, debería ser directamente proporcional a la cantidad de procesadores. Ya no es posible mantener esta idea por la misma razón dada anteriormente: no es posible relacionar o cuantificar la relación entre la cantidad de procesadores (o máquinas) con la potencia de cálculo de la máquina paralela.

La Figura 4.9 muestra los valores de speedup máximos en una red de cinco computadoras ws0, ..., ws4, cada una con su potencia de cálculo relativa dada por

pw(ws0) = 1.0 pw(ws1) = 0.8 pw(ws2) = 0.7
pw(ws3) = 0.5 pw(ws4) = 0.3

En la Figura 4.9-a) los valores de speedup se muestran con barras y en la Figura 4.9-b) los valores se unen con líneas, mostrando un poco más claramente cómo el máximo speedup posible para estas cinco computadoras se “aleja” de la recta $y = x$ a medida que se agregan computadoras. También la Figura 4.9 muestra la tendencia a incorporar las computadoras más rápidas de las que están disponibles. En el caso de utilizar las computadoras ws0 y ws1, la que más probablemente se incorpore es ws2 ya que es la que tiene mayor capacidad de cálculo de las tres disponibles: ws2, ws3 y ws4, es por eso que en los gráficos de speedup se incorporan las máquinas de “mayor a menor” de acuerdo a su capacidad de cálculo, a menos que explícitamente se determine otro criterio.



a) b)

Figura 4.9: Speedup de Cinco Computadoras Heterogéneas.

Es importante notar que esta forma de calcular el máximo valor de speedup asume que *todas* las computadoras y en particular la más veloz tienen *siempre* la misma capacidad de cálculo. Cuando las computadoras se utilizan a su máxima capacidad configurada en cuanto a espacio de swap es posible y de hecho muy probable que se puedan resolver tamaños de problemas mayores de los que permite la memoria principal disponible y se utilice el espacio de swap. Esto a su vez ocasiona dos inconvenientes desde el punto de vista del rendimiento que son bien conocidos:

1. Mientras se lleva a cabo cálculo hay mayor actividad del sistema operativo por el manejo de los datos que deben transferirse desde y hacia el espacio de swap (normalmente en disco).
2. Es posible y de hecho muy probable que el procesador deba esperar por datos que están en espacio de swap (disco) hasta que se transfieran a memoria principal desde donde el procesador puede utilizarlos para operar con ellos.

Y de hecho, el rendimiento se degrada notablemente, aunque la cuantificación de esta degradación depende de la computadora (velocidad de disco, subsistema de entrada/salida, etc.) y también del problema (patrón de acceso a los datos, cantidad de datos en espacio de swap, etc.)

Por lo tanto, si el problema resuelto en la computadora secuencial o en la computadora con mayor capacidad de cálculo en el caso de las redes locales, implica la utilización de espacio de swap se tendrá como referencia un tiempo de ejecución afectado por la utilización del espacio de swap. Cuando se considera la posibilidad de ejecutar en paralelo en las computadoras de una red local, de hecho se está implícitamente asumiendo la distribución de los datos del problema y por lo tanto es posible que el mismo tamaño de problema se pueda resolver de forma tal que en cada máquina solamente se utiliza la memoria principal y no se recurre al espacio de memoria swap con la degradación de rendimiento que eso implica. Por lo tanto, dependiendo del tamaño del problema y de las máquinas utilizadas, el valor de speedup que se puede obtener puede ser mayor que el óptimo calculado de

acuerdo con la Ecuación (4.1). Puesto de otra manera, el valor óptimo de speedup calculado según la Ecuación (4.1) asume que la computadora más veloz tiene *siempre* la capacidad de cálculo determinada con la ejecución de parte del problema en espacio de swap, o lo que es similar, es más lenta de lo que *realmente* es (cuando todo el problema puede manejarse en memoria principal sin recurrir al espacio de swap).

Dado que se realizan experimentos secuenciales que implican la utilización del espacio de swap, se debería proporcionar un valor óptimo para el speedup que no sea "desviado" por esta razón. En el contexto de los problemas numéricos, conviene recurrir una vez más a la idea de capacidad de cálculo dada en cantidad de operaciones de punto flotante por segundo, o Mflop/s. Dado que se utilizan directamente los Mflop/s para esta "nueva" forma de cálculo del speedup óptimo también se deben tener en cuenta la cantidad de operaciones que se tienen que realizar. Retomando el ejemplo de las cinco computadoras ws0, ..., ws4, anterior, se debería utilizar ahora la capacidad de cada una de ellas en términos de Mflop/s que pueden ser, por ejemplo

Mflop/s(ws0) = 1000
Mflop/s(ws1) = 800
Mflop/s(ws2) = 700
Mflop/s(ws3) = 500
Mflop/s(ws4) = 300

Y también ahora es necesario conocer la cantidad de operaciones de punto flotante que se necesitan llevar a cabo para resolver el problema, que pueden ser, por ejemplo, 10^9. Por lo tanto, si la computadora con mayor capacidad, ws0, puede resolver el problema sin recurrir al espacio de swap, entonces resuelve los cálculos a su máxima capacidad, es decir a razón de 1000×10^6 operaciones por segundo. En este caso, el máximo speedup "coincide" con el calculado utilizando la Ecuación (4.1), es decir:

$$SpeedupOptimo = \sum_{i=0}^{4} pw(proc_i) = \sum_{i=0}^{4} \frac{Mflop/s(ws_i)}{1000} = 3.3$$

Si, por el contrario, la computadora con mayor capacidad, ws0, utiliza el espacio de swap durante la resolución del problema, ya no realiza los cálculos a su máxima velocidad. Suponiendo que la degradación por la utilización del espacio de swap durante los cálculos es del 30%, esto implica que realiza los cálculos a razón de 700×10^6 operaciones por segundo, por lo tanto, asumiendo que durante la ejecución paralela todas las computadoras operan a su máxima capacidad, utilizando la Ecuación (4.3) se tiene que el speedup óptimo sería

$$SpeedupOptimo = \frac{\sum_{i=0}^{4} Mflop/s(ws_i)}{700} = 4.71$$

que evidentemente es mayor al calculado según los valores *pw*(ws0), ..., *pw*(ws4). En todo caso, se tendrían dos puntos de vista para el cálculo del speedup óptimo. El calculado

según según la Ecuación (4.1), que a su vez depende de las potencias de cálculo relativas, $pw(ws_i)$, calculadas según la Ecuación (4.5), que asume que las computadoras tienen siempre la misma capacidad de cálculo, y que se podría llamar "speedup óptimo de cómputo según las velocidades relativas", o Comp(rsf).

$$\text{Comp}(\text{rsf}) = \sum_{i=0}^{P-1} pw(proc_i) \tag{4.7}$$

La idea subyacente sobre la que se apoya el cálculo de Comp(rsf) es básicamente: si se agrega una máquina, se agrega su potencia de cálculo relativa a la de mayor capacidad. Siguiendo el ejemplo dado con ws0, ..., ws4, esto significa que si en vez de utilizar solamente ws0 se utilizan ws0 y ws1 entonces se debería tener una computadora (paralela) que tiene 1.8 veces la capacidad de ws0.

El speedup óptimo calculado según la Ecuación (4.3) asume que todas las computadoras ejecutan siempre al máximo de su capacidad, independientemente de que sea necesaria la utilización del espacio de swap durante el procesamiento o no. Por lo tanto, se podría llamar "speedup óptimo de cómputo según las capacidades de cálculo de cada computadora dadas en Mflop/s", o Comp(Mf).

$$\text{Comp}(\text{Mf}) = \frac{\sum_{i=0}^{P-1} Mflop/s(ws_i)}{\underset{j=0..P-1}{máx}\ (Mflop/s(ws_j))} \tag{4.8}$$

La idea subyacente sobre la que se apoya el cálculo de Comp(Mf) es básicamente: si se agrega una máquina, se agrega directamente su potencia de cálculo en Mflop/s. Siguiendo el ejemplo dado con ws0, ..., ws4, esto significa que si en vez de utilizar solamente ws0 se utilizan ws0 y ws1 entonces se debería tener una computadora (paralela) que tiene una capacidad de cálculo de 1000+800 Mflop/s = 1800 Mflop/s, lo que significa que la computadora paralela es 1800/700 ≅ 2.57 veces la capacidad de ws0, dado que el valor de referencia en cuanto a Mflop/s de ws0 es 700 Mflop/s, que se tiene utilizando el espacio de swap.

Evidentemente el cálculo del speedup óptimo será el mismo, Comp(rsf) = Comp(Mf) si se toman como referencia los valores de Mflop/s máximos, es decir sin que se utilice el espacio de swap de cada máquina. Comp(rsf) es derivado directamente de la forma clásica del cálculo de speedup y Comp(Mf) sería el que hay que observar con cierta atención cuando el tiempo de ejecución secuencial está afectado por la utilización del espacio de swap. De alguna manera, cuando durante la ejecución secuencial se utiliza el espacio de memoria swap, Comp(Mf) podría entenderse como lo que se ha llamado ocasionalmente "speedup superlineal". Siguiendo con el ejemplo desde este punto de vista:

- Al resolver un problema en ws0 se tiene el tiempo de ejecución derivado de la capacidad de ws0 al utilizar espacio de swap, es decir directamente proporcional a 700 Mflop/s.
- Al resolver el mismo problema con cómputo paralelo y utilizar ws1, es "esperable" que

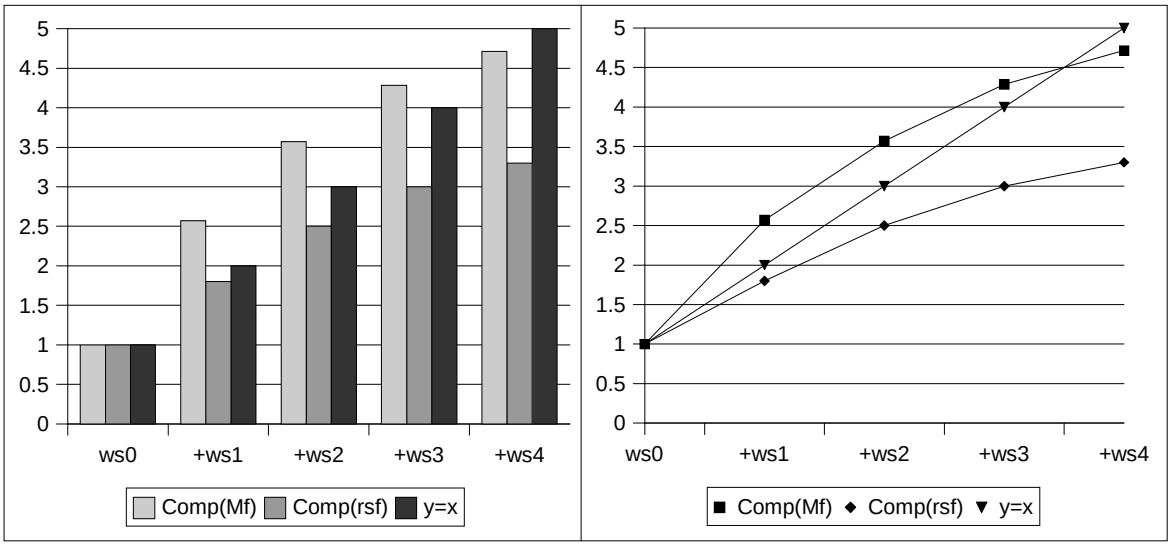
se resuelva el problema considerando una computadora con una capacidad de (700 + 0.8×700) Mflop/s = 1260 Mflop/s dado que ws1 tiene una capacidad de cálculo de 0.8 veces la capacidad de ws0.

- Si la distribución de todo el problema entre ws0 y ws1 hace innecesaria la utilización del espacio de swap, se tendrá que ambas computadoras resuelven los cálculos a su máxima capacidad y por lo tanto el tiempo de ejecución paralelo será proporcional a (1000+800) Mflop/s = 1800 Mflop/s y por lo tanto el tiempo de ejecución será menor que el "esperable" considerando solamente las velocidades relativas.

La Figura 4.10 muestra los diferentes valores de speedup óptimos, Comp(Mf) y Comp(rsf), considerando las cinco computadoras del ejemplo, cuyas características de rendimiento están resumidas en la siguiente tabla

Computadora	***Mflop/s (Máximo)***	***pw***
ws0	1000	1
ws1	800	0.8
ws2	700	0.7
ws3	500	0.5
ws4	300	0.3

y además considerando que la computadora con mayor capacidad de cálculo utiliza el espacio de swap para resolver el problema dado con su consiguiente penalización en rendimiento del 30%.



a) b)

Figura 4.10: Cálculos de Speedup Optimos para Cinco Computadoras.

En la Figura 4.10-a) los valores se muestran con barras y en la Figura 4.10-b) los valores se unen con líneas, donde se puede ver claramente que:

- Las diferentes formas de cálculo de speedup óptimo proporcionan distintos valores en el

caso de que la solución secuencial implica la utilización del espacio de swap.
- Se confirma que la recta $y = x$ no proporciona información significativa en el contexto de procesadores heterogéneos.

Por último, se debe hacer notar que en el cálculo del speedup óptimo no hay ninguna consideración respecto de las comunicaciones, siempre se tiene en cuenta *solamente* la capacidad de cálculo de todos los procesadores (computadoras) utilizadas.

4.4 Análisis de Rendimiento de los Algoritmos

En general, el cálculo analítico del rendimiento de los algoritmos paralelos tiene dos propósitos generales:
- Determinar si el algoritmo es capaz de aprovechar el rendimiento de la computadora paralela sobre la cual puede implementarse.
- Comparar y evaluar distintos algoritmos diseñados para una misma tarea.

El cálculo analítico de rendimiento de los algoritmos tiene (o debe tener) en cuenta no solamente las características de cómputo de las computadoras sino que también incorpora como mínimo otro factor que afecta el rendimiento: las comunicaciones. Aunque se conozca la arquitectura de una computadora paralela con todos los detalles necesarios, es muy difícil cuantificar el impacto que tienen las comunicaciones sobre el rendimiento de las aplicaciones que se resuelven. Sin embargo, la situación cambia al diseñar un algoritmo paralelo específico dado que éste determina con claridad todo lo necesario relacionado con las comunicaciones y la sincronización entre los procesos. En general, se consideran los puntos de sincronización entre procesos como una clase de comunicación en particular.

Aunque son muy similares, los algoritmos a considerar para analizar son dos, y fueron presentados en el capítulo anterior:
- Con los mensajes explícitamente secuenciales respecto del cómputo. Es decir que en todo instante de tiempo una computadora puede estar haciendo una de dos tareas:
  - Cómputo local, es decir resolviendo un cálculo parcial de la porción de la matriz resultado que debe computar.
  - Resolviendo comunicaciones de datos, más específicamente enviando o recibiendo un mensaje broadcast.
- Con los mensajes solapados, de forma tal que la mayoría de los mensajes se pueden transmitir mientras se lleva a cabo cómputo local (*simultáneamente*). Para que esto ocurra en realidad cada computadora debe ser capaz de realizar cómputo y transmisiones de datos a la vez.

Ambos algoritmos se presentaron en el capítulo anterior, junto con la forma analítica para el cálculo del rendimiento de cada uno de ellos y se denominarán SeqMsg y OverMsg respectivamente. En todos los casos, se asume que los períodos de cómputo se llevan a cabo a la máxima capacidad de cálculo de las computadoras involucradas.

4.4.1 SeqMsg: Cómputo y Comunicaciones Secuenciales

De acuerdo con lo explicado en el capítulo anterior, el tiempo paralelo del algoritmo en el cual los períodos de cómputo y comunicaciones se llevan a cabo de forma secuencial está dado por

$$t_{par\_seqmsg} = P\alpha + \beta n^2 + \frac{2n^3 - n^2}{pw}$$

donde

- P es la cantidad de computadoras que se utilizan.
- n es el orden de las matrices cuadradas que se multiplican.
- α es el tiempo de latencia de la red de comunicaciones.
- $1/\beta$ es el ancho de banda (tasa de transferencia) asintótico de la red de comunicaciones, expresado en términos del tipo de elementos de las matrices que se multiplican.
- pw es la suma de todas las capacidades de cálculo de las computadoras utilizadas expresadas en términos de Mflop/s, es decir

$$pw = \sum_{i=0}^{P-1} Mflop/s(ws_i) \tag{4.9}$$

Aunque no explícitamente definido, se tiende a asumir que

- El tiempo de latencia no es demasiado importante siempre y cuando los mensajes sean suficientemente grandes, o lo que es lo mismo, cuando el problema es suficientemente grande, dado que el tamaño de los mensajes está directamente relacionado con el tamaño del problema [71] [52] [146].
- El ancho de banda asintótico de la red de comunicaciones es independiente de la cantidad de procesos que se comunican con un broadcast o, más específicamente, la cantidad de procesos receptores de cada mensaje broadcast. Esto es posible en las redes Ethernet siempre y cuando las rutinas de comunicaciones aprovechen las características del hardware de comunicaciones.

Por lo tanto, se puede simplificar un poco la forma de cálculo del tiempo de ejecución paralelo eliminando completamente el tiempo de latencia de los mensajes (o, lo que es igual, considerándolo igual a cero) y se llega a

$$t_{par\_seqmsg} = \beta n^2 + \frac{2n^3 - n^2}{pw}$$

Y este tiempo paralelo es el que se utiliza para el cálculo del speedup óptimo que este algoritmo puede obtener en una red de computadoras, dando lugar a lo que se denominará **SeqMsg(Mf)**.

4.4.2 OverMsg: Cómputo y Comunicaciones Solapadas

De acuerdo con lo explicado en el capítulo anterior, el tiempo paralelo del algoritmo en el cual gran parte de los períodos de cómputo y comunicaciones se llevan a cabo de forma solapada (*simultánea*) está dado por

$$t_{par\_overmsg} = t_{bcast} + (P\text{-}1)\ máx(t_{bcast}, t_{cómp}) + t_{cómp}$$

donde

$$t_{bcast} = \alpha + \beta\, n^2/P \qquad \text{y} \qquad t_{cómp} = \frac{2n^3 - n^2}{P\ pw}$$

Y este tiempo paralelo es el que se utiliza para el cálculo del speedup óptimo que este algoritmo puede obtener en una red de computadoras, dando lugar a lo que se denominará **OverMsg(Mf)**.

Con esta forma de calcular analíticamente el tiempo de ejecución paralelo se asume que:
- Todas las computadoras son capaces de solapar cómputo con comunicaciones.
- El solapamiento de las comunicaciones no afecta el tiempo de cómputo local ni el tiempo de comunicaciones de los mensajes broadcast.

Es de destacar que ambas suposiciones son muy difíciles de verificar al menos en las computadoras estándares de las redes locales instaladas.

4.5 Redes Locales y Algoritmos

Dado que ya se dispone de:
1. rendimiento secuencial de todas las computadoras de todas las redes locales: Mflop/s(ws_i)
2. forma analítica de calcular el rendimiento de cada una de las redes locales: Comp(Mf) y Comp(rsf).
3. forma analítica de los dos algoritmos propuestos: t_{par_seqmsg} y $t_{par\_overmsg}$.
4. estimación, a menos a nivel de hardware, del ancho de banda asintótico de cada una de las redes locales: Mb/s de las redes Ethernet.

ya es posible estimar el rendimiento tanto de cada una de las redes como de los algoritmos sobre estas redes, al menos en lo que se refiere al máximo *teórico*.

4.5.1 Red Local del CeTAD

La Figura 4.11 muestra los máximos valores de speedup a considerar en la red local del CeTAD cuando todos los datos del problema se pueden contener en memoria principal, de forma tal que no es necesaria la utilización de memoria swap durante los cálculos. En este caso, la computadora de referencia (la de mayor capacidad de cálculo) es **purmamarca** y el

tamaño de matrices es n = 2000. Además, dado que se utiliza una red Ethernet de 10 Mb/s, se asume que por la degradación producida por todas las capas del sistema operativo se pueden transferir datos entre procesos de usuario a razón de 2^{20} bytes (1 MB) por segundo. Podría considerarse una suposición optimista, pero aceptable dado que se estiman valores máximos.

En la Figura 4.11 se puede notar que:

- La capacidad de cómputo relativa de las computadoras proporciona un orden razonable para ser utilizadas en paralelo. Cuando se usa una cantidad determinada de computadoras para cómputo paralelo se tiende a incluir las de mayor capacidad de cálculo entre las disponibles.
- Las computadoras **cetadfomec1** y **cetadfomec2** se muestran con los nombres **cf1** y **cf2** respectivamente.
- Como se esperaba, Comp(Mf) y Comp(rsf) coinciden, dado que la capacidad de cómputo de referencia de **purmamarca** es la máxima (aproximadamente 324 Mflop/s, Figura 4.5), porque no utiliza el espacio de swap durante la ejecución secuencial.
- El speedup calculado para el algoritmo con los mensajes solapados OverMsg(Mf) es similar al de cómputo, Comp(Mf), hasta utilizar la computadora **sofia** inclusive, pero agregando más computadoras casi no hay mejora en cuanto a rendimiento. Expresado de otra manera, a partir de la incorporación de **fourier**, el tiempo de comunicaciones es mayor que el de cómputo y por lo tanto casi no hay mejora en cuanto a tiempo de ejecución paralelo por la incorporación de más computadoras.
- El peso relativo del tiempo de comunicación con respecto al tiempo de cómputo se evidencia para el algoritmo con los mensajes y cómputo secuenciales. El speedup calculado para este algoritmo, SeqMsg(Mf) así lo muestra por la diferencia en los valores con respecto a los demás cálculos de speedup, incluyendo el del algoritmo con los mensajes solapados, que tiene en cuenta al menos una parte del tiempo total de comunicaciones.
- El total de la capacidad de cálculo de las diez computadoras es poco menos que 4.5 veces la capacidad de **purmamarca**.

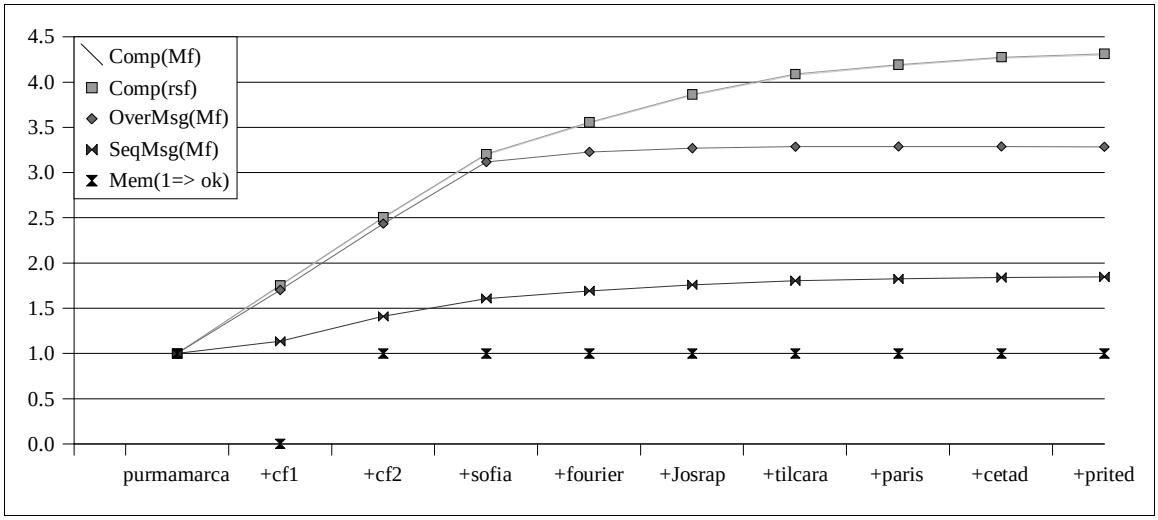


Figura 4.11: Análisis de Speedup de la Red del CeTAD para n = 2000.

También en la Figura 4.11 se muestra una estimación "empírica" de los requerimientos de memoria que se señala en el gráfico como Mem(1 => ok). Cuando se muestra con valor igual a 0 implica que es muy probable que en una o más computadoras se necesite recurrir al espacio de swap durante la ejecución. Cuando se muestra con valor igual a 1 indica que es muy probable que no sea necesario recurrir al espacio de swap en *ninguna* de las computadoras durante la ejecución del algoritmo. Nótese que en el gráfico se muestra con valor igual a 0 solamente cuando se utilizan dos computadoras en paralelo: **purmamarca** y **cf1**. Esto es así porque:

- Cuando se utiliza solamente **purmamarca**, que tiene 64 MB de memoria principal (Tabla 4.1), ya se conoce (de la experimentación misma) que no es necesaria la utilización del espacio de swap.
- Al incorporar **cf1**, que tiene 32 MB de memoria, es muy probable que **purmamarca** no necesite recurrir al espacio de swap pero **cf1** sí.
- Al incorporar **cf2**, ya son tres las máquinas entre las que se distribuyen los datos y a partir de aquí es muy probable que no haya inconvenientes por la memoria.

Aunque esta estimación de memoria no sea muy precisa (y de hecho es muy difícil hacer una que lo sea), siempre es útil tener al menos una idea de referencia dado que, como se ha aclarado antes, la capacidad de cálculo puede ser muy afectada.

La Figura 4.12 muestra los máximos valores de speedup a considerar en la red local del CeTAD para el máximo tamaño de problema que puede resolver la computadora de mayor capacidad de cálculo (recurriendo a la memoria swap). La computadora de referencia sigue siendo **purmamarca** y el tamaño de matrices es n = 3200. Como para la estimación anterior, se asume que se pueden transferir datos entre procesos de usuario a razón de 2^{20} bytes (1 MB) por segundo.

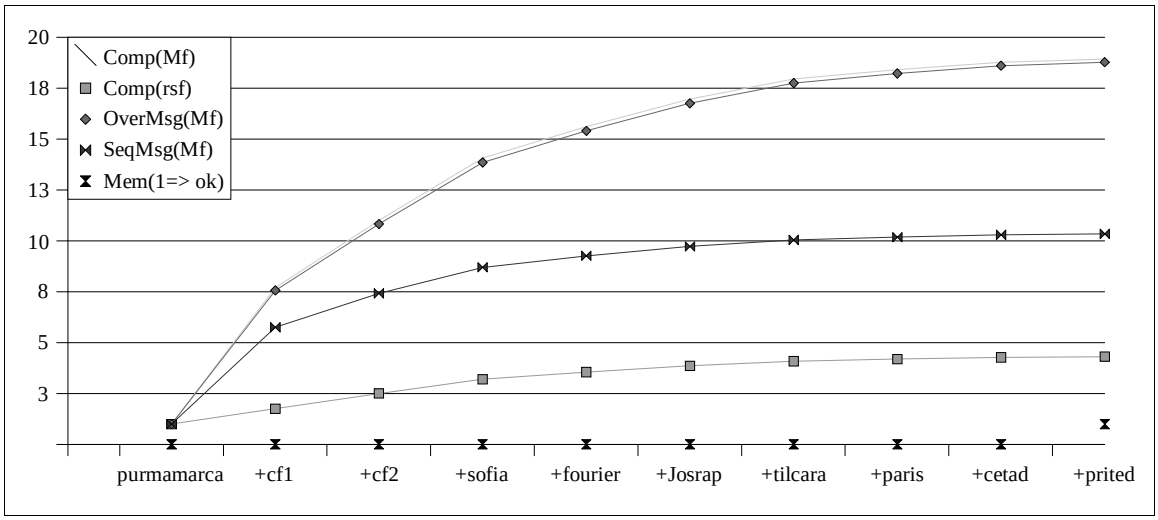


Figura 4.12:Análisis de Speedup de la Red del CeTAD para n = 3200.

Dado que el mayor tamaño de problema (con matrices de 3200×3200 elementos), que puede resolver **purmamarca** implica la utilización del espacio de swap, la capacidad de cálculo de referencia es aproximadamente 74 Mflop/s (Figura 4.5). Por lo tanto, ya no es posible que coincidan Comp(Mf) y Comp(rsf), más específicamente, el valor de speedup óptimo calculado con la capacidad máxima de cálculo (en Mflop/s) necesariamente será

mayor al valor de speedup óptimo teniendo en cuenta las velocidades relativas entre las computadoras. Esto implica que el máximo valor de speedup utilizando todas las máquinas del CeTAD (10 computadoras), da como resultado una computadora paralela que tiene casi 19 veces la capacidad de cálculo de **purmamarca** para matrices de 3200×3200 elementos, es decir cuando **purmamarca** tiene que recurrir al espacio de swap durante la ejecución.

Además, de la Figura 4.12 se desprende que:

- Los valores de Comp(rsf) no cambian con respecto a los de la Figura 4.11, dado que las velocidades relativas se asumen iguales.
- Asumiendo que cada computadora puede resolver efectivamente sus cálculos y comunicaciones de manera simultánea, los valores de OverMsg(Mf) son casi idénticos a los de Comp(Mf).
- El peso del tiempo de las comunicaciones sigue siendo relativamente alto respecto del tiempo de cómputo, y esto se identifica claramente por las diferencias entre los valores de Comp(Mf) y SeqMsg(Mf). Más específicamente, cuando se contabiliza el tiempo de comunicación además del tiempo de cómputo (tal como debe hacerse para el algoritmo con los períodos de transmisión de mensajes y cómputo resueltos de manera secuencial), los valores óptimos de speedup se reducen notablemente con respecto a los que se obtienen con la suma de las capacidades de cálculo.
- Dado que para n = 3200 **purmamarca** se "transforma" en una computadora con mucho menor capacidad de cálculo que para n = 2000, aún con cálculos y comunicaciones secuenciales se espera que la ganancia sea sustancial, y bastante mayor que la que es esperable de acuerdo con las velocidades relativas. Los valores de Comp(rsf) bastante menores que todos los demás así lo muestran.
- La estimación de requerimientos de memoria en cada computadora, Mem en el gráfico, muestra que recién con la utilización de todas las máquinas es poco probable que se requiera la utilización del espacio de memoria swap en todas las computadoras. Esto es debido básicamente a que dos de las computadoras con mayor capacidad de cálculo relativa, **cf1** y **cf2**, tienen poca capacidad de memoria también en términos relativos a la de mayor capacidad de cálculo (**purmamarca**).

4.5.2 Red Local del LQT

La Figura 4.13 muestra los máximos valores de speedup a considerar en la red local del LQT cuando todos los datos del problema se pueden contener en memoria principal, de forma tal que no es necesaria la utilización de memoria swap durante los cálculos. En este caso, la computadora de referencia (la de mayor capacidad de cálculo) es **lqt_07** y el tamaño de matrices es n = 5000. También en este caso, dado que se utiliza una red Ethernet de 10 Mb/s, se asume que por la degradación producida por todas las capas del sistema operativo se pueden transferir datos entre procesos de usuario a razón de 2^{20} bytes (1 MB) por segundo.

La computadora paralela *obtenida* utilizando la totalidad de la máquinas proporciona en el mejor de los casos poco más de 4.5 veces la capacidad de **lqt_07**, aproximadamente 2.9 Gflop/s. Una vez más, el cálculo analítico del máximo speedup posible de obtener con el algoritmo de cómputo y comunicaciones secuenciales muestra el peso relativo del tiempo

de comunicaciones con respecto al tiempo de cómputo, y se mantiene en valores aproximadamente iguales a la mitad de los valores de las demás estimaciones.

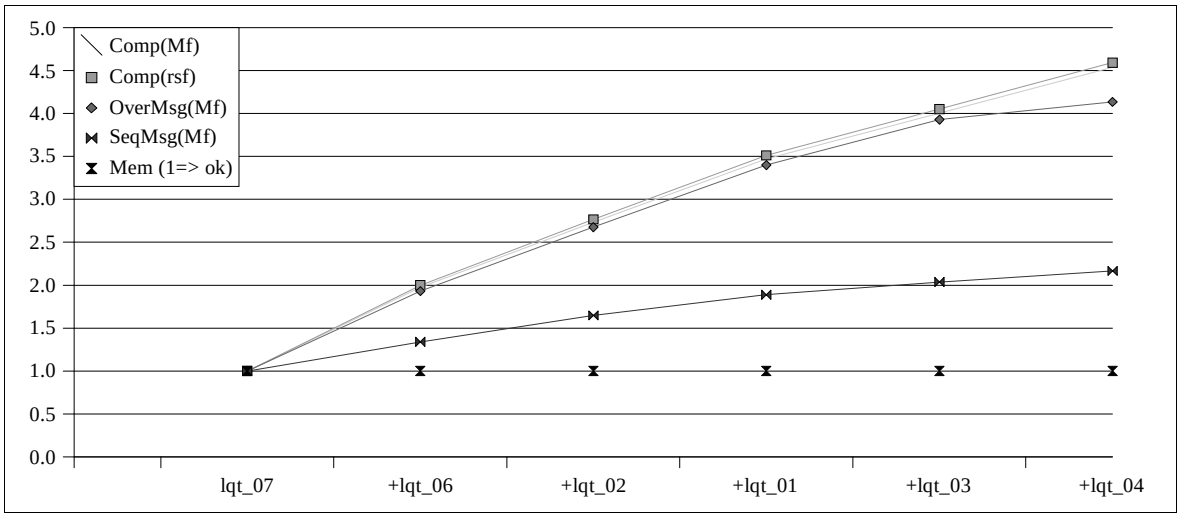


Figura 4.13: Análisis de Speedup de la Red del LQT para n = 5000.

La Figura 4.14 muestra los máximos valores de speedup a considerar en la red local del LQT para el máximo tamaño de problema que puede resolver la computadora de mayor capacidad de cálculo (recurriendo a la memoria swap). La computadora de referencia sigue siendo **lqt_07** y el tamaño de matrices es n = 9000. Como para la estimación anterior, se asume que se pueden transferir datos entre procesos de usuario a razón de 2^{20} bytes (1 MB) por segundo.

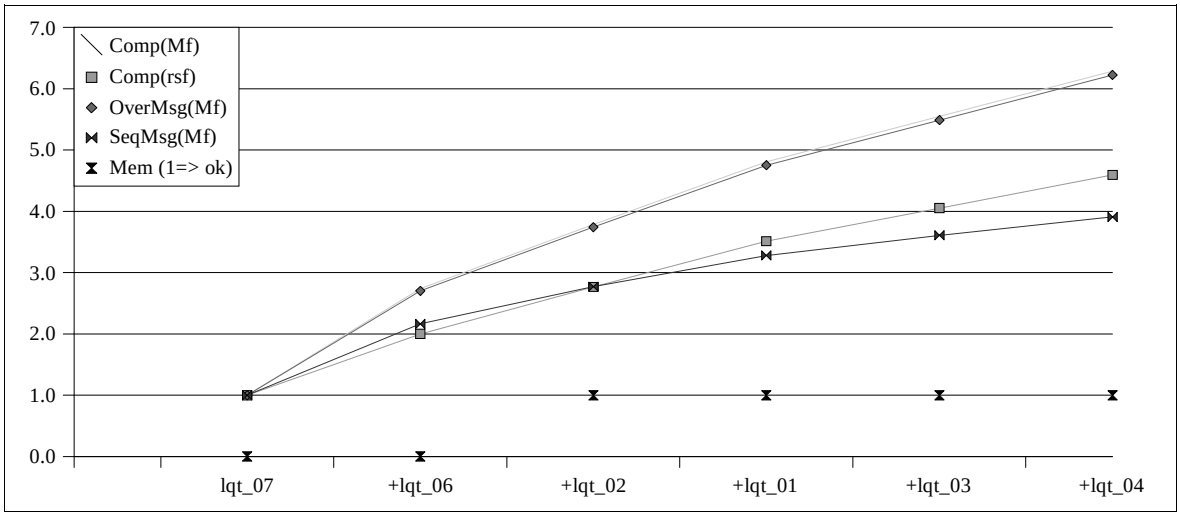


Figura 4.14: Análisis de Speedup de la Red del LQT para n = 9000.

A partir de la Figura 4.14 también se deduce que:

- Al utilizar todas las computadoras para resolver el problema de la multiplicación de matrices de 9000×9000 elementos se *podría* reducir el tiempo de ejecución más de seis veces respecto del tiempo de ejecución de **lqt_07**, aún cuando según las velocidades

relativas la reducción no llegaría a cinco veces.
- Si es posible solapar totalmente los cálculos locales con las comunicaciones, entonces el máximo speedup posible de obtener es similar al máximo absoluto. Expresado de otra manera, el tiempo de cómputo es igual o mayor que el de las comunicaciones.
- Cuando los mensajes y el cómputo local se llevan a cabo de manera secuencial el tiempo de ejecución será mayor que el esperado teniendo en cuenta las velocidades relativas de las computadoras a partir de la inclusión de **lqt_01**, es decir que Comp(rsf) > SeqMsg(Mf) a partir de la inclusión de **lqt_01**.
- Según las estimaciones de memoria no sería necesario recurrir a la utilización del espacio de memoria swap a partir de la inclusión de **lqt_02**.

4.5.3 Red Local del LIDI

La Figura 4.15 muestra los máximos valores de speedup a considerar en la red local del LQT cuando todos los datos del problema se pueden contener en memoria principal, de forma tal que no es necesaria la utilización de memoria swap durante los cálculos. En este caso, la computadora de referencia (la de mayor capacidad de cálculo) es **lidipar14** pero se debe recordar que todas las computadoras son iguales, y el tamaño de matrices es $n = 2000$. A diferencia de las redes locales anteriores se utiliza una red Ethernet de 100 Mb/s, y se asume que por la degradación producida por todas las capas del sistema operativo se pueden transferir datos entre procesos de usuario a razón de 10×2^{20} bytes (10 MB) por segundo. Como en los casos anteriores, podría considerarse una suposición optimista, pero aceptable dado que se estiman valores máximos.

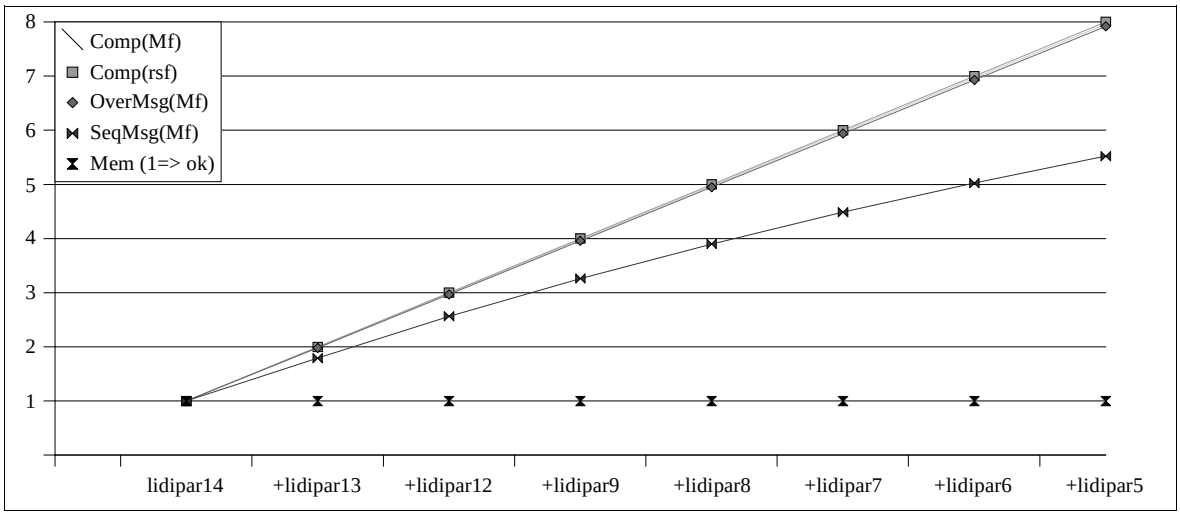


Figura 4.15: Análisis de Speedup de la Red del LIDI para $n = 2000$.

Los valores de speedup que se muestran en la Figura 4.15 son más o menos comunes con los de las computadoras paralelas *clásicas* (homogéneas), dado que:
- Los valores de Comp(rsf) se *corresponden* con la recta $y = x$.
- Al menos hasta la inclusión de **lidipar5** todos los valores máximos estimados para el speedup siguen un patrón de crecimiento lineal respecto de la cantidad de la cantidad de

procesadores (computadoras) utilizados, incluso considerando el algoritmo de cómputo y comunicaciones secuenciales.

También de la Figura 4.15 se desprende que:

- Comparando esta red con las anteriores, al tener una red de interconexión diez veces mejor en cuanto a ancho de banda las comunicaciones no tienen un peso relativo tan grande. De hecho, el algoritmo que lleva a cabo cómputo con comunicaciones de manera solapada tiene speedup óptimo igual al calculado sin tener en cuenta las comunicaciones, OverMsg(Mf) ≅ Comp(Mf).
- Con el algoritmo que lleva a cabo cómputo y comunicaciones secuenciales, al utilizar las ocho computadoras disponibles se obtiene en el mejor de los casos una computadora (paralela) que tiene poco más de 5.5 veces mayor capacidad de cálculo que **lidipar14** (o cualquiera de las demás, dado que son todas iguales).
- Como todas las computadoras son iguales se puede identificar más claramente que en los casos anteriores (CeTAD y LQT) la relación entre el tiempo de comunicaciones y el de cómputo con los valores máximos de speedup del algoritmo con cómputo y comunicaciones secuenciales, SeqMsg(Mf). Dado que a medida que hay mayor cantidad de máquinas se reparte el mismo trabajo entre todas ellas, el tiempo de cómputo total disminuye (hay mayor cantidad de computadoras procesando simultáneamente) pero el tiempo total de comunicaciones se mantiene igual. Por lo tanto, a medida que se agregan máquinas el tiempo de comunicaciones afecta de manera más significativa al tiempo total de ejecución (cómputo más comunicaciones).

Para ejemplificar este último punto se pueden tomar los valores específicos calculados para cuatro y ocho máquinas. Independientemente de la cantidad de computadoras que se utilizan, cuando el tamaño de las matrices es el mismo (en este caso n = 2000) la cantidad de datos que se comunican es la misma ya que siempre deben transmitirse los datos de la matriz B entre las computadoras. El tiempo estimado de transmisión de la matriz B (suma de tiempos de los mensajes broadcast del algoritmo) en una red Ethernet de 100 Mb/s es aproximadamente 1.5 segundos. Cuando se utilizan **lidipar14**, **lidipar13**, **lidipar12**, y **lidipar9**, el tiempo estimado de cómputo es aproximadamente 13.8 segundos. Cuando se agregan a las anteriores las computadoras **lidipar8**, **lidipar7**, **lidipar6**, y **lidipar5**, el tiempo estimado de cómputo es aproximadamente 3.4 segundos. Por lo tanto, cuando los mensajes y las comunicaciones se ejecutan de manera secuencial:

- El tiempo total de ejecución cuando se utilizan cuatro computadoras es (sumando tiempo de cómputo y tiempo de comunicaciones), 1.5 + 13.8 = 15.3 segundos. Esto implica que aproximadamente el 10% del tiempo total de ejecución es utilizado para las comunicaciones.
- El tiempo total de ejecución cuando se utilizan ocho computadoras es (sumando tiempo de cómputo y tiempo de comunicaciones) 1.5 + 3.4 = 4.9 segundos. Esto implica que aproximadamente el 30% del tiempo total de ejecución es utilizado para las comunicaciones.

La Figura 4.16 muestra los máximos valores de speedup a considerar en la red local del LIDI para el máximo tamaño de problema que puede resolver la computadora de mayor capacidad de cálculo (recurriendo a la memoria swap). La computadora de referencia sigue siendo **lidipar14** y las matrices son de 3200×3200 elementos. Como para la estimación anterior, se asume que se pueden transferir datos entre procesos de usuario a razón de

10×2^{20} bytes (10 MB) por segundo. Además de mostrarse que los valores máximos de speedup calculados utilizando las velocidades relativas, Comp(rsf), no cambian respecto de los que se muestran en la Figura 4.15, en la Figura 4.16 también se muestra que:

- El problema de multiplicar matrices de 3200×3200 elementos se *podría* resolver casi 35 veces más rápido en las ocho computadoras que en una de ellas. Esto se confirma no solamente por la potencia de cálculo de las ocho máquinas procesando a su máxima capacidad (sin tener en cuenta las comunicaciones), Comp(Mf), sino también con el algoritmo que ejecuta cálculo solapado con las comunicaciones, OverMsg(Mf).
- Con el algoritmo que resuelve de manera secuencial las comunicaciones respecto de los cálculos, el problema de multiplicar matrices de 3200×3200 elementos se *podría* resolver más de 25 veces más rápido en las ocho computadoras que en una de ellas.
- La penalización en rendimiento por el uso del espacio de swap en **lidipar14** para matrices de 3200×3200 elementos como la tasa de transferencia de datos de la red de interconexión se combinan para tener estos valores de speedup "superlineales".
- Excepto para una y dos máquinas, las estimaciones de requerimientos de memoria no identifican posibles problemas en cuanto a necesidad de utilización del espacio de swap.

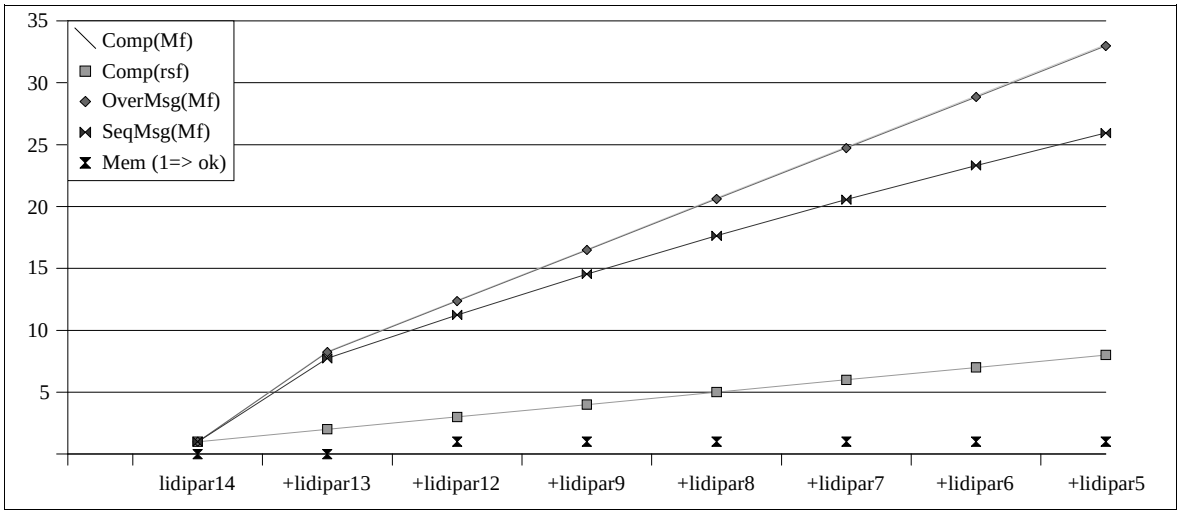


Figura 4.16: Análisis de Speedup de la Red del LIDI para n = 3200.

4.6 Rendimiento Real de las Redes Locales Utilizando PVM

Los algoritmos propuestos en el capítulo anterior se implementaron directamente utilizando la biblioteca PVM (Parallel Virtual Machine) para las rutinas de comunicaciones entre procesos. En cada una de las computadoras se utiliza el mejor código secuencial para los periodos de cálculo local. En cada una de las redes locales (CeTAD, LQT y LIDI) se llevaron a cabo los mismos experimentos, es decir:

- Multiplicación de matrices con el algoritmo de cómputo y mensajes secuenciales (SeqMsg).

- Multiplicación de matrices con el algoritmo de cómputo y mensajes solapados (OverMsg).
- Tamaños de matrices tales que en la máquina con mayor capacidad de cálculo
  - No se utiliza espacio de swap.
  - Más grande posible.

  En cada red local esto se corresponde con matrices de orden
  - $n = 2000$ y $n = 3200$ respectivamente en las redes locales del CeTAD y del LIDI.
  - $n = 5000$ y $n = 9000$ respectivamente en la red local del LQT.

4.6.1 Red Local del CeTAD

La Figura 4.17 muestra los valores de speedup obtenidos en la red local del CeTAD por los algoritmos de cómputo y comunicación secuenciales, y cómputo solapado con comunicaciones, implementados utilizando la biblioteca de pasaje de mensajes PVM, SeqMsg(PVM) y OverMsg(PVM) respectivamente, para $n = 2000$, junto con los que se mostraron anteriormente en la Figura 4.11.

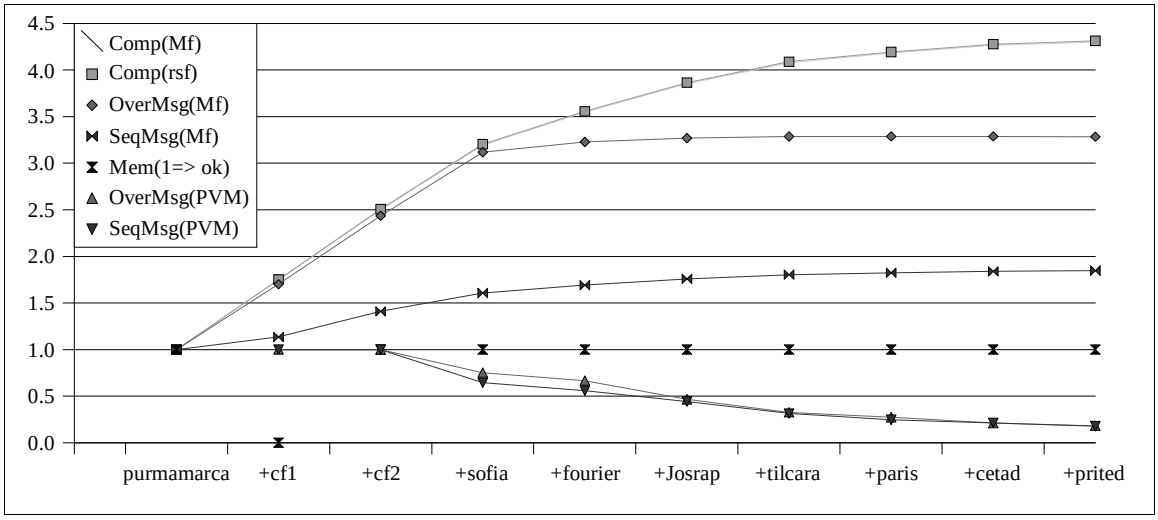


Figura 4.17: Speedup de los Algoritmos con PVM en la Red del CeTAD para $n = 2000$.

Claramente, los resultados están lejos de ser satisfactorios. De hecho, las dos conclusiones más desalentadoras son

- Ninguno de los tiempos de ejecución, es decir independientemente de la cantidad de computadoras utilizadas, fue mejor que el tiempo de la ejecución secuencial, con el problema resuelto en **purmamarca**.
- A medida que se utilizan más computadoras, el tiempo de ejecución aumenta en vez de disminuir.

Más aún, los valores de speedup de los algoritmos que se muestran iguales a uno, es decir para los casos en que se utilizan **purmamarca** y **cf1** y **purmamarca**, **cf1** y **cf2** respectivamente ni siquiera son *reales*. En los dos casos, la ejecución de los procesos del programa paralelo en **cf1** y/o **cf2** se cancela por falta de memoria disponible. Por lo tanto,

se muestran iguales a uno porque de hecho se considera que la única posibilidad de solución para la multiplicación de matrices en ese contexto es la ejecución secuencial (utilizando **purmamarca** solamente). A pesar de que se tiene una aproximación respecto de los requerimientos de memoria (Mem, en los gráficos), evidentemente al menos con PVM esa aproximación no es acertada.

Aún descartando el problema de memoria, el problema de rendimiento es evidente. En principio, las alternativas en cuanto a las causas del bajo rendimiento obtenido pueden ser:

- Bajo rendimiento de cómputo, al resolver cada uno de los cálculo intermedios. Este problema está básicamente relacionado con el rendimiento en cuanto a cómputo de cada una de las computadoras.
- Bajo rendimiento de las comunicaciones, al enviar y recibir los mensajes broadcast. Este problema está básicamente relacionado con PVM y con la red de interconexión. En este sentido, se tienen dos posibilidades:
  - PVM no implementa de manera óptima los mensajes broadcast o
  - la comunicación entre procesos de usuario de distintas computadoras sean muy penalizados en rendimiento con respecto a la capacidad de la red de interconexión.

La Figura 4.18 muestra los valores de speedup obtenidos en la red local del CeTAD por los algoritmos de cómputo y comunicación secuenciales, y cómputo solapado con comunicaciones, implementados utilizando la biblioteca de pasaje de mensajes PVM, SeqMsg(PVM) y OverMsg(PVM) respectivamente, para n = 3200, junto con los que se mostraron en la Figura 4.12.

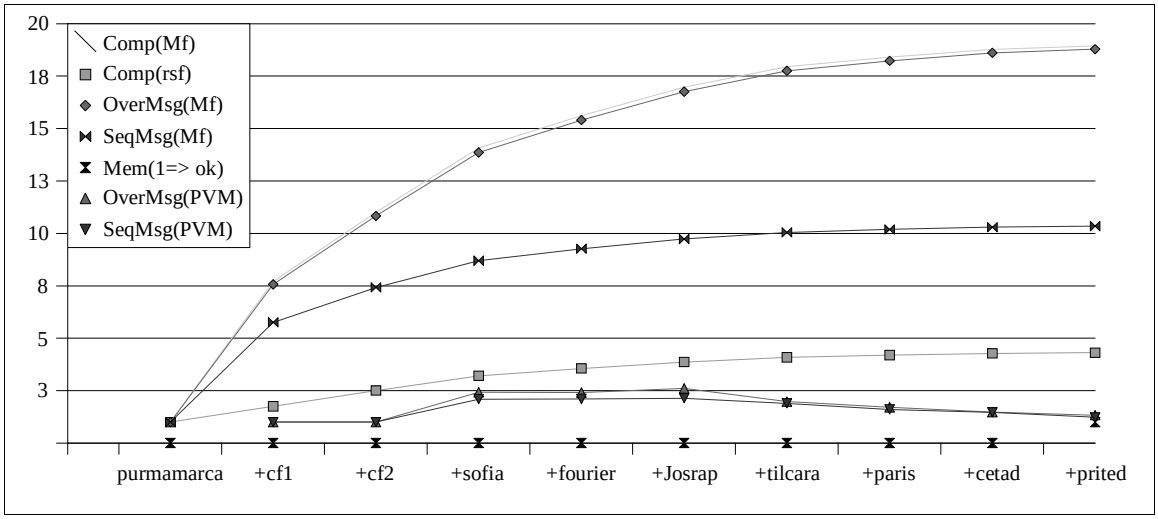


Figura 4.18: Speedup de los Algoritmos con PVM en la Red del CeTAD para n = 3200.

En este caso, es decir tomando como referencia la capacidad de cálculo de **purmamarca** para multiplicar matrices cuadradas de orden n = 3200:

- Una vez más los requerimientos de memoria en **cf1** y **cf2** hacen imposible la ejecución del programa paralelo y los procesos son cancelados por el sistema operativo.
- En el mejor de los casos, se obtienen valores de speedup cercanos a tres cuando todas las estimaciones son superiores para la misma cantidad de máquinas, incluso la que

tiene en cuenta solamente las velocidades relativas: Comp(rsf).
- A partir de la inclusión de **tilcara** en la máquina paralela el tiempo de ejecución paralela empeora, llegando a tener con las diez máquinas valores de speedup de aproximadamente 1.33 con el algoritmo que lleva a cabo cómputo y comunicación solapados y 1.23 con el algoritmo que lleva a cabo cómputo y comunicación secuenciales, OverMsg(PVM) y SeqMsg(PVM) respectivamente.
- Todas las estimaciones de speedup de los algoritmos son muy lejanas de los valores obtenidos.

4.6.2 Red Local del LQT

La Figura 4.19 muestra los valores de speedup obtenidos en la red local del LQT por los algoritmos de cómputo y comunicación secuenciales, y cómputo solapado con comunicaciones, implementados utilizando la biblioteca de pasaje de mensajes PVM, SeqMsg(PVM) y OverMsg(PVM) respectivamente, para n = 5000, junto con los que se mostraron anteriormente en la Figura 4.13.

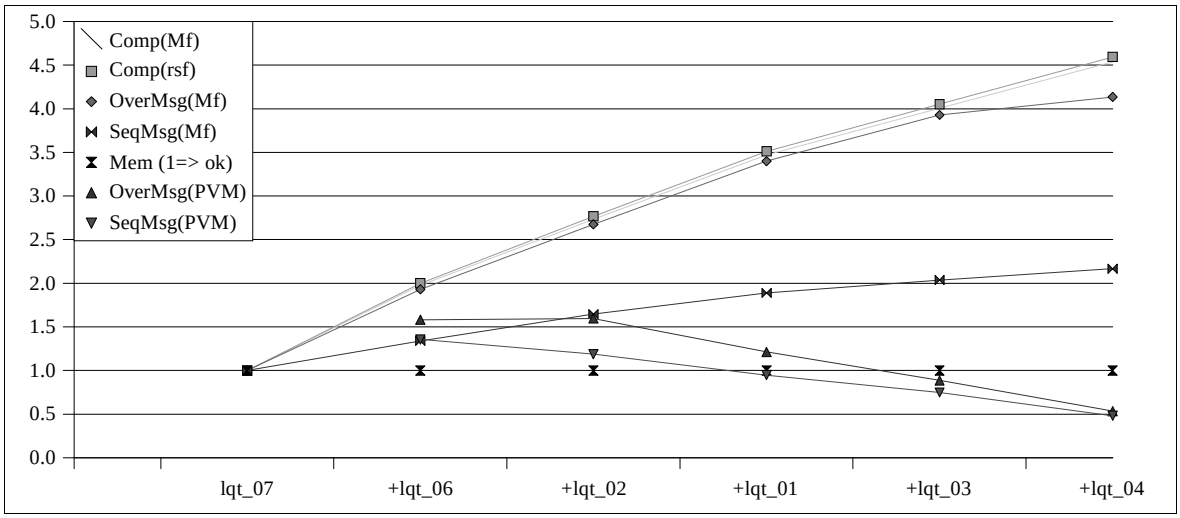


Figura 4.19: Speedup de los Algoritmos con PVM en la Red del LQT para n = 5000.

Comparando estos resultados con los que se corresponden del CeTAD, que se muestran en la Figura 4.17, la similitud es notable. Básicamente las características de los valores de speedup obtenidos es la misma
- Casi no se gana rendimiento por utilizar máquinas procesando en paralelo,
- En la mayoría de los casos, agregar máquinas para procesar en paralelo implica pérdida de rendimiento,

a pesar de que las computadoras y el tamaño del problema son muy diferentes entre sí. Por lo tanto, estos resultados no hacen más que confirmar que hay uno o varios problemas y que el o los problemas no son específicos de la red local del CeTAD ni de la red local del LQT.

La Figura 4.20 muestra los valores de speedup obtenidos en la red local del LQT por los algoritmos de cómputo y comunicación secuenciales, y cómputo solapado con

comunicaciones, implementados utilizando la biblioteca de pasaje de mensajes PVM, SeqMsg(PVM) y OverMsg(PVM) respectivamente, para n = 9000, junto con los que se mostraron en la Figura 4.14. Las similitudes en cuanto a los valores de speedup con respecto al CeTAD en el contexto similar (Figura 4.18), son una vez más bastante evidentes, a pesar de las diferencias entre las máquinas y el tamaño del problema:

- Los requerimientos de memoria que impone el cómputo paralelo con las rutinas de comunicaciones de PVM hacen que cuando se utilizan dos computadoras, **lqt_07** y **lqt_06**, el sistema operativo cancele uno o varios de los procesos involucrados por falta de memoria. Es por esto que en el gráfico se muestra speedup igual a uno para dos computadoras para los dos algoritmos.
- Hasta una determinada cantidad de computadoras, el speedup aumenta. En este caso hasta la inclusión de **lqt_02** para el algoritmo de cómputo y comunicación secuenciales, SeqMsg(PVM) en el gráfico, y hasta la inclusión de **lqt_01** para el algoritmo de comunicaciones solapadas con cómputo OverMsg(PVM) en el gráfico.
- La utilización de todas las máquinas no aporta casi ningún beneficio en cuanto a rendimiento con respecto a la alternativa de solución secuencial. Los valores reales de speedup cuando se utilizan las seis computadoras son 1.3 para OverMsg(PVM) y 1.08 para SeqMsg(PVM).

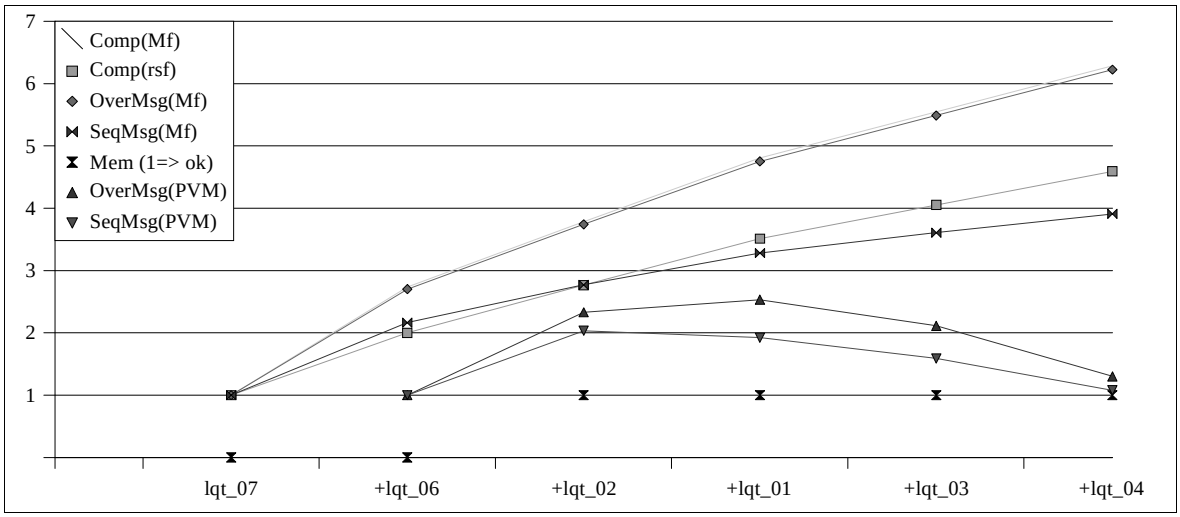


Figura 4.20: Speedup de los Algoritmos con PVM en la Red del LQT para n = 9000.

Comparando los resultados que se muestran en la Figura 4.20 con los de la Figura 4.18, también se pueden encontrar algunas diferencias:

- Algunos valores de speedup obtenidos son bastante más cercanos a los estimados, al menos para tres y cuatro máquinas, es decir cuando se utilizan **lqt_07**, **lqt_06**, **lqt_02**, y **lqt_07**, **lqt_06**, **lqt_02** y **lqt_01** respectivamente.
- El algoritmo de que lleva a cabo cómputo solapado con comunicaciones tiene mejor rendimiento que el que no intenta aprovechar ningún solapamiento. La diferencia llega a ser de poco más del 30% cuando se utilizan **lqt_07**, **lqt_06**, **lqt_02**, **lqt_01** y **lqt_03**.

4.6.3 Red Local del LIDI

La Figura 4.21 muestra los valores de speedup obtenidos en la red local del LIDI por los algoritmos de cómputo y comunicación secuenciales, y cómputo solapado con comunicaciones, implementados utilizando la biblioteca de pasaje de mensajes PVM, SeqMsg(PVM) y OverMsg(PVM) respectivamente, para n = 2000, junto con los que se mostraron anteriormente en la Figura 4.15. Como en las redes del CeTAD y del LQT en el contexto *correspondiente* (Figura 4.17 y Figura 4.19):

- Las estimaciones de speedup son bastante lejanas respecto de los valores obtenidos. A partir de la utilización de cuatro computadoras inclusive la diferencia es cada vez mayor.
- En términos generales, agregar computadoras implica pérdida de rendimiento, las excepciones se dan para dos y tres computadoras, dado que el rendimiento en esos casos aumenta cuando se usan más máquinas.

A diferencia de las redes del CeTAD y del LQT:

- No se llega a tener peor tiempo de cómputo que para la solución secuencial, aunque la tendencia a medida que aumenta la cantidad de computadoras utilizadas indica que se puede llegar a esta situación (con valores de speedup menores que uno).
- La pérdida de rendimiento a medida que aumenta la cantidad de computadoras utilizadas es bastante más gradual.

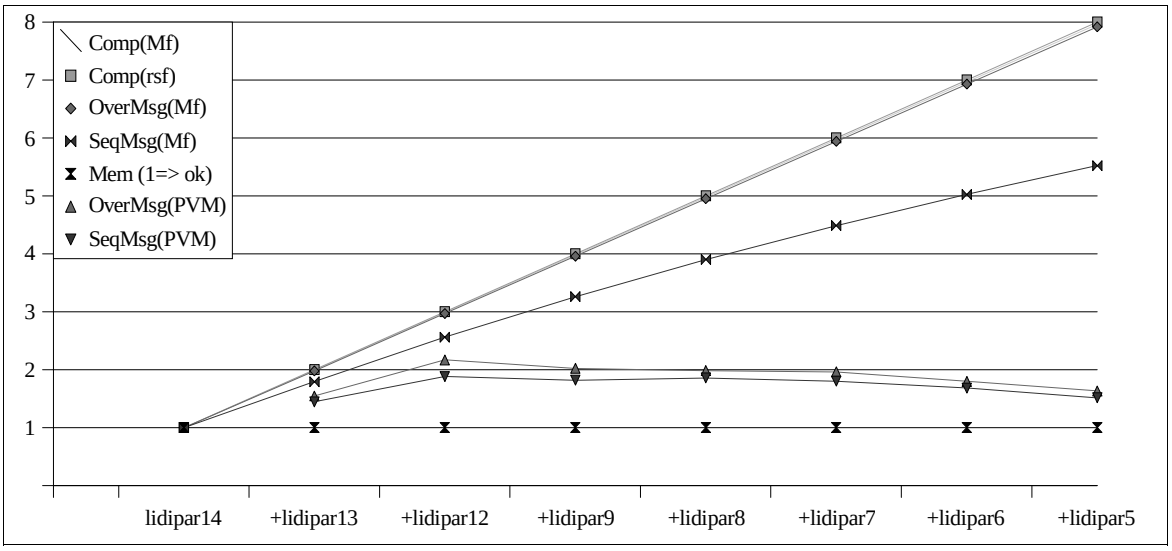


Figura 4.21: Speedup de los Algoritmos con PVM en la Red del LIDI para n = 2000.

La Figura 4.22 muestra los valores de speedup obtenidos en la red local del LIDI por los algoritmos de cómputo y comunicación secuenciales, y cómputo solapado con comunicaciones, implementados utilizando la biblioteca de pasaje de mensajes PVM, SeqMsg(PVM) y OverMsg(PVM) respectivamente, para n = 3200, junto con los que se mostraron anteriormente en la Figura 4.16.

A pesar de que los valores obtenidos de la experimentación no son cercanos a los estimados para los algoritmos, es la primera vez que se tienen valores de speedup mayores

al menos que los valores de speedup calculados según las velocidades relativas. De acuerdo con la Figura 4.22 también se tiene que:

- En el mejor de los casos, que se da al utilizar **lidipar14**, **lidipar13**, **lidipar12**, **lidipar9** y **lidipar8**, se resuelve la multiplicación de matrices un poco más de diez veces más rápidamente que en **lidipar14** (se debe recordar que **lidipar14** utiliza memoria swap para resolver este problema).
- A partir de la inclusión de **lidipar6** el rendimiento disminuye con los dos algoritmos, lo cual implica que se reducen tanto OverMsg(PVM) como SeqMsg(PVM).
- Se tienen valores de speedup "superescalares" (mayores que la cantidad de procesadores homogéneos), dado que el tiempo de referencia de la resolución del problema de manera secuencial en **lidipar14** está penalizado en cuanto a rendimiento por la utilización del espacio de swap durante los cálculos, hecho que se nota claramente en la Figura 4.8.

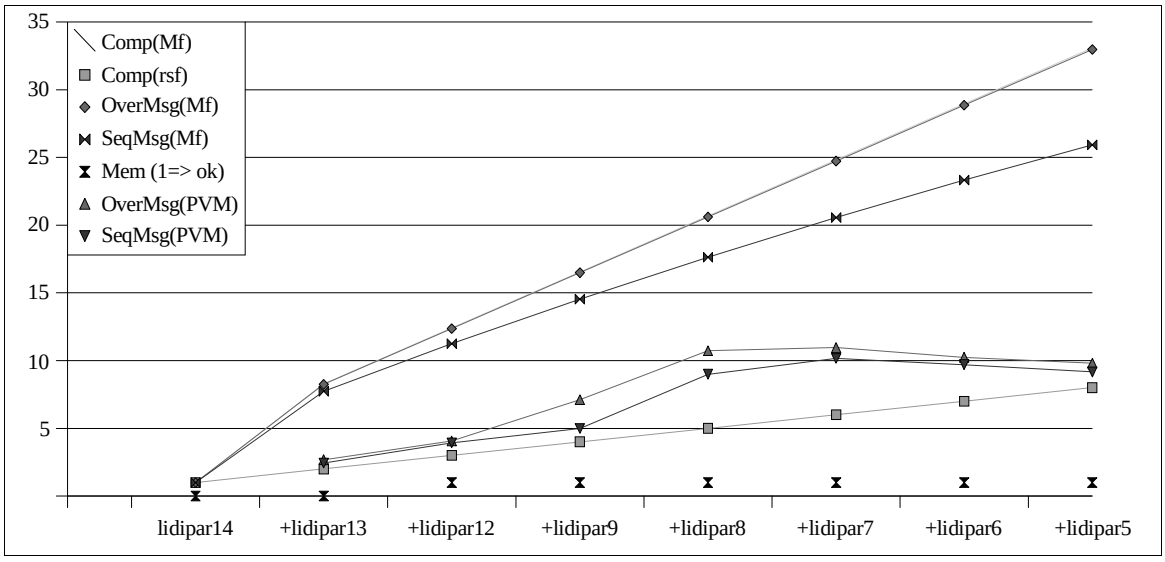


Figura 4.22: Speedup de los Algoritmos con PVM en la Red del LIDI para $n = 3200$.

Los valores obtenidos de la experimentación en la red local del LIDI parecen ser bastante mejores que los que se obtuvieron de la experimentación en las redes locales del CeTAD y del LQT. Más allá de las diferencias entre las máquinas en particular, desde el punto de vista de las computadoras paralelas que se construyen con cada red local, las principales diferencias son:

- La red de interconexión del LIDI es diez veces más rápida que la del CeTAD y del LQT.
- La computadora paralela construida con la red local del LIDI es homogénea, mientras que tanto la del CeTAD como la del LQT son (*muy*) heterogéneas.

Intuitivamente, la razón más importante por la cual se tienen mejores resultados en la red local del LIDI es la capacidad muy superior de la red de interconexión, pero evidentemente se necesitan más datos para dar una explicación mejor fundamentada.

4.7 Perfiles de Ejecución en las Redes Locales Utilizando PVM

Para tener mayor precisión respecto de los tiempos de ejecución paralela y de las razones por las cuales las estimaciones de speedup son tan lejanas respecto del speedup obtenido en la experimentación, se llevaron a cabo los mismos experimentos pero con un mínimo de instrumentación de forma tal que:

- Se identifica claramente qué parte del tiempo total de ejecución se utiliza para cómputo y qué parte del tiempo total se utiliza para comunicaciones. Esta información es muy útil para identificar si el problema son las comunicaciones o no.
- Se identifica gráficamente en qué estado de ejecución está cada proceso (computadora de la red local) durante todo *instante* de tiempo de ejecución. Este tipo de información tiene un poco más de detalle que la anterior, y es útil para identificar si hay alguna computadora en particular que produce un retraso general. Por ejemplo: si por alguna razón local una computadora no envía un mensaje broadcast en el tiempo esperado, todas las demás se verán afectadas porque no lo recibirán.

Dado que los experimentos son muy numerosos como para mostrar el perfil de tiempos de ejecución de cada uno de ellos y además como en su mayoría son similares, se muestran y explican los más significativos en cada una de las redes locales.

4.7.1 Red Local del CeTAD

En la Figura 4.23 se muestra el perfil de ejecución al utilizar las cinco mejores máquinas de la red local del CeTAD para una multiplicación de matrices en paralelo de 2000×2000 elementos, con el algoritmo de cómputo y comunicaciones secuenciales, donde:

- El tiempo se muestra en segundos.
- En todo instante de tiempo, cada computadora puede estar:
  - Ejecutando un cálculo parcial, mostrado como “Cómputo” en el gráfico.
  - Enviando o recibiendo un mensaje broadcast, mostrado como “Bcasts” en el gráfico, y durante el cual los cálculos no se pueden realizar y por lo tanto se “Espera” la terminación del broadcast para seguir con el cómputo.
- Bcasts identifica cada mensaje broadcast, donde un proceso que se ejecuta en una computadora envía datos a todos los demás (en este caso a otros cuatro procesos) que se ejecutan en las demás computadoras (cuatro computadoras).
- Dado que en PVM el envío de *todos* los mensajes es solapado con respecto al cómputo cuando se envía un broadcast simultáneamente se lleva a cabo un período de cálculo parcial de la matriz resultado.

Más allá de algunos detalles de tiempo de ejecución particulares, se nota claramente que durante la mayor parte del tiempo de ejecución de cada computadora se espera por la conclusión de un mensaje broadcast (más precisamente, por la recepción de un mensaje broadcast desde otra computadora).

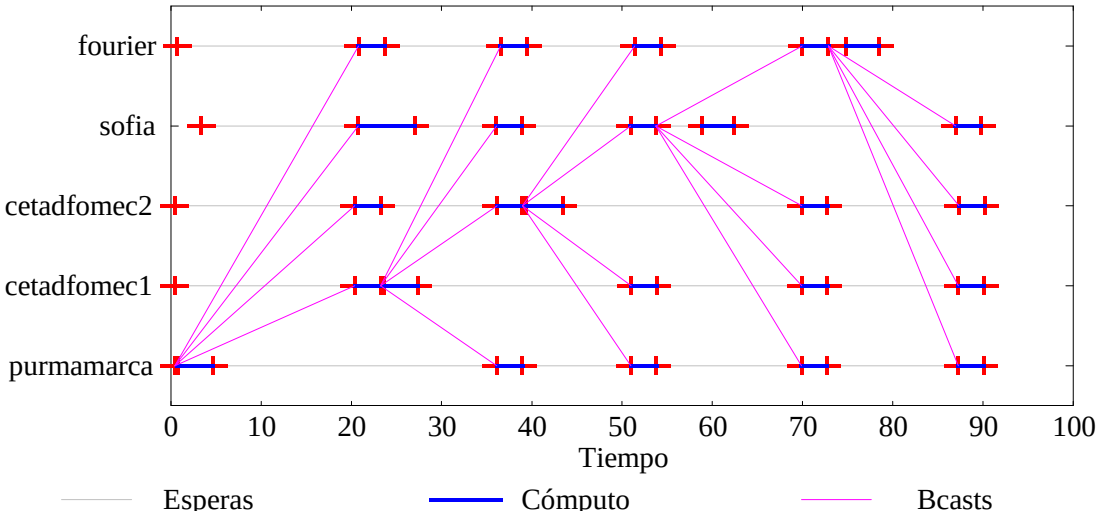


Figura 4.23: Perfil de SeqMsg(PVM) con Cinco Máquinas y *n* = 2000 en el CeTAD.

La Tabla 4.4 muestra la información resumida de la ejecución del programa paralelo que se corresponde con el perfil de ejecución de la Figura 4.23, donde

- ***Nombre*** identifica el nombre de la computadora utilizada.
- ***Filas*** identifica la cantidad de filas de la matriz resultado que fueron asignadas a cada computadora, que es proporcional a la velocidad relativa de cada computadora con respecto a la computadora paralela.
- ***Tot. Cómp.*** identifica la cantidad de tiempo local de ejecución durante el cual se ejecutaron operaciones con números en punto flotante.
- ***Por It.*** identifica la cantidad de tiempo local de ejecución de un paso de cómputo (operaciones con números en punto flotante).
- ***Tot. Msg.*** identifica la cantidad de tiempo local de ejecución utilizado para la espera de la conclusión de los mensajes broadcast (un envío y cuatro recepciones, dado que son cinco computadoras en total).

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	555	15.17	3.03	68.29
cf1	426	15.05	3.01	68.39
cf2	426	15.59	3.12	67.80
sofia	394	17.00	3.40	65.22
fourier	199	15.25	3.05	55.91

Tabla 4.4: Resumen de SeqMsg(PVM) con Cinco Máquinas y *n* = 2000 en el CeTAD.

La Figura 4.24 muestra el perfil de ejecución cuando se utilizan todas las máquinas (diez) disponibles del CeTAD. También a partir de lo que se muestra en el gráfico, queda claro que la mayoría del tiempo de ejecución se emplea para los mensajes.

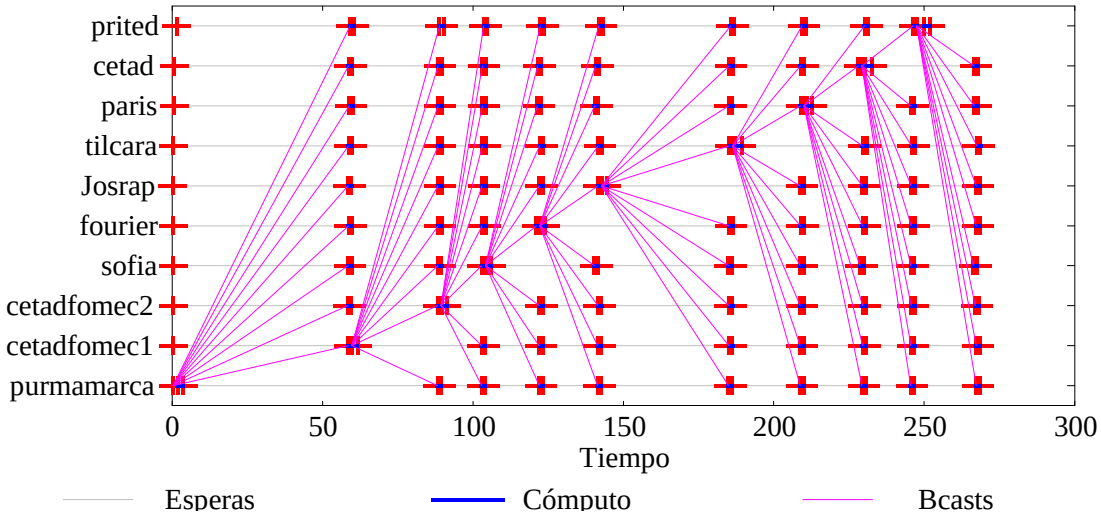


Figura 4.24: Perfil de SeqMsg(PVM) con Diez Máquinas y *n* = 2000 en el CeTAD.

En la Figura 4.24 también se nota bastante claramente que tanto el primer broadcast, enviado desde **purmamarca** como el sexto, enviado desde **Josrap**, utilizan mayor tiempo de transmisión que los demás. Pero aunque estos dos mensajes utilizaran el tiempo promedio de comunicaciones que utilizan los demás, el tiempo total de comunicaciones seguiría siendo dominado por las comunicaciones. Por lo tanto, el primer problema a resolver según estos dos perfiles de ejecución mostrados (Figura 4.23 y Figura 4.24), es el del tiempo de comunicaciones excesivo.

La Tabla 4.5 muestra la información resumida de la ejecución del programa paralelo que se corresponde con el perfil de ejecución de la Figura 4.24, donde se puede cuantificar bastante mejor el peso relativo de las comunicaciones (***Tot. Msg.***) con respecto al cómputo (***Tot. Cómp.***).

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	454	12.35	1.24	255.78
cf1	349	12.44	1.24	255.75
cf2	349	12.48	1.25	255.31
sofia	324	12.21	1.22	255.05
fourier	164	12.85	1.28	255.22
Josrap	142	12.24	1.22	256.03
tilcara	104	13.29	1.33	254.94
paris	48	12.08	1.21	255.01
cetad	38	12.84	1.28	254.16
prited	28	13.68	1.37	236.92

Tabla 4.5: Resumen de SeqMsg(PVM) con Diez Máquinas y *n* = 2000 en el CeTAD.

La situación no es muy diferente cuando se utiliza el algoritmo de comunicaciones solapadas con cómputo, tal como lo muestra la Figura 4.25 para las cinco mejores computadoras. En resumen, la Figura 4.25 muestra el perfil de ejecución al utilizar las cinco mejores máquinas de la red local del CeTAD para una multiplicación de matrices en paralelo de 2000×2000 elementos, con el algoritmo de cómputo y comunicaciones solapados. También en este caso, durante la mayor parte del tiempo de ejecución de cada computadora se espera por la conclusión de un mensaje broadcast (más precisamente, por la recepción de un mensaje broadcast desde otra computadora).

Evaluando la evolución de la ejecución en cada computadora es sencillo identificar que aunque se intentan solapar las comunicaciones con el cómputo local, de todas maneras transcurre más tiempo esperando por la recepción de los datos que se reciben del resto de las computadoras (identificados como “Esperas” en la Figura 4.25) que haciendo los cálculos que corresponden a las distintas iteraciones del algoritmo paralelo.

Comparando los tiempos de ejecución totales de los algoritmos OverMsg(PVM) de la Figura 4.25 y SeqMsg(PVM) de la Figura 4.23, el tiempo de ejecución total de OverMsg(PVM) es menor que el de SeqMsg(PVM) y esto se debe al solapamiento que enmascara en parte el peso de los tiempos de comunicaciones. En este sentido, aunque las computadoras de la red local del CeTAD son muy diferentes entre sí, de todas maneras son capaces de solapar cómputo local y comunicaciones al menos en parte.

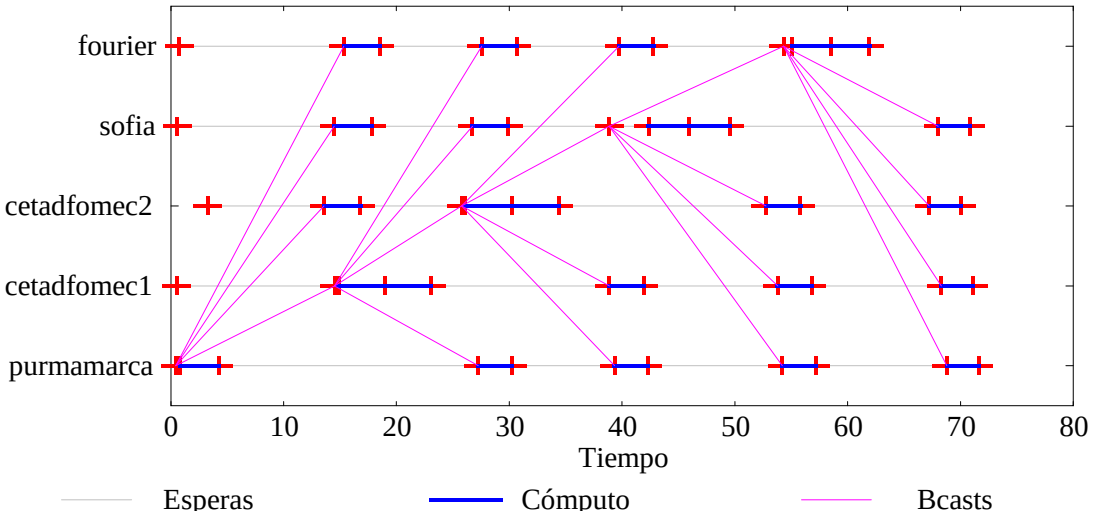


Figura 4.25: Perfil de OverMsg(PVM) con Cinco Máquinas y n = 2000 en el CeTAD.

El resumen de ejecución que se muestra en la Tabla 4.6 es similar al que se muestra en la Tabla 4.4 aunque los tiempos de comunicaciones son menores dado que parte del tiempo de cada mensaje broadcast se *superpone* con cómputo local.

Cuando se toma como referencia el máximo tamaño que se puede resolver en la computadora con mayor capacidad de cálculo del CeTAD, es decir n = 3200, las características en cuanto a perfiles de ejecución y rendimiento siguen siendo las mismas. La Figura 4.26 muestra el perfil de ejecución al utilizar las siete mejores máquinas de la red local del CeTAD para una multiplicación de matrices en paralelo de 3200×3200 elementos,

con el algoritmo de cómputo y comunicaciones solapados.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	555	15.20	3.04	55.97
cf1	426	17.07	3.41	53.54
cf2	426	17.31	3.46	49.44
sofia	394	16.50	3.30	53.78
fourier	199	16.18	3.24	44.97

Tabla 4.6: Resumen de OverMsg(PVM) con Cinco Máquinas y n = 2000 en el CeTAD.

Quizás recién en este momento se nota gráficamente algo que se podía esperar desde la proposición misma del algoritmo pensado para solapar cómputo local con comunicaciones: el tiempo de cómputo local aumenta cuando se llevan a cabo las comunicaciones y es probable que el envío de mensajes sea más *costoso* en términos de tiempo de ejecución que la recepción. Esto se confirma gráficamente en la Figura 4.26 solamente para las computadoras **cetadfomec1** y **cetadfomec2**, para las cuales el tiempo de procesamiento para cálculos parciales (que se muestran como "Cómputo" en la Figura 4.26) mientras envían datos son mayores que el resto de los tiempos de procesamiento para cálculos parciales. Sin embargo, la proporción de aumento de tiempos de cómputo sigue sin ser importante cuando son comparadas con los tiempos de comunicaciones, comparación que se puede cuantificar a partir de los datos en la Tabla 4.7.

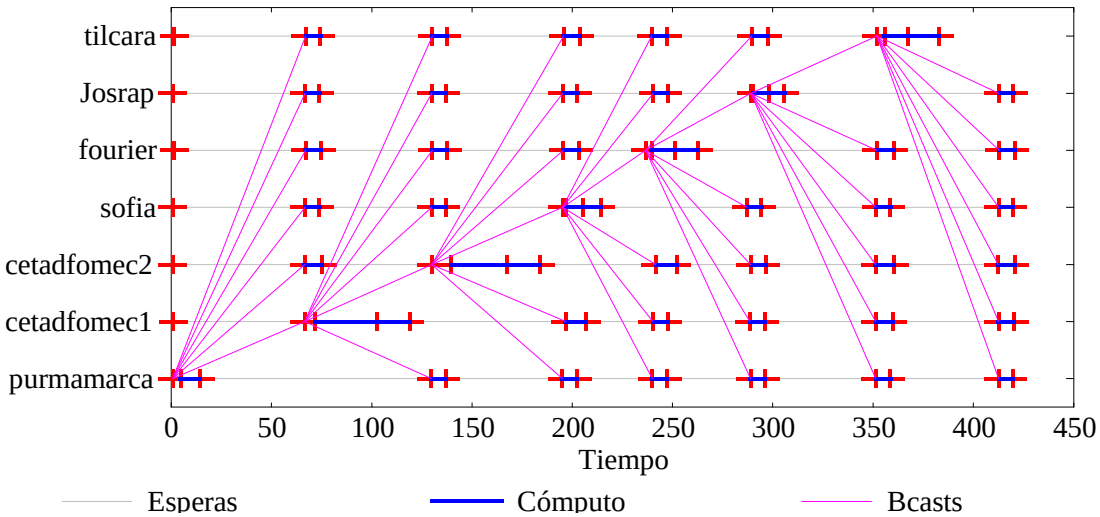


Figura 4.26: Perfil de OverMsg(PVM) con Siete Máquinas y n = 3200 en el CeTAD.

La Tabla 4.7 *completa* la información de la Figura 4.26 con el resumen de la ejecución, mostrando

- las cantidades de filas asignadas (***Filas***),
- los tiempos totales de cómputo y de comunicaciones (***Tot. Cómp*** y ***Tot. Msg.***),
- el tiempo dedicado en cada iteración a cómputo local (***Por It.***),

en cada computadora.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	771	53.36	7.62	365.40
cf1	593	88.55	12.65	330.97
cf2	593	89.01	12.72	330.66
sofia	549	53.26	7.61	365.44
fourier	276	62.05	8.86	356.99
Josrap	242	51.96	7.42	367.13
tilcara	176	65.33	9.33	316.02

Tabla 4.7: Resumen de OverMsg(PVM) con Siete Máquinas y n = 3200 en el CeTAD.

La Figura 4.27 y la Tabla 4.8 muestran todo lo referido a una multiplicación de matrices en paralelo de 3200×3200 elementos, con el algoritmo de cómputo y comunicaciones solapados, utilizando las diez máquinas disponibles del CeTAD.

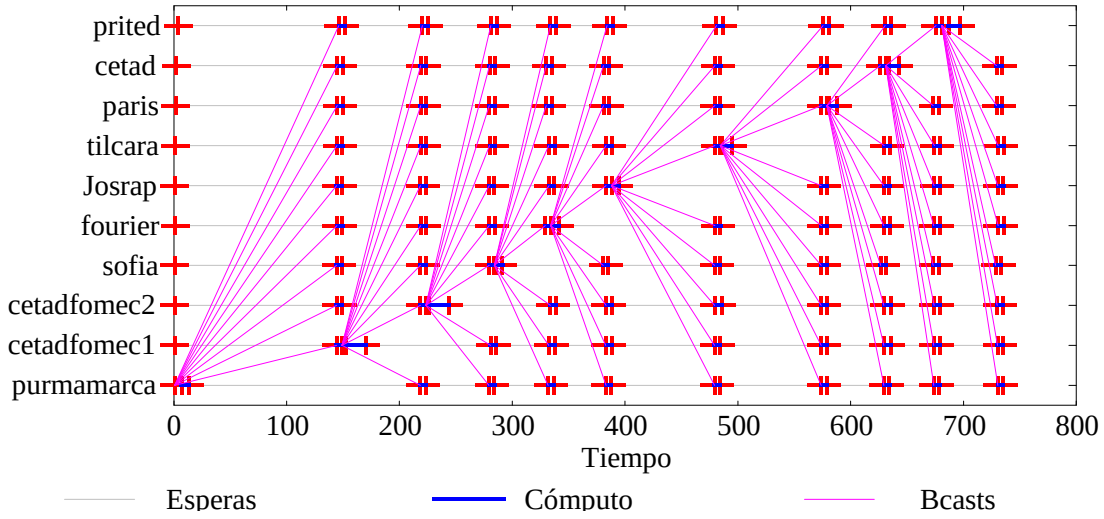


Figura 4.27: Perfil de OverMsg(PVM) con Diez Máquinas y n = 3200 en el CeTAD.

Es muy interesante comparar los totales en cuanto a tiempo de cómputo (***Tot. Cómp.***) y de comunicaciones (***Tot. Msg.***) que se muestran en la Tabla 4.7 y en la Tabla 4.8. El promedio de tiempo total de cómputo en cada computadora es aproximadamente 66.22 segundos cuando se utilizan las siete computadoras con mayor capacidad de cálculo del CeTAD. Cuando se utilizan todas las computadoras, este promedio es aproximadamente 53.5 segundos.

Comparando ahora los tiempos de comunicaciones dados en la Tabla 4.7 con los dados en la Tabla 4.8 deberían ser *iguales* (o, en realidad, muy similares) porque en todos los casos se deben transferir entre las computadoras (vía mensajes broadcast) todos los datos de la matriz B y por lo tanto el tiempo de comunicaciones debería mantenerse más o menos invariante. Esto es asumiendo que la implementación de cada mensaje broadcast entre

procesos aprovecha al máximo la capacidad de broadcast de las redes Ethernet, donde a lo sumo podría haber un pequeño aumento de tiempo por la mayor cantidad de procesos receptores de cada uno de los mensajes que se deben llevar a cabo. Sin embargo, el promedio de tiempo total de comunicaciones cuando se utilizan las siete mejores máquinas del CeTAD es de 347.52 segundos y cuando se utilizan todas las computadoras es de 676.14 segundos, es decir casi el doble de tiempo para transferir los mismos datos.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	726	49.16	4.92	685.10
cf1	559	63.12	6.31	671.01
cf2	559	63.72	6.37	670.37
sofia	518	48.72	4.87	683.82
fourier	261	50.20	5.02	684.24
Josrap	228	48.08	4.81	686.53
tilcara	166	52.02	5.20	682.36
paris	77	50.20	5.02	682.36
cetad	61	52.03	5.20	680.20
prited	45	57.78	5.78	635.42

Tabla 4.8: Resumen de OverMsg(PVM) con Diez Máquinas y n = 3200 en el CeTAD.

Dado que el problema de rendimiento está dado por las comunicaciones, conviene seguir analizando más detalladamente los tiempos de comunicaciones, y en este sentido, los resúmenes de las ejecuciones paralelas pueden proveer más información. Como ya se ha comentado, para un tamaño de matrices dado, la cantidad de datos que se deben comunicar entre las computadoras es el mismo e independiente de la cantidad de computadoras que se utilicen en paralelo. Desde el punto de vista de los mensajes, siempre se deben transferir los datos de la matriz B (C = A×B), entre todas las computadoras.

De acuerdo con la Tabla 4.4, con el algoritmo sin solapamiento de cómputo y comunicaciones en cinco computadoras se utilizan en promedio 65.12 segundos en cada computadora para la comunicación de los datos de la matriz B. Según la Tabla 4.5, Cuando se utilizan las diez computadoras (el mismo algoritmo y el mismo tamaño de matrices), el promedio de tiempo es de 253.42 segundos. Aunque los valores absolutos son totalmente distintos, con el algoritmo diseñado para solapar las comunicaciones con cómputo local y para matrices de orden n = 3200, la situación general no parece cambiar demasiado. De acuerdo con la Tabla 4.7, el tiempo promedio de comunicaciones es de 347.52 segundos cuando se utilizan siete computadoras y de acuerdo con la Tabla 4.8 el tiempo promedio de comunicaciones es de 676.14 segundos cuando se utilizan diez computadoras.

La Tabla 4.9 muestra un resumen de lo que sucede con las comunicaciones en cuanto a rendimiento dado en MB/s (2^{20} bytes pos segundos) con los datos de las tablas mencionados anteriormente, donde

- ***n*** es el orden de las matrices que se multiplican (y el orden de la matriz B que se transfiere entre las máquinas.
- ***Cant. Máq.*** es la cantidad de computadoras que se utilizan en paralelo y donde se instrumentó el código para obtener, entre otros, los tiempos de comunicaciones.
- ***Algoritmo*** indica el algoritmo paralelo utilizado.
- ***Tiempo*** es el tiempo local promedio empleado para las comunicaciones.
- ***MB/s*** denota el rendimiento de la red de interconexión dado en Megabytes (2^{20} bytes) por segundo, calculado de acuerdo al tiempo de transferencia de la matriz B (que es la única que se debe transferir entre las computadoras).

n	*Cant. Máq.*	*Algoritmo*	*Tiempo*	*MB/s*
2000	5	SeqMsg	65.12	0.23
2000	10	SeqMsg	253.42	0.06
3200	7	OverMsg	347.52	0.04
3200	10	OverMsg	676.14	0.02

Tabla 4.9: Rendimiento de las Comunicaciones en el CeTAD.

Las dos conclusiones más importantes con respecto al rendimiento de las comunicaciones utilizando los datos de la Tabla 4.9 son:

1. En general, es muy bajo, ya que en el mejor de los casos se utiliza menos del 25% de la capacidad máxima teórica de la red de interconexión.
2. Para una cantidad dada de datos a transferir se cumple que con mayor cantidad de computadoras el rendimiento disminuye.

4.7.2 Red Local del LQT

Tanto los perfiles de ejecución de cada uno de los algoritmos como los resúmenes de tiempos de ejecución de cómputo y de mensajes son muy similares a los que se mostraron del CeTAD. Es claro que no coinciden en términos absolutos por las diferencias en cuanto a computadoras y tamaños de matrices con las que se lleva a cabo el procesamiento. Lo que es significativamente similar es el "comportamiento" en cuanto a rendimiento de las diferentes máquinas paralelas y su consiguiente conclusión en cuanto al problema a resolver: sigue siendo el de las comunicaciones.

La Figura 4.28 muestra el perfil de ejecución que corresponde al mejor tiempo paralelo utilizado para resolver una multiplicación de matrices de 5000×5000 elementos. Este tiempo se obtiene con el algoritmo de cómputo y comunicación solapados utilizando tres computadoras. Es interesante notar dos aspectos bastante relevantes a partir del perfil de ejecución de la Figura 4.28:

- Se aprovecha bastante efectivamente el solapamiento de las comunicaciones con cómputo local. Más específicamente, las computadoras que reciben los mensajes broadcast enviados desde **lqt_06** y **lqt_02** esperan por estos datos muy poco tiempo más del esperado.

- Casi todo el tiempo de comunicaciones que afecta el tiempo total de ejecución es el que corresponde al primer broadcast, por el que las máquinas tienen que esperar inicialmente, es decir el que se envía desde **lqt_07** y que se recibe en **lqt_06** y **lqt_02**.

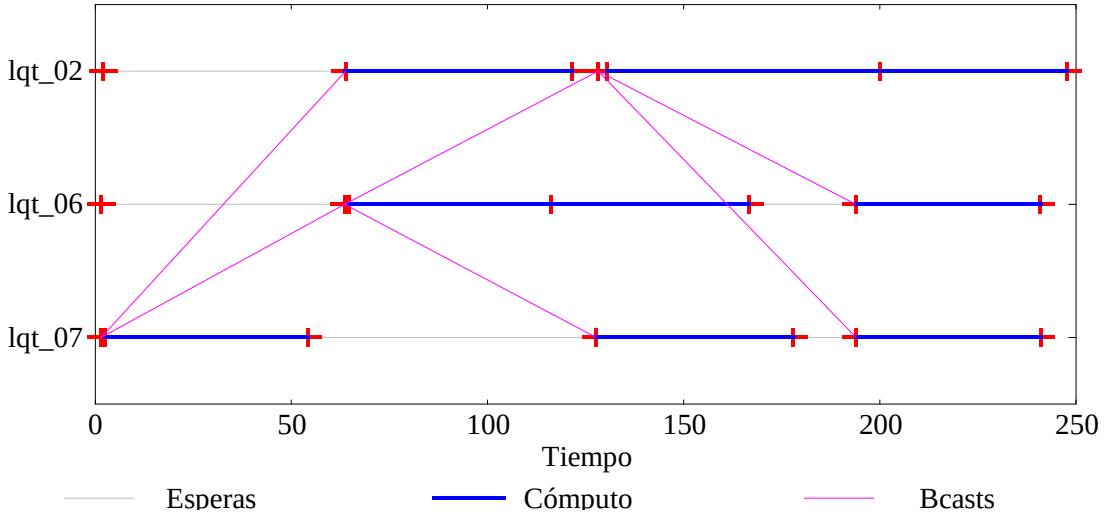


Figura 4.28: Perfil de OverMsg(PVM) con Tres Máquinas y n = 5000 en el LQT.

La Tabla 4.10 muestra el resumen de la ejecución paralela correspondiente al perfil de la Figura 4.28.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
lqt_07	1808	148.87	49.62	90.60
lqt_06	1807	149.06	49.69	90.34
lqt_02	1385	175.20	58.40	70.66

Tabla 4.10: Resumen de OverMsg(PVM) con Tres Máquinas y n = 5000 en el LQT.

Como ya se aclaró con los valores de speedup obtenidos por los algoritmos, el rendimiento empeora cuando se utilizan más máquinas de la red local del LQT. Esto se visualiza claramente en la Figura 4.29, que muestra el perfil de ejecución de una multiplicación de matrices de 5000×5000 elementos utilizando el algoritmo con comunicaciones y cómputo solapados y todas las computadoras disponibles del LQT.

También en la Figura 4.29 es evidente que hay una computadora en particular que se comporta de manera diferente al resto, o por lo menos que el mensaje broadcast enviado desde **lqt_01** utiliza un tiempo de transmisión significativamente mayor que los demás. Una vez más, como cuando esto se identificó la misma situación en la red local del CeTAD (Figura 4.24), se debe aclarar que aunque este broadcast utilizara el tiempo promedio de comunicaciones que utilizan los demás, el tiempo total de comunicaciones seguiría siendo dominado por la suma de los tiempos utilizados para las comunicaciones.

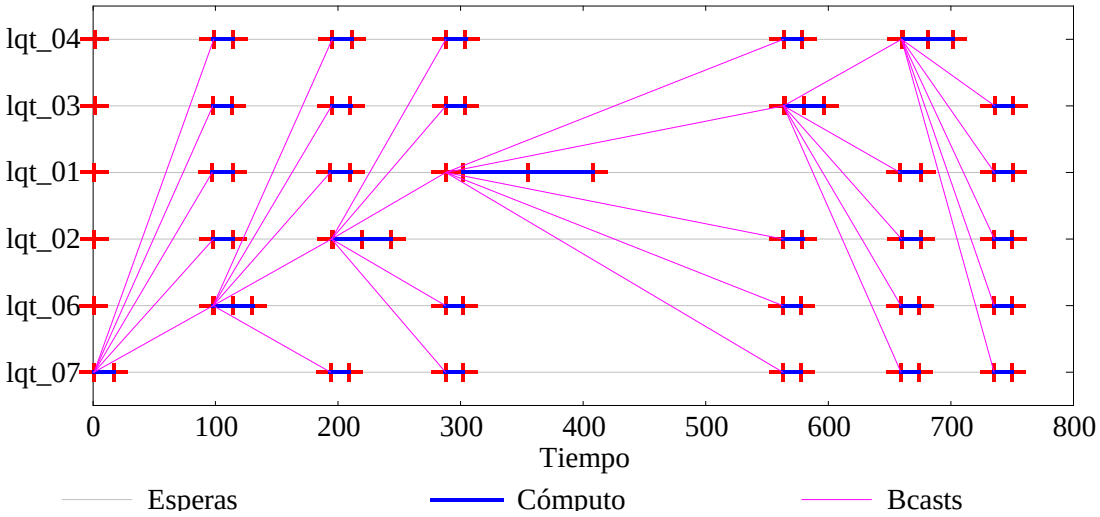


Figura 4.29: Perfil de OverMsg(PVM) con Seis Computadoras y n = 5000 en el LQT.

La Tabla 4.11 muestra el resumen de ejecución de ejecución de la multiplicación de matrices de 5000×5000 elementos utilizando el algoritmo con comunicaciones y cómputo solapados y todas las computadoras disponibles del LQT que *corresponde* a la Figura 4.29. Como ya fue posible identificar a partir de la experimentación con las computadoras del CeTAD, lo que cambia significativamente en cuanto a rendimiento es el tiempo de comunicaciones. Este comportamiento se corrobora en el LQT, comparando la columna que muestra los tiempos utilizados para comunicaciones en cada máquina (***Tot. Msg.***) de la Tabla 4.10 con la de la Tabla 4.11. Cuando se utilizan más máquinas, la misma cantidad de datos (los elementos de la matriz B) se transfieren entre las computadoras en mucho mayor tiempo.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
lqt_07	1089	89.21	14.87	657.71
lqt_06	1089	89.15	14.86	657.57
lqt_02	835	109.13	18.19	637.87
lqt_01	811	174.52	29.09	572.56
lqt_03	589	92.79	15.47	654.50
lqt_04	587	102.52	17.09	592.84

Tabla 4.11: Resumen de OverMsg(PVM) con Seis Máquinas y n = 5000 en el LQT.

En el caso de las matrices de orden n = 9000, la situación en cuanto a rendimiento es similar. La Figura 4.30 muestra el perfil de ejecución que corresponde al mejor tiempo paralelo utilizado para resolver una multiplicación de matrices de 9000×9000 elementos. Este tiempo se obtiene con el algoritmo de cómputo y comunicación solapados utilizando las cuatro computadoras con mayor capacidad de cálculo.

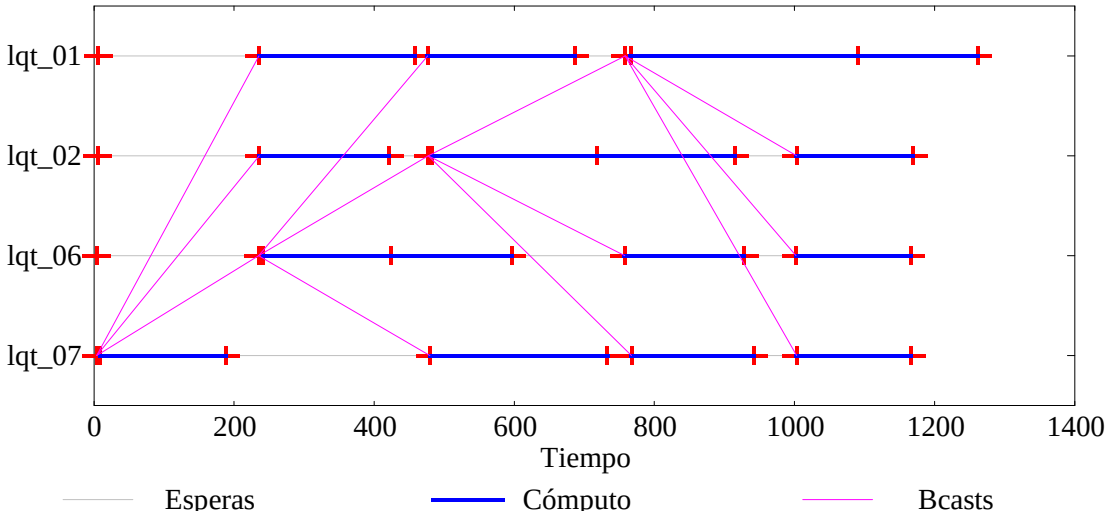


Figura 4.30: Perfil de OverMsg(PVM) con Cuatro Computadoras y n = 9000 en el LQT.

Una vez más, cuando se utilizan más máquinas para resolver el mismo problema (una multiplicación de matrices de 9000×9000 elementos), con el mismo algoritmo (que resuelve las comunicaciones solapadas con cómputo local), el rendimiento empeora y el tiempo total de ejecución es mayor. La Figura 4.31 muestra el perfil de ejecución para resolver una multiplicación de matrices de 9000×9000 elementos en paralelo utilizando todas las computadoras disponibles del LQT y el algoritmo con cómputo y comunicaciones solapados. Los resúmenes de la ejecución no se muestran en este caso, pero no hacen más que confirmar lo que se puede concluir visualmente a partir de la Figura 4.30 y la Figura 4.31: la pérdida de rendimiento se debe al excesivo tiempo de comunicaciones al utilizar mayor cantidad de computadoras.

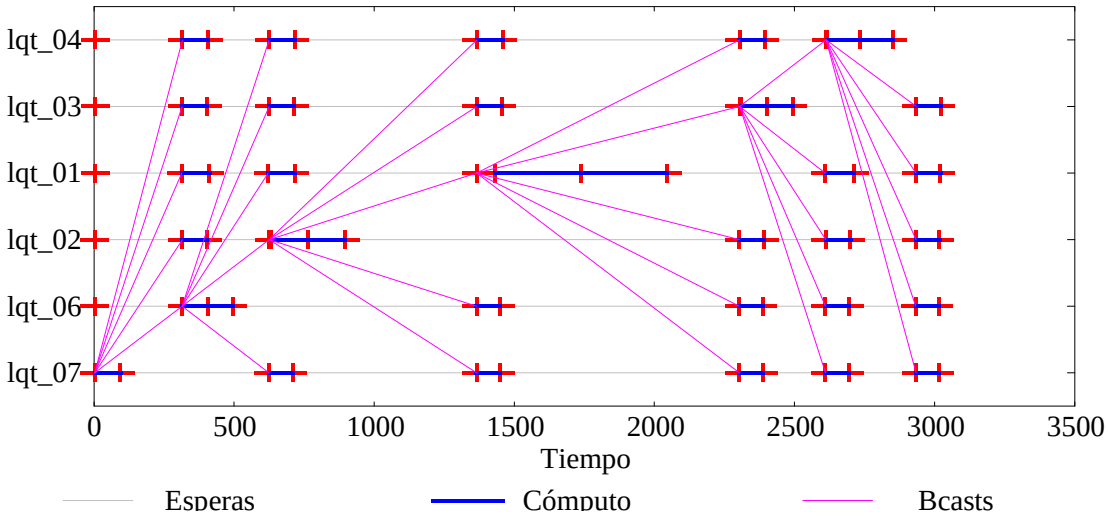


Figura 4.31: Perfil de OverMsg(PVM) con Seis Computadoras y n = 9000 en el LQT.

Específicamente con respecto al rendimiento de las comunicaciones, al analizar los valores de tiempo de ejecución para matrices de orden n = 5000 (dados en la Tabla 4.10 y en la Tabla 4.11), y de orden n = 9000 (que no se muestran explícitamente), sucede lo mismo

que en la red local del CeTAD:
1. En general, es muy bajo comparado con el tiempo de comunicaciones.
2. Para una cantidad dada de datos a transferir se cumple que con mayor cantidad de computadoras el rendimiento disminuye. El tiempo necesario para realizar un broadcast aumenta linealmente con la cantidad de computadoras involucradas.

4.7.3 Red Local del LIDI

En la red local del LIDI se confirman los resultados anteriores pero con la diferencia dada por el mejor rendimiento de la red de interconexión, que es diez veces mejor que el de las redes locales del CeTAD y del LQT. En este sentido, los perfiles de ejecución (y los resúmenes de tiempos de cómputo y comunicaciones) muestran que:
- A diferencia de lo que sucede en las redes locales del CeTAD y del LQT, el tiempo de comunicaciones no tiene un peso tan determinante en el tiempo total de los programas paralelos.
- Tal como sucede en las redes locales del CeTAD y del LQT, el tiempo de comunicaciones aumenta notablemente a medida que se utilizan más computadoras para resolver un mismo problema.

A modo de ejemplo, la Figura 4.32 muestra el perfil de la ejecución del programa paralelo con cómputo y comunicaciones secuenciales utilizando cuatro computadoras para resolver una multiplicación de matrices de orden n = 2000.

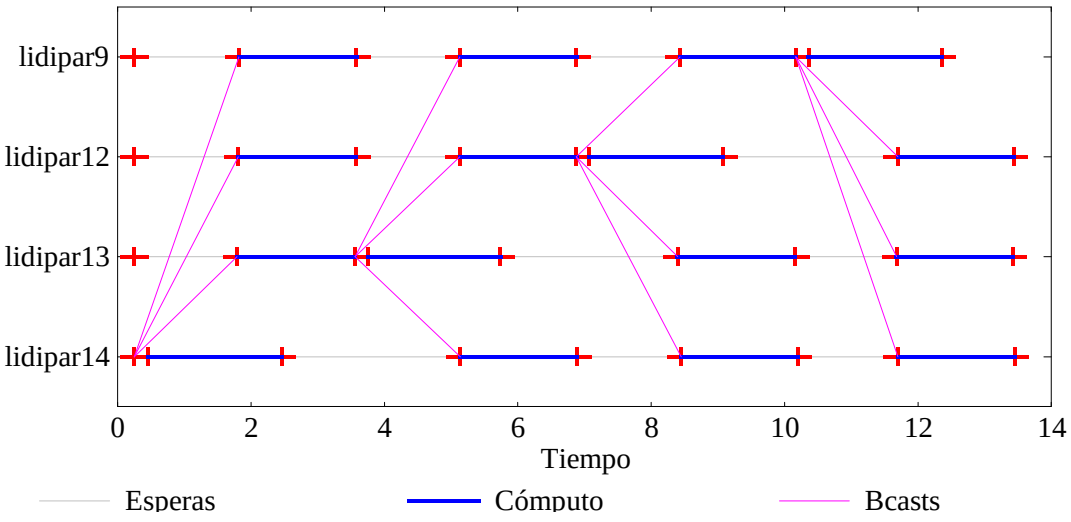


Figura 4.32: Perfil de SeqMsg(PVM) con Cuatro Computadoras y n = 2000 en el LIDI.

Además, Tabla 4.12 muestra el resumen de la ejecución del programa paralelo con cómputo y comunicaciones secuenciales utilizando cuatro computadoras para resolver una multiplicación de matrices de orden n = 2000. Los datos de esta tabla nuevamente completan la visión gráfica de la Figura 4.32 y además permite cuantificar aspectos importantes como la relación entre los tiempos de cómputo y comunicaciones que cada máquina utiliza durante la ejecución del programa.

Nombre	*Filas*	*Tot. Cómp.*	*Por It.*	*Tot. Msg.*
lidipar14	500	7.25	1.81	6.30
lidipar13	500	7.29	1.82	6.16
lidipar12	500	7.26	1.81	6.31
lidipar9	500	7.24	1.81	5.24

Tabla 4.12: Resumen de SeqMsg(PVM) con Cuatro Máquinas y n = 2000 en el LIDI.

La Figura 4.33 y la Tabla 4.13 muestran el perfil y el resumen de la ejecución del programa paralelo con cómputo y comunicaciones secuenciales utilizando todas las computadoras para resolver una multiplicación de matrices de orden n = 2000. Es decir que se mantienen el algoritmo y el tamaño de las matrices (respecto de la Figura 4.32 y la Tabla 4.12) pero se aumenta al doble la cantidad de computadoras que se utilizan en paralelo para resolver el problema. En teoría, el tiempo total de cómputo debería reducirse a la mitad del que se tiene en la Figura 4.32, pero no solamente no es la mitad sino que es mayor.

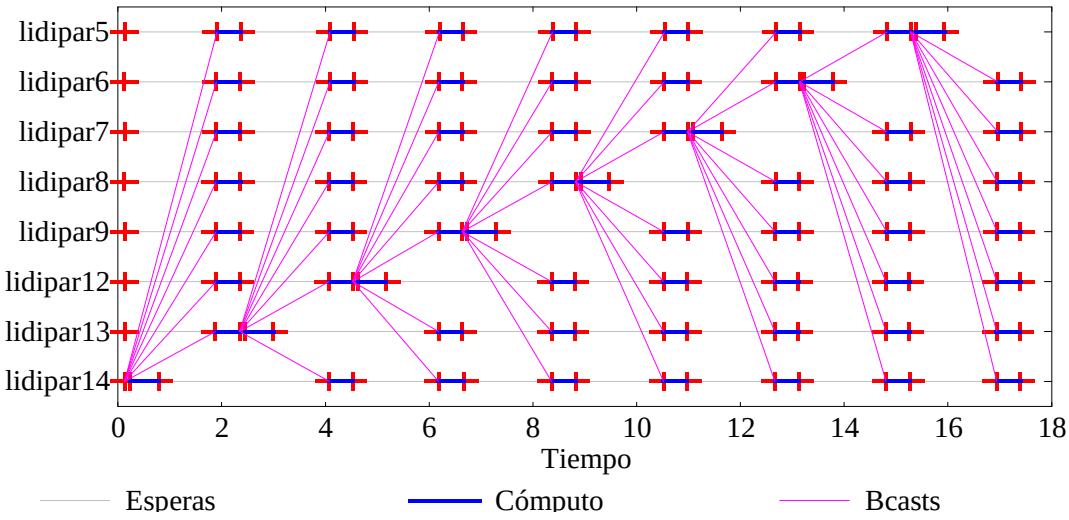


Figura 4.33: Perfil de SeqMsg(PVM) con Ocho Computadoras y n = 2000 en el LIDI.

Comparando los valores de la Tabla 4.12 y la Tabla 4.13:

- En la Tabla 4.12 (donde se utilizan cuatro computadoras) se nota que cada computadora utiliza un poco más de tiempo en cómputo local que en comunicaciones. La proporción de tiempos es aproximadamente 45% - 55%.
- La situación cambia cuando se utilizan ocho computadoras, tal como se muestra en los valores de la Tabla 4.13, ya que cada computadora utiliza para las comunicaciones un promedio de más de tres veces mayor que el tiempo que utiliza para cómputo local. Nuevamente se debe recordar que la cantidad de datos que se deben comunicar son exactamente los *mismos* y además vía mensajes broadcast en una red de interconexión que es capaz de manejar mensajes broadcast a nivel físico.

Nombre	*Filas*	*Tot. Cómp.*	*Por It.*	*Tot. Msg.*
lidipar14	250	3.78	0.47	13.49
lidipar13	250	3.75	0.47	13.52
lidipar12	250	3.73	0.47	13.54
lidipar9	250	3.74	0.47	13.54
lidipar8	250	3.73	0.47	13.55
lidipar7	250	3.74	0.47	13.54
lidipar6	250	3.72	0.46	13.56
lidipar5	250	3.73	0.47	12.08

Tabla 4.13: Resumen de SeqMsg(PVM) con Ocho Máquinas y n = 2000 en el LIDI.

Es interesante notar lo que sucede cuando se resuelve la multiplicación de matrices de orden n = 3200. La Figura 4.34 y la Figura 4.35 muestran los perfiles de ejecución del programa paralelo con cómputo y comunicaciones secuenciales que resuelven una multiplicación de matrices de orden n = 3200 utilizando cuatro y ocho computadoras respectivamente. Evidentemente, el tiempo de ejecución con cuatro computadoras (poco más de 90 segundos, Figura 4.34) es bastante mayor que el tiempo de ejecución con ocho computadoras (aproximadamente 50 segundos, Figura 4.35), pero esto no asegura por sí solo que el rendimiento es aceptable.

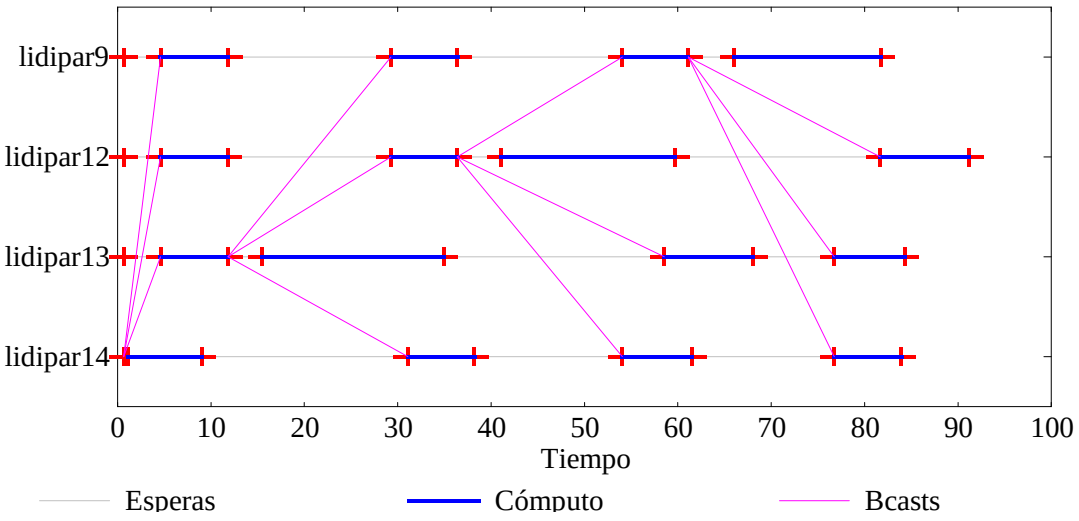


Figura 4.34: Perfil de SeqMsg(PVM) con Cuatro Computadoras y n = 3200 en el LIDI.

La información gráfica de la Figura 4.34 se completa con los datos de la Tabla 4.14 que muestra el resumen de tiempos de ejecución y, de manera análoga, la información gráfica de la Figura 4.35 se completa con los datos de la Tabla 4.15 que también muestra el resumen de tiempos de ejecución.

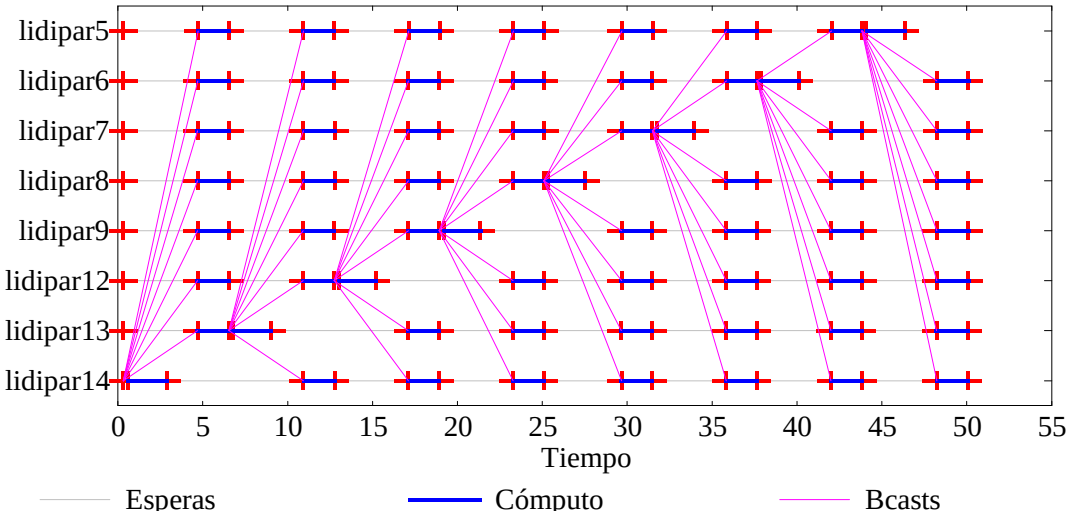


Figura 4.35: Perfil de SeqMsg(PVM) con Ocho Computadoras y *n* = 3200 en el LIDI.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
lidipar14	800	30.20	7.55	55.40
lidipar13	800	43.00	10.75	42.79
lidipar12	800	43.24	10.81	49.37
lidipar9	800	38.20	9.55	44.69

Tabla 4.14: Resumen de SeqMsg(PVM) con Cuatro Máquinas y *n* = 3200 en el LIDI.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
lidipar14	400	14.97	1.87	35.13
lidipar13	400	14.98	1.87	35.12
lidipar12	400	14.97	1.87	35.13
lidipar9	400	14.94	1.87	35.17
lidipar8	400	14.95	1.87	35.16
lidipar7	400	14.98	1.87	35.14
lidipar6	400	14.89	1.86	35.22
lidipar5	400	15.01	1.88	31.47

Tabla 4.15: Resumen de SeqMsg(PVM) con Ocho Máquinas y *n* = 3200 en el LIDI.

Se puede apreciar en la Figura 4.34 que hay períodos de cómputo que son sustancialmente mayores que otros. Más específicamente, en las computadoras **lidipar13**, **lidipar12**, y **lidipar9**, el período de cómputo posterior al envío del mensaje broadcast es mayor que todos los demás períodos de cómputo locales. Por la definición misma del algoritmo, en todos los períodos de cómputo se lleva a cabo la misma tarea y por lo tanto el tiempo de

cómputo debería ser igual.

Una de las principales razones para que una computadora tenga menor rendimiento (asumiendo que se realiza exactamente la misma tarea, como en este caso), es la utilización de la memoria swap que se ha explicado antes. Por lo tanto, la explicación más *aceptable* está relacionada justamente con la memoria. Dado que para llevar a cabo un mensaje broadcast se utiliza PVM, cada una de las rutinas de comunicaciones tiene su propio requerimiento de memoria agregado (overhead), básicamente para almacenamiento de los mensajes en *buffers* (memoria intermedia) que PVM se encarga luego de transferir entre las máquinas. En general es aceptado que la cantidad de memoria extra en este sentido es como mínimo igual a la cantidad de datos que se transfieren. Esta sobrecarga (overhead) de memoria hace que la memoria principal disponible no sea suficiente y por lo tanto se utiliza el espacio de memoria swap. En realidad, esto genera una reducción del rendimiento de

- Cómputo, dado que parte de los datos a procesar quizás están asignados en memoria swap.
- Comunicaciones, dado que parte de los datos a transferir quizás están asignados en memoria swap.

Y es por esta razón los períodos de cómputo posteriores a un broadcast son “más lentos” que los demás.
A diferencia de lo que ha sucedido hasta ahora (en las redes locales del CeTAD y del LQT), el tiempo total de comunicaciones en promedio es bastante mayor para cuatro máquinas que para ocho, lo cual se puede comprobar cuantitativamente con los datos de la Tabla 4.14 y la Tabla 4.15 respectivamente.

El perfil y el resumen de ejecución para el mismo problema pero con seis máquinas que se muestran en la Figura 4.36 y en la Tabla 4.16 confirman que el problema de rendimiento anterior (que se puede visualizar en la Figura 4.34) está relacionado con la memoria ya que

- Todos los períodos de cómputo en todas las computadoras utilizan aproximadamente el mismo tiempo de ejecución.
- El tiempo total de comunicaciones es menor que para cuatro y ocho computadoras.

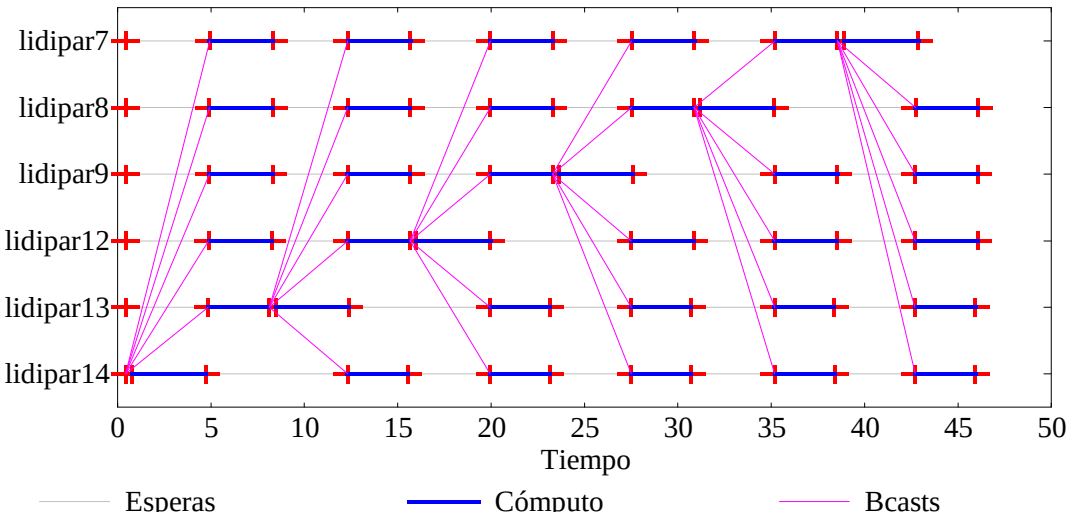


Figura 4.36: Perfil de SeqMsg(PVM) con Seis Computadoras y n = 3200 en el LIDI.

Nombre	*Filas*	*Tot. Cómp.*	*Por It.*	*Tot. Msg.*
lidipar14	534	19.99	3.33	25.53
lidipar13	534	19.98	3.33	25.53
lidipar12	533	20.75	3.46	24.89
lidipar9	533	20.73	3.46	24.91
lidipar8	533	20.75	3.46	24.90
lidipar7	533	20.77	3.46	21.68

Tabla 4.16: Resumen de SeqMsg(PVM) con Seis Máquinas y n = 3200 en el LIDI.

4.8 Rendimiento Real de las Redes Locales Utilizando "UDP"

Dado que el problema de rendimiento es generado casi exclusivamente por las comunicaciones, se llevaron a cabo experimentos específicos para evaluar el rendimiento de los mensajes broadcast y también punto a punto utilizando la biblioteca PVM. Si bien los resultados se muestran detalladamente en el Apéndice C, las principales conclusiones son [120]:

- Las formas de enviar un mismo mensaje a más de un proceso destino cuando cada proceso está asignado a una máquina distinta se implementan con múltiples mensajes punto a punto. Tanto
  - la operación de multicast, **pvm_mcast()**,
  - como la operación de broadcast en un grupo, **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.
- El tiempo de latencia de los mensajes depende de las computadoras origen y destino del mismo. Sin embargo, para mensajes suficientemente grandes el tiempo de latencia no tiene relevancia con respecto al tiempo de transferencia de los datos y por lo tanto el tiempo total del mensaje es independiente de las máquinas que se comunican.

La rutina de comunicaciones broadcast propuesta está orientada al aprovechamiento de las características físicas (específicamente broadcast) de las redes Ethernet que la biblioteca de comunicaciones (PVM) no utiliza. Esto tiene como consecuencia directa que el tiempo de comunicaciones broadcast con PVM es mucho mayor que el esperado y por lo tanto no se tienen buenos resultados de rendimiento total. 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 [123]. 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 las características 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 que 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 [123] [132]:
- 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 la 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 [31] [25] [GAMMA]. 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 [15] [14] [16] 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) [95] sobre IP (Internet Protocol) [96] 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 cuenta, siempre que exista una implementación del protocolo UDP se podrá utilizar el broadcast propuesto, independientemente de que la red de interconexión sea Ethernet o no [93].
- 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 dispone de *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 nueva rutina de mensajes broadcast se

vuelven a realizar los mismos experimentos y los resultados se muestran en las subsecciones que siguen.

Los resultados de la experimentación muestran varias características que no han sido encontradas hasta ahora. La aparición de la mayoría de estas características está dada por:

- El rendimiento marcadamente superior del broadcast basado en UDP sobre el de la biblioteca PVM. Esto genera que el tiempo de comunicaciones sea comparable (al menos en el mismo orden de magnitud) con el del cómputo y por lo tanto el rendimiento de cómputo de cada computadora es importante dado su peso en el tiempo de ejecución total. Hasta ahora, el tiempo de comunicaciones ha sido tan superior que básicamente todo o la gran mayoría del tiempo de ejecución paralela se debe a las comunicaciones.
- La heterogeneidad de las computadoras, lo cual en sí mismo aporta un grado importante de innovación a lo que generalmente se muestra sobre rendimiento de máquinas paralelas.

Dado que ahora la heterogeneidad y más específicamente las diferencias de rendimiento de cómputo de las computadoras se vuelve relevante, la mayoría de las nuevas características aparecen en la red local del CeTAD, que es la más heterogénea de las tres en las que se lleva a cabo la experimentación.

En general, en las redes locales del LQT y del LIDI se repiten algunas características que aparecen en la red local del CeTAD o directamente no aparecen. A lo sumo se acentúa en el caso de la red local del LQT la influencia de los requerimientos de cómputo cuando los problemas a resolver son (o pueden ser) mayores por la mayor cantidad de memoria disponible. Por esta razón los resultados de la experimentación en la red local del CeTAD serán explicados con el mayor nivel de detalle posible y en el caso de las otras dos redes locales se verán solamente las diferencias en cuanto a “comportamiento” siempre desde el punto de vista del rendimiento.

4.8.1 Red Local del CeTAD

Con el objetivo de presentar y separar mejor los resultados de la experimentación en la red local del CeTAD, todo el análisis de los datos obtenidos se divide en las dos subsecciones que siguen de acuerdo con el tamaño del problema. La primera se dedica a los resultados de speedup tomando como referencia el tiempo de ejecución de la multiplicación de matrices de orden $n = 2000$. Posteriormente se analizan los resultados de speedup tomando como referencia el tiempo de ejecución de la multiplicación de matrices de orden $n = 3200$.

4.8.1.1 Matrices de 2000×2000 Elementos

La Figura 4.37 muestra los valores de speedup obtenidos en la red local del CeTAD por los algoritmos implementados utilizando UDP, SeqMsg(UDP) y OverMsg(UDP) para matrices de orden $n = 2000$ junto con los que se mostraron anteriormente en la Figura 4.17.

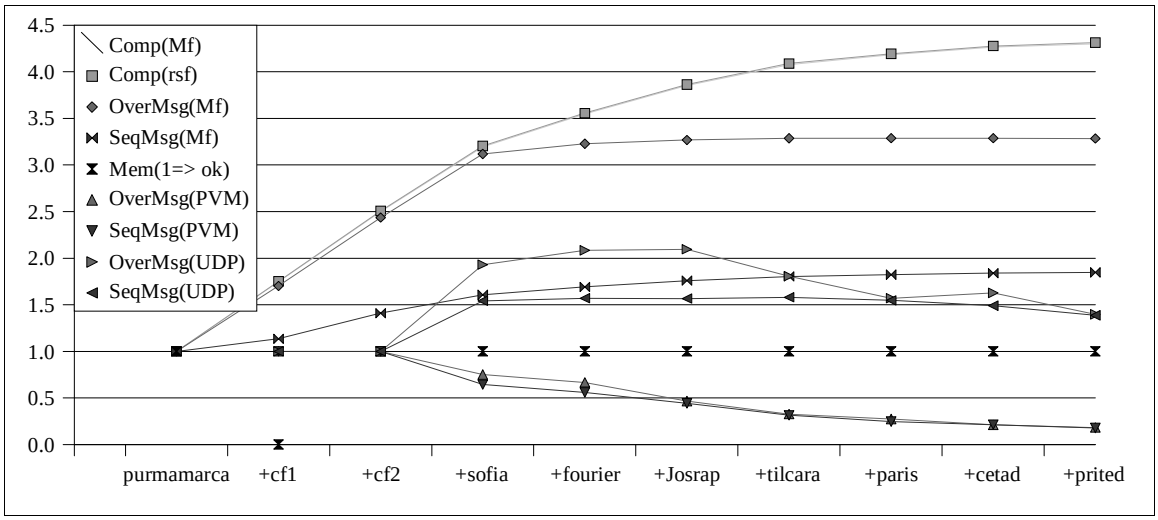


Figura 4.37: Speedup de los Algoritmos con UDP en la Red del CeTAD para n = 2000.

Tal como se puede apreciar en la Figura 4.37, el rendimiento de los algoritmos SeqMsg(UDP) y OverMsg(UDP) (que son los únicos que se agregan con respecto a los que se, muestran en la Figura 4.17), varía dependiendo de la cantidad de computadoras que se utilizan. Esta variación de los algoritmos no parece, *a priori*, estar relacionada. Inicialmente se analizarán detalladamente los resultados del algoritmo con los períodos de cómputo y comunicaciones secuenciales, SeqMsg(UDP), desde tres puntos de vista:

- La relación de los resultados de rendimiento obtenidos comparándolos con los que se obtienen al utilizar la biblioteca de comunicaciones PVM.
- La capacidad del algoritmo de utilizar al máximo la capacidad de cómputo de las máquinas, en los períodos de cómputo.
- La capacidad del algoritmo de utilizar al máximo la capacidad de la red de comunicaciones, en los períodos de comunicaciones.

Posteriormente, el mismo tipo de análisis de resultado (siempre desde el punto de vista del rendimiento en general) se hará con el algoritmo orientado al aprovechamiento de la capacidad de solapar cómputo con comunicaciones, OverMsg(UDP).

SeqMsg(UDP). El algoritmo que lleva a cabo los períodos de cómputo y comunicaciones de manera secuencial, SeqMsg(UDP), no parece alejarse demasiado de la estimación del speedup máximo que le corresponde, SeqMsg(Mf), al menos hasta que se utilizan las diez máquinas disponibles.

La Tabla 4.17 muestra el resumen de ejecución cuando se utilizan seis máquinas para multiplicar matrices de 2000×2000 elementos con este algoritmo. Comparando los valores que se muestran en la Tabla 4.17 con los que se muestran en la Tabla 4.4, donde lo único diferente es que se utiliza la rutina de broadcast provista por PVM se puede ver que:

- En términos de tiempo de cómputo local son muy similares, cuando se utiliza PVM el promedio total es de 15.61 segundos y con la rutina basada directamente en UDP el promedio es de 14.31 segundos.
- Los tiempos de comunicaciones son absolutamente diferentes, cuando se utiliza PVM el promedio total es de 65.12 segundos y con la rutina basada directamente en UDP el promedio es de 15.69 segundos, un poco más de cuatro veces menor.

Nombre	*Filas*	*Tot. Cómp.*	*Por It.*	*Tot. Msg.*
purmamarca	555	14.15	2.83	15.89
cf1	426	14.14	2.83	15.86
cf2	426	14.09	2.82	15.91
sofia	394	14.40	2.88	15.71
fourier	199	14.78	2.96	15.08

Tabla 4.17: Resumen de SeqMsg(UDP) con Cinco Máquinas y n = 2000 en el CeTAD.

La diferencia en tiempo de comunicaciones evidentemente está dada por la forma en que PVM resuelve los mensajes broadcast: múltiples mensajes punto a punto entre las computadoras involucradas. Pero más allá de la comparación con la implementación basada en PVM, es necesario verificar si cada computadora se utiliza al máximo de su capacidad y si la red de interconexión se utiliza con rendimiento óptimo. Este análisis se hará para el caso en el cual se utilizan las diez computadoras, dado que el rendimiento disminuye y se pueden identificar un poco mejor las causas de esta degradación del rendimiento.

La Tabla 4.18 muestra el resumen de ejecución cuando se utilizan todas las máquinas (diez) para multiplicar matrices de 2000×2000 elementos con el algoritmo que lleva a cabo cómputo y comunicaciones secuenciales, SeqMsg(UDP). Comparando los valores de la Tabla 4.18 con los de la tabla anterior se puede notar que:

- En términos de cómputo, no hay muchos cambios dado que las cinco computadoras que se agregan en realidad tienen muy poca capacidad de cálculo con respecto al total de las demás. La cantidad total de procesamiento de las computadoras es directamente proporcional a la cantidad de filas asignadas, que a su vez está dada por la velocidad relativa de cada computadora, y las primeras cinco máquinas tienen a su cargo el 82% del total.
- El tiempo de comunicaciones ha aumentado de 15.69 segundos con cinco máquinas a 21.2 segundos con diez máquinas, lo cual representa una penalización en el rendimiento de las comunicaciones de un poco más de 35%. Evidentemente un 35% de degradación en el rendimiento de las comunicaciones es mucho mejor que el 500% que implica utilizar PVM pero de todas maneras parece muy elevado. Sin llegar a detalles muy específicos de lo que sucede con los mensajes broadcast en términos de rendimiento se puede decir que:
    - A medida que más procesos estén involucrados en comunicaciones colectivas y en particular en mensajes broadcast, es de esperar una mayor penalización en términos de rendimiento. Cuando todos los procesos están ejecutándose en computadoras diferentes y además las computadoras están interconectadas con una red Ethernet de 10 Mb/s la degradación será mayor.
    - Las computadoras que se incorporan son relativamente mucho más lentas que las demás. Aún cuando la capacidad de transferencia de las placas de red sea de 10 Mb/s independientemente de las computadoras en las que estén instaladas, está demostrado al menos experimentalmente que como mínimo el tiempo de latencia de los mensajes depende de la capacidad de cómputo de las máquinas. A medida

que aumenta la cantidad de computadoras, el tamaño de los mensajes disminuye y por lo tanto aumenta la importancia relativa del tiempo de latencia en el tiempo total de cada mensaje.

- Y estos factores (más máquinas y con mayor tiempo de latencia de comunicaciones), se combinan para llegar a poco más de 35% de penalización en el rendimiento de las comunicaciones.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	454	11.81	1.18	21.63
cf1	349	11.84	1.18	21.58
cf2	349	11.82	1.18	21.60
sofia	324	11.97	1.20	21.48
fourier	164	12.67	1.27	20.72
Josrap	142	12.10	1.21	21.43
tilcara	104	12.08	1.21	21.27
paris	48	11.59	1.16	21.53
cetad	38	11.98	1.20	21.03
prited	28	12.65	1.27	19.75

Tabla 4.18: Resumen de SeqMsg(UDP) con Diez Máquinas y n = 2000 en el CeTAD.

En este punto se podría analizar cuál es el rendimiento de *esta* red de interconexión para los mensajes broadcast cuando se utilizan cinco y diez máquinas. Se debe recordar que en todos los casos la cantidad total de datos a transferir está dada por la cantidad de elementos de la matriz B (C = A×B). En el caso de matrices de orden n = 2000 y con números en punto flotante de precisión simple, la cantidad dada en bytes es exactamente 4×2000^2. Desde el punto de vista del rendimiento, entonces:

- Cuando se utilizan las cinco computadoras con mayor capacidad de cálculo del CeTAD el tiempo total de las comunicaciones en cada máquina es en promedio 15.69 segundos. Esto representa un ancho de banda real entre procesos de un poco más de 8 Mb/s. Por lo tanto se tiene menos de 20% de penalización respecto del máximo absoluto del hardware en la comunicación de los mensajes broadcast entre procesos.
- Cuando se utilizan todas las computadoras del CeTAD, el tiempo total de las comunicaciones en cada máquina es en promedio 21.2 segundos. Esto representa un ancho de banda real entre procesos de un poco más de 6 Mb/s. Por lo tanto se tiene menos de 40% de penalización respecto del máximo absoluto del hardware en la comunicación de los mensajes broadcast entre procesos.

Hasta aquí se ha simplificado la métrica de tiempo de los mensajes al asumir que el tiempo dedicado a la transferencia de los datos entre las computadoras es igual al tiempo de espera por la comunicación de los datos en cada computadora. Sin embargo, es necesario recordar que el tiempo de comunicaciones del algoritmo en cada computadora en realidad es el tiempo durante el cual se debe esperar a que se complete un mensaje broadcast. Esto

significa que todo desbalance de carga de procesamiento genera que en algunas computadoras se espera más o menos por los datos. En la Tabla 4.18, por ejemplo, se puede notar que en general se cumple que las computadoras con mayor tiempo dedicado a cómputo, **fourier** y **prited**, tienen menor tiempo de comunicaciones, porque en realidad una parte del tiempo de comunicaciones que se contabiliza en las demás computadoras es este tiempo "extra" de cómputo que las computadoras utilizan para procesamiento de los datos de las matrices. Dado que el desbalance de carga en términos de tiempo de cómputo no llega a ser del 10% este tiempo al menos por ahora no se tiene en cuenta para discriminarlo como diferente del tiempo de las comunicaciones.

Además de considerar el rendimiento de las comunicaciones, evidentemente también se puede considerar la evaluación de dos aspectos que tienen una gran influencia en el rendimiento paralelo: el rendimiento de cómputo de cada una de las máquinas en particular y el balance de carga *real* de la máquina paralela.

Rendimiento de cómputo local-secuencial de SeqMsg(UDP). Desde el punto de vista del rendimiento de cómputo de cada máquina se tendría que analizar cuál es la penalización con respecto a la máxima capacidad de procesamiento por utilizar las computadoras en paralelo. En este sentido, quizás el factor más importante a identificar o cuantificar es la "interferencia" de las comunicaciones sobre el rendimiento de cómputo secuencial de cada computadora. Expresado de otra manera, desde el punto de vista del rendimiento no es lo mismo *solamente* computar-procesar datos que llevar a cabo períodos de cómputo *y* períodos de comunicaciones. Es claro que la memoria cache, por ejemplo, debe ser compartida o utilizada en todos los períodos, y por lo tanto no es lo mismo hacer toda una multiplicación de matrices que procesar un tercio, comunicar datos, procesar el segundo tercio, comunicar datos y procesar el último tercio.

Cuando se tienen máquinas heterogéneas, el impacto de las comunicaciones sobre el rendimiento de cómputo no necesariamente será el mismo. Siguiendo con el ejemplo de la utilización de la memoria cache, en principio el impacto en el rendimiento de cómputo puede ser distinto dependiendo del tamaño de memoria cache. En el Apéndice A se puede comprobar que las máquinas del CeTAD tienen una gran *variedad* de tamaños y tipos de memoria cache.

Se puede evaluar el rendimiento de cómputo local de cada computadora utilizando los valores que resumen la ejecución paralela para cinco y diez máquinas dados en la Tabla 4.17 y en la Tabla 4.18 respectivamente. La Tabla 4.19 muestra los Mflop/s de cada máquina cuando se considera la paralelización de una multiplicación de matrices de 2000×2000 elementos en cinco computadoras con el algoritmo SeqMsg(UDP). La Tabla 4.20 muestra los Mflop/s cuando se utilizan todas las computadoras. Es importante notar que:

- El rendimiento en cada computadora permanece relativamente invariante a medida que se agregan máquinas y por lo tanto a medida que se disminuye la granularidad. Se debe recordar que aumentar la granularidad implica aumentar la cantidad de mensajes para este algoritmo y por lo tanto se tiene mayor cantidad de períodos de cómputo en los cuales se llevan a cabo menor cantidad de operaciones.
- El rendimiento de cada computadora es bastante cercano al óptimo, tomando como el óptimo el que se tiene cuando *solamente* se ejecutan operaciones de cálculo

secuenciales. En este sentido, se tiene como referencia la experimentación realizada para evaluar la capacidad de cómputo relativa y que para el caso de las máquinas del CeTAD se muestra en la Figura 4.4 y en la Figura 4.5 para **purmamarca** en particular.

Nombre	***Filas***	***Tot. Cómp.***	***Mflop/s***
purmamarca	555	14.15	314
cf1	426	14.14	241
cf2	426	14.09	242
sofia	394	14.40	219
fourier	199	14.78	108

Tabla 4.19: Mflop/s de SeqMsg(UDP) con Cinco Máquinas y *n* = 2000 en el CeTAD.

Nombre	***Filas***	***Tot. Cómp.***	***Mflop/s***
purmamarca	454	11.81	307
cf1	349	11.84	236
cf2	349	11.82	236
sofia	324	11.97	216
fourier	164	12.67	104
Josrap	142	12.10	94
tilcara	104	12.08	69
paris	48	11.59	33
cetad	38	11.98	25
prited	28	12.65	18

Tabla 4.20: Mflop/s de SeqMsg(UDP) con Diez Máquinas y *n* = 2000 en el CeTAD.

Balance de carga de SeqMsg(UDP). Como se puede calcular a partir de los valores que resumen los tiempos de ejecución paralelos en las diferentes tablas, aparecen algunas diferencias en cuanto a tiempo dedicado a cómputo local que debería explicarse de acuerdo con el balance de carga del definido para el algoritmo. Las diferencias de tiempo de cómputo local que se pueden identificar son:

- En la Tabla 4.19 el menor tiempo de cómputo es el de **cf1**, 14.09 segundos, y el mayor tiempo de cómputo es el de **fourier**, 14.78 segundos. El porcentaje de tiempo que implica este desbalance es menor al 5% del menor tiempo de cálculo.
- En la Tabla 4.20 el menor tiempo de cómputo es el de **paris**, 11.59 segundos, y el mayor tiempo de cómputo es el de **fourier**, 12.67 segundos. El porcentaje de tiempo que implica este desbalance es menor al 10% del menor tiempo de cálculo.

Es claro que aunque desde el punto de vista teórico se puede llegar a una ecuación o expresión que permita un balance de carga exacto, pero que se encuentren diferencias en

tiempo de ejecución. Solamente a modo de ejemplo, las computadoras **cf1** y **cf2** son *iguales* (Tabla 4.1 y Apéndice A), y tanto en la Tabla 4.19 como en la Tabla 4.20 se puede comprobar que las computadoras tienen asignada la misma carga de procesamiento, pero sin embargo tienen distintos tiempos de cómputo local. En general, independientemente del tipo de computadoras, algoritmos y formas de balancear la carga siempre se tendrá un mínimo de diferencias en cuanto a tiempos de ejecución.

Otra fuente de un mínimo desbalance de carga está dada por la definición misma del algoritmo. Tal como se explica en el capítulo anterior, el balance de carga se implementa asignando la cantidad de filas de la matriz resultado que corresponden a la capacidad de cálculo relativa de cada computadora. Por lo tanto, la cantidad de datos a calcular en cada computadora está dada en *cantidad de filas* de la matriz resultado. Sin embargo, no siempre la capacidad de cálculo relativa de las computadoras se pueden expresar como múltiplos unas de otras (o múltiplos de una de referencia) y además no siempre la cantidad total de filas podrá ser distribuida completamente siguiendo esta estrategia. Por lo tanto, el factor de corrección “de una fila” en la distribución de los datos que se explica en el capítulo anterior se puede mencionar como causante de un posible desbalance.

Por último, aunque no necesariamente menos importante otra posible fuente para el desbalance de carga es el que implica la propia heterogeneidad de las computadoras. Aunque cada computadora se mantiene en cuanto a capacidad de cálculo muy cercana al óptimo, no necesariamente esta “cercanía” será igual en todos los casos. Expresado de otra manera, todas las computadoras tienen un porcentaje mínimo de penalización de rendimiento en cuanto a cálculo secuencial cuando se hace cómputo paralelo (por código y *otros* procesos en la computadora necesarios para comunicaciones) y este porcentaje puede variar dependiendo de las características físicas de las computadoras. La variación en este porcentaje de penalización también genera cierto desbalance.

A pesar de que todas las fuentes de desbalance de carga son acumulativas, y a pesar de la gran heterogeneidad en cuanto a capacidad de cálculo de las computadoras, lo peor que se tiene en la experimentación en cuanto a desbalance de carga es menor al 10% del mejor tiempo de cómputo local.

OverMsg(UDP). Todo el detalle que se ha hecho en cuanto al análisis de resultados corresponde al algoritmo con los períodos de cómputo y comunicación secuenciales en cada una de las máquinas. Lo que sucede con el algoritmo que está orientado al solapamiento de las comunicaciones con el cómputo local es diferente y en la Figura 4.37 es evidente que algunos valores son bastante diferentes de los que se esperan. Para poder hacer una comparación más completa de los resultados obtenidos por los dos algoritmos y además identificar mejor las diferencias con la implementación basada en PVM se muestra primero el resumen de la ejecución paralela con cinco máquinas en la Tabla 4.21. Comparando estos valores con los de la Tabla 4.6 correspondientes a la implementación utilizando el broadcast de PVM, las conclusiones son similares a las que resultaron de la comparación de las implementaciones del algoritmo SeqMsg, es decir SeqMsg(PVM) con SeqMsg(UDP):

- Los tiempos de cómputo son casi los mismos, es decir que en términos de rendimiento de cómputo local cada computadora tiene casi la misma capacidad de cálculo (en Mflop/s, por ejemplo).

- Los tiempos de comunicaciones son muy diferentes, en este caso el rendimiento del broadcast utilizando UDP es aproximadamente diez veces mejor que el de PVM.

Nombre	*Filas*	*Tot. Cómp.*	*Por It.*	*Tot. Msg.*
purmamarca	555	16.63	3.33	5.96
cf1	426	17.30	3.46	5.45
cf2	426	17.31	3.46	5.42
sofia	394	18.29	3.66	4.33
fourier	199	16.53	3.31	5.98

Tabla 4.21: Resumen de OverMsg(UDP) con Seis Máquinas y n = 2000 en el CeTAD.

Cuando se comparan los valores de la Tabla 4.21 con los de la Tabla 4.17 se puede notar que:

- Los tiempos de cómputo local de OverMsg(UDP) son mayores que los de SeqMsg(UDP), lo cual es esperable ya que no solamente se llevan a cabo transferencias de datos entre computadoras sino que además en un mismo intervalo de tiempo los procesos de cómputo en cada computadora se ejecutan con los procesos de comunicaciones y por lo tanto la competencia por la CPU y la memoria cache, por ejemplo, es mayor y eso se traduce en mayor tiempo de cómputo.
- Los tiempos de comunicaciones de OverMsg(UDP) son aproximadamente un tercio de los obtenidos con SeqMsg(UDP). Dado que en realidad la misma cantidad de datos se transfieren entre las computadoras, gran parte de cada transferencia de datos transcurre *mientras* se está resolviendo un cálculo parcial de la matriz resultado en cada computadora. En este sentido, al menos para cinco computadoras, el solapamiento de las comunicaciones puede considerarse satisfactorio.

Sin embargo, como se puede notar claramente en la Figura 4.37, a partir de la inclusión de **tilcara** el rendimiento no solamente es menor que el esperado sino que además disminuye notablemente. Aunque se puede comprobar con la inclusión de cada una de las máquinas a partir de **tilcara** (**paris**, **cetad** y **prited**), lo que sucede se puede identificar de forma bastante clara directamente con el resumen de la ejecución de todas las máquinas que se muestra en la Tabla 4.22.

Tal como lo muestra la Tabla 4.22, el tiempo de cómputo local de las seis computadoras con mayor capacidad de cálculo del CeTAD (**purmamarca**, **cetadfomec1**, **cetadfomec2**, **sofia**, **fourier** y **Josrap**) es bastante similar entre sí, aproximadamente 13±0.45 segundos. El tiempo de cómputo de las cuatro computadoras restantes es bastante mayor y con mayor disparidad entre sí, aproximadamente 20±5 segundos. La explicación de lo que sucede es bastante sencilla y tiene relación con la competencia por los recursos mencionada antes. En este algoritmo que lleva a cabo las comunicaciones de manera solapada con el cómputo local, es claro que deben existir uno o más procesos en cada computadora que se ejecutan (transfieren datos) mientras se resuelve un período de cálculos parciales. Esto significa que durante la ejecución de un período de cálculo los recursos más importantes (CPU y memoria cache) serán compartidos con el o los procesos de comunicaciones. Esto como

mínimo genera una degradación del rendimiento de cálculo, pero además, según los valores de la Tabla 4.22 algunas computadoras pueden solapar cómputo y comunicaciones mejor que otras.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	454	12.39	1.24	21.08
cf1	349	12.99	1.30	20.48
cf2	349	13.01	1.30	20.46
sofia	324	13.45	1.34	20.07
fourier	164	13.28	1.33	20.18
Josrap	142	12.82	1.28	20.80
tilcara	104	18.34	1.83	15.08
paris	48	16.91	1.69	16.39
cetad	38	23.26	2.33	9.90
prited	28	24.81	2.48	7.91

Tabla 4.22: Resumen de OverMsg(UDP) con Diez Máquinas y n = 2000 en el CeTAD.

Por lo tanto se llega a que las diferencias en el rendimiento de cómputo solapado con comunicaciones dependen de varios factores, como la arquitectura del bus del sistema o del método de entrada/salida con el cual los datos se comunican a través de las placas de interfase de red instaladas. Si bien es muy difícil explicar más detalladamente lo que sucede en cada computadora, esto significa que algunas computadoras en realidad no son capaces de solapar *realmente* cómputo local con comunicaciones sino que secuencializan el cómputo con las comunicaciones y por lo tanto su rendimiento es el mismo que el que tienen con el algoritmo de cómputo y comunicaciones secuenciales. Incluso puede ser peor porque con OverMsg se tiene mayor sobrecarga (overhead) a nivel de procesos que se ejecutan y que el sistema operativo tiene que manejar (generando mayor cantidad de cambios de contexto, por ejemplo).

Si bien desde el punto de vista del rendimiento total esta explicación no es satisfactoria (de hecho, impone un límite físico dado que depende del hardware de la propia computadora) el propio algoritmo con cómputo solapado con las comunicaciones se *transforma* en un *benchmark*. Expresado de otra manera, OverMsg(UDP) permite de manera automática e independiente de las computadoras conocer con bastante precisión si son capaces de solapar cómputo con comunicaciones o no. De hecho, en este caso hasta se podría dar una cuantificación aproximada que puede estar basada en las diferencias de tiempo empleado para cómputo en cada computadora. Este benchmark toma mayor relevancia cuando se tiene en cuenta que se están utilizando computadoras de cómputo secuencial estándares, donde la posibilidad de solapar cómputo con comunicaciones vía una o más interfases de red no parece estar entre los objetivos prioritarios del diseño.

Aunque se tiene una explicación aproximada a la pérdida de rendimiento de cómputo paralelo a partir de la inclusión de **tilcara**, parece razonable analizar con un poco más de

detalle lo que sucede con el rendimiento de cómputo de cada una de las máquinas en particular y el balance de carga *real* de la máquina paralela.

Rendimiento de cómputo local-secuencial de OverMsg(UDP). Como era de esperar, y por las razones que ya se expusieron, el rendimiento de cálculo de las computadoras disminuye respecto de la situación óptima, es decir cuando solamente se realiza cómputo local. Dado que las máquinas son heterogéneas, la disminución también es *heterogénea*, dependiente de la arquitectura de cada computadora. Dejando de lado las máquinas que secuencializan el cómputo y las comunicaciones (a partir de **tilcara**), puede decirse que el costo de solapar las comunicaciones con el cómputo local es menor que el beneficio, dado que de hecho, el rendimiento total aumenta.

Balance de carga de OverMsg(UDP). Dado que el balance de carga se hace utilizando el rendimiento de cómputo secuencial y por la interferencia *heterogénea* de los procesos de transferencia de datos sobre los de cómputo, el balance de carga de OverMsg(UDP) puede llegar a ser peor que el de SeqMsg(UDP). Quitando del análisis del balance de carga real a las computadoras que no pueden solapar las comunicaciones con cómputo local de manera "aceptable" desde el punto de vista del rendimiento, el desbalance no llega a ser del 10% en el caso de las computadoras del CeTAD. Sin embargo, cuando este desbalance llegue a ser significativo se puede utilizar la información del resumen de ejecución para redefinir la distribución de los datos y distribuir los datos en función del rendimiento de cómputo asumiendo solapamiento de comunicaciones.

4.8.1.2 Matrices de 3200×3200 Elementos

Cuando se toma como referencia el tiempo de ejecución del mayor tamaño de problema que se puede resolver en la computadora de mayor capacidad de cómputo, es decir matrices de orden n = 3200 en **purmamarca**, los valores de speedup obtenidos se muestran en la Figura 4.38 junto con los que se mostraron Figura 4.18.

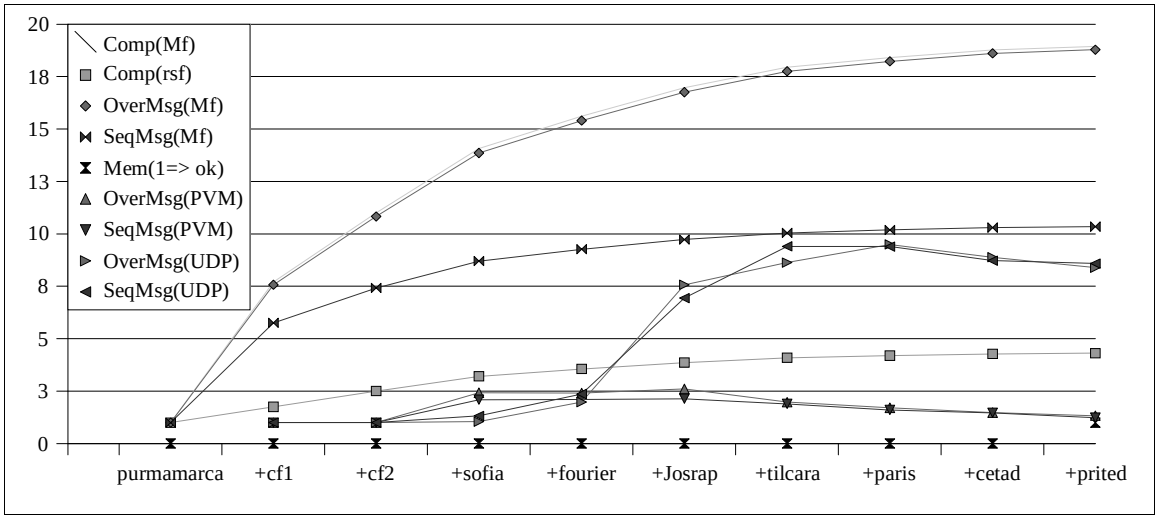


Figura 4.38: Speedup de los Algoritmos con UDP en la Red del CeTAD para n = 3200.

Como se puede notar en la Figura 4.38, en ningún caso se supera la estimación de SeqMsg(Mf) que es el rendimiento óptimo teniendo en cuenta solamente el aporte de cómputo de cada una de las computadoras sin tener en cuenta ningún tiempo relacionado con las comunicaciones.

El muy bajo rendimiento obtenido hasta la inclusión de **fourier** está directamente relacionado con los requerimientos de memoria del algoritmo en cada computadora. Específicamente, **cetadfomec1**, **cetadfomec2** y **fourier** tienen solamente 32 MB de memoria instalada y por lo tanto son las que tienen mayor probabilidad de que se utilice espacio de memoria swap durante los cálculos parciales de la matriz resultado.

Los valores que resumen la ejecución paralela que se muestran en la Tabla 4.23 aclaran que **fourier** es la computadora que requiere mucho tiempo más que las demás para resolver los cálculos locales.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	887	56.94	11.39	334.81
cf1	682	78.11	15.62	316.04
cf2	682	79.75	15.95	314.29
sofia	631	57.31	11.46	334.65
fourier	318	193.88	38.78	224.60

Tabla 4.23: Resumen de OverMsg(UDP) con Cinco Máquinas y n = 3200 en el CeTAD.

Los valores de la Tabla 4.23 también muestran que una gran cantidad de tiempo de ejecución en todas las máquinas está dedicado a la espera de la transferencia de los mensajes broadcast y en este caso también este tiempo puede estar afectado por la utilización del espacio de swap durante la ejecución.

De alguna manera, la estimación de memoria, Mem, en la Figura 4.38, está indicando que recién con todas las computadoras es probable que no haya problemas de memoria. La necesidad de recurrir a la memoria swap genera que

- Tanto los procesos de cómputo como los de comunicaciones pueden tener parte de su código ejecutable en memoria secundaria y por lo tanto pueden ser suspendidos en algún momento de su ejecución para recuperar el código.
- Tanto los procesos de cómputo como los de comunicaciones pueden tener parte de sus datos en memoria secundaria y por lo tanto pueden ser suspendidos en algún momento de su ejecución para recuperar esa información.
- Para todos los procesos el tiempo de swap genera un retraso en cuanto a tiempo de ejecución, pero en el caso de los procesos de comunicaciones la penalización puede ser mayor dado que pueden llegar a perder datos que se les están enviando durante el tiempo de swap.

Según la Figura 4.38, a partir de la inclusión de **Josrap**, el rendimiento es mucho mayor, aunque no tiene mayores variaciones con la inclusión de las demás computadoras. A partir de la inclusión de **Josrap** no se utiliza la memoria swap o la utilización no penaliza

significativamente la ejecución de los procesos de cómputo y de comunicaciones. La Tabla 4.24 muestra el resumen de la ejecución paralela con siete computadoras. Se tiene un gran avance en rendimiento de cálculo así como en el rendimiento de las comunicaciones.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
purmamarca	771	56.16	8.02	45.35
cf1	593	77.28	11.04	25.67
cf2	593	78.11	11.16	25.37
sofia	549	59.70	8.53	42.75
fourier	276	58.26	8.32	44.41
Josrap	242	52.93	7.56	47.07
tilcara	176	73.54	10.51	29.25

Tabla 4.24: Resumen de OverMsg(UDP) con Siete Máquinas y n = 3200 en el CeTAD.

En este caso particular, el tiempo total de cómputo está "dominado" por el tiempo de cómputo local de la computadora **cetadfomec2**, que sigue utilizando el espacio de memoria swap. Dado que los requerimientos de memoria sobre **fourier** se han disminuido respecto de la utilización de cinco computadoras, su tiempo de ejecución dedicado a cómputo ha disminuido notablemente respecto del que se muestra en la Tabla 4.23.

Por lo tanto, cuando deja de tener influencia la utilización de memoria swap en una o más máquinas en el tiempo total de cómputo paralelo se comienza a tener el rendimiento cercano al esperado del algoritmo que intercala períodos de cómputo con períodos de comunicación. En el caso específico de SeqMsg(UDP) esto es así hasta que el tiempo de comunicaciones se vuelve muy relevante o domina el tiempo total, lo cual está dando una idea de que la granularidad del problema no es adecuada para más allá de ocho máquinas. Aún así no se pierde demasiado en rendimiento. En el caso específico de OverMsg(UDP), una vez que se pueden aprovechar las máquinas a su capacidad máxima (o a un porcentaje relativamente alto de la capacidad máxima), se comienzan a utilizar las computadoras que no pueden solapar cómputo con comunicaciones y por lo tanto se tiene el mismo rendimiento que con SeqMsg(UDP).

Si se toma como referencia la estimación de memoria para establecer la cantidad de computadoras (en una elección *a priori*), Mem en los gráficos de speedup, se deberían utilizar todas las máquinas. El rendimiento obtenido para la multiplicación de matrices de 3200×3200 elementos utilizando las diez máquinas del CeTAD es aproximadamente nueve veces el rendimiento de **purmamarca** para el mismo tamaño de memoria. Este rendimiento se podría considerar "superlineal" dado que la suma de las potencias de cálculo relativas de todas las máquinas del CeTAD es menos de cinco veces la capacidad de cálculo de **purmamarca**. Una vez más, se debe recordar que el tiempo de referencia para este cálculo de speedup es el tiempo de ejecución "real" de multiplicación para matrices de orden n = 3200 que en **purmamarca** implica la utilización de memoria swap y la penalización es muy grande con respecto a la máxima capacidad de procesamiento de esta computadora (sin utilización de espacio de memoria swap).

4.8.1.3 Conclusiones Generales de la Experimentación en CeTAD

En un primer análisis, desde el punto de vista del rendimiento y de la escalabilidad *global* de los algoritmos, los resultados que se obtienen en la red local del CeTAD no parecen ser muy alentadores, ya que tanto SeqMsg(UDP) como OverMsg(UDP) a partir de una determinada cantidad de computadoras disminuyen en vez de aumentar (Figura 4.37 y Figura 4.38). Sin embargo, con la experimentación realizada se puede obtener información muy importante, además de la que se refiere estrictamente a rendimiento y escalabilidad global :

- El algoritmo OverMsg(UDP) se puede utilizar como *benchmark* para identificar cuáles computadoras son capaces de solapar efectivamente cómputo con transferencias de datos.
- En general, la estimación de speedup óptimo no es la que finalmente se obtiene por la sobrecarga (overhead) que imponen las placas de red, el sistema operativo, etc., pero conociendo las computadoras que son capaces de solapar cómputo con comunicaciones será posible definir la cantidad de computadoras a utilizar para un problema dado. Más específicamente: *cuáles*.
- La estimación de requerimientos de memoria, aunque no exacta, puede ser útil cuando es probable que algunas o todas las computadoras tengan problemas en cuanto a la utilización del espacio de memoria swap. Cuando se tiene esta información será posible elegir qué algoritmo utilizar (SeqMsg o OverMsg) dependiendo de si las computadoras a incluir en el cómputo paralelo pueden o no solapar efectivamente cómputo con comunicaciones.

4.8.2 Red Local del LQT

La mayoría de las cosas que se explicaron en el contexto de la experimentación en la red local del CeTAD son aplicables (quizás en otra escala) a la experimentación en la red local del LQT.

La Figura 4.39 muestra los valores de speedup obtenidos en la red local del LQT por los algoritmos implementados utilizando UDP, SeqMsg(UDP) y OverMsg(UDP), para matrices de orden $n = 5000$ junto con los que se mostraron anteriormente en la Figura 4.19. La estimación de speedup para el algoritmo de cómputo y comunicaciones secuenciales es casi exacta respecto de lo que se obtiene en la experimentación, lo cual es muy satisfactorio. Además, confirma que el problema del bajo rendimiento de SeqMsg(PVM), es decir del algoritmo implementado utilizando la biblioteca PVM, es justamente el broadcast de PVM que no aprovecha las características de las redes Ethernet.

Es evidente también que el $O(n^3)$ en cuanto a requerimiento de operaciones de punto flotante de este problema es igual de influyente en el tiempo total de ejecución paralela que el $O(n^2)$ de la cantidad de datos a transferir por medio de los mensajes broadcast. Es por esta razón que tanto la estimación de speedup óptimo SeqMsg(Mf) como el speedup obtenido en la experimentación SeqMsg(UDP), es aproximadamente el 50% del speedup

óptimo independientemente del algoritmo utilizado y de la red de comunicaciones Comp(Mf).

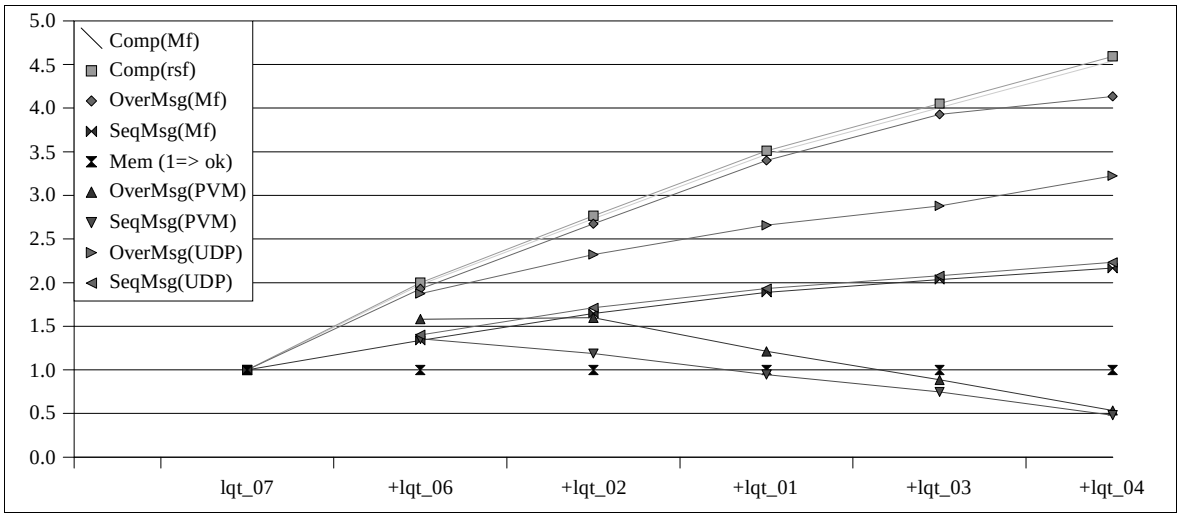


Figura 4.39: Speedup de los Algoritmos con UDP en la Red del LQT para n = 5000.

Aunque el algoritmo con cómputo y comunicaciones solapadas no llega al speedup óptimo calculado para este algoritmo, OverMsg(Mf), en la experimentación no solamente tiene mejor rendimiento que el algoritmo de cómputo y comunicaciones secuenciales sino que muestra al menos tres aspectos:

- La capacidad de cada una de las computadoras de solapar al menos en parte el cómputo local con las comunicaciones vía la red de interconexión. De hecho, tal como se puede visualizar en la Figura 4.39, el resultado final en términos de rendimiento tiene el efecto de solapar aproximadamente el 50% de las comunicaciones.
- La influencia de la sobrecarga que impone el sistema operativo entre otras capas de software no es despreciable, y de hecho es la que "consume" la diferencia entre OverMsg(Mf) y OverMsg(UDP).
- La granularidad de la aplicación hace que el speedup obtenido, OverMsg(UDP), se incremente a medida que se agregan mayor cantidad de computadoras. Expresado de otra manera, el tiempo de cómputo local es comparable con el tiempo de las comunicaciones y un buen porcentaje de las comunicaciones se puede realizar de manera solapada con el procesamiento.

Con respecto al solapamiento del cómputo con las comunicaciones, lo que sucede en realidad es lo mismo (o similar) que se explicó en la red del CeTAD. La Tabla 4.25 muestra el resumen de ejecución de SeqMsg(UDP) con todas las máquinas del LQT para resolver una multiplicación de matrices de orden n = 5000 en paralelo y la Tabla 4.26 muestra el resumen de ejecución de OverMsg(UDP).

De la comparación de los tiempos de ejecución que se muestran en cada tabla se llega a que:

- Los períodos de cálculo en la ejecución de OverMsg(UDP) son un poco mayores que los de SeqMsg(UDP) por la competencia por los recursos con los procesos que se encargan de las comunicaciones "en background".

- En la ejecución de OverMsg(UDP) un gran porcentaje de las comunicaciones se lleva a cabo *durante* el tiempo de cómputo local. Mientras que cada computadora debe esperar en promedio aproximadamente 87±3 segundos para la transferencia de datos durante la ejecución de SeqMsg(UDP), la espera es en promedio aproximadamente 25±4 segundos durante la ejecución de OverMsg(UDP).

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
lqt_07	1089	85.58	14.26	89.66
lqt_06	1089	86.36	14.39	88.94
lqt_02	835	87.94	14.66	87.41
lqt_01	811	90.54	15.09	85.17
lqt_03	589	90.15	15.02	85.20
lqt_04	587	89.94	14.99	85.36

Tabla 4.25: Resumen de SeqMsg(UDP) con Seis Máquinas y n = 5000 en el LQT.

Nombre	***Filas***	***Tot. Cómp.***	***Por It.***	***Tot. Msg.***
lqt_07	1089	91.40	15.23	28.69
lqt_06	1089	91.11	15.19	28.95
lqt_02	835	99.53	16.59	20.59
lqt_01	811	95.30	15.88	24.38
lqt_03	589	96.68	16.11	22.80
lqt_04	587	96.43	16.07	22.89

Tabla 4.26: Resumen de OverMsg(UDP) con Seis Máquinas y n = 5000 en el LQT.

La Figura 4.40 muestra los valores de speedup obtenidos en la red local del LQT por los algoritmos implementados utilizando UDP, SeqMsg(UDP) y OverMsg(UDP), para matrices de orden n = 9000 junto con los que se mostraron anteriormente en la Figura 4.20 (experimento que le *corresponde*).

Una vez más, la estimación del rendimiento óptimo del algoritmo de cómputo y comunicaciones secuenciales, SeqMsg(Mf), es casi exactamente la que se obtiene en la experimentación, que es SeqMsg(UDP). En el caso de la estimación del speedup óptimo para el algoritmo de cómputo y comunicaciones solapadas, OverMsg(Mf) es muy cercano al que se obtiene, OverMsg(UDP). La diferencia relativa entre los valores es bastante menor a la que se muestra en la Figura 4.39 y esto está dado por la influencia que tiene el $O(n^3)$ de requerimientos de cómputo sobre el $O(n^2)$ del requerimiento de comunicaciones. Y evidentemente esa diferencia hace que el porcentaje de tiempo de cómputo sea bastante superior al de las comunicaciones para matrices de 9000×9000 elementos que para matrices de 5000×50000 elementos.

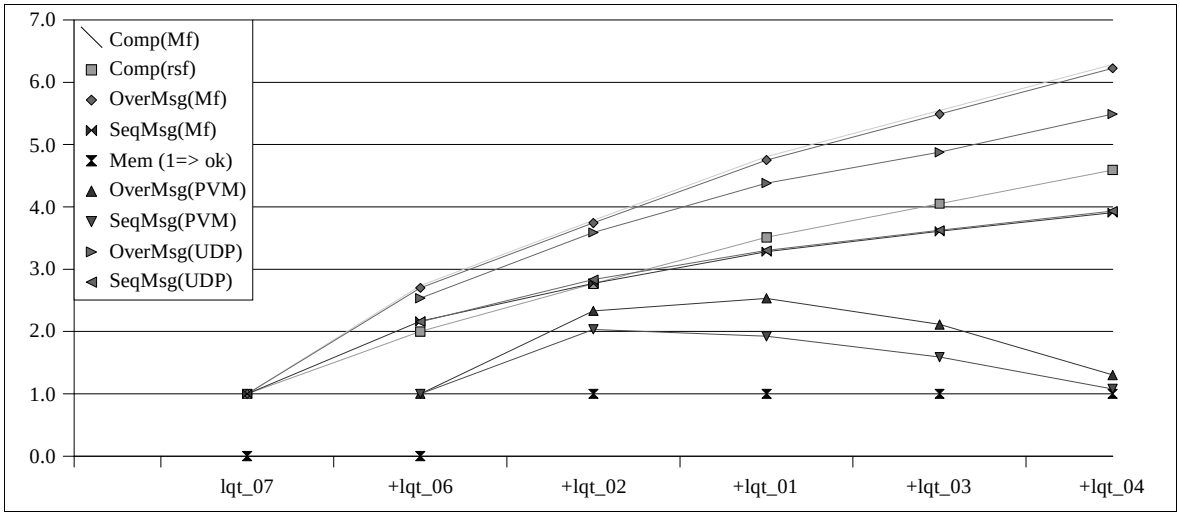


Figura 4.40: Speedup de los Algoritmos con UDP en la Red del LQT para n = 9000.

El rendimiento obtenido para la multiplicación de matrices de 9000×9000 elementos utilizando las seis máquinas del LQT es aproximadamente 5.5 veces el rendimiento de **lqt_07** para el mismo tamaño de memoria. Este rendimiento se podría considerar "superlineal" dado que la suma de las potencias de cálculo relativas de todas las máquinas del LQT es aproximadamente 4.5 veces la capacidad de cálculo de **lqt_07**. Una vez más, se debe recordar que el tiempo de referencia para este cálculo de speedup es el tiempo de ejecución "real" para matrices de orden n = 9000 que en **lqt_07** implica la utilización de memoria swap.

4.8.3 Red Local del LIDI

La Figura 4.41 muestra los valores de speedup obtenidos en la red local del LIDI por los algoritmos implementados utilizando UDP, SeqMsg(UDP) y OverMsg(UDP), para la multiplicación de matrices de orden n = 2000 junto con los que se mostraron anteriormente en la Figura 4.21.

Para comparar la estimación de los valores óptimos de speedup de los algoritmos, SeqMsg(Mf) y OverMsg(Mf) con los obtenidos, SeqMsg(UDP) y OverMsg(UDP) se debe recordar que los tiempos de comunicaciones se estiman en base a una red diez veces más rápida que las del CeTAD y del LQT. Esto a su vez genera que cualquier sobrecarga (overhead) sobre el tiempo de cómputo óptimo tendrá un impacto mayor en la red del LIDI que en las otras dos. Como conclusiones generales a partir de la Figura 4.41 se tienen:

- Tanto SeqMsg(UDP) como OverMsg(UDP) son menores a las estimaciones que les corresponden. El más alejado de la estimación es OverMsg(UDP).
- Los dos algoritmos tienen mejor rendimiento a medida que se aumenta la cantidad de computadoras. Es evidente que la mayor capacidad de la red de comunicaciones así como el rendimiento aceptable que se obtiene con los mensajes broadcast son factores importantes para que esto suceda.

- El algoritmo que está diseñado para solapar las comunicaciones con el cómputo local en cada una de las máquinas, OverMsg, es mejor que el de cómputo y comunicaciones secuenciales, SeqMsg. Esto a su vez indica que las computadoras son capaces de solapar efectivamente al menos en parte cómputo local con comunicaciones a través de la red de interconexión.

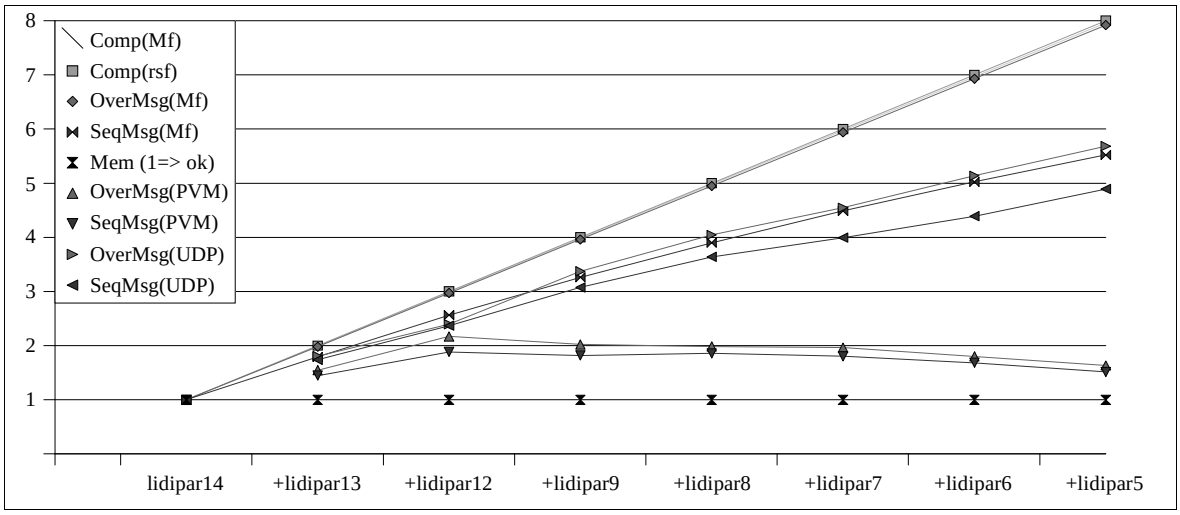


Figura 4.41: Speedup de los Algoritmos con UDP en la Red del LIDI para n = 2000.

La Figura 4.42 muestra los valores de speedup obtenidos en la red local del LIDI por los algoritmos implementados utilizando UDP, SeqMsg(UDP) y OverMsg(UDP), para la multiplicación de matrices de orden n = 3200 junto con los que se mostraron anteriormente en la Figura 4.22.

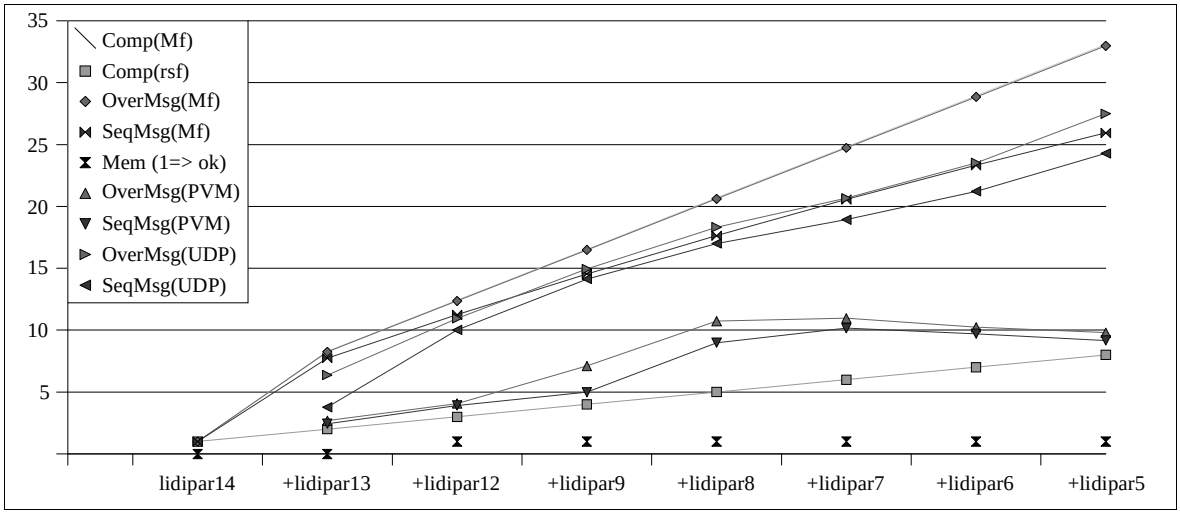


Figura 4.42: Speedup de los Algoritmos con UDP en la Red del LIDI para n = 3200.

Quizás éstos sean los mejores resultados en cuanto a rendimiento de todos los obtenidos. Tal como lo muestra la Figura 4.42, las ocho computadoras del LIDI pueden resolver una multiplicación de matrices de 3200×3200 elementos más de 25 veces más rápido que la

misma multiplicación en una de ellas (son todas iguales). De todas maneras, se debe recodar que el tiempo de referencia secuencial para la multiplicación de matrices de orden n = 3200 está penalizado con respecto al óptimo por la utilización de memoria swap durante los cálculos.

Una vez más, OverMsg(UDP) es mejor que SeqMsg(UDP) dado que las computadoras pueden efectivamente solapar comunicaciones y cómputo local y el algoritmo aprovecha también efectivamente esta capacidad. También se observa en la Figura 4.42 que siempre el rendimiento de los algoritmos mejora cuando se aumenta la cantidad de computadoras. Esto indica que, como era esperado, el algoritmo es escalable (al menos hasta ocho computadoras) y en particular la implementación de los mensajes broadcast entre procesos de usuario utilizando UDP también es escalable. A diferencia de lo que sucede con las matrices de orden n = 2000, para las matrices de orden n = 3200 los valores de speedup obtenidos son más cercanos a los estimados para los algoritmos, lo cual no hace nada más que confirmar la importancia del $O(n^3)$ de cómputo sobre el $O(n^2)$ de las comunicaciones que sobre la red local del LIDI se acentúa comparándola con las del CeTAD y del LQT.

4.9 Conclusiones-Resumen de la Experimentación

Una vez identificadas las características de las redes locales en cuanto a computadoras y redes de interconexión que se presentaron al principio del capítulo, se presentaron los resultados de la experimentación utilizando PVM como la biblioteca de pasaje de mensajes en general y de mensajes broadcast en particular. De esta experimentación surge que:

1. El rendimiento con PVM no es aceptable. En todas las redes locales, es decir independientemente de cantidad de máquinas, heterogeneidad u homogeneidad de las mismas, tamaños de matrices utilizados y rendimiento de la red de interconexión física los resultados fueron los mismos: el rendimiento empeora a medida que se utilizan mayor cantidad de computadoras.
2. Al hacer el perfil de ejecución y con un mínimo de instrumentación queda claro que el problema de rendimiento es siempre ocasionado por la rutina broadcast de la biblioteca PVM, que es implementada por múltiples mensajes punto a punto.

Se propone e implementa un mensaje broadcast basado directamente en el protocolo UDP, dado que en principio ninguna biblioteca de pasaje de mensajes de propósito general puede asegurar *a priori* rendimiento optimizado de los mensajes broadcast. Se repiten los mismos experimentos hechos anteriormente con PVM y se llega a que:

3. El algoritmo con períodos de cómputo y comunicaciones ejecutados de manera secuencial (SeqMsg) proporciona en casi todos los casos el rendimiento esperado. Las excepciones pueden darse en el caso de la utilización de memoria swap en algunas computadoras.
4. El algoritmo SeqMsg proporciona rendimiento que mejora a medida que se agregan computadoras en todas las redes, aún cuando el tamaño de las matrices no sea el mayor posible (o *escale* junto con la cantidad de computadoras).
5. El algoritmo organizado para solapar los períodos de cómputo y comunicaciones (OverMsg) tiene mejor rendimiento en todos los casos que SeqMsg, con lo cual el

rendimiento obtenido por este algoritmo puede considerarse aceptable en todas las redes locales utilizadas.

6. El Algoritmo OverMsg puede utilizarse de manera bastante sencilla como *benchamark* con dos propósitos:
   1. Identificar de manera sencilla la proporción de comunicaciones que puede ser llevada a cabo solapadamente (en *background*) mientras se realiza cómputo local en cada computadora.
   2. Las computadoras que no pueden ejecutar cómputo y comunicación de manera solapada y que por lo tanto penalizan el rendimiento paralelo de **toda** la red utilizada. En este sentido, OverMsg puede ser utilizado para descartar tales computadoras, o para dar un máximo de computadoras utilizables para una aplicación dada.
7. Tanto SeqMsg como OverMsg proporcionan speedup muy satisfactorio si la aplicación o, más específicamente, el tamaño de la aplicación hace que el rendimiento secuencial sea afectado por la memoria swap de la computadora en la cual se resuelve.
8. Tanto SeqMsg como OverMsg pueden utilizarse satisfactoriamente en clusters homogéneos como heterogéneos ya que en todos los casos obtienen rendimiento optimizado de
   1. Cómputo local de cada computadora y específicamente balanceado.
   2. Comunicaciones sobre la red de interconexión Ethernet con los mensajes broadcast.