D. **CASIANO RODRÍGUEZ LEÓN**, Doctor en Ciencias Matemáticas y Profesor Titular de Universidad del Área de Lenguajes y Sistemas Informáticos adscrito al Departamento de Estadística, I.O. y Computación de la Universidad de La Laguna,

Certifica

Que la presente memoria titulada "**Nuevos Modelos de Predicción en Computación Paralela**" ha sido realizada bajo su dirección por el Licenciado D. José Luis Roda García y constituye su Tesis para optar al grado de Doctor Ingeniero en Informática.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos que haya lugar, firma la presente en La Laguna, a veintinueve de mayo de mil novecientos noventa y ocho.

Fdo: Casiano Rodríguez León

Agradecimientos

Quiero agradecer al Doctor **Casiano Rodríguez León** por la gran ayuda en la realización de este trabajo. La gran cantidad de horas empleadas se ven "amortizadas" con la satisfacción de un trabajo donde hemos logrado nuestros objetivos.

Agradecer también a **Félix, Paco, Kiko, Coro, Dani, Patri y Jesús Alberto** por el apoyo que me han brindado en todo momento.

También a los **compañeros del Departamento,** al personal del Centro de Cálculo del Centro Superior de Informática y del Instituto de Astrofísica de Canarias.

Por supuesto a los **amigos**, a **mis familias** y a **Cris**, por la suerte de tenerlos.

ÍNDICE

Capítulo 2. MODELOS LogP, C3 y Patrones

Capítulo 3. MODELO BULK SYNCHRONOUS PARALLEL (BSP)

1.1 INTRODUCCIÓN

Un modelo de computación define el comportamiento de una máquina teórica. El objetivo de un modelo es permitir el diseño y análisis de algoritmos paralelos que se puedan ejecutar eficientemente en una amplia variedad de arquitecturas. Esta definición conlleva el uso de metodologías para diseñar los algoritmos y calcular el tiempo invertido en su ejecución. Estas metodologías imponen restricciones a los lenguajes de programación y a la implementación de los compiladores dependientes de las arquitecturas. Un obstáculo importante para la utilización masiva de las máquinas paralelas en aplicaciones de propósito general, ha sido la carencia de un único modelo de computación paralela ampliamente aceptado. En la actualidad, el gran reto está en desarrollar algún modelo que consiga englobar las diferentes propuestas existentes.

La computación secuencial ha resuelto este problema desde 1946 cuando von Neumann hizo su propuesta en [Bur46]. Los últimos 50 años tanto el hardware como el software se han basado en la idea de "máquina von Neumann". Una de las ventajas más importantes del modelo es la gran portabilidad de los programas de unas máquinas a otras con sólo pequeños cambios. En la computación en paralelo, esto no ocurre, siendo en muchos casos muy difícil o imposible transportar los programas de unas máquinas a otras, cada una con su arquitectura específica.

La computación en paralelo debe cubrir dos objetivos que entre sí son contradictorios. Por un lado debe permitir desarrollar software independiente de la arquitectura y tecnologías existentes. Por otro, los algoritmos deben obtener las máximas ventajas de la arquitectura subyacente. Un modelo de computación paralela debe reflejar una predicción lo más precisa posible del rendimiento, debe ser capaz de ser aplicado sobre máquinas actuales y futuras, y debe reflejar las restricciones impuestas por las máquinas determinadas. Este modelo debe cumplir el mismo papel que cumple el de computación secuencial al proporcionar un camino bien definido entre el software y el hardware.

La aparición de un modelo de computación paralela simple y preciso que englobe las necesidades actuales es fundamental para poder diseñar, analizar y transportar algoritmos paralelos. Entre los modelos de computación propuestos más importantes nos encontramos con el modelo **PRAM**, el modelo de **Redes**, el modelo **BSP**, el modelo **LogP** y el modelo $\mathbf{C}^3$.

1.2 MODELO PRAM

El modelo Parallel Random Access Machine (PRAM) propuesto por [For78] ha sido aceptado ampliamente para analizar la complejidad de los algoritmos paralelos. El modelo es simple y útil para representar un gran número de algoritmos paralelos pero no es realista debido a que asume que todos los procesadores de la máquina trabajan síncronamente y que acceden a cualquier celda de memoria en tiempo constante. En el modelo también se considera que el ancho de banda de las comunicaciones entre procesadores es infinito, no existe latencia ni sobrecarga en la transmisión de los mensajes y los procesadores sufren la contención cuando intentan acceder a la misma celda de la memoria compartida.

La simplicidad y generalidad del modelo *PRAM* lo han convertido en una herramienta de trabajo ampliamente aceptada, por lo que existe un gran número de algoritmos diseñados para el mismo. Sin embargo se suele cuestionar el realismo de este modelo no tanto en su calidad de modelo de programación como en su aspecto de modelo de análisis de rendimiento. Para que la memoria compartida pueda ser accedida en tiempo constante debe existir un camino físico desde el procesador hasta la posición de memoria. Se necesitarán puertas lógicas que decodifiquen la dirección. Si la memoria tiene M posiciones, el coste de la circuitería de decodificación se expresaría como $f(M)$, siendo f alguna función de coste. Si P procesadores comparten esa memoria, el coste sube a $P \times f(M)$ lo que para valores de M y P grandes será excesivo. La suposición de que los accesos a memoria compartida ocurren en tiempo constante conduce en ocasiones a la concepción de algoritmos cuya eficiencia puede ser pobre en las máquinas paralelas actuales, especialmente si la labor de adaptación del algoritmo *PRAM* es dejada totalmente en manos de un compilador.

1.3 MODELO DE REDES

El modelo de Redes [Lei92] considera que una máquina paralela esta constituida por nodos de procesamiento conectados según un grafo que permite la transmisión de datos entre procesadores vecinos. El modelo promueve la creación de algoritmos paralelos que se adapten a la topología de la máquina donde se va a realizar la ejecución. Si al modelo PRAM se le acusa de poco realista, a este modelo se le suele acusar de exceso de dependencia de la topología. El argumento utilizado es que la necesidad de adaptar el algoritmo a la topología implica una pérdida de portabilidad.

Matemáticamente, la red de interconexión se puede ver como un modelo de grafo $G = (V, A)$ dirigido o no: Los N procesadores están localizados en los vértices (nodos) del grafo y se comunican a través de los arcos (aristas). El grafo debe estar definido mediante una función recursiva (computable) en el número N de nodos.

El uso práctico de estas estructuras está limitado tanto por restricciones de cableado, como por principios de diseño y fabricación. Teniendo en cuenta estas limitaciones, se han sugerido una serie de criterios para evaluar estas organizaciones. Estos criterios ayudan a entender la efectividad de las redes en la codificación de algoritmos paralelos eficientes sobre el hardware real. Estos criterios son los siguientes:

- **Grado o número de aristas por nodo**: Interesa que el número de aristas por vértice sea constante e independiente del tamaño de la red, ya que así es posible construir redes con un número de procesadores arbitrariamente grande utilizando el mismo componente básico.
- **Diámetro**: El diámetro de una red es la distancia más larga entre dos nodos cualesquiera. El diámetro es una cota inferior en la complejidad de los algoritmos paralelos que requieren comunicaciones entre pares de vértices arbitrarios.
- **Ancho de Bisección**: Es el mínimo número b de aristas que deben ser eliminadas con el fin de dividir la red en dos mitades desconectadas del mismo tamaño (salvo una

unidad). Tales aristas nos miden el ancho del "cuello" de botella para cierto tipo de comunicaciones. Si todos los procesadores en una de las mitades del "cuello" intenta simultáneamente enviar un volumen de datos D a la otra mitad, todos los datos deberán atravesar el cuello de botella. Por tanto, una cota inferior del tiempo tardado será $\Omega(D/b)$.

- **Ancho de Entrada/Salida**: Si sólo se dispone de *io* puertos de entrada/salida a la red y es necesario introducir los D datos del problema en la red (o sacarlos), es obvio que se tardará $\Omega(D/io)$.

En el modelo de Redes, las comunicaciones sólo están permitidas entre los procesadores conectados directamente y cualquier otra comunicación se realizará utilizando procesadores intermedios. En un instante determinado, los nodos se pueden comunicar con los vecinos y operar con los datos almacenados localmente.

Muchos son los algoritmos diseñados bajo este modelo que muestran una conducta perfecta en la arquitectura para la que ha sido diseñada. La perfecta correspondencia entre el algoritmo y la estructura de la máquina para la que ha sido diseñado un algoritmo, lo hace poco flexible si queremos transportarlo a otra máquina. Por tanto, los algoritmos sufren la dependencia de la estructura subyacente en la máquina.

1.4 MODELOS ACTUALES Y PROPUESTOS

Otros modelos parten de los descritos anteriormente. La convergencia existente en las arquitecturas actuales han provocado la aparición de modelos que tienen en cuenta características de las nuevas máquinas. En la actualidad, la gran mayoría de las computadoras consisten en módulos de procesadores y memoria muy rápidos interconectados por una red de comunicaciones de altas prestaciones. La tendencia futura parece que mantendrá esta idea de computadora en general y existirán casos particulares de supercomputadoras específicas. Es más, algunos autores, McColl [Mco95b] y Culler [Cul93] apuestan por computadoras de altas prestaciones constituidas por procesadores de uso doméstico tipo Pentium II y una red de interconexión muy potente. Las redes de PCs conectados en red son una oportunidad de explotar nuevas aplicaciones paralelas.

Las nuevas arquitecturas han provocado la aparición de modelos más realistas y aplicables a nuevos problemas. Entre ellos podemos destacar principalmente el modelo BSP, el modelo LogP, el modelo C^3, además de los modelos de Patrones y el BSP Sin Barreras propuestos en esta memoria.

Lo más interesante de las diferentes contribuciones es que estamos en búsqueda del modelo que pueda abarcar un gran número de arquitecturas y aplicaciones. Lo que parece un consenso es que todo modelo debería tener en cuenta los siguientes retos: debe soportar no sólo las características de las multicomputadoras sino también los requerimientos de memoria y de comunicaciones de las agrupaciones de estaciones de trabajo y de ordenadores personales; debe permitir analizar la computación paralela asíncrona y debe contemplar las operaciones de entrada/salida además de las comunicaciones.

El modelo Bulk Synchronous Parallel (BSP) [Val90] fue uno de los primeros modelos que contabiliza las comunicaciones y caracteriza la arquitectura subyacente por un conjunto reducido de parámetros: latencia y ancho de banda. En el modelo BSP se propone que todo programa sea dividido en superpasos donde se realizan los cómputos con los datos existentes en la memoria local y se realizan las comunicaciones. Entre cada superpaso existe un sincronismo por barreras, donde tienen lugar los intercambios de información y a partir del cual cada procesador podrá utilizar los datos que le han sido enviados. El coste de un paso es la suma del mayor de los cómputos de los procesadores y el coste del procesador que más comunica. El modelo calcula el coste total de un programa como la suma de los costes de todos los superpasos.

En los últimos años han surgido entornos y librerías no estándar que siguen el modelo BSP [Hil97],[Ski96],[Mco95a],[Sim94]. En ellas se definen aproximadamente una veintena de funciones con las cuales se realiza cualquier programa BSP.

Un inconveniente importante de este modelo es que la naturaleza asíncrona de muchos programas hacen que sea imposible o muy ineficiente realizar programas en BSP. Otro inconveniente es que no se trata de una librería estándar de uso generalizado. El estándar actual de paso de mensajes es MPI [Pac97].

El modelo LogP propuesto por Culler [Cul93] tiene como objetivo principal caracterizar una máquina por cuatro parámetros: latencia, sobrecarga (*overhead*), inversa del ancho de banda y el número de procesadores. Comparado con el modelo BSP, el LogP proporciona un mecanismo de paso de mensajes más restrictivo y carece de sincronismos explícitos. La característica principal del modelo es que en las redes actuales se observa que existe un coste asociado al trabajo de cada procesador al colocar en la tarjeta de red los datos y mientras esto se produce, se pueden estar enviado datos a otro procesador. Por tanto, tiene en cuenta un efecto de solapamiento entre el cómputo y las comunicaciones. Dos son los inconvenientes más importantes del modelo. Por un lado, las plataformas software actuales no son adecuadas para el análisis y evaluación de algoritmos LogP. La excesiva carga en los protocolos de comunicaciones imponen un elevado coste a la hora de enviar y recibir datos, como se muestra en los trabajos [Cul96], [Ros96], [Kee95], [Mar96]. Hay que utilizar librerías de comunicaciones específicas, diseñadas con la idea de rebajar estos costes. Entre estas librerías se encuentran Active Messages [Eic92] y Fast Messages [Pak95]. Por otro, el análisis de algoritmos complejos no es una tarea fácil.

El modelo C^3 [Ham96] introduce la congestión de las comunicaciones en los links y en el procesador, además de la latencia y el ancho de banda. El modelo tiene en cuenta los tipos de encaminamiento y los modos de envíos y recepción de datos (con y sin bloqueos). En general es un modelo para el estudio de las comunicaciones con patrones usuales tipo uno a todos, todos a uno, etc.. Las complejidades son obtenidas de comunicaciones a muy bajo nivel y casi impracticable de utilizar para aplicaciones convencionales.

El modelo de Patrones es una solución que hemos propuesto en [Rod96], [Rod97] y [Rod98a] donde se obtienen las predicciones más fiables ya que se miden la latencia y ancho de banda para cada uno de los patrones básicos que encontramos en la mayoría de los algoritmos. El problema principal de este modelo es que no es general y ofrece soluciones particulares para cada uno de los patrones. Esto hace que no pueda ser considerado como un modelo de carácter general.

Por último, el modelo descrito ampliamente en el Capítulo 4 de esta memoria, el BSP Sin Barreras (BSP Without Barriers, BSPWB) es una variante más precisa del modelo BSP donde se introduce un nuevo concepto de paso, el M-paso. En todo algoritmo BSPWB deben cumplirse las siguientes condiciones: el número total de *M-pasos* ejecutados por cada uno de los procesadores es el mismo y las comunicaciones siempre tienen lugar entre procesadores que están en el mismo M-paso.

Una ventaja importante de este modelo es el uso de librerías estándar como PVM o MPI para el diseño y análisis de algoritmos paralelos. El BSPWB ha resultado un modelo válido en la computación paralela como se demuestra en los diferentes ejemplos que mostramos en esta memoria [Rod98c], [Rod98d] y [Rod98e]. Los experimentos se han realizado sobre aplicaciones reales, las plataformas software fueron PVM y MPI y plataformas hardware: una red de estaciones de trabajo conectadas por un conmutador de altas prestaciones que denominaremos **LAN UTP**, la misma red de máquinas pero conectadas por un medio compartido como un cable coaxial que denominaremos **LAN COA**, y las multicomputadoras **CRAY T3E**, **DIGITAL ALPHA SERVER 8400**, **IBM SP2** y **ORIGIN 2000**.

1.5 PLATAFORMAS SOFTWARE

1.5.1 LA MAQUINA VIRTUAL PARALELA (PVM)

La PVM es un programa que permite que máquinas heterogéneas, tanto paralelas como secuenciales, colaboren como un recurso computacional concurrente único. La idea principal es conectar diferentes computadoras, en principio con sistema operativo UNIX, para usarlas como un único computador paralelo. De esta forma, se podrán resolver problemas de elevado coste computacional.

La Máquina Virtual Paralela (PVM), comenzó a desarrollarse en 1989, en los laboratorios Oak Ridge National (ORNL). En el proyecto están involucrados: Vaidy Sunderam de la Universidad de Emory, Al Geist del ORNL; Robert Manchek de la Universidad de Tenesis, Adam Beguelin de la Universidad de Carnegie Mellon y del Centro de Supercomputación de Pitsburgo, Weicheng Jiang de la Universidad de Tenesis, Jack Dongarra de la Universidad de Tenesis y ORNL.

El modelo computacional de la PVM sigue la idea de "máquina virtual". Una máquina virtual es una colección de computadoras conectadas en red, que trabajan en un entorno de computación concurrente.Las características principales son:

a) Heterogeneidad, escalabilidad, múltiple representación de datos y tolerancia ante fallos.

b) Uso de máquinas simultáneo de diferentes arquitecturas con múltiples procesadores y diferentes tipos: escalares, vectoriales, etc.

c) Interfaz gráfica para optimizar, mejorar, depurar y analizar los programas bajo este entorno.

La PVM está compuesta por un conjunto de primitivas de usuario y por un programa que controla la máquina virtual. La computación concurrente se consigue conectando vía red una serie de elementos de proceso (EP) que se comunican utilizando el sistema de intercambio de mensajes. Estos elementos de computación o de proceso están conectados en una o más redes que pueden ser de distintas topologías y tecnologías (Ethernet, fibra óptica, par trenzado, coaxial, etc.).

Una tarea es una unidad de computación equivalente al proceso en Unix. Estas tareas se podrán escribir en C o en Fortran y la única diferencia será la incorporación de las rutinas para realizar el paralelismo (creación de una nueva tarea, paso de mensajes, etc.). Denominaremos "aplicación pvm", al programa realizado por el usuario que utiliza la forma de trabajo de la máquina virtual.

La plataforma PVM se compone de dos partes:

a) Un demonio: programa llamado "*pvmd*" que se debe ejecutar en aquellas computadoras que constituyan la máquina virtual. Está diseñado para que cualquier persona pueda instalarlo en su máquina. Cuando un usuario quiere ejecutar una aplicación PVM ejecuta primero el demonio en una de las máquinas, y éste se encarga de activar el resto de demonios de las máquinas que forman la PVM. A partir de aquí, el programa de aplicación podrá iniciar su ejecución desde cualquiera de las máquinas. Una cuestión importante es que se pueden estar ejecutando varias máquinas virtuales de forma solapada donde cada usuario trabaja con su propia máquina virtual.

b) Librería de rutinas: Contiene las funciones que se podrán realizar una vez esté activa la PVM. Entre otras, contiene rutinas para :

- Control de procesos
- Crear, enviar y recibir mensajes
- Información del sistema
- Configuraciones dinámicas (añadir y eliminar máquinas)
- Señalización
- Mensajes de error
- Empaquetar datos en mensajes
- Gestionar grupos de procesos

Cuando un usuario activa un demonio "*pvmd*" en un EP, puede especificar un fichero de entrada que contiene una lista de todos los EP que van a formar parte de la máquina virtual. Este demonio se encarga de activar cada uno de los demonios de los EP que forman la máquina virtual. A partir de estos momentos, se establecen los puertos de comunicaciones entre cada uno de estos demonios. Toda actividad de control entre los demonios, el control del tráfico de mensajes entre ellos y la llegada correcta de los paquetes, se realiza en estos instantes.

Los puertos de comunicaciones (denominados en inglés *sockets*) se establecen por medio de circuitos virtuales (TCP) o datagramas (UDP) entre cada demonio existente en la máquina virtual y sus respectivas tareas locales, o también entre tareas en el mismo EP o en diferentes EP. Estos *sockets* realizan la salida estándar y los mensajes de error estándar, que se utilizan principalmente para la depuración de los programas.

Los programas de aplicaciones están compuestos de "componentes" que son las tareas. Cuando se ejecuta una de estas componentes, múltiples instancias de cada componente podrían iniciarse. Para formar parte de la máquina virtual, un proceso PVM debe realizar una llamada a una rutina de la librería, la cual se encarga de establecer un *socket* TCP entre estas componentes y el demonio local. El demonio local entonces se encarga de informar al resto de las computadoras que conforman la máquina virtual, para que actualicen sus tablas de localización de componentes.

Una componente sólo se comunica con su demonio local. Toda petición para iniciar otro proceso en otra máquina, o peticiones de envió de mensajes a otras máquinas son coordinadas por los demonios. Como cada demonio contiene las tablas de componentes, en todo momento sabrá a quien le tiene que mandar los mensajes. Cuando un demonio recibe un mensaje de otro demonio, pasa el mensaje a la componente adecuada en la máquina local.

El diseño global de la PVM con el uso de demonios conectados por *sockets* UDP, fue elegido por tres razones:

a) La red no se bloqueará.

b) Los *sockets* UDP implican una menor sobrecarga para configurar y mantener que los sockets TCP.

c) Las componentes no necesitan conocer la localización de ninguna otra componente ni tienen que interrumpir sus tareas para manejar los datos que llegan.

Los programas de aplicaciones ven el sistema PVM como un recurso de computación paralelo general y flexible.

1.5.1.1 Características principales en PVM

a) Interfaz de usuario

La interfaz con el usuario es una consola donde se pide información del estado de la máquina virtual. Nos indicará que procesos están activos, que computadoras forman la máquina, etc.

b) Identificación de las tareas

Todo proceso dentro de la PVM tiene asignado un número de identificación de su tarea. Se ha denominado "*tid*", identificador de tarea. Cada *tid* es único dentro de toda la máquina virtual, y está controlado directamente por los demonios, nunca por el usuario. Este sólo puede leer el *tid* de una tarea en cuestión. En la máquina virtual PVM 3.3, existen varias rutinas de la librería que manejan los *tids* (*pvm_mytid()*, *pvm_spawn()*, *pvm_parent()*, *pvm_bufinfo()*).

c) Control de los procesos

El software PVM nos proporciona rutinas que permiten al proceso del usuario convertirse en una aplicación *pvm* y luego volver a ser un proceso del usuario al terminar su labor. Existen rutinas para añadir y eliminar máquinas de la configuración de la máquina virtual. También existen rutinas para enviar señales a otras tareas y rutinas para encontrar información acerca de la máquina virtual.

d) Tolerancia ante fallos

En cualquier momento puede que una máquina integrada en la PVM deje de estar activa, con lo cual la PVM deberá actualizar sus tablas. El estado de un EP puede ser requerido por cualquier aplicación. En cualquier momento puede ser necesario añadir un nuevo EP a la máquina virtual. El programador de aplicaciones tiene toda la responsabilidad para gestionar esta característica. La PVM no realiza en ningún momento, intentos de recuperación automática de tareas que fueron eliminadas debido a un fallo del EP.

e) Grupo de procesos dinámicos

Un proceso puede pertenecer a varios grupos de procesos. Estos grupos se manejan a través de funciones específicas que incorporan un proceso a un grupo o lo sacan de él. Existen muchas situaciones donde todos los procesos deben conjuntar algunas operaciones, etc.

f) Señalización

La PVM proporciona dos métodos para mandar señales entre tareas. Un método sería enviar una señal Unix a otra tarea. El segundo método consiste en que una tarea notifica a un grupo de tareas acerca de un evento enviando un mensaje con una etiqueta específica, definida por el usuario y que la otra tarea puede interpretar.

g) Comunicaciones

La PVM aporta rutinas para empaquetar, desempaquetar, enviar y recibir mensajes entre tareas. Existen envíos asíncronos a una tarea o a un grupo de ellas. Los *buffers* para los mensajes se habilitan de forma dinámica. El tamaño máximo de un mensaje que podrá ser enviado o recibido está únicamente limitado por la cantidad de memoria disponible en la máquina.

h) Integración de sistemas multiprocesadores

La PVM fue desarrollada en un principio para unir máquinas conectadas a una red. Se han incorporado herramientas para que los sistemas multiprocesadores puedan convivir en este entorno. Los mensajes entre dos procesadores de la misma máquina van directos entre ellos, en cambio los mensajes a otras máquinas de la red o de otras redes van a través de sus demonios. Estos se encargarán de encaminarlos al lugar apropiado ya que conocen, por sus tablas, la configuración de la máquina virtual.

1.5.1.2 Primitivas PVM

1.5.1.2 Primitivas PVM

a) Control de procesos

pvm_mytid()

Esta función incorpora a la PVM al proceso que la llama, proporcionándole un número de tarea único en toda la máquina virtual, denominado *tid*. Esta función debe ser la primera rutina que se invoque antes de llamar a cualquier otra.

pvm_exit()

Comunica al demonio local que el proceso está abandonando la PVM. Esta primitiva no mata el proceso en sí, sino que continua pero como otro proceso cualquiera de UNIX.

pvm_spawn()

Inicializa un conjunto de tareas que se incorporarán en la PVM. Para determinar en que EP se van a crear estas nuevas tareas, la heurística utilizada podría estar en función de las medidas de carga de los EP de la PVM y el grado de computación de los mismos. La función devuelve el número de tareas creadas en una o más maquinas de la configuración.

Si las tareas han sido inicializadas correctamente, como parámetro de salida, devuelve un vector con los *tids* y en caso que alguna de estas no hayan podido ser inicializadas, devuelve en su correspondiente componente del vector, un código de error, indicando porque no ha podido iniciarse dicha tarea.

pvm_kill()

Elimina la tarea identificada por el parámetro *tid* de la PVM.

b) Información de la máquina virtual

pvm_parent()

Devuelve el *tid* del proceso que creó la tarea que llama esta función. Si existe algún error devuelve un error tipo *PvmNoParent*.

pvm_pstat()

Devuelve el estado de una tarea PVM identificada por *tid*. Devuelve *PvmOk* si la tarea está en ejecución, *PvmNoTask* si no lo está, y *PvmBadParam* si el *tid* es incorrecto.

pvm_mstat()

Devuelve *PvmOk* si el EP está ejecutando tareas PVM, *PvmHostFail* si no podemos acceder al EP, o *PvmNoHost* si ese EP no está en la PVM. Esta función nos podrá ayudar a crear aplicaciones tolerantes a fallos. En todo momento podemos preguntar por el estado de cualquiera de las máquinas involucradas en la máquina virtual.

pvm_config()

Devuelve información acerca de la PVM, incluyendo el número de EP, tipo de arquitecturas, etc.

pvm_tasks()

Devuelve información de las tareas en ejecución en la PVM. El control de las tareas es muy importante, ya que podemos controlar las aplicaciones.

c) Configuraciones dinámicas

pvm_addhost (), pvm_delhost ()

Estas primitivas nos permiten añadir o eliminar un EP de la configuración inicial de la PVM.

d) Primitivas de señalización

pvm_sendsig()

Manda la señal *signum* a otra tarea PVM identificada por un *tid*.

pvm_notify()

Provoca el envío de un mensaje a una serie de tareas especificadas, al ocurrir un evento en la PVM. Los posibles eventos que pueden ocurrir son: una tarea ha finalizado, una máquina ha caído o ha sido eliminada, o si se ha añadido una nueva máquina.

e) Primitivas de mensajes de error

pvm_error()

Imprime el estado de error de la última primitiva PVM.

pvm_serror()

Seleccionar esta rutina permite enviar mensajes de error de forma automática, y de esta forma cualquier error que se produzca, automáticamente escribirá el mensaje de error asociado.

f) Envío y recepción de mensajes

El envío de mensajes conlleva los pasos siguientes:

a) Se debe inicializar un *buffer* de envío con una primitiva particular.

b) Los datos que van a ser enviados deben ser empaquetados (primitivas de empaquetado).

c) El mensaje se envía a otro proceso (primitivas de emisión).

Un mensaje podrá ser recibido a través de una primitiva de recepción con o sin bloqueo. Después debemos desempaquetar cada ítem. En la PVM, sólo existe un *buffer* de

envío activo y uno de recepción activo, por proceso en un instante dado. El usuario es responsable de gestionar que tipo de mensaje está activo en cada momento, pudiendo crear los que sean necesarios.

pvm_mkbuf()

Crea un *buffer* de envío vacío y especifica un código para designarlo. Existen varias opciones para la codificación de los datos en este mensaje, dependiendo del valor de la constante *encoding*:

a) *PvmDataDefault*. Codificación estándar XDR.

b) *PvmDataRaw*. No realizar codificación.

c) *PvmDataInPlace*. Los datos se quedan en el mismo EP.

pvm_initsend()

Limpia el *buffer* de envío y lo prepara para empaquetar nuevos datos. Esta primitiva se debe ejecutar antes de empaquetar los datos. Los valores de *encoding* son los mismos que en la función anterior.

pvm_freebuf()

Elimina el *buffer* identificado por el parámetro *bufid*. Se debe ejecutar esta primitiva siempre que se deje de utilizar este *buffer*.

pvm_getsbuf()

Devuelve el número del *buffer* de envío activo en ese momento.

pvm_getrbuf()

Devuelve el número del *buffer* de recepción activo.

pvm_setsbuf()

Selecciona el nuevo *buffer* de envío activo y devuelve el identificador del anterior.

pvm_setrbuf()

Selecciona el nuevo *buffer* de recepción activo, retornando el identificador del anterior. El empaquetado de los datos se realiza con diferentes primitivas que empaquetan un conjunto de elementos del mismo tipo en el *buffer* de envío activo. Estas primitivas pueden ser llamadas múltiples veces y en cualquier orden. De esta forma, un mensaje puede contener varios vectores de datos de distintos tipos. No existen restricciones en cuanto complejas pueden ser estos mensajes. La única condición que existe es que se deben desempaquetar en el mismo orden en que fueron empaquetados. Tenemos primitivas para empaquetar bytes, enteros, reales, complejos, caracteres, etc.

El envío y recepción de datos se lleva a cabo con las siguientes primitivas.

pvm_send()

Esta primitiva etiqueta un mensaje con un valor entero, *msgtag*, que indica un número de mensaje, y se lo envía a una tarea especificada por un *tid*.

pvm_mcast()

Etiqueta el mensaje con un identificador entero, *msgtag*, y lo manda a las tareas especificadas por *tids*.

pvm_nrecv()

Recepción sin bloqueo. Si el mensaje pedido no ha sido recibido, esta primitiva devuelve un cero. Esta rutina puede ser llamada múltiples veces para pedir el mensaje y comprobar si ha llegado mientras se puede seguir realizando otro trabajo entre dichas llamadas.

Si llega un mensaje con etiqueta específica desde una tarea *tid* concreta, entonces se colocará el mensaje en el *buffer* activo.

pvm_recv()

Primitiva de recepción con bloqueo. Espera hasta que un mensaje haya llegado con la etiqueta especificada y/o con *tid* determinado.

pvm_bufinfo()

Devuelve información acerca del mensaje con el identificador especificado.

pvm_recvf()

Uso de definiciones propias de la primitiva de recepción de mensajes.

g) Primitivas de grupos de procesos

Estas funciones nos permiten controlar un conjunto de procesos que por cuestiones del problema a resolver, interesa que estén agrupadas. Cuestiones como la sincronización de los procesos, se puede llevar a cabo permitiendo que todos los procesos pertenezcan a un grupo. Dentro de un grupo, se pueden realizar diferentes funciones que aquí comentamos. Cualquier tarea PVM puede incorporarse o abandonar un grupo de procesos en cualquier momento sin tener que informar a los demás del grupo.

pvm_joingroup ()

Permite que una tarea se incorpore al grupo. Crea un grupo con el nombre especificado y pone una tarea en ese grupo. Cada tarea, al incorporarse a un grupo, se le asigna un número, a través del cual se gestionan los grupos.

pvm_lvgroup()

Una tarea abandona un grupo llamando a esta primitiva. En caso de volver a incorporarse, el número será seguramente distinto al que tenía antes. Los números se asignan dinámicamente.

pvm_gettid()

Devuelve el *tid* del proceso que está en un grupo con un número determinado.

pvm_getinst()

Devuelve el número que tiene una tarea dentro de un grupo.

pvm_gsize()

Devuelve el número de miembros que existen en un grupo.

pvm_barrier()

Cuando se llama a esta función, el proceso se bloquea en espera que todos las tareas implicada ejecuten esta función.

pvm_cast()

Etiqueta un mensaje con un identificador entero, y manda el mensaje a todas las tareas de un grupo determinado. Si una tarea se incorpora a un grupo durante la llegada de este tipo de mensajes, podría no recibir el mensaje.

PVM es una herramienta que tuvo un gran auge entre los años 1993 y 1997. A partir de la aparición del "estándar" de paso de mensajes MPI, los desarrolladores de PVM se integraron en la nueva plataforma constituyendo la librería de paso de mensajes más importante que existe hoy en día.

1.5.2 MESSAGE PASSING INTERFACE (MPI)

MPI (Message Passing Interface) es un sistema estandarizado y portable de paso de mensajes desarrollado por un grupo de investigadores de la academia y de la industria con soporte en una amplia variedad de computadores paralelos. El estándar define la sintaxis y la semántica de un núcleo de rutinas de librería que resultan de gran utilidad a un amplio conjunto de usuarios que desarrollan programas de paso de mensajes en C o en Fortran.

El esfuerzo de estandarización de MPI implicó a más de 80 personas de 40 organizaciones, principalmente de Estados Unidos y de Europa. La mayoría de los vendedores de computadoras paralelas del momento estuvieron implicados en MPI, junto a investigadores de universidades, de laboratorios gubernamentales y de la industria. El proceso de estandarización comienza con la reunión de trabajo "Standards for Message Passing in a Distributed Memory Environment", celebrada en Abril de 1992 en Virginia.

En Noviembre de 1992, una reunión del grupo de trabajo de MPI propone imprimir un carácter más formal al proceso de estandarización y decide realizar reuniones periódicas con una cierta regularidad a lo largo del año 1993. El borrador de MPI se presentó en Noviembre de 1993 en la conferencia Supercomputing´93. Después de un periodo de comentarios públicos se produjeron algunos cambios en la versión original de MPI, la versión 1.0 apareció en Junio de 1994. Este conjunto de encuentros y la discusión de la

nueva plataforma, constituyó lo que se conoce como el Forum MPI, abierto a todos los miembros de la comunidad de la computación de altas prestaciones.

MPI constituye una librería de paso de mensajes, esto es, una colección de rutinas que facilita la comunicación entre los procesadores de un programa paralelo de memoria distribuida. Se presenta como la primera librería estándar y portable que ofrece buenos rendimientos. No se trata de un estándar real puesto que no fue diseñada por una organización como ANSI o ISO. Por el contrario, se trata de un estándar por consenso diseñada en el Forum MPI.

Algunas de las cualidades más importantes que ofrece MPI son las siguientes: comunicación asíncrona, gestión eficiente de los *buffers* de mensajes, gestión eficiente de grupos, un amplio conjunto de operaciones de comunicación colectiva, topologías virtuales, creación de tipos derivados de datos y varios modos de comunicación. Estas características confieren una gran riqueza y funcionalidad al numeroso conjunto de funciones (alrededor de 125) que constituyen la librería MPI.

Las primeras implementaciones de MPI comenzaron a desarrollarse en paralelo con la definición del estándar y sirvieron al grupo MPI como fuente de experiencias en plataformas reales. Entre las implementaciones más importantes cabe destacar las de MPICH [Gro94], LAM [Bur94] y CHIMP [Ala94]. De todas ellas MPICH goza de gran popularidad y corresponde con la versión instalada en las plataformas hardware que hemos utilizado.

1.5.2.1 La plataforma MPICH

MPICH es una implementación gratuita de MPI que contiene todas las características del estándar, donde se mantiene la portabilidad del código MPI entre las diferentes máquinas paralelas y se intenta mantener al máximo su rendimiento.

MPICH pudo implementarse rápidamente debido a la existencia de librerías bien definidas en diferentes sistemas y a la experiencia de los autores en librerías similares. Entre ellos podemos citar P4 [But92], Chameleon [Gro93] y Zipcode [Skj94]. En P4 se incluyen funciones básicas para el paso de mensajes y para memoria compartida. En un principio, MPICH estaba implementado utilizando P4 en los casos de redes TCP/IP y memoria compartida. Chameleon consiste en una librería de paso de mensajes de alto rendimiento que se implementa como macros de C y que se utiliza en muchas librerías de fabricantes de máquinas: NX de Intel y MPL de IBM, por ejemplo. Por último, Zipcode es un sistema para desarrollar librerías escalables. En MPICH se introdujeron ideas de Zipcode, como los contextos, los grupos y los comunicadores. Zipcode también contiene muchas rutinas colectivas que trabajan con topologías virtuales, conceptos que también heredó MPICH.

1.5.2.2 Arquitectura de MPICH

1.5.2.2 Arquitectura de MPICH

En el diseño de MPICH se tuvieron en cuenta las dos características más importantes: portabilidad y eficiencia. En el desarrollo de MPICH se intentó por un lado maximizar la cantidad de código general que sirviera a una gran cantidad de plataformas sin reducir el rendimiento del sistema. Por otro, se ofrece una estructura que permita que MPICH sea transportado a otra plataforma con el menor número de cambios (sólo los dependientes de la plataforma). Las características de los grupos, atributos, comunicadores son compatibles en las implementaciones de los sistemas soportados.

El mecanismo que permite la portabilidad y eficiencia es una especificación denominada ADI (Abstract Device Interface) en MPICH. Todas las funciones MPI están implementadas en términos de funciones y macros definidas en la ADI. Todas las funciones soportadas en este nivel son portables. La especificación ADI debe proporcionar cuatro tipos de funciones: especificar que se va a enviar o recibir un mensaje, transferir datos entre la interfaz de programación del usuario y el hardware de paso de mensajes, gestión de mensajes en cola e información del entorno de ejecución. De esta forma, la ADI consta de funciones en término de las cuales las funciones MPI quedan expresadas. En el caso de MPICH, la ADI contiene además el código de empaquetado de mensajes, gestión múltiple de *buffers*, detectar que los mensajes entrantes pueden leerse o deben encolarse y gestionar las comunicaciones heterogéneas.

En MPICH existen diferentes implementaciones de la ADI. Una de ellas es el "Channel Interface" (C.I.) que está compuesto por cinco funciones como mínimo y proporciona la forma más rápida de obtener una versión MPICH bajo otra plataforma. A esta implementación se le pueden incorporar nuevas funciones específicas de las arquitecturas para obtener mayor rendimiento.

Podemos describir los niveles existentes en la implementación de MPICH de la siguiente manera. En el nivel superior están las funciones MPI. Si estas son colectivas suelen estar implementadas en un nivel inferior con funciones MPI punto a punto (MPI_Send, MPI_Recv, etc.). Debajo de este nivel está la especificación ADI. Continúan las diferentes implementaciones que dependen de la plataforma hardware. El caso más portable es el Channel Interface y existen otras implementaciones, específicas para cada una de las plataformas: Meiko, T3D, SGI.

En el nivel más bajo, el C.I., lo que necesitamos es enviar datos desde el espacio de direcciones de un proceso origen a otra en el proceso destino. Esto se puede realizar con muy pocas funciones (tan sólo cinco funciones en algunos casos). La implementación de CI se realiza con Chameleon, memoria compartida o con versiones especiales propias de las plataformas.

En el C.I. existen tres mecanismos para implementar el intercambio de datos:

1) "Eager": en este protocolo los datos son enviados al destino de forma inmediata. Si el destino no está en espera de los datos, el receptor debe proporcional espacio para almacenar los datos de forma local. Esta opción ofrece el mayor rendimiento, sobretodo cuando los niveles inferiores proporcionan las funciones de control y almacenamientos. Tiene la desventaja que con mensajes grandes puede dar problemas al no disponer de memoria suficiente el receptor. Es la opción por defecto en MPICH.
2) "Rendezvous". Los datos son enviados al destino sólo cuando este los solicita.

Cuando se ejecuta el "receive", el receptor envía una petición al origen solicitando los datos. Este protocolo es el más robusto pero puede ser el más ineficiente dependiendo de los niveles inferiores. Para utilizar este protocolo MPICH debe configurarse con el protocolo –use_rndv.

3) "Get": los datos son leídos por el receptor. En este caso es necesario algún método para copiar los datos de la memoria de un procesador a otro. Este mecanismo ofrece altos rendimientos pero requiere de hardware especial como memoria compartida u operaciones sobre memoria remota.

1.5.2.3 Primitivas MPI

En MPI existen aproximadamente 125 funciones divididas en diferentes grupos según sus características. A continuación detallaremos aquellas funciones que hemos considerados más importantes, entre las que destacamos las utilizadas en este trabajo.

a) Envíos y recepciones punto a punto

MPI_Get_count()

Devuelve el número de elementos recibidos por una operación. Esta función inicializa la variable denominada *status*, que nos permite comprobar lo que ha ocurrido con la ejecución de la función.

MPI_Send()

Envía datos con bloqueo a un procesador específico.

MPI_Recv()

Espera a recibir datos de un origen. Existe un parámetro de salida que nos permite diferenciar entre los tipos de mensajes que hemos recibido.

MPI_Sendrecv()

Envía el contenido de un *buffer* especificado a un receptor y recibe otro mensaje de otro emisor.

b) Modos de comunicaciones en punto a punto

MPI_Bsend()

Función de envío de datos pero utilizando *buffers* predefinidos por el usuario.

MPI_Rsend()

Función de envío de datos donde el receptor esta esperando por los datos.

MPI_Ssend()

Envío síncrono. Esta función no continuará hasta que exista el correspondiente "receive" y la llegada de los datos al destino haya comenzado.

c) Asignación de espacio (*buffer*)

MPI_Buffer_attach()

Permite informar al sistema que se debe utilizar almacenamientos para los mensajes que se van a enviar. Permite evitar bloqueos para mensajes grandes.

MPI_Buffer_detach()

Libera la memoria anteriormente asignada.

d) Comunicaciones sin bloqueos

MPI_Ibsend()

Realiza un envío sin bloqueos con asignación previa de "*buffers*".

MPI_Irecv()

Comienza una recepción sin bloqueos con asignación previa de "*buffers*".

MPI_Irsend()

Realiza un envío sin bloqueos en el estado "ready".

MPI_Isend()

Realiza un envío sin bloqueos en el estado "estándar".

MPI_Issend()

Comienza una recepción sin bloqueos en el estado "síncrono".

MPI_Test()

Comprueba si la operación no bloqueante asociada al "handler" devuelto por la función que realizó el envío ha finalizado.

MPI_Wait()

Queda pendiente de que la operación que inicializó el "handler" haya terminado.

e) Comunicaciones colectivas

Las comunicaciones colectivas hacen uso de los comunicadores. Es importante el concepto de comunicador. En la ejecución de un programa un conjunto de procesos quedan ligados en un mismo entorno. Cuando alguien quiere enviar, a través de una función colectiva, no hace referencia al nombre de cada proceso, más bien al comunicador al que todos los procesos pertenecen.

MPI_Barrier()

Bloquea la ejecución del proceso donde se ejecuta, hasta que todos los procesos asociados al mismo "comunicador" hayan ejecutado la misma función.

MPI_Bcast()

Envía el contenido del *buffer* de datos de salida a todos los procesadores con el mismo comunicador. El parámetro *root* indica quien es el emisor de los datos.

MPI_Scatter() y MPI_Scatterv()

Envía diferentes datos desde el emisor a cada uno de los receptores con el mismo comunicador. La diferencia entre ambas funciones reside en que la primera envía la misma cantidad de datos a todos los procesos y la segunda esta cantidad se puede personalizar.

MPI_Gather() y MPI_Gatherv()

El receptor recibe de forma no personalizada o personalizada los datos de cada uno de los procesos que forman parte del comunicador.

MPI_AlltoAll() y MPI_AlltoAllv()

En este caso, los datos no personalizados y personalizados son enviados desde todos los procesos a todos los procesos del comunicador.

MPI_Allgather() y MPI_Allgatherv()

Equivalente a MPI_Gather() y MPI_Gatherv(), pero con la características que todos reciben de todos.

MPI_Reduce()

Combina los contenidos de los operandos de cada uno de los procesadores utilizando una función predeterminada.

MPI_Allreduce()

Todos los procesadores del grupo realizan la operación anterior.

MPI_Reduce_scatter()

Combina los contenidos de los operando de los procesos y devuelve con una operación de uno a todos personalizada, los valores que a cada uno le corresponden.

MPI_Scan()

Ejecuta una operación de prefijos en paralelo.

f) Grupos, Contextos y comunicadores

MPI_Comm_group()

Devuelve el grupo al que pertenece el comunicador.

MPI_Comm_create()

Crea un nuevo comunicador con el conjunto de procesos que se le pasa a la función.

MPI_Comm_rank()

Devuelve el número de proceso dentro del comunicador. Es el nombre lógico del procesador.

MPI_Comm_size()

Devuelve el número de procesos involucrados con el mismo grupo.

g) Gestión de información del sistema y de errores

MPI_Get_processor_name()

Devuelve el nombre del proceso donde se ha ejecutado la función.

MPI_Wtick()

Devuelve la precisión de la función MPI_Wtime.

MPI_Wtime()

Devuelve un número de doble precisión indicando el número de segundos que han pasado desde un punto determinado.

MPI_Abort()

Aborta todos los procesos del comunicador.

MPI_Finalize()

Finaliza la ejecución del programa MPI.

MPI_Init()

Inicializa MPI para comenzar la ejecución de un programa.

1.6 PLATAFORMAS HARDWARE

1.6.1 Redes de área local

En una red de área local pueden existir máquinas diferentes interconectadas cooperando para resolver un problema paralelo. Se puede considerar que se comportan como un único ordenador que dispone de varios nodos interconectados mediante la red. Los componentes básicos son los ordenadores conectados a la red y las características de los dispositivos de la red. Existen muchas variantes de redes de ordenadores, las características más importante son las siguientes: tipo de medio físico, la topología y tipo de acceso al medio.

1.6.1.1 Tipos de medio físico

Las redes de área local pueden estar conectadas utilizando diversos medios físicos. La conexión por par trenzado consiste en un par de cables de cobre trenzados entre sí y que pueden estar apantallados o no. Este tipo de cable viene heredado de las instalaciones de la telefonía fija y en la actualidad está ampliamente extendido. Se usa en la mayoría de redes locales del tipo Ethernet. Otra forma de conectar equipos en una red es con cable coaxial, que se compone de un hilo conductor central rodeado de una malla muy fina de hilos de cobre. El espacio que queda entre el hilo y la malla está ocupado por un material aislante. Todo está cubierto por un aislante exterior. La transmisión se realiza sin modulación alguna.

Los cables coaxiales de banda ancha son muy parecidos al coaxial de banda base anterior, sin embargo, en este tipo de cable se transporta la información a diferentes frecuencias. El cable de fibra óptica se compone de varios filamentos convenientemente protegidos. Cada uno de ellos consta de un núcleo cubierto por un revestimiento. La diferencia de índices de refracción entre estos dos materiales provoca que las señales luminosas se transmitan a través del núcleo.

1.6.1.2 Topologías

Con la electrónica adecuada, la topología que puede tener una red local podría ser muy variada, sin embargo, en la mayoría de las redes locales se utiliza una topología de anillo o de bus. Con el crecimiento del número de estaciones conectadas a la red y la consiguiente degradación en el rendimiento de la misma, se plantea la necesidad de segmentación de la red mediante la introducción de una técnica "*crossbar switch"*. Si es necesario se puede llegar incluso a una solución "*crossbar switch"* completa en la que cada estación esta directamente conectada al "*switch"*.

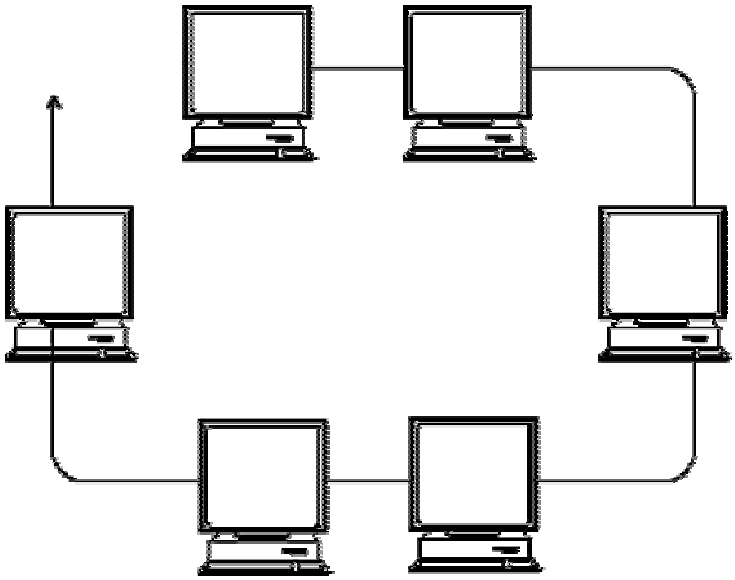

Figura 1. 1 Red de estaciones de trabajo.

1.6.1.3 Modos de Acceso

Control centralizado: Polling

En las redes que utilizan control centralizado, existen dos tipos de nodos: el nodo principal que controla el acceso al medio de transmisión y los nodos secundarios. El nodo principal pregunta a cada uno de los nodos secundarios si tienen algo que comunicar. En caso afirmativo envían el mensaje y el nodo principal continua preguntando al resto de nodos de forma secuencial.

Control Distribuido: Paso de testigo

Consiste en la circulación continua de una secuencia de bits especial llama *testigo*. Uno de los bits del testigo indica su estado: libre u ocupado. Un nodo gana el derecho de acceso cuando recibe un testigo libre. Si tiene algún mensaje preparado lo transmite y genera un nuevo testigo que envía al siguiente nodo.

Paso de testigo en bus: Token-Bus

El paso del testigo se realiza superponiendo un anillo lógico sobre un bus físico, asignando direcciones de destino adecuadas a cada uno de los nodos. La ventaja principal de este sistema es la flexibilidad, mientras que el principal problema consiste en la latencia que se introduce debido a que el testigo debe ser totalmente leído antes de comenzar a ser transmitido.

Paso de testigo en anillo: Token-Bus

El cableado físico proporciona ya el camino que va a seguir el testigo. Un testigo circula continuamente en el anillo, cuando una estación lo recibe y detecta que está libre, cambia su estado a ocupado y envía el mensaje justo después del testigo.

Acceso aleatorio

La idea de los métodos de acceso aleatorio consiste en acceder al medio siempre que se encuentre vacío. Debido a la latencia propia de la red varias máquinas pueden comenzar a enviar un mensaje sobre un medio en principio vacío, evidentemente la información se va a mezclar. Esta situación se denomina *colisión*. La diferencia entre los distintos métodos consiste en el mecanismo de contención o resolución de colisiones.

Acceso aleatorio a bus: CSMA/CD

Cada nodo inspecciona continuamente el estado del medio de transmisión e inicia una transmisión sólo si el medio se encuentra vacío. Debido a la propagación de las señales puede que otro nodo haya comenzado a transmitir también con lo que se produce una colisión. Para evitar esto los nodos esperan un tiempo aleatorio después de que el medio queda libre antes de comenzar a transmitir.

Acceso aleatorio a un anillo: Inserción de registro

Cuando un nodo tiene un mensaje que transmitir lo coloca en un registro de desplazamiento. Cuando se crea en el anillo un hueco apropiado, el registro se conecta en serie al anillo y comienza a transmitir. Si mientras tanto aparece una trama de datos por el lado de recepción, se va almacenando en el registro, esperando su transmisión posterior (excepto si éste es el nodo de destino). Por tanto, el nodo debe asegurarse que tiene espacio suficiente para almacenar los mensajes que le lleguen.

Las ejecuciones se han realizado sobre una red local compuesta por estaciones de trabajo Sun conectadas a un Etherswitch que denominamos LAN-UTP, y directamente con un cable coaxial que denominados LAN- COA, con acceso tipo CSMA/CD en ambos casos. Las estaciones de trabajo utilizadas han sido Sparc-5 con una velocidad de reloj de 70MHz.

1.6.2 IBM SP2

La multicomputadora IBM SP2 es un ordenador paralelo con memoria distribuida en el que los procesadores están interconectados mediante un subsistema de comunicaciones. Los nodos están basados en los procesadores PowerPC o P2SC. Pueden ser clasificados en tres tipos: *thin*, *wide* o *high*. IBM oferta varias posibilidades para el subsistema de comunicaciones, desde tecnología de red estándard, como Ethernet, FDDI (Fiber Distributed Data Interface) o ATM (Asyncronous Transfer Mode), hasta desarrollos propios de IBM como el *High Performance Switch*, que ofrece los mejores rendimientos de las comunicaciones.

El subsistema de comunicaciones de altas prestaciones está compuesto por el *high-performance switch* y los adaptadores que conectan cada nodo al conmutador. Los adaptadores disponen de un microprocesador que descarga al nodo del trabajo relativo al paso de mensajes entre nodos, además dispone de una memoria que usa como *buffer*. Un mecanismo de acceso directo a memoria se encarga de mover la información entre el nodo y el adaptador.

IBM describe el *high-performance switch* como "una red multietapa de conmutación de paquetes entre cualquier origen y destino, similar a una red *omega*". En esta red, el ancho crece linealmente con el tamaño del sistema, lo que garantiza la escalabilidad del sistema. El núcleo de la red es un chip crossbar que dispone de ocho puertos bidireccionales, que se puede usar para construir pequeños sistemas SP2. Los sistemas mayores deben usar tarjetas que contienen dos etapas de cuatro chips cada una, con lo que se dispone de un total de 32 puertos bidirecionales. Estos sistemas disponen además de al menos una etapa adicional para aumentar la redundancia de caminos. Existen configuraciones para 16, 48, 64 y 128 nodos.

La máquina con la que se realizaron las ejecuciones dispone de 12 + 32 procesadores, 12 GB de memoria principal, 431 GB en disco y una velocidad punta de 18,55 Gflop/s.

1.6.3 Silicon Graphics Origin 2000

Cualquier sistema Origin se compone de un número de nodos interconectados entre sí mediante una fibra. Cada nodo consta de uno o dos procesadores, memoria, un directorio para coherencia de cache, y dos interfaces: una que conecta con el sistema de entrada/salida (XIO) y la otra con el sistema de interconexión (CrayLink).

LOS MÓDULOS DE PROCESADORES

Cada módulo de procesadores contiene uno o dos procesadores R10000, cache de segundo nivel (1 ó 4 Mbytes), memoria principal, un directorio para la coherencia de la cache, un *hub*, una interfaz de entrada y salida y una interfaz para conectarse a la fibra.

Los sistemas Origin usan memoria compartida distribuida. La memoria se encuentra distribuida en los módulos de procesadores pero es accesible por todos los procesadores. Evidentemente el coste de acceder a la memoria disponible en el módulo local es mucho más bajo que el coste de acceder a cualquier otro banco de memoria de otro módulo.

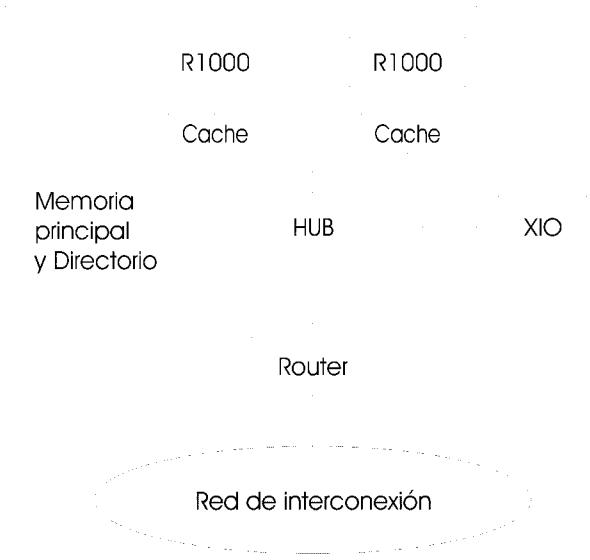

Figura 1. 2 Estructura de un nodo de un Origin 2000.

El *hub* es un *crossbar switch* que dispone de cuatro puertos que conecta a los procesadores, la memoria principal y su memoria de directorio asociada, el sistema de intercomunicación y el subsistema de entrada y salida. Estas cuatro interfaces están interconectadas mediante un *crossbar* interno. Las interfaces se comunican enviando mensajes a través del *crossbar*. Los puertos para la memoria y los procesadores son bidireccionales y trabajan a 780Mb/s. Los puertos para entrada y salida y el *CrayLink* son dos half-duplex para cada uno que trabajan a 780 Mb/s. El *hub* controla tanto la comunicación intranodo entre cualquiera de los cuatro subsistemas que conecta como la comunicación con otros nodos. El *hub* se encarga de convertir los mensajes internos que usan un formato de petición/respuesta a los formatos externos que usan el sistema de entrada y salida y el CrayLink. Todos los mensajes internos son lanzados por el procesador o los dispositivos de entrada y salida. Las cuatro interfaces del *hub* actúan como controladores individuales de su propio subsistema. Cada interfaz tiene dos *buffers* FIFO, uno para los mensajes entrantes y otro para los mensajes salientes. Cuando llega un mensaje desde el exterior la interfaz lo convierte al formato *intra-hub* y lo coloca en la cola correspondiente. Los mensajes pueden ser clasificados como peticiones y respuestas. Cada *buffer* FIFO está dividido en dos a nivel lógico: uno para peticiones y otro para respuestas. El protocolo de coherencia de cache y los caminos separados para peticiones y respuestas garantizar que el sistema esté libre de interbloqueos.

EL SISTEMA DE INTERCONEXIÓN

El sistema de interconexión es un conjunto de conmutadores, llamados *"routers"*, que están unidos por cables en varias configuraciones. Entre este sistema y el bus existen diferencias importantes:

- La fibra de interconexión es una malla de múltiples enlaces punto a punto conectados por conmutadores. Estos enlaces y conmutadores permiten que ocurran múltiples transacciones simultáneamente.
- Los enlaces permiten una conmutación extremadamente rápida.
- La fibra de interconexión no requiere arbitraje para su acceso ya que su uso esta limitado por contención.
- Cuando se incrementa el número de nodos se aumenta también el número de *routers* y de enlace, aumentando de esta forma el ancho de banda. En un sistema de *bus* compartido el ancho de banda es fijo.
- La topología del *CrayLink* permite que el ancho de banda de bisección crezca linealmente con el número de nodos en el sistema.

El ordenador Origin 2000 en el que se realizaron las ejecuciones dispone de 32 + 32 procesadores R10000 (196 MHz), 8 GB de memoria principal, 288 GB en disco y una velocidad punta de 25,08 Gflop/s.

1.6.4 Digital Alpha Server 8400

Es un sistema de bus compartido alrededor del cual se pueden conectar diversos dispositivos: uno o varios procesadores, módulos de memoria y módulos de entrada y salida.

El *bus* trabaja de forma síncrona con los procesadores a un submúltiplo de reloj de estos y puede llegar hasta los 100Mhz. Se compone de dos caminos separados: uno para datos de 256 bits de ancho y otro de instrucciones y direcciones de 40 bits de ancho. Para enlazar las direcciones o instrucciones que viajan por un *bus* con los datos que viajan por el otro se usa un número de secuencia. A este *bus* se conectan, a través de los *slots*, tanto los módulos que contienen procesadores como los que contienen memoria o dispositivos de entrada y salida. Dependiendo del modelo se dispone de un número diferente de *slots*, siendo nueve el mayor para el modelo 8400. Cualquier sistema debe disponer de al menos un módulo con procesadores, uno con memoria y uno de entrada y salida. El resto de *slots* se pueden completar con las siguientes limitaciones: hasta siete módulos con procesadores, hasta siete con memoria y hasta tres de entrada y salida.

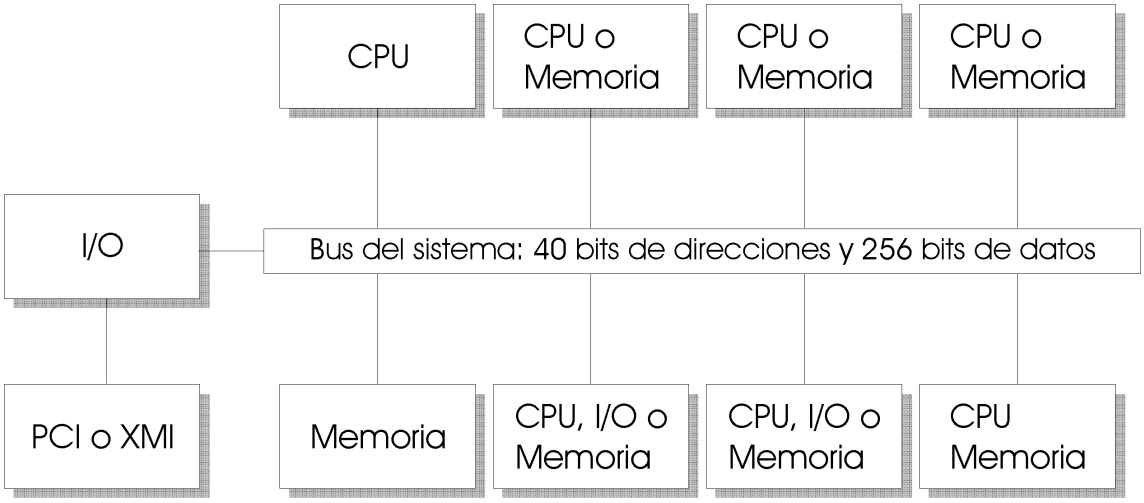

Figura 1. 3 Esquema interno de la DIGITAL Alpha 8400.

Un modulo de procesador puede tener uno o dos microprocesadores Alpha 21164. Cada procesador dispone de su propia conexión con el *bus* y de su propia cache. Dentro de cada procesador hay 8 Kbytes disponibles para cache de instrucciones, 8 Kbytes para cache de datos y 96 Kbytes de cache de segundo nivel. Además existe una cache de tercer nivel en el módulo para cada procesador con un tamaño de 4 Mbytes. Un sistema Alpha 8400 puede disponer de hasta siete módulos de procesadores, y por tanto, un total de 14 procesadores.

La memoria máxima que se puede conectar al sistema es de 28 Gbytes dividida en siete módulos de memoria de 4Gbytes cada uno. La memoria está dividida en bloques para permitir memoria entrelazada. Cada módulo de 2 Gbytes dispone de dos vías de acceso a memoria. Un sistema con 28 Gbytes dispone de 16 vías. El número de vías disponibles depende de la configuración de la memoria (número de módulos y tamaño de cada uno de ellos).

Existen dos tipos de módulos de entrada y salida, los que implementan buses PCI (Peripherical Component Interconnect) y los que implementan buses XMI. En ambos casos cada módulo dispone de 12 slots.

Las pruebas se realizaron en una máquina Digital AlphaServer 8400 con 10 procesadores Alpha 21164 (440 Mz), 2 GBytes de memoria principal, 60 GBytes en disco y una velocidad punta de 8,80 Gflop/sec.

1.6.5 CRAY T3E

El CRAY T3E es un sistema multiprocesador de memoria compartida distribuida que soporta hasta 2048 procesadores interconectados mediante un toro 3D. Cada nodo contiene un procesador Alpha 21164, un chip de control del sistema, memoria local y un *router*. La lógica del sistema trabaja a una velocidad de 75 Mhz, mientras que los procesadores trabajan a algún múltiplo de esta velocidad (300 Mhz en el CRAY T3E o 450 Mhz en el CRAY T3E-900). Los enlaces del toro tienen un ancho de banda agregado de 600 MB/s en cada dirección y un ancho de banda útil de entre 100 y 480 MB/s dependiendo del tipo de tráfico. El sistema de entrada y salida utiliza un canal llamado *GigaRing* con un ancho de banda de 267 MB/s para cada cuatro procesadores.

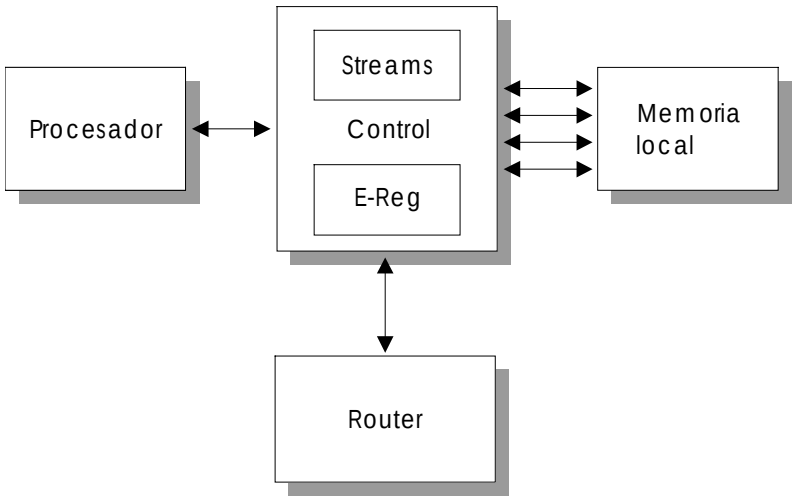

Figura 1. 4. Esquema interno del CRAY T3E.

La memoria local consiste en un conjunto de cuatro chips controladores de memoria, que controlan directamente a ocho bancos de memoria física. Cada controlador de memoria esta conectado con el chip controlador principal mediante un bus de 32 bits, con lo que dispone de una capacidad máxima de 1,2 GB/s. Este ancho de banda está mejorado además mediante un conjunto de *stream buffers*. Estos detectan automáticamente referencias a direcciones consecutivas y realizan una prebúsqueda de la información en la memoria local. Un nodo de CRAY T3E no dispone de memoria *cache* de segundo nivel.

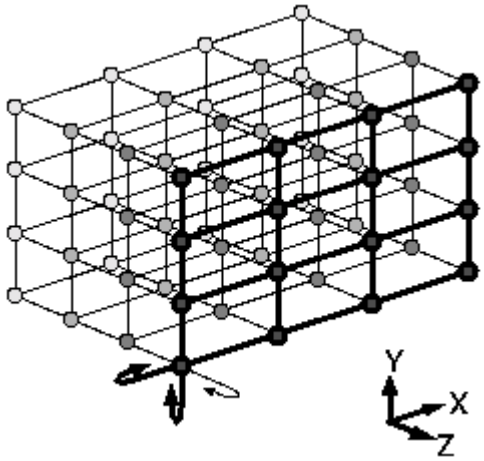

Figura 1. 5. Estructura del toro del Cray T3E.

Las pruebas se realizaron en una máquina CRAY T3E con 32 procesadores DEC 21164 (Alpha EV-5), con una velocidad de reloj a 300 *MHz*, juego de instrucciones RISC de 64 bits, potencia de pico de 600 Mflops por procesador, Red toroidal 3D de interconexión de baja latencia, escalable hasta 136 procesadores, con l28 MBytes de memoria distribuida por procesador, escalable hasta 2Gbytes, con capacidad de l30 GBytes de disco duro SCSI. El sistema de entrada/salida paralela está basado en la tecnología *GigaRing* de SGI/Cray.

2.1 INTRODUCCIÓN

El objetivo principal del modelo LogP es proponer un modelo de computación paralela que sirva como base para el análisis y diseño de cualquier algoritmo con el fin de poder implementarlo sobre un amplio conjunto de arquitecturas paralelas actuales y futuras.

Como se ha destacado en el primer capitulo de esta memoria, diferentes modelos han sido propuestos (el modelo PRAM y sus variantes, modelos de Redes, el modelo BSP y sus variantes, etc.) pero ninguno ha resultado ser lo suficientemente realista, robusto y preciso para las máquinas paralelas actuales y futuras. El modelo PRAM por la incapacidad de crear máquinas reales que soporten eficientemente el modelo. El modelo de Redes es dependiente de la arquitectura que se esté utilizando. Un algoritmo para una determinada red no servirá para redes diferentes, siempre habrá que realizar modificaciones, pudiendo ser muy costoso. El modelo BSP es bastante estricto respecto a la idea del superpaso y las barreras. En cada paso, existe un conjunto de operaciones de cómputo seguidos por una relación de comunicaciones entre los diferentes procesadores. El modelo LogP, según sus autores [Cul93a], intenta ser lo más realista posible, teniendo en cuenta los principales factores de rendimiento de los procesadores y de la red de interconexión. Los autores indican que con el LogP se facilitará el análisis y diseño de los algoritmos y se ayudará a obtener mejores predicciones en los rendimientos y comportamientos de las comunicaciones y el cómputo.

Uno de los puntos de partida de este modelo fue el modelo Bulk Synchronous Parallel Model (BSP). Los principios de ese modelo ofrecieron a los autores una visión de lo que debería ser un modelo para la computación paralela: realista y simple de usar, además de permitir el diseño de algoritmos que funcionen bien sobre un amplio conjunto de máquinas.

El otro punto de partida del modelo, fue la convergencia existente de las arquitecturas actuales. La creación de un modelo de computación paralela general ha sido imposible debido a la diversidad de arquitecturas paralelas, cada una con sus propias particularidades y modelos asociados (MIMD, SIMD, memoria compartida, sistólicos, flujo de datos, paso de mensajes). Pero por razones tecnológicas y de costes, este espectro se ha reducido a una colección de computadoras completas, consistentes en módulos con procesadores, memoria cache y memoria DRAM; y todos conectados a través de una red de comunicaciones muy potente. El espectacular aumento de las prestaciones de los procesadores y de la capacidad de la memoria ha provocado la aparición de nuevos modelos que se ajustan más a las arquitecturas dominantes. Es importante destacar como las grandes compañías de máquinas paralelas utilizan procesadores típicos de estaciones de trabajo e incluso de los ordenadores personales. Todo esto parece indicar que el futuro de las máquinas paralelas estará más bien en tener cientos o miles de procesadores estándar de 64 bits, que tener millones de procesadores de 1 bit.

Respecto a la tecnología de las redes también el avance es espectacular pero no al nivel de los dos casos anteriores. Actualmente, el ancho de banda de la red está muy por debajo del ancho de banda entre la memoria y el procesador. El coste de mover datos entre procesadores es mucho mayor que dentro de la misma computadora entre la memoria y el procesador. El ancho de banda de la red está limitado fundamentalmente por

el excesivo coste de entrada y salida de datos del canal y la interfaz de red. Por tanto, los problemas de las redes de interconexión siguen siendo los grandes valores de latencia, el insuficiente ancho de banda y el "*overhead*" del procesador (trabajo realizado por el procesador para poder enviar o recibir datos).

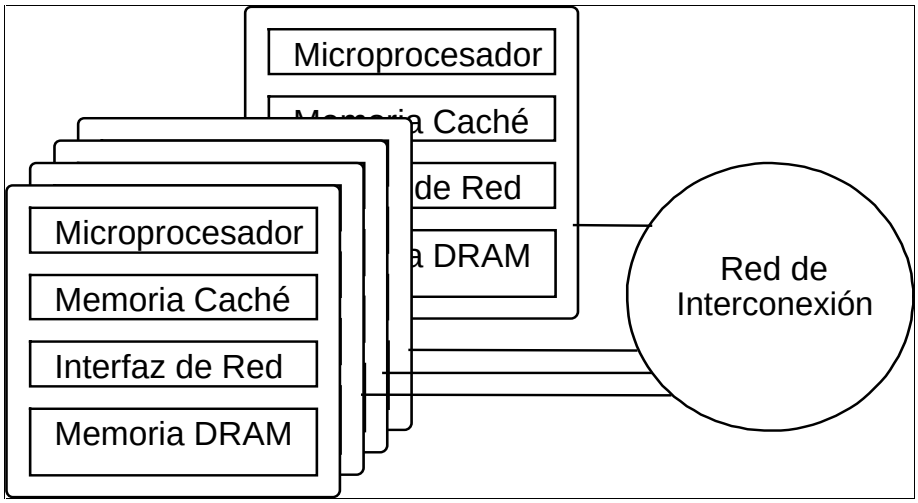

Figura 2. 1 Esquema básico de las arquitecturas actuales.

Respecto a la topología de las redes de interconexión, no existe un consenso acerca de las mismas. Cada compañía propone un esquema propio e incluso variando entre sus diferentes máquinas. Esto provoca sobre los algoritmos, la dependencia de la red subyacente para obtener unos buenos rendimientos.

El modelo LogP caracteriza una máquina paralela por un conjunto reducido de parámetros, ocultando características específicas de las mismas como son la topología y los algoritmos de encaminamiento.

2.2 EL MODELO LogP

Culler et. al. [Cul93] desarrollaron el modelo LogP para computadoras de memoria distribuida donde los procesadores realizan las comunicaciones a través de mensajes punto a punto. Actualmente, la mayoría de las computadoras siguen este esquema. En el modelo se tienen en cuenta las características de rendimiento de la red pero no su estructura interna de interconexión (multietapa, hipercubo, toro, etc.).

Una de las características más importantes del modelo y que lo diferencian de otros, es su consideración sobre el solapamiento de las comunicaciones y el cómputo. Mientras el mensaje viaja por la red, el procesador puede estar realizando cómputo útil y avanzar en su trabajo sin esperar a que el mensaje llegue a su destino. En los ejemplos expuestos en este capítulo se observará claramente este efecto de solapamiento.

El Modelo LogP queda caracterizado por los siguientes parámetros:

- L: una cota superior de la latencia, o retraso, que aparece cuando se envía un mensaje de tamaño pequeño M, desde un módulo

procesador/memoria origen hasta el modulo procesador/memoria destino.

- *o*: el *overhead*, definido como la cantidad de tiempo en que un procesador está ocupado en la transmisión o recepción de un mensaje. Durante este tiempo, el procesador no puede realizar otras operaciones. Típicas funciones de *overhead* son el empaquetado y desempaquetado de datos y la gestión de *buffers*.
- *g*: el gap, definido como el intervalo mínimo de tiempo entre la transmisión o recepción de dos mensajes de tamaño M consecutivos. El inverso de *g* es el ancho de banda disponible por procesador.
- P: el número de módulos procesador/memoria.

Para obtener los valores de los parámetros, el modelo no tiene en cuenta factores que podrían afectar a los mismos como son la saturación de la red, los mensajes largos, hardware especial de encaminamiento y los patrones de comunicaciones. Los parámetros L, o y g se miden normalmente en unidades en función de múltiplos de ciclos de reloj.

El modelo es asíncrono en el sentido que los procesadores trabajan de forma asíncrona y la latencia de cualquier mensaje es imprevisible, pero siempre acotada por L. Además se debe tener en cuenta que debido a las variaciones de la latencia, los mensajes dirigidos a un destino concreto podrían no llegar en el mismo orden en que fueron enviados. La única limitación impuesta por el LogP, es la limitación de la capacidad de la red con un máximo de $\lceil L/g \rceil$ mensajes que pueden estar en tránsito de un procesador a otro en cualquier instante de tiempo. Si un procesador intenta transmitir por encima de este límite, se quedará en un estado de espera, hasta que el mensaje se pueda enviar.

La figura 2.2 ilustra claramente el ámbito de cada parámetro. La transmisión de un mensaje de longitud M bits en una red débilmente cargada consiste en cuatro pasos bien diferenciados. En primer lugar el *overhead* de envío T_{send}, que consiste en el tiempo en que el procesador está ocupado colocando el mensaje en la interfaz de red y aún no ha colocado ningún bit en la red. En segundo lugar, el mensaje es enviado a la red poco a poco según el ancho del canal w. El tiempo para introducir el último bit de un mensaje de M bits en la red es $\lceil M/w \rceil$ ciclos. Actualmente la mayoría de las máquinas de supercomputación existentes utilizan métodos de "cut-through routing" [Dua97], de forma que el tiempo que tarda en atravesar la red el último dato, hacia el procesador destino es $H\,r$, donde r es el retraso que ocurre en cada uno de los H nodos intermedios, independientemente del tamaño del mensaje. Por último, el *overhead* de recepción T_{recv} es el tiempo que se consume desde que llegan los datos a la tarjeta del procesador destino hasta que este puede utilizarlos. Por tanto, el tiempo total para comunicar un mensaje de longitud M bits a través de H nodos intermedios viene dado por:

$$T(M, H) = T_{send} + \lceil M/w \rceil + H\,r + T_{recv} \qquad (2.1)$$

El modelo básico asume que los mensajes tienen un tamaño fijo pequeño M. Otros autores, como Alexandrov et. al. en su trabajo "LogGP: Towards a Realistic Model of Parallel Computation" [Ale95], ofrecen una versión del modelo donde queda caracterizado tanto para tamaños pequeños como grandes. En esta versión del modelo, los autores argumentan que muchas de las máquinas actuales utilizan almacenamientos internos (*buffers*) que provocan que el modelo para tamaños pequeños no se ajuste correctamente y el LogGP si lo haga.

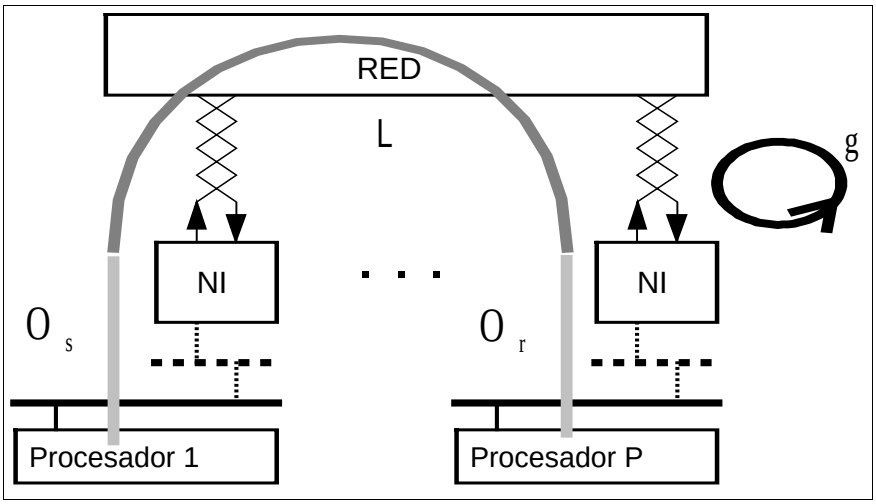

Figura 2. 2 Ambito de actuación de los parámetros del modelo LogP.

Una de las autocríticas que realizan los autores es que el LogP considera muchos parámetros. Esto hace que el análisis de algoritmos sea más complejo. Afortunadamente, no todos los parámetros son igualmente importantes en todas las situaciones. En mucho casos, uno o más de estos parámetros se pueden ignorar sin debilitar el análisis. Por ejemplo:

- En algoritmos que comunican datos de forma esporádica, se podrían ignorar el ancho de banda g y la capacidad de la red $\lceil L/g \rceil$.
- En algoritmos que envían flujos de datos continuos, (por ejemplo en forma de pipeline), el tiempo de transmisión de un mensaje está dominado por el gap entre mensajes g y la latencia L podrían descartarse.
- En algunas máquinas el *overhead* o, domina sobre el gap ($o>>g$) y por tanto g se puede eliminar.

La elección de estos parámetros representa un compromiso entre las características de máquinas reales y la posibilidad de ofrecer a los usuarios de algoritmos paralelos, una metodología para el análisis y diseño de los mismos.

2.2.1 Cálculo de los Parámetros

Las interfaces de red convencionales y los protocolos asociados constituyen actualmente un cuello de botella importante en las comunicaciones de las computadoras. Con protocolos como TCP/IP y librerías de comunicaciones como MPI o PVM, las prestaciones de las comunicaciones son poco eficientes y las restricciones muy elevadas. Estos imponen una sobrecarga muy elevada a la hora de enviar o recibir mensajes: entre cientos y miles de instrucciones por mensaje. Los nuevos protocolos y niveles de comunicaciones como Active Messages [Eic92] o Fast Messages [Pac95] disminuyen enormemente este coste: sólo decenas de instrucciones por mensaje. Los trabajos [Kim95] y [Ros96] son una muestra de los estudios realizados sobre la influencia de los protocolos actuales, además de proponer métodos alternativos utilizando Active Messages y Fast Sockets respectivamente como nuevo nivel para las comunicaciones. También en [Mar96] podemos encontrar un estudio detallado de la influencia del *overhead* en diferentes aplicaciones que se ejecutan sobre las redes de estaciones de trabajo. Se muestra como

aumentando los valores del *overhead*, las aplicaciones pueden ser hasta 60 veces más lentas con 32 procesadores. Se trata de un estudio muy interesante al proporcionar un método sistemático para observar la gran influencia del *overhead* sobre las aplicaciones convencionales.

Culler et. al. en su trabajo "LogP Performance Assessment of Fast Network Interfaces", [Cul95] describen una metodología completa basada en Active Messages [Eic92], para la obtención de los parámetros del LogP. Presentan unas métricas que denominan "microbenchmarks", que permiten medir los valores de los parámetros del modelo. En este trabajo utilizan como plataforma software Active Messages y como plataformas hardware: Intel Paragon, Meiko CS-2 y un cluster de estaciones SparcStations conectadas con una red de altas prestaciones Myrinet [Bod95].

Aunque los microbenchmarks que Culler et. al. han sido diseñados para Active Messages, los autores argumentan que se pueden utilizar de igual forma sobre librerías como MPI y PVM en cualquier máquina que lo soporte [Cul95].

Las métricas se basan en el tiempo que invierte un mensaje en viajar desde un procesador origen a otro destino: $2o+L$. Este tiempo se divide en una parte de *overhead*, o (suele ser de empaquetamientos y almacenamientos internos) y otra parte de viaje del mensaje por la red, L. El *overhead* está dominado por el software de comunicaciones, el coste de acceso de la interfaz de red a la memoria y el coste de acceso al bus de entrada salida. La latencia depende del tiempo invertido en la interfaz de red, en el canal de comunicaciones y en los retrasos debidos a los nodos intermedios de la red. Durante el tiempo de una latencia, el procesador puede realizar cómputo efectivo sobre alguna tarea (efecto de solapamiento anteriormente descrito).

El tercer parámetro del modelo, el gap g, refleja el tiempo en el que la tarjeta de red puede enviar o recibir dos mensajes consecutivos. El inverso del gap, es el ancho de banda efectivo en mensajes por unidad de tiempo. De esta forma, enviar N mensajes consecutivos de un procesador a otro requiere un tiempo de: $o+(N-1)g+L+o$. En caso de envíos desde diferentes i procesadores al mismo destino, el ancho de banda efectivo queda limitado por el receptor con un mensaje cada g unidades. El gap depende del tiempo invertido en la interfaz de comunicaciones y del ancho de banda del canal.

2.2.2 Algoritmos para computar los parámetros

El tiempo consumido en enviar y recibir un mensaje entre dos procesadores sin saturar la red, se conoce como el tiempo de "PingPong" (TPP) y corresponde con la suma de un *overhead* de envío, una latencia de viaje del mensaje por la red y un *overhead* de recepción: $o + L + o$.

El algoritmo para la evaluación de los parámetros consiste en realizar un conjunto de envíos de M mensajes de tamaño pequeño y realizar una firma gráfica de su comportamiento. El código de la figura 2.3, basado en Active Messages, ilustra esta situación:

```
Comienzo del reloj
        Repetir M veces
                Realizar peticiones
Parar el reloj
Recepción de mensajes solicitados.
```

Figura 2. 3 Microbenchmark básico.

Para pequeños valores de *M*, el emisor realiza todas peticiones antes de recibir ninguna respuesta. Esto sucede para todo valor de *M*, menor que el tiempo del PingPong, dividido por el tiempo que se tarda en realizar un *overhead* de envío (TPP/o_s). Según hacemos crecer *M*, una parte de las respuestas llegarán aún cuando se siguen realizando peticiones, y el coste de enviar y recibir los mensajes aumenta. Y ¿cuánto deberían estar separadas estas respuestas? Pues, exactamente *g*, ya que es el tiempo en que se puede transmitir dos mensajes consecutivos.

Si *M* sigue creciendo, el número de mensajes en la red aumenta y se alcanzará la limitación impuesta por la capacidad de la red. En este momento se deberán recoger los mensajes de respuesta antes de que un nuevo mensaje sea insertado en la red. De nuevo, el coste de los mensajes es *g*.

De esta forma, el coste medio de envíos y recepciones de mensajes, en función de *M*, debería seguir una curva como la que se indica en la figura 2.4. En esta figura se observan tres comportamientos diferentes: "estado de sólo envíos" para valores pequeños de *M*, "estado estacionario" para valores de *M* grandes, y el "estado de transición", para valores intermedios de *M*.

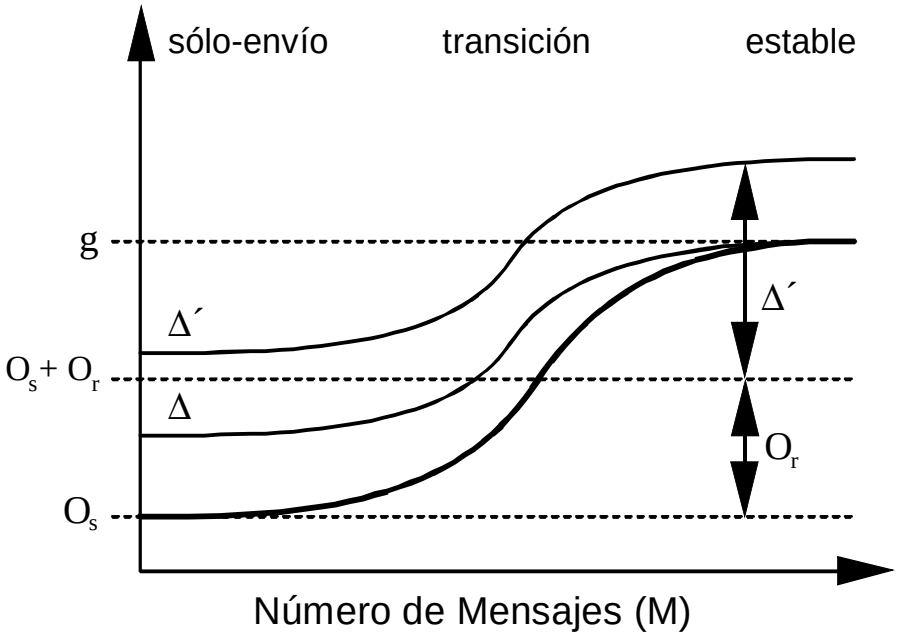

Figura 2. 4 Caracterización gráfica de los parámetros.

El tiempo medio por mensaje en el estado de "sólo envíos" nos proporciona o_s. El régimen de transición comienza en (TPP/o_s), cuando llega el primer mensaje de vuelta o cuando se alcanza la capacidad de la red. Se alcanza asintóticamente g cuando estamos en el estado estacionario. Del tiempo del pingpong obtenemos la suma: $o_s + (N-1)\ g + L + o_r$. Nos falta obtener o_r para poder obtener L.

En el estado de transición de la figura 2.6 se observa que $o_s + o_r + Ocioso = g$ donde *Ocioso* es el tiempo que el emisor emplea esperando para obtener un mensaje de la red. No podemos medir directamente *Idle* ni o_r, ya que la petición de respuesta se colgará esperando por un mensaje de vuelta y luego utilizará un *overhead* de recepción para retirarlo de la red.

La propuesta para calcular o_r consiste en realizar una cierta cantidad de computación perfectamente controlable entre mensajes consecutivos: *DELTA* (Δ). Como muestra la figura 2.6 para valores de *DELTA* mayores que *Idle*, el emisor se convierte en

```
Comienzo del reloj
    Repetir M veces
        Realizar peticiones
        Computar durante DELTA
Parar el reloj
Recepción de mensajes solicitados.
```

Figura 2.5 Microbenchmark con DELTA.

el cuello de botella y el coste medio del mensaje es $o_s + o_r + DELTA = g'$, donde g' es el nuevo cuello de botella. Ya que conocemos *DELTA* y podemos medir o_s y g', podemos obtener o_r.

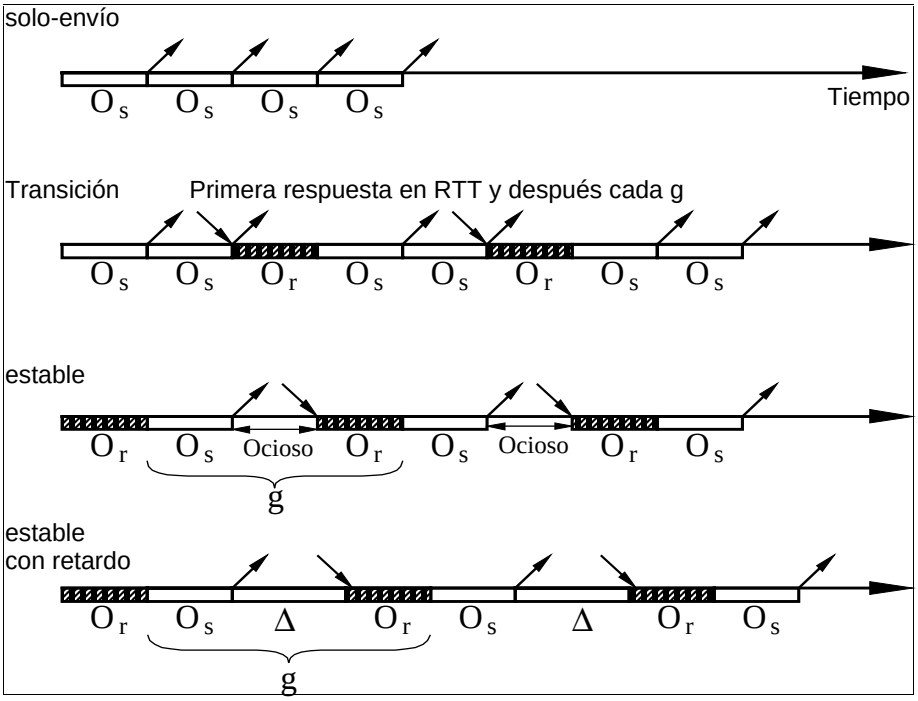

Figura 2.6 Estados según aumenta el número de mensajes.

Ejecutando el algoritmo de la figura 2.5 para un rango de *M* y *DELTA*, podemos construir la figura 2.4. Cualquier valor de *DELTA* mayor que *Ocioso* tendrá un coste de mensaje en estado estacionario de $g' > g$. Del gráfico se puede observar directamente los valores de g, o_s y o_r. Y por tanto, del tiempo obtenido del algoritmo PingPong (TPP) despejamos el valor de la latencia.

2.2.3 Metodología para calcular los parámetros en MPI

Describiremos en esta sección los experimentos para obtener los valores de L, o y g en MPI en vez de usar *Active Messages* o *Fast Messages* como se hace en el apartado anterior. Para ello seguimos el trabajo de Kort y Trystram [Kor98] "Assesing LogP Model Parameters for the IBM SP2". En este punto necesitamos unas definiciones previas:

Definición 2.1: *Una primitiva de envío se dice "local" si su finalización no depende del estado del proceso receptor.*

Definición 2.2: *Una comunicación se dice "eager" si el mensaje se envía sin esperar a que se produzca una petición de recepción.*

Definición 2.3: *Una comunicación diremos que es "rendez-vous" si el mensaje no se envía a menos que exista una petición desde el receptor*

Definición 2.4: *El "delay" de una primitiva de envío o recepción es el tiempo que tarda en ejecutarse la llamada.*

Este apartado se concentra en las tres primitivas *MPI_Bsend*, *MPI_Ssend* y *MPI_Send*. El objetivo es responder a las siguientes preguntas. ¿Es la primitiva de envío *local*? ¿Se envían los mensajes según los protocolos *eager* o *rendez-vous*? ¿Es el *overhead* igual al *delay* o las capas encargadas del transporte actúan concurrentemente consumiendo tiempo del procesador principal? Para responder a estas cuestiones, se diseñan tres experimentos. El primer experimento, figura 2.7, nos dice si la primitiva es *local* o no. Si el tiempo medido por este primer experimento depende de la cantidad de tiempo empleada por la computación en el segundo procesador la primitiva no es local. En el segundo experimento (figura 2.8) medimos el *delay* del *MPI_Recv* y estudiamos la variación en dicho tiempo cuando aumentamos el tiempo de computación. El protocolo utilizado es *eager* si este retraso disminuye con la cantidad de tiempo invertida en computación. En caso contrario deducimos que estamos trabajando con un protocolo *rendez-vous*.

La tabla 2.1 muestra la variedad de conductas que se observan en términos del tamaño del mensaje en una IBM SP2.

Primitiva Send	n ≤ 4096	n > 4096
MPI_Ssend	No local, rendez-vous	No local, rendez-vous
MPI_Bsend	Local, eager	Local, rendez-vous
MPI_Send	Local, eager	No local, rendez-vous

Tabla 2.1 Conducta de las primitivas de envío en una IBM SP2.

En el tercer experimento detectamos si el *overhead* es igual al *delay*, es decir igual al tiempo que se invierte en la llamada a la rutina de envío. Si el cómputo realizado después de la llamada a la primitiva de envío "estorba" y retrasa el tiempo cronometrado de la recepción, podemos deducir que los procesos de liberación del mensaje no terminan con la finalización de la rutina. En ese caso, existe un *overhead* posterior a la finalización del *delay*.

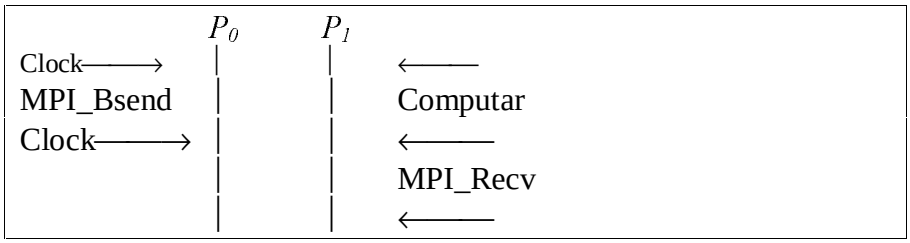

Figura 2.7. Comprobando si es *local*.

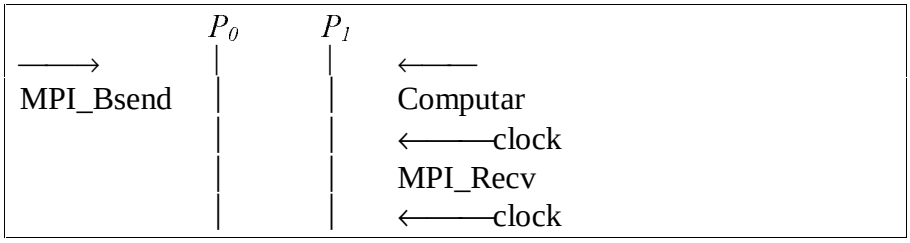

Figura 2.8. Comprobando si es *eager*.

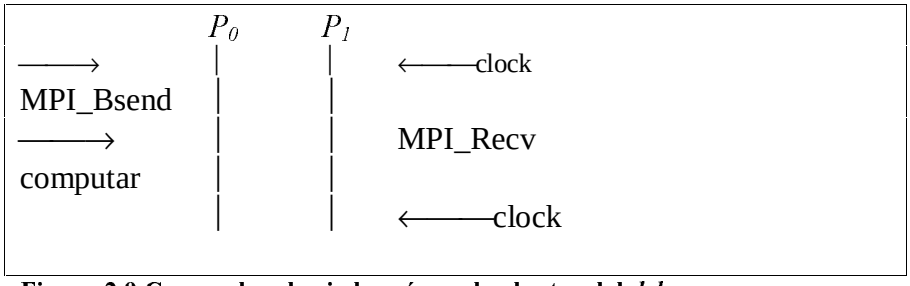

Figura 2.9 Comprobando si el envío acaba dentro del *delay*.

Los experimentos realizados en la IBM SP2 muestran que en esta máquina el tiempo del *MPI_Recv* no depende de la computación en el primer procesador si el mensaje es de tamaño menor que 1840 bytes. A partir de ese tamaño el tiempo medido crece. Se deduce que a partir de ese tamaño los mensajes se envían en bloques.

Este resultado condiciona el experimento para evaluar el *overhead* de envío. Tenemos que averiguar cuando ha sido totalmente procesado el mensaje a enviar. Para ello se utiliza la primitiva *MPI_Test*. Aunque esta primitiva fue concebida para comprobar la terminación de comunicaciones sin bloqueo, puede ser llamada con argumentos vacíos para asegurarnos del progreso de la comunicación. El experimento para evaluar el *overhead* de envío o_s consiste en:

1. Estimar el tiempo *TestDelay* que lleva la función *MPI_Test* cuando no existen comunicaciones pendientes.
2. Dado un mensaje de *n* bytes, estimar el número *TestCount* necesario de llamadas a *MPI_Test* para completar la comunicación. Para obtenerlo se realiza un experimento similar a la figura 2.9. En este caso el cómputo consiste en que P_0 llama *TestCall* veces a la función *MPI_Test*. Se repite el experimento con valores crecientes de

TestCalls. Nos detenemos cuando el retraso (*delay*) de la función *MPI_Recv* deja de crecer.

3. Medimos o_s con el siguiente programa:

```
Código para P0
MPI_Barrier(MPI_COMM_WORLD);
Os = MPI_Wtime();
MPI_Bsend(m,n,MPI_CHAR,1,TAG,MPI_COMM_WORLD);
for(i=0;i<TestCount;i++)
  MPI_Test(MPI_REQUEST_NULL,&flag,&status);
Os = MPI-Wtime()-TestCount*TestDelay;

Código para P1
MPI_Barrier(MPI_COMM_WORLD);
MPI_Recv(m,n,MPI_CHAR,TAG,MPI_COMM_WORLD,&status);
```

La diferencia que se observa entre las primitvas *MPI_Bsend* y *MPI_Ssend* en la tabla 2.2 para paquetes de más de *48KB* es debida a que, para tamaños grandes, *MPI_Bsend* utiliza el protocolo *rendez-vous* y además tiene que hacer copia del mensaje debido a su carácter de rutina *local*.

	MPI_Bsend	**MPI_Ssend**
0 ≤n≤4KB	o_s =0,04*n+30	o_s =0,04*n+110
4KB <n ≤16KB	o_s =0,081*n+90	o_s =0,053*n+20
16KB<n≤32KB	o_s =0,073*n+70	o_s =0,054*n+175
32KB<n≤48KB	o_s =0,058*n+530	o_s =0,037*n+700
n >48KB	o_s =0,050*n+870	o_s =0,034*n+890

Tabla 2.2 *Overhead* de envío para la IBM SP2.

Para evaluar el overhead de recepción o_r haremos que los dos procesadores se sincronicen. Luego P0 envía un mensaje a P1. El procesador P1 lee el reloj y emite una petición de recepción sin bloqueo. Después permanece en un bucle llamando a la función MPI_Test hasta todos los paquetes del mensaje han sido recibidos. Al tiempo medido se le resta el tiempo que se invierte en el bucle de llamadas a MPI_test cuando no existen mensajes pendientes. Para la IBM SP2 el parámetro o_r crece linealmente con el tamaño del mensaje de acuerdo con las ecuaciones:

$$o_r = 0,059*n+10 \text{ si } n \leq 16KB \text{ y } o_r = 0,035*n+420 \text{ si } n > 16KB$$

El código utilizado es el siguiente:

```
Código para P0
MPI_Barrier(MPI_COMM_WORLD);
MPI_Bsend(m,n,MPI_CHAR,1,TAG,MPI_COMM_WORLD);

Código para P1
MPI_Barrier(MPI_COMM_WORLD);
call = 0;
Or = MPI_Wtime();
MPI_Irecv(m,n,MPI_CHAR,TAG,MPI_COMM_WORLD,&status);
do
{
  MPI_Test(&request, &flag, &status);
   call++;
}
while  (flag == false);
Or = MPI_Wtime()-Or-loopDelay(call);
```

La evaluación de g y L es más complicada que la del *overhead*. Se consideran aquí sólo mensajes de tamaño pequeño. Para evaluar g suele utilizarse el microtest propuesto por Culler en el apartado anterior. El microtest mide el tiempo *delay(M)* de realizar un bucle de M peticiones (*requests*) separados entre si por una cierta cantidad de cómputo Δ. En este experimento una "petición" es una llamada asíncrona a procedimiento remoto.

Cada curva en la figura 2.4 representa la variación del cociente *Delay(M)/M* en M para diferentes valores de Δ. Si el cuello de botella de las comunicaciones es la red, entonces el tiempo promedio invertido en la petición será g. Por otra parte, si el cuello de botella es el procesador, el tiempo promedio invertido en la petición será la suma de los *overheads* de envío y recepción o_s+o_r. Para distinguir en que caso estamos los experimentos se realizan con diferentes valores de Δ. Si para valores grandes de M la curva crece incluso para valores pequeños de Δ es que los procesadores son más lentos. Si no varía para valores de Δ pequeños es que la red es el cuello de botella. Este es el caso de la IBM SP2. Para un paquete de tamaño *1000* bytes, tomando $\Delta = 0$ y $\Delta = 20$ microsegundos resulta que las dos curvas tienen diferentes valores de donde se deduce que el *overhead* es el cuello de botella en la IBM SP2. Los valores para esta máquina son $g = 150\mu s$, $o_s = 72\mu s$ y $o_s = 82\mu s$. Comparativamente la latencia L de la SP2 es muy pequeña y puede ser despreciada.

2.2.4 DISEÑO DE ALGORITMOS EN EL MODELO LogP

En los trabajos de [Cul93],[Ale95],[Kar93][Ian97] se ilustra como trabaja el modelo y su capacidad de ser un esquema válido para muchos algoritmos. En estos trabajos se muestran como soluciones eficientes de algoritmos usuales como emisión de mensajes de uno a muchos, patrones de todos a todos, etc. tienen bajo el modelo LogP una solución eficiente sobre las arquitecturas actuales. El primer trabajo de los reseñados, considera algoritmos óptimos para los problemas de las sumas en árbol y de la emisión de un mensaje al resto de procesadores, "broadcast". La solución tradicional a estos problemas

consistía en simples árboles perfectamente balanceados. Bajo el modelo LogP, el algoritmo óptimo de sumas en árbol y de "broadcast" resulta ser un árbol no balanceado con un número de nodos dependiente de los parámetros L, o y g.

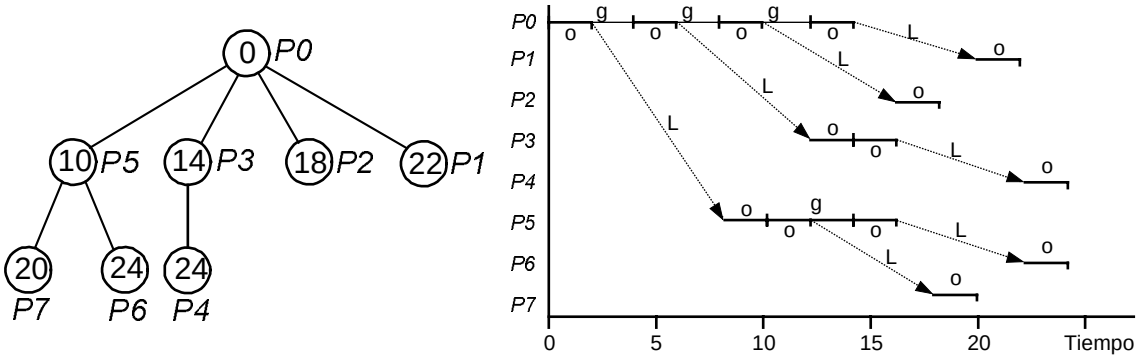

Figura 2.10 Emisión de un mensaje a muchos. L=6, g=4, o=2 y P=8.

La figura 2.10 ilustra el ejemplo de la emisión de un mensaje desde el procesador raíz al resto de procesadores del sistema *P-1*. Se basa en que todo procesador que ha recibido el mensaje, lo transmite lo antes que puede, asegurando que ningún procesador recibe más de 1 mensaje. De esta forma, el procesador fuente del broadcast comienza la transmisión del mensaje en tiempo 0. El dato entra en la red después de haber sido consumido por el procesador un *overhead* o e invierte una latencia L para llegar al destino que tendrá que consumir de nuevo un *overhead*. Por tanto el tiempo total para la recepción del primer dato es $2o + L$. El procesador fuente pueden iniciar la transmisión de un nuevo mensaje a otro procesador en tiempo g. Y así sucesivamente en tiempo $2g$, $3g$, etc., suponiendo que $g \geq o$. De esta forma, varios procesadores actuarán como fuentes de un árbol de emisión y finalizarán de enviar los últimos mensajes a los últimos procesadores.

Como se aprecia en la figura 2.10 el árbol óptimo de emisión es un árbol no balanceado con un número de hijos en cada nodo determinados por los valores de los parámetros o, L y g.

El código de la figura 2.11 muestra una implementación del algoritmo de emisión. Para ello necesitamos construir un árbol de emisión cuyos nodos están etiquetados con el tiempo en el que el mensaje los alcanzará. Se etiqueta la raíz del árbol con el tiempo *0*. En general, un nodo con etiqueta t tiene hijos con etiquetas $t + L + 2*o + j*g$, $j \geq 0$. El árbol de emisión óptimo es el formado con los P nodos con las P menores etiquetas (en caso de empates cualesquiera de los nodos "empatados" vale).

Una vez construido el árbol óptimo con raíz *0*, es posible escribir el procedimiento de "broadcast" de la figura 2.11. Los vectores *father* y *numchildren* y la matriz *child* contienen la información que caracteriza el árbol óptimo. Se asume que todos los procesadores conocen quien es el procesador raíz de la emisión.

```
if (NAME == root) {
  for (i = 0; i < numchildren[0]; i++),
    send x to (child[0][i]+root);
}
else {
  id  = (NAME-root);
  receive x from (father[id]+root);
  for (i = 0; i < numchildren[id]; i++)
  send x to (child[id][i]+root);
}
```

Figura 2.11 Código de una Emisión . Todas las operaciones son módulo P.

Otro ejemplo bien estudiado es el de la suma en árbol. La figura 2.12 ilustra este ejemplo donde la cantidad de datos a sumar por cada nodo depende directamente de *L*, *o* y *g*. La idea principal consiste en que un procesador puede realizar tanto cómputos internos como el tiempo que tarda en llegar un mensaje con los datos sumados desde otro procesador. Claro está que el tiempo de enviar un mensaje depende como hemos visto de los parámetros del modelo y que caracterizan la máquina sobre la que realizamos el experimento.

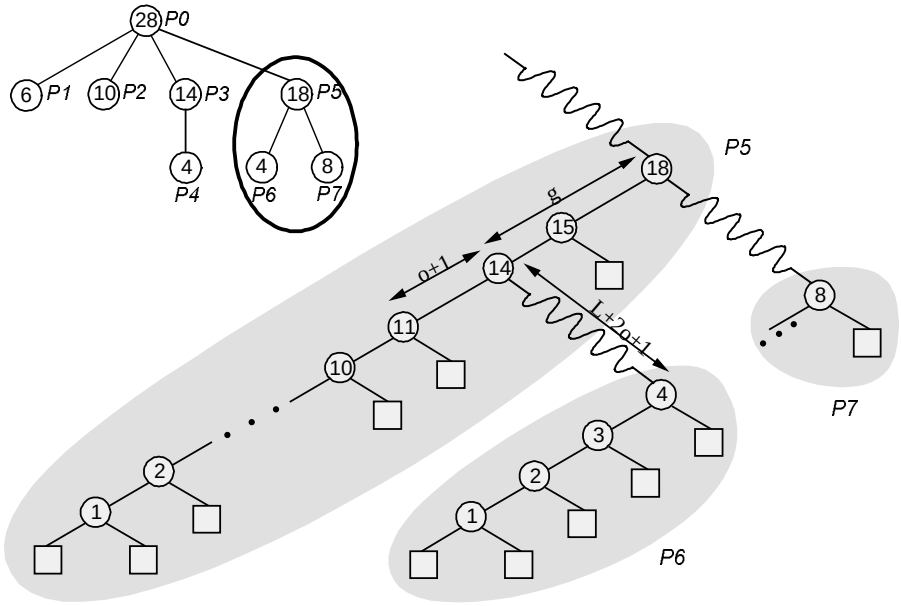

Figura 2.12 Ejemplo de las sumas en árbol. T=28, L=5, g=4, o=2 y P=8.

En este caso, para obtener el algoritmo óptimo para la suma de *N* valores de entradas, consideramos primero cómo sumar el máximo número de valores en un tiempo determinado *T*. El patrón de comunicaciones entre los diferentes procesadores de nuevo forma un árbol con la misma forma que el árbol óptimo de broadcast. Cada procesador debe sumar un conjunto de elementos y luego enviarlos a su procesador padre. Los elementos que se suman en un procesador están desde un principio en la memoria de cada procesador junto con las sumas parciales recibidas de los procesadores colocados en los nodos inferiores del árbol.

Si $T \leq L + 2o$ entonces la solución óptima es sumar los $T + 1$ elementos de forma secuencial en un único procesador, ya que se tardaría demasiado para recibir datos de otro procesador. Si $T \geq L + 2o$ entonces el último paso realizado por el procesador raíz, en tiempo *T-1*, es sumar el valor que ha calculado localmente con los valores que ha recibido del procesador anterior. Este procesador tuvo que haber enviado los datos al raíz en tiempo $(T - 1 - L - 2o)$. Si continuamos desarrollando recursivamente, obtendremos el árbol de sumas correspondiente. Como el procesador raíz puede recibir un mensaje cada g unidades de tiempo, sus procesadores hijos tuvieron que haber terminado sus sumas en tiempos $T - (2 o + L + 1)$, $T - (2 o + L + 1 + g)$, $T - (2 o + L + 1 + 2 g)$, $T - (2 o + L + 1 + 3 g)$,..... . El procesador raíz puede realizar las primeras sumas y luego $(g - o - 1)$ sumas de datos locales entre cada llegada de mensaje. Este esquema de comunicaciones debe modificarse teniendo en cuenta la siguiente característica: debido a que un procesador invierte o ciclos en recibir una suma parcial de otro procesador, todas las sumas parciales transmitidas deben representar como mínimo o sumas. Basándonos en esta idea, se puede obtener de forma directa el número de valores de entrada que deben estar distribuidos en cada procesador.

En la figura 2.13 se muestra un ejemplo del árbol de sumas, con sus cómputos y comunicaciones. Cada nodo está etiquetado con el tiempo de su cómputo, las líneas curvas indican sumas parciales enviadas entre procesadores y las cajas cuadradas representan los valores de entrada. El trabajo inicial de cada procesador queda definido por los enlaces rectos que unen los nodos de sumas parciales. Si un procesador no es un nodo hoja en el árbol de comunicaciones, recibirá sucesivamente una valor que sumará a lo que él ha computado y realizará $(g - o - 1)$ sumas de valores. De nuevo aparece un solapamiento entre el cómputo local y la llegada de mensajes. El *overhead* del procesador receptor comienza tan pronto como el mensaje ha llegado.

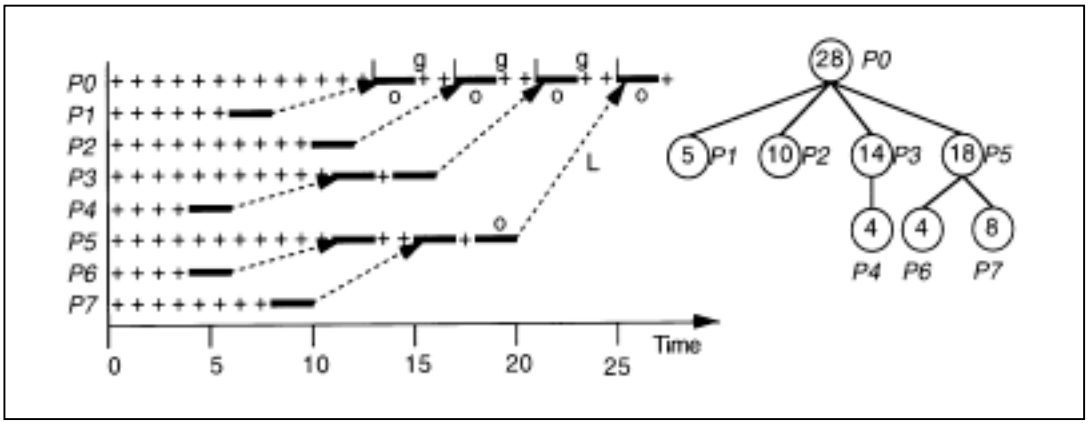

Figura 2.13. Esquema de comunicaciones del problema de las sumas.

Un tercer ejemplo es la reducción Todos a Todos. Sea + una operación binaria conmutativa y asociativa. Cada procesador aporta un operando x. Se trata de obtener el resultado $x_0+...+x_{P-1}$ en todos los procesadores. Supondremos que el tiempo requerido para realizar la operación + es despreciable. Como consecuencia de esta hipótesis, no se puede sacar ventaja de solapar los tiempos de operar con los tiempos de comunicaciones y el "*overhead*" no juega ningún papel en este algoritmo. Podemos simplificar notaciones y denotar por L al valor $L+2o$. Veremos que la reducción AllToAll puede realizarse en el mismo tiempo que un Broadcast.

Denotemos por *f(t)* el máximo número de procesadores que pueden alcanzarse en un "broadcast" de un único elemento. Es claro que *f(t) = 1* para valores de *t* menores que *L+2o*, pues este tiempo es necesario para la llegada del primer mensaje. Para simplificar el razonamiento supondremos que *L+2o* es múltiplo de *g*. En tal caso, los mensajes siempre llegan en tiempos *t* múltiplos de *g*. Supongamos entonces que *t* es múltiplo de *g*. Razonando inductivamente está claro que los procesadores "alcanzados" en tiempo *t* se dividen en dos subconjuntos: los que son alcanzados exactamente en el instante *t* y aquellos que fueron alcanzados anteriormente. Los primeros debieron ser receptores de un mensaje transmitido por un procesador "alcanzado" *L+2o* unidades de tiempo antes. Se deduce que hay *f(t-L-2o)* procesadores en esta categoría. Los segundos fueron alcanzados antes del instante *t* y por tanto lo fueron no más tarde de *t-g*. Son *f(t-g)* nodos. Por tanto.

$$f(t) = f(t-g)+f(t-L-2o) \qquad (2.2)$$

que es una secuencia de tipo Fibonacci. Sea *T[P]* el tiempo mínimo para que el algoritmo de emisión alcance *P* procesadores. El valor *T[P]* puede obtenerse de la tabla *f* fácilmente. Por ej. *T[8]= 5* para el caso de la Figura 2.13.

t =	0	1	2	3	4	5
f =	1	1	2	3	5	8

Figura 2.14. Valores de *f(t)* para el problema. L=2, g=1, o=0 y P=8.

```
for(j= 1; j < L; j ++)
  send x to NAME + j;
for (j = L; j <= T[P]- L; j++) {
  receive y from NAME-f[j-1 ];
  x +=  y;
  send x to NAME + f[j+L-1];
}
for(j = T[P]-L+1;j < T[P];j++) {
  receive y from NAME-f[j-1 ];
  x + = y;
}
```

Figura 2.15 Reducción Todos a Todos.

El algoritmo de reducción AllToAll se escribe en términos de la tabla *f*. En los intervalos *j* anteriores a *L = L+2o* cada procesador *NAME* envía su valor a (*NAME+j)%P*. Después comienzan a llegar valores que son combinados y enviados al procesador (*NAME+f[j+L-1])%P*. En los últimos *L* intervalos se reciben los valores restantes.

P	*s*	*y*	*s*	*y*	*s*	*y*	*s*	*y*	*r*	*y*	*r*	*y*
0	*1*	*x0*	*2*	*x0*	*3*	*x7+x0*	*5*	*x6+x7+x0*	*5*	*x4+x5+x6+x7+x0*	*3*	*x1+...+x0*
1	*2*	*x1*	*3*	*x1*	*4*	*x0+x1*	*6*	*x7+x0+x1*	*6*	*x5+x6+x7+x0+x1*	*4*	*x2+...+x1*
2	*3*	*x2*	*4*	*x2*	*5*	*x1+x2*	*7*	*x0+x1+x2*	*7*	*x6+x7+x0+x1+x2*	*5*	*x3+...+x2*
3	*4*	*x3*	*5*	*x3*	*6*	*x2+x3*	*0*	*x1+x2+x3*	*0*	*x7+x0+x1+x2+x3*	*6*	*x4+...+x3*
4	*5*	*x4*	*6*	*x4*	*7*	*x3+x4*	*1*	*x2+x3+x4*	*1*	*x0+x1+x2+x3+x4*	*7*	*x5+...+x4*
5	*6*	*x5*	*7*	*x5*	*0*	*x4+x5*	*2*	*x3+x4+x5*	*2*	*x1+x2+x3+x4+x5*	*0*	*x6+...+x5*
6	*7*	*x6*	*0*	*x6*	*1*	*x5+x6*	*3*	*x4+x5+x6*	*3*	*x2+x3+x4+x5+x6*	*1*	*x7+...+x6*
7	*0*	*x7*	*1*	*x7*	*2*	*x6+x7*	*4*	*x5+x6+x7*	*4*	*x3+x4+x5+x6+x7*	*2*	*x0+...+x7*
t	*0*	*0*	*1*	*1*	*2*	*2*	*3*	*3*	*4*	*4*	*5*	*5*

Figura 2.16 Traza del algoritmo.

La Figura 2.16 muestra la traza del algoritmo para el ejemplo de la figura 2.14. La primera columna, etiquetada *P* contiene los nombres de procesador. La fila *t* indica el tiempo(valor de *j* en el bucle). Las columnas *s* contienen el nombre del procesador al que se realiza el presente envío. Análogamente, las columnas *r* muestran el número de procesador desde el que se espera recibir el mensaje. Las columnas *y* presentan el valor acumulado. Obsérvese que en el instante *j* el procesador *NAME* tiene el valor

$$x[NAME\text{-}f[j]+1]+x[NAME\text{-}f[j]+1]+\ldots+x[NAME\text{-}1]+x[NAME] \quad (2.3)$$

cuando $j = T$ resulta $f[T] = P$ y por tanto,

$$NAME\text{-}f[j]+1 = NAME\text{-}P+1 = NAME-1 \quad (2.4)$$

(donde las operaciones son en aritmética módulo *P*). La combinación total está en todos los procesadores.

Otros ejemplos encontrados en la literatura son el algoritmo de emisión de *k* datos (k-items Broadcast Problem), la emisión de un dato a todos los procesadores (All-To-All Broadcast Problem) y la combinación de resultados por operaciones de emisión. Todos estos problemas se estudian en [Kar92], donde realizan consideraciones específicas como la re-asignación de los datos, para obtener los resultados eficientes.

Otros casos aparecen en [Cul93] y [Dus96] donde se abordan problemas de aplicaciones reales: la FFT y la ordenación. En el primer caso se justifica la realización de asignaciones cíclicas y por bloques a procesadores para obtener un esquema de las comunicaciones más eficiente sin variar la cantidad de cómputo a realizar. La ventaja de esta re-ordenación de los datos en los procesadores es realmente original y provoca un menor número de comunicaciones. Pero al aplicar el modelo, la transmisión de un bloque de datos de un procesador corresponde a una latencia más el máximo del gap *g* y dos veces el *overhead* más un cierto factor predeterminado. Resulta cuanto menos curioso, que no se aplique la característica más importante del modelo: el solapamiento. Al realizar el análisis cuantitativo aplican una función lineal para las comunicaciones. Este es un ejemplo claro de cómo puede simplificarse el modelo LogP para poder ser aplicado a problemas algo más complejos que los estudiados anteriormente.

2.3 EL MODELO C^3

El modelo C^3 propuesto por Hambrush y Khokhar [Ham96] evalúa para una máquina y algoritmo dados, la complejidad de la ***C***omputación, los patrones de ***C***omunicaciones y la ***C***ongestión debida a las operaciones de comunicaciones. A través de estos tres parámetros los autores argumentan su validez para desarrollar algoritmos que posean una relación grande de cómputo-comunicaciones. A diferencia del modelo LogP, el modelo C^3 se caracteriza porque las comunicaciones dependen de la congestión debida a la cantidad de comunicaciones a realizar, del esquema de encaminamiento y protocolos utilizados, aparte de la cantidad de datos recibidos por un procesador y de la latencia de los mensajes.

En este modelo la computación queda sincronizada a través de una barrera de sincronismo similar a la del modelo BSP. Los algoritmos se dividen en etapas, y cada etapa conlleva una fase de computación seguida de los envíos y recepciones de los mensajes. El rendimiento de una etapa, o del programa global, se establece en términos de unidades de cómputo y comunicaciones. El número de unidades de cómputo se mide en

relación con la cantidad de cómputo realizado en cada etapa. El número de unidades de comunicaciones contabilizadas dependerá de la cantidad de datos recibidos por un procesador, de la latencia de los mensajes, de la congestión debida a la cantidad de comunicaciones a realizar y del encaminamiento y protocolos utilizados.

Un mensaje se compone de paquetes. Un paquete es la unidad de comunicación lógica entre dos procesadores. Denotaremos por l la longitud del paquete (expresada en bytes), s denota el coste de arranque o "set-up" del mensaje y h denota la latencia. Para contabilizar la latencia se utiliza la distancia promedio. En lo que sigue se asume que el ancho de banda de inyección del procesador y el ancho de banda del canal es r.

Numerosas aplicaciones contienen partes en las cuales los p procesadores se dividen en q conjuntos disjuntos $S1,...,Sk$, tal que Si contiene p_i procesadores y los envíos y recepciones sólo ocurren entre procesadores en el mismo conjunto. Se asume que cada conjunto S_i se corresponde a una versión reducida de la máquina. Si el "hardware" y el "software" permiten la ejecución de operaciones dentro de S_i, que sean independientes de las operaciones en otras submáquinas, la eficiencia de las operaciones de "routing" aumenta. La importancia de la capacidad de operar en submáquinas independientes ha sido incorporada en algunos estándares como MPI. En la evaluación de las comunicaciones con el modelo C^3 se asume que es posible hacer "routing" independiente en las diferentes submáquinas.

2.3.1 Unidades de Computación

El rendimiento bajo el modelo C^3 queda determinado por unidades de computación y unidades de comunicación. Supongamos que el procesador P_i accede a t_i octetos en una etapa. Esta etapa se carga con un coste de $max_{i=0,p-1}\ \{ceil(\ t_i\ /\ r\ )\}$ unidades de computación. La razón para normalizar las unidades de computación por r es que una computación excesivamente pequeña, forzosamente tiene un impacto negativo en el rendimiento. Así, si $t_i < r$, se carga con una unidad de computación y se penaliza el hecho de no haber accedido a un número suficiente de bytes para llenar un paquete.

2.3.2 Unidades de Comunicación

En el modelo C^3 las unidades de comunicación con las que se carga una etapa reflejan el tiempo que se va en el envío y recepción de los mensajes, el tiempo que los mensajes están en ruta, la cantidad de congestión que podría ocurrir y una estimación del retraso resultante. El número de unidades cargadas también depende del esquema de "routing" y del tipo de primitivas de envío y recepción utilizadas. Los dos esquemas que se consideran en el modelo C^3 son “store-and-forward” y “wormhole” routing. La mayoría de las máquinas actuales permiten tanto protocolos con bloqueo como sin bloqueo para los envíos y las recepciones. Estos protocolos difieren en los métodos de sincronización utilizados. El modelo C^3 diferencia entre “nonblocking receives” y, “blocking” y “non-blocking sends”.

Un “blocking send” se define en el modelo C^3 como una operación de envío iniciada por un procesador fuente, la cual no termina hasta que el mensaje es recibido en el procesador de destino. En un envío "non-blocking send" el procesador fuente, una vez

ha colocado los datos en su *buffer* de envío tiene que esperar sólo hasta que el mensaje este seguro en ese almacenamiento temporal.

Supongamos que un procesador P_i envía un mensaje de $L_{i,j}$ octetos al procesador Pj, tal que $0 \leq i, j \leq p\text{-}1$. Este mensaje consume un tiempo de envío $s_{i,j}$, el cual representa una estimación del tiempo que necesita el mensaje cuando no existe congestión. Cuando se utilizan "non-blocking sends", "non-blocking receives" y encaminamiento "store-and-forward" el tiempo de la comunicación es:

$$s + ceil(L_{i,j} / r) * h \qquad (2.5)$$

y para encaminamiento "worm-hole routing":

$$s + h + ceil(L_{i,j} / r) \qquad (2.6)$$

Para "blocking sends" y "nonblocking receives" con "store and forward" el tiempo se incrementa:

$$s + h + ceil(L_{i,j} / r) * h + s + h \qquad (2.7)$$

y con "worm-hole routing":

$$s + h + ceil(L_{i,j} / r) + h + s + h \qquad (2.8)$$

Sean $n_s(i)$ y $n_r(i)$ el número de procesadores a los que P_i envía y de los que recibe un mensaje respectivamente. El tiempo de envío y recepción total S_i y R_i para el procesador P_i en un superpaso representa una cota del tiempo de envío y recepción en un entorno libre de congestión.

Para "worm-hole routing", "nonblocking send" y "nonblocking receive" el tiempo de envío es:

$$S_i = s*n_s(i)+h+\sum_{j=0,\, p-1} ceil(L_{i,\,j} / r) \qquad (2.9)$$

Y el tiempo de recepción:

$$R_i = \sum_{j=0,\, p-1} ceil(L_{j,i} / r) \qquad (2.10)$$

La cantidad S_i+R_i nos da una cota del tiempo invertido por el procesador en el envío y recepción de mensajes. Al cargar la etapa con

$$max_{i\,=0,\, p-1} \{ S_i+R_i \} \qquad (2.11)$$

el modelo refleja el tiempo total experimentado por la máquina en envíos y recepciones, sin incluir los retrasos causados por la congestión en los enlaces y en los procesadores.

La congestión es difícil de evaluar y tiene mucha influencia en el tiempo requerido para completar un encaminamiento. Es un fenómeno global que depende de factores como la topología, la cantidad de datos inyectados en la red y de los caminos tomados por los mensajes. En el modelo C^3 se distingue entre congestión en los enlaces C_l y congestión en los procesadores C_p. Para simplificar el estudio, se supone que todos los

mensajes se inyectan simultáneamente. Los parámetros utilizados para medir la congestión son:

- *p*, el número de procesadores.
- *cong*, el número total de parejas de procesadores que se comunican.
- *b*, el ancho de banda de bisección de la máquina, y
- L_a, el número promedio de paquetes comunicados entre procesadores.

La congestión en los enlaces está estrechamente relacionada con el ancho de bisección de la máquina. En una máquina con un ancho de bisección *b*, enviar *k* paquetes desde un procesador en una mitad a los procesadores en la otra mitad, conlleva *ceil(k / b)* pasos. Definimos la congestión sobre los enlaces por la siguiente fórmula:

$$C_l = L_a * ceil(cong / b) \qquad (2.12)$$

La idea es que en el peor caso, los procesadores fuente de las *cong* parejas que se comunican están en un lado del cuello determinado por el ancho de bisección y los procesadores destino están al otro lado del cuello.

Definimos la congestión sobre los procesadores por la siguiente fórmula:

$$C_p = L_a * ceil(cong / p) * H \qquad (2.13)$$

La cantidad *ceil(cong/p)* representa el número promedio de mensajes por procesador al comienzo de las comunicaciones. Se utiliza L_a, la longitud promedio del mensaje, para estimar la pérdida de velocidad que experimenta el mensaje. La idea es que un mensaje de tamaño L_a que está atravesando *H* enlaces y por lo tanto, está compitiendo por los recursos con otros mensajes en cada uno de los *H-1* nodos intermedios se ve frenado por un factor *ceil(cong/p)* en cada procesador.

El número total de unidades de comunicación con las que se carga una etapa en el modelo C^3 viene dado por:

$$\boldsymbol{max}_{i=0,p-1} \{ S_i + R_i \} + C_l + C_p \qquad (2.14)$$

Por supuesto, con vistas a calcular el tiempo real de ejecución es necesario ponderar con pesos relativos las unidades de comunicación y cómputo. Estos pesos deberán estar basados en la velocidad del reloj del procesador y la velocidad del reloj de la red así como en el ancho de banda de los procesadores.

2.3.3 Ejemplos

El modelo C^3 es especialmente adecuado para el diseño de primitivas de comunicación. En los ejemplos que se presentan consideraremos la técnica de encaminamiento "*worm-hole*" con "non-blocking sends" y "non-blocking receives".

PATRÓN DE COMUNICACIONES EXCHANGE

Consideremos un patrón de comunicaciones Exchange, en el que cada procesador envía el mismo número *L* de octetos a un destino diferente. En este caso

$$n_s(i) = 1\ ,\ n_r(i) = 1,\ \ 0 \le i \le p\text{-}1;\ \ cong = p \text{ y } L_a = ceil(\ L\ /\ r\ );\qquad (2.15)$$

sustituyendo en *(2.9)* y *(2.10)*, tenemos:

$$S_i = s + h + ceil(\ L\ /\ r\ )\ ;\ \ R_i = ceil(\ L\ /\ r\ );\qquad (2.16)$$

del mismo modo, de *(2.12)* y *(2.13)* se obtiene:

$$C_l = ceil(\ L\ /\ r\ )\ *\ ceil(\ p\ /\ b\ )\ ;\ C_p = ceil(\ L\ /\ r\ )\ *\ h\qquad (2.17)$$

Que sustituidas en (*2.14*) nos dan el tiempo invertido:

$$s+h+ceil(L/r)+ceil(L/r)+ceil(L/r)*ceil(p/b)+\ ceil(L/r)*h =$$
$$= s+h+ceil(L/r)\{2+ceil(p/b)+h)\}\qquad (2.18)$$

Se observa que, en general, la congestión en los enlaces y en los procesadores es la que domina el tiempo de un Exchange. Por supuesto, esto depende del ancho de bisección de la red particular que se considere. Por ejemplo, en una malla cuadrada con *p* procesadores es, $b = p^{1/2}$ y $h = (2*p^{1/2})/3$. La fórmula (*2.9*) se convierte en:

$$s+(2*p^{1/2})/3+ceil(L/r)\{2+(5*p^{1/2})/3)\}\qquad (2.19)$$

En un hipercubo con p procesadores, $b = p/2$ y $h = log(p)/2$. El tiempo será:

$$s+log(p)/2+ceil(L/r)\{4+log(p)/2)\}\qquad (2.20)$$

En un árbol binario $b = 1$ y $h = log(p)$, el tiempo es:

$$s+log(p)+ceil(L/r)\{2+p+log(p))\}\qquad (2.21)$$

PATRÓN DE COMUNICACIONES UNO A TODOS

En un patrón uno a todos en el que un procesador *t* envía *p-1* mensajes diferentes a diferentes destinos, se tiene $n_s(t) = p\text{-}1$, $n_r(i) = 1$ para $i \neq t$, $0 \le i \le p\text{-}1$, y $cong = p\text{-}1$. El tiempo del procesador *t* domina el del resto.

$$S_i = (p\text{-}1)*(s+ceil(L/r))+h,\ i = t\qquad (2.22)$$
$$R_i = ceil(L/r)\ ,\ i \neq t\qquad (2.23)$$
$$C_l = ceil(L/r)*ceil((p\text{-}1)/b)\qquad (2.24)$$
$$C_p = ceil(L/r)*h\qquad (2.25)$$

El tiempo invertido será:

$$(p\text{-}1)*(s+ceil(L/r))+h\ +ceil(L/r)*\ ceil((p\text{-}1)/b)+ceil(L/r)*h$$
$$= (p\text{-}1)*s+h+\ ceil(L/r)*((p\text{-}1)*(b+1)/b)+h)\qquad (2.26)$$

PATRÓN DE COMUNICACIONES TODOS A UNO

En un patrón todos a uno, todos los procesadores envían a un procesador común, P_t. En este caso tenemos $ns(i) = 1$ para $i \neq t$, $0 \le i \le p\text{-}1$, $nr(t) = p\text{-}1$ y $cong = p\text{-}1$. El tiempo total de recepción en el procesador P_t domina el tiempo de las comunicaciones, quedando:

$$S_i = ceil(L/r)) + h \quad (2.27)$$
$$R_i = (p-1) * ceil(L/r) \quad (2.28)$$
$$C_l = ceil(L/r) * ceil((p-1)/b) \quad (2.29)$$
$$C_p = ceil(L/r)*h \quad (2.30)$$

PATRÓN DE COMUNICACIONES TODOS A TODOS

Por último, estudiamos cuando todos los procesadores envían un mensaje a todos los demás, es decir el patrón todos a todos o exchange total. En este caso, las unidades de comunicaciones quedan definida por $ns(i) = p-1$, $nr(i) = p-1$, para $0 \leq i \leq p-1$, y $cong = p(p-1)$. La fórmula completa de este patrón indica como la congestión del procesador y de los links domina sobre las comunicaciones:

$$S_i = (p-1)*(s+ceil(L/r))+h \quad (2.31)$$
$$R_i = (p-1)*ceil(L/r) \quad (2.32)$$
$$C_l = ceil(L/r)*ceil(p(p-1)/b) \quad (2.33)$$
$$C_p = ceil(L/r)*h*(p-1) \quad (2.34)$$

2.4 EL MODELO DE PATRONES

Las comunicaciones punto a punto entre procesadores suelen describirse como una función lineal del tamaño del mensaje enviado. El modelo LogP contempla además el overhead que invierte el procesador en preparar el mensaje. El modelo C^3 tiene en cuenta además el encaminamiento y la congestión de la red. La necesidad de modelos sencillos que describan correctamente las comunicaciones en aplicaciones reales, nos ha hecho proponer lo que hemos denominado el modelo de Patrones. Este modelo se basa en un conjunto de leyes empíricas que se obtienen a partir de los patrones de comunicaciones más usuales en los algoritmos paralelos.

El modelo de Patrones, al igual que el modelo LogP, contempla el solapamiento entre el cómputo y las comunicaciones, pero ofrece una ventaja importante sobre los modelos LogP y C^3, el análisis de la complejidad de los algoritmos se convierte en abordable. Es un modelo de compromiso entre la eficiencia y la sencillez de uso.

A la hora de considerar los parámetros del modelo se debe tener en cuenta una serie de factores como la latencia, la velocidad de transferencia, el ancho de banda, el ancho de banda total y el ancho de banda asintótico.

Se define la *latencia* como el tiempo necesario para transmitir un mensaje vacío entre dos computadoras. Este tiempo incluye los retrasos para adquirir el canal de comunicaciones y el tiempo invertido en la red. Estos retrasos suelen estar condicionados principalmente por los protocolos software de comunicaciones, por las señales e interrupciones del hardware y por el tiempo de acceso al canal. Cuando intercambiamos mensajes de tamaño pequeño, la latencia es el parámetro a tener en cuenta. La sobrecarga inicial domina los tiempos de transmisión. En este caso, el modelo LogP diferencia en la latencia dos componentes o y L, de sobrecarga en el procesador y de viaje por la red.

La *velocidad de transferencia de datos* indica la velocidad, expresada en octetos por segundo, en que los datos pueden ser transferidos entre dos computadoras una vez comenzada la transmisión.

El *ancho de banda* es el máximo número de octetos por segundo que se puede transmitir por el canal de comunicaciones. El *ancho de banda total* del sistema se define como el número total de octetos que se pueden transferir por la red en un periodo de tiempo determinado. En una red basada en un bus, una vez que comienza la transmisión, el mensaje ocupa el canal de forma exclusiva y el ancho de banda total del sistema coincide con la velocidad de transmisión. Sin embargo, cuando existen diferentes canales de comunicación, múltiples mensajes pueden transmitirse simultáneamente por la red y el ancho de banda total de la red supera la velocidad de transferencia de datos del canal. El *ancho de banda asintótico* es la velocidad límite de transferencia entre dos máquinas de la red al hacer crecer el tamaño del mensaje enviado. Debido a las técnicas de encaminamiento [Ni93] utilizadas en la mayoría de las redes actuales, el ancho de banda asintótico coincide con el ancho de banda.

En la literatura diferentes autores [Sch94], [Don95] modelizan el tiempo que consumen las comunicaciones en sistemas basados en el paso de mensajes como una función lineal en el tamaño del paquete a enviar de la forma:

$$Tiempo\_de\_enviar\_N\_octetos = \alpha + \beta \bullet N + (h-1) \bullet \gamma \qquad (2.35)$$

donde:

- α: es la latencia, entendida como la suma del coste de preparar un mensaje para ser enviado y el coste de tránsito de un paquete de tamaño pequeño desde un origen a un destino. Se suele denominar tiempo de iniciación (*startup*).
- β: es el inverso de la velocidad de transferencia, es decir, el tiempo mínimo que se tarda en enviar 1 octeto desde un origen a un destino una vez comenzada la transmisión.
- N: es el número de octetos que se desean enviar.
- h: es el número de enlaces intermedios entre origen y destino.
- γ: es el retraso debido a los enlaces.

Los parámetros α, β y γ dependen de cada arquitectura y deben ser computados experimentalmente para cada caso. En muchas arquitecturas paralelas, el retraso debido a los enlaces, γ es despreciable debido a las técnicas de encaminamiento y al pequeño diámetro de la red de comunicaciones. Así pues el tercer sumando de *(2.35)* puede desestimarse utilizando la expresión simplificada:

$$Tiempo\_de\_enviar\_N\_octetos = \alpha + \beta \bullet N \qquad (2.36)$$

Esta fórmula es exacta cuando los procesadores están directamente conectados a través de la red. En el caso particular de un bus compartido, si un procesador necesita enviar datos, intenta acceder al bus y los envía a través de él si consigue el acceso. Un número elevado de procesadores intentando acceder al bus hacen que el tiempo de espera por el acceso crezca para cada procesador.

Se puede observar de la fórmula *(2.36)* como los parámetros que realmente influyen en este modelo son dos: la latencia o iniciación y el inverso del ancho de banda.

2.4.1 APLICACIONES DEL MODELO DE PATRONES

2.4.1.1 MULTIPLICACIÓN DE MATRICES.

Un ejemplo bien conocido es la multiplicación de matrices por bloques. El algoritmo consiste en enviar a todos los procesadores la matriz A completa de tamaño $N \times N$. De la otra matriz se enviará a cada procesador sólo el bloque que le corresponda de tamaño $N \times N / P$, siendo P el número de procesadores. Luego se realizan los cómputos y las devoluciones de los trozos de las matrices de los respectivos procesadores al procesador recolector [Rod98a].

Los experimentos se han realizado en dos plataformas hardware diferentes: la IBM SP2 y en una red de estaciones de trabajo conectadas por un bus. La plataforma software en ambos casos ha sido PVM.

En PVM, la operación de envío de datos entre procesadores conlleva tres fases: inicializar el buffer de envío (pvm_initsend()), codificar y empaquetar los datos a enviar (pvm_pk*()) y enviar los datos al destino (pvm_send()). La operación de recepción implica dos fases: recoger los datos en el buffer de recepción (pvm_recv()) y desempaquetarlos (pvm_upk*()). El tiempo invertido en la transmisión de datos se ve afectado por el protocolo de comunicaciones utilizado y por el tipo de codificación.

Los protocolos de comunicaciones admitidos en PVM son dos: datagramas (UDP) y circuito virtual (TCP). Cuando se utilizan datagramas, son los demonios los que llevan el control de las comunicaciones. Cuando un proceso envía un mensaje a otro situado en una máquina diferente, es pasado al demonio local que lo envía al demonio de la máquina con el proceso destino. Por tanto se realizan tres copias de los datos (del proceso local al demonio local, del demonio local al demonio destino y del demonio destino al proceso destino). En el caso de circuito virtual (TCP), la comunicación ocurre directamente entre los procesos, ahorrando dos de las copias. Desafortunadamente este protocolo no es escalable en agrupaciones de estaciones de trabajo debido al reducido número de descriptores de ficheros admitidos por el sistema (típicamente 64 en SUNOS e incluso 32 en otros casos).

Respecto al tipo de codificación de los datos a enviar, existen tres posibilidades: PvmDataDefault, PvmDataRaw y PvmDataInPlace. La primera opción (PvmDataDefault) convierte los datos a la representación XDR, permitiendo la interacción entre máquinas con diferentes arquitecturas. En la segunda opción (PvmDataRaw) no se realiza ninguna codificación de los datos. Esto limita su uso a comunicaciones entre máquinas con la misma arquitectura. Por último, la opción PvmDataInPlace no realiza ni codificación ni copia de los datos a enviar. Se accede a los datos a través de un puntero, ahorrando una copia de los mismos. Esto implica que el usuario no puede modificar los datos entre el tiempo de empaquetado y el tiempo de envío. Debido a que la mayoría de las máquinas instaladas en red suelen corresponder a una misma arquitectura y a la necesidad de garantizar la coherencia de los datos, PvmDataRaw es la codificación más utilizada. Es por ello que haremos especial énfasis en esta codificación.

La estimación de los parámetros α y β dependerá entonces de cada uno de los factores anteriores. Los parámetros α y β son calculados mediante ajuste por mínimos cuadrados a partir de los tiempos reales del paso de mensajes frente a la longitud de los mismos. Experimentalmente, los tiempos para tal ajuste se obtienen utilizando el algoritmo ping-pong *(PP)* que calcula el tiempo invertido en enviar un paquete de tamaño N entre dos procesadores, midiendo el tiempo de ida y vuelta del paquete y dividiéndolo por dos. La figura 2.17 muestra el pseudo código PVM del algoritmo PingPong.

Según el modelo establecido en el apartado anterior, el tiempo de enviar la matriz a todos los procesadores es de orden $P\,(\alpha + \beta Ni)$, donde Ni es el número de octetos de la matriz A ($Ni = N^2$ x longitud en octetos del tipo de datos utilizado). En el envío de las submatrices de B se invierte un tiempo del orden de $P\,(\alpha + \beta Ni/P)$. A continuación, cada procesador realiza el cómputo de las submatrices con coste $O(N^3/P)$ y devuelve al procesador principal la submatriz con un coste de finalmente $(\alpha + \beta Ni/P)$.

Procesador Origen:

tiempo 1
inicializar buffer de envío
empaquetar N octetos
enviar N octetos al destino
esperar por los N octetos
desempaquetar los N octetos
tiempo 2

Total:(tiempo2-tiempo1)/2

Procesador Destino:

recibir N octetos
desempaquetar N octetos recibidos
empaquetar N octetos
enviar N octetos

Figura 2.17 Algoritmo Ping-Pong (PP)

De este modo, la complejidad teórica del algoritmo paralelo viene dada por la ecuación *(2.37)*, donde el último sumando se corresponde con el tiempo de recepción desde el último procesador:

$$P\ (\alpha + \beta Ni) + P\ (\alpha + \beta Ni/P) + D\ N^3 / P + (\alpha + \beta Ni/P) \qquad (2.37)$$

El algoritmo de la multiplicación de matrices por bloques se ha ejecutado sobre una máquina de altas prestaciones y sobre una red de área local de bajas prestaciones. La IBM SP2 consiste en procesadores conectados por una red de comunicaciones de alto rendimiento. En el caso de la red de bajas prestaciones, las máquinas se conectan por medio de un canal compartido: un bus coaxial.

IBM SP2:

Para obtener los resultados del modelo para la IBM SP2, de la fórmula *(2.37)* necesitamos computar los valores de α y β del modelo. Estos valores se han obtenido por ajuste lineal de los tiempos de envíos de paquetes de 1 entero a 1048576 enteros, al ejecutar el algoritmo PingPong anteriormente expuesto. La constante de cómputo D que aparece en la ecuación *(2.37)* se ha obtenido experimentalmente. Su valor es $D = 7{,}30 \times 10^{-7}$.

En la tabla siguiente se muestran los resultados de enviar N bytes entre dos procesadores para la IBM SP2 usando las funciones *pvm_send()* y *pvm_recv()* de PVMe[IBM95], la versión adaptada a las características de PVM.

Bytes	PingPong
4	0,000079
8	0,000079
16	0,000079
32	0,000080
64	0,000079
128	0,000083
256	0,000100
512	0,000112
1024	0,000139
2048	0,000186
4096	0,000272
8192	0,000404
16384	0,000721
32768	0,001353
65536	0,002677
131072	0,005635
262144	0,011088
524288	0,022057
1048576	0,044107
2097152	0,088171
4194304	0,176442

Tabla 2.3 Valores del PingPong

Los valores de α y β se calculan de estos valores por un ajuste de una recta por mínimos cuadrados. Los valores resultantes son 0,000079 para α y 4,20E-8 para β con codificación PvmDataRaw. La figura 2.18 siguiente ilustra claramente el comportamiento de la curva. Se observa como el mejor ajuste para valores grandes a partir de 256 bytes la curva se define perfectamente como una recta. Para valores menores de 256 bytes se muestra que también sigue una conducta lineal. Por tanto, para valores asintóticos se toman los valores de α y β anteriores y en caso de trabajar para valores más reducidos, se debería tomar los valores que den del ajuste entre 4 bytes y 256 bytes. De la intersección que formarían estas dos rectas podemos obtener con gran aproximación el tamaño de paquete de la IBM SP2.

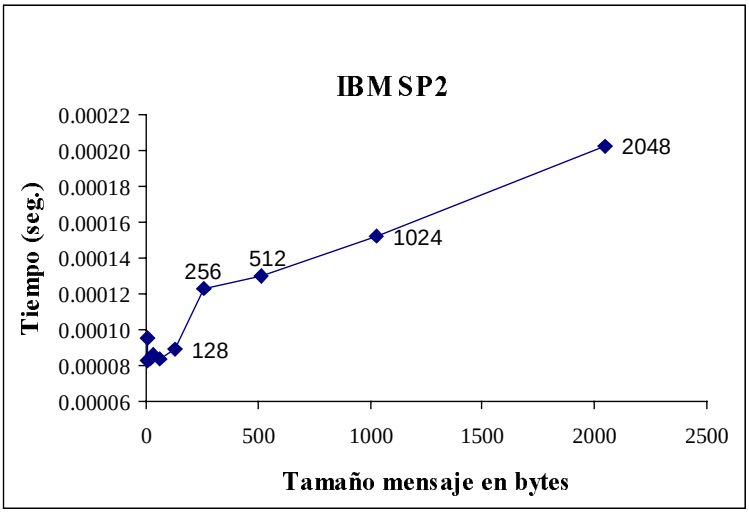

Figura 2.18 Tiempos para la IBM SP2 para tamaños pequeños.

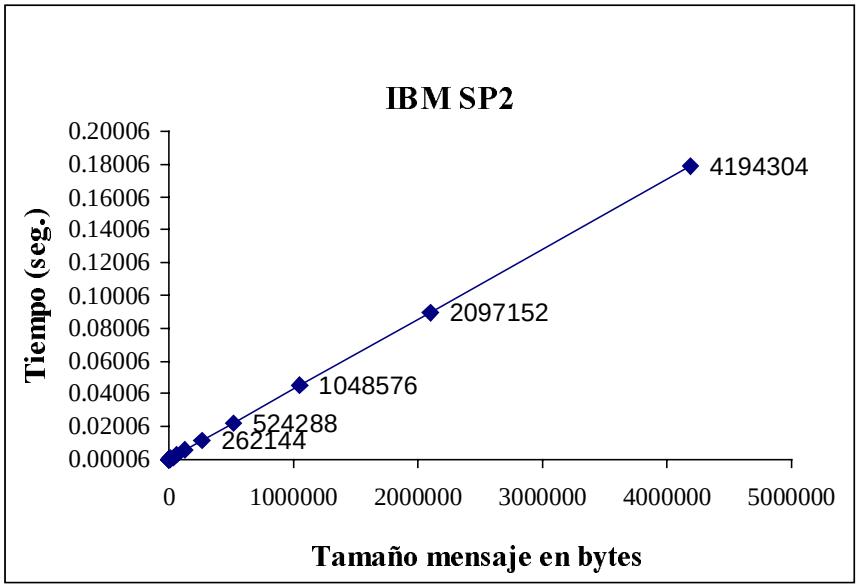

Figura 2.19 Tiempos para la IBM SP2 para tamaños grandes.

Una vez computados α y β, e integrándolos en la fórmula *(2.37)*, se compara con los valores que ofrece el tiempo de ejecución real de la multiplicación de matrices de 840 x 840 enteros con 2, 3, 4, 5, 6 y 7 procesadores. De la tabla 2.4 se comprueba como la predicción es válida con errores máximos de 2,06 %. De la gráfica 2.20 se observa que las curvas prácticamente se solapan demostrando la validez del análisis con el modelo.

	2	3	4	5	6	7
Real DataRaw	218,18	145,88	111,09	87,79	73,36	63,61
Modelo DataRaw	216,76	144,75	108,80	87,28	72,98	62,80
Error (%)	0,65	0,78	2,06	0,58	0,52	1,28

Tabla 2.4 Resultados de la multiplicación de matrices para la IBM SP2.

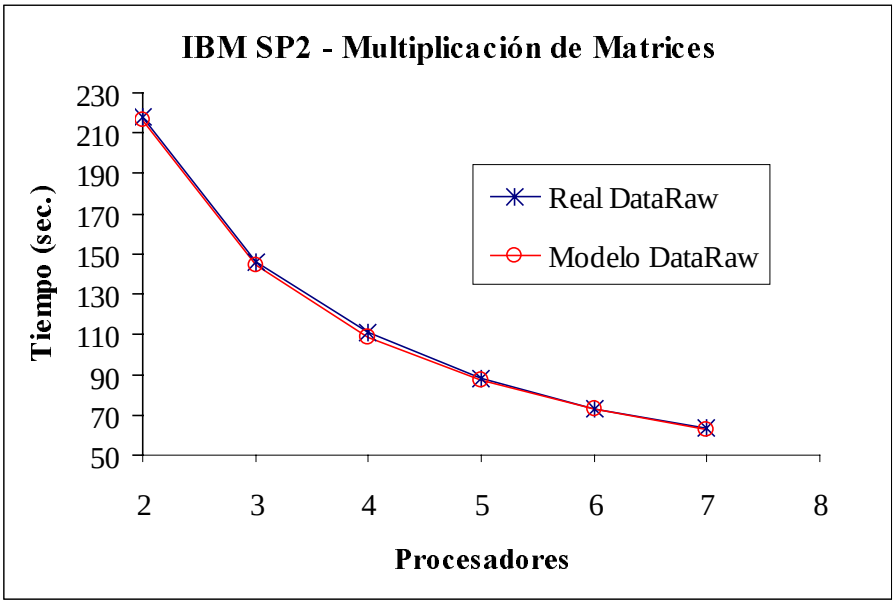

Figura 2.20 Multiplicación de matrices: valores reales y estimados.

RED DE ESTACIONES DE TRABAJO:

En este apartado se describe el modelo de patrones según la ecuación *(2.36)* para una agrupación de estaciones de trabajo conectadas mediante una red tipo bus. Para ser más precisos, el modelo considerado contempla una red de área local Ethernet a 10 Mbits en la que todas las máquinas son homogéneas (Sun Sparc 5) y se encuentran en el mismo segmento. Además se consideran únicamente situaciones estáticas en las que no hay interacción con otros usuarios del sistema, el nivel de tráfico en la red se mantiene constante y bajo y en las máquinas la carga es inferior al 10 %.

Para el desarrollo de la experiencia se ha utilizado el software PVM. Es importante tener conocimiento de su funcionamiento interno para entender el comportamiento de los resultados que se exponen. Recordando la estructura interna de PVM, en el sistema podemos diferenciar dos elementos principales: el demonio y la librería de rutinas. El primero es responsable de la creación de procesos en las diferentes máquinas, de la señalización y del paso de mensajes en los casos necesarios. Reside en todas las máquinas que constituyen la máquina paralela virtual. La librería contiene un repertorio funcionalmente completo de primitivas necesarias para la cooperación entre procesos: paso de mensajes, gestión dinámica de las máquinas, etc.

Como cabría esperar, por motivo de las copias se realizan, los valores obtenidos para el protocolo TCP son inferiores a los obtenidos con el protocolo UDP. En la tabla 2.5 se observa claramente que la proporción del parámetro β respecto a los protocolos es menor cuanto más costosa es la codificación (desde 1,75 veces inferior para PvmDataInPlace, hasta 1,48 para PvmDataDefault). El valor de α se ha obtenido para un tamaño de paquete igual a 4 octetos y sólo presenta diferencias significativas según el protocolo utilizado.

	α–UDP	α–TCP	β–UDP	β–TCP
PvmDataDefault	0,00434	0,00142	$1{,}93 \times 10^{-6}$	$1{,}30 \times 10^{-6}$
PvmDataRaw	0,00347	0,00127	$1{,}63 \times 10^{-6}$	$9{,}60 \times 10^{-7}$
PvmDataInPlace	0,00380	0,00158	$1{,}59 \times 10^{-6}$	$9{,}10 \times 10^{-7}$

Tabla 2.5 Valores de α y β obtenidos con el algoritmo PingPong.

Las figuras 2.21 y 2.22 muestran los valores las curvas de tiempos para tamaños pequeños y grandes respectivamente. El punto de inflexión ocurre para valores superiores de 2048 bytes.

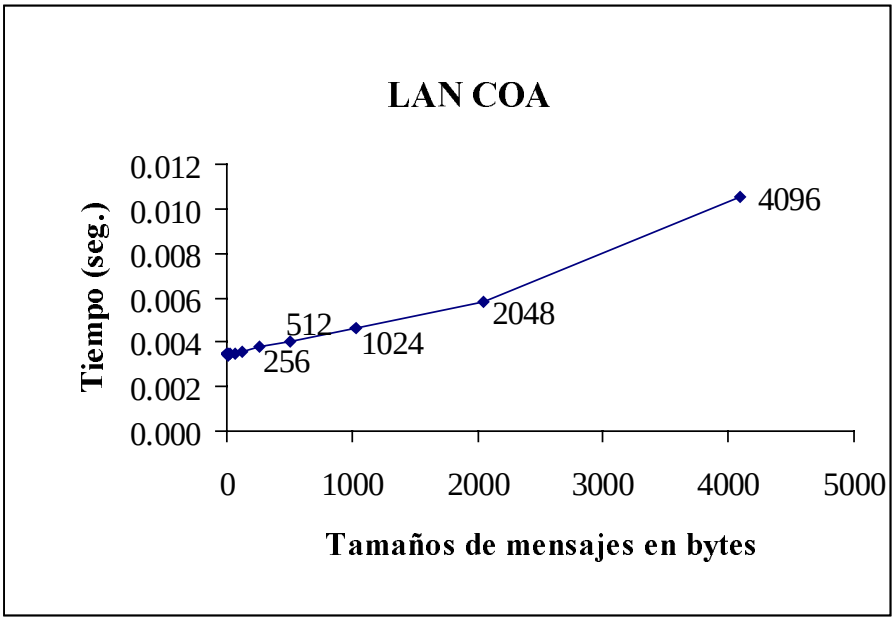

Figura 2.21 Tiempos del PingPong para tamaños pequeños.

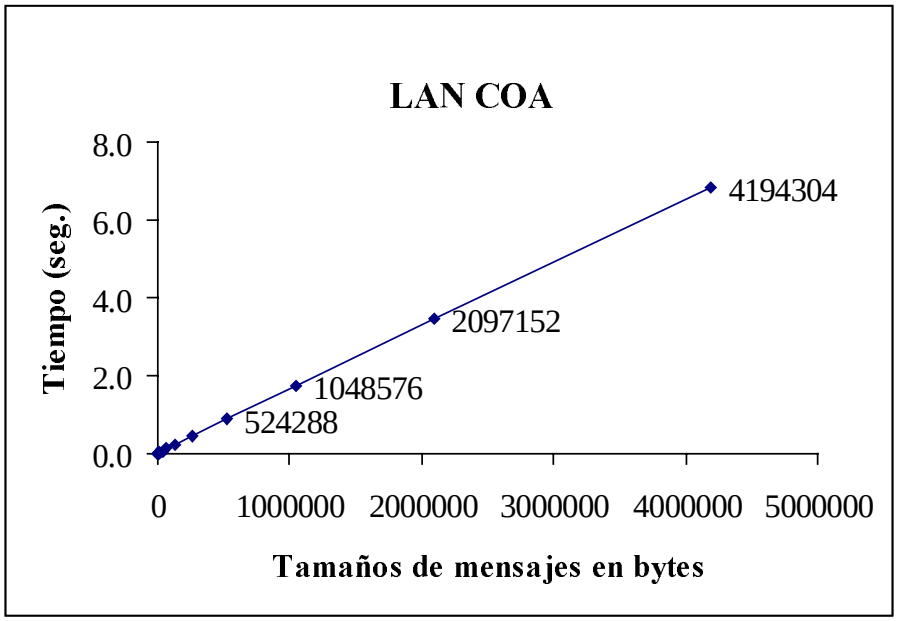

Figura 2.22 Tiempos del PingPong para tamaños grandes.

De nuevo hemos considerado la multiplicación de matrices de tamaño 840 x 840 enteros de 4 octetos y hemos variado el número de máquinas entre 2 y 8. La tabla 2.6 muestran para todas las posibles codificaciones los tiempos reales considerando el protocolo TCP. La tabla muestra también los valores estimados teóricamente según el modelo descrito en la ecuación *(2.37)*. Los valores de α y β utilizados son los mostrados en la tabla 2.3. La constante de cómputo $D = 1{,}28 \times 10^{-6}$ que aparece en la ecuación *(2.37)* se ha obtenido experimentalmente.

	Real TCP			Modelo TCP		
	DataDefault	DataRaw	DataInPlace	DataDefault	DataRaw	DataInPlace
2	399,33	397,29	397,50	392,18	388,82	388,33
3	275,35	272,73	270,27	268,80	264,64	264,03
4	212,64	210,10	207,14	208,94	203,90	203,16
5	177,64	175,30	172,81	174,50	168,55	167,67
6	154,75	151,74	148,50	152,76	145,88	144,87
7	139,51	135,99	133,15	138,28	130,46	129,32
8	128,85	125,16	122,52	128,34	119,58	118,30

Tabla 2.6 Multiplicación de Matrices con el protocolo TCP

La gráfica 2.23 nos permite comprobar los resultados mostrados en la tabla 2.6:

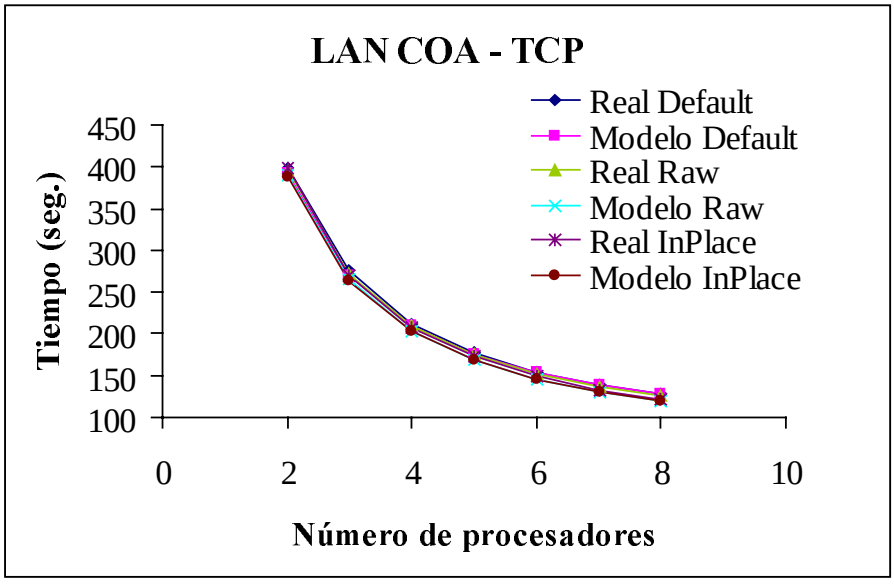

Figura 2.23 Valores estimados y reales de la Multiplicación de matrices.

No sólo el comportamiento en TCP es el predicho sino que los resultados son muy parecidos, con un error de un 4,46% en la codificación PvmDataRaw. El resto de errores se muestran en la tabla 2.7. Este error se ha medido como 100 * (REAL – MODELO) /REAL.

Procesadores	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Error Default	1,79	2,38	1,74	1,77	1,29	0,88	0,40
Error Raw	2,13	2,97	2,95	3,85	3,86	4,06	4,46
Error InPlace	2,31	2,31	1,92	2,97	2,44	2,88	3,45

Tabla 2.7 Errores para LAN TCP

En la tabla 2.8 se muestran los resultados para el caso del protocolo UDP. Se observa que en las columnas etiquetadas “DataRaw” de la tabla 2.8 (protocolo UDP), que el tiempo estimado por el modelo se mantiene inferior al tiempo real hasta 4 procesadores. A partir de ese número se invierte el comportamiento y mientras el tiempo teórico predice un crecimiento, se observa que el tiempo real continua aún decreciendo.

Puesto que el valor de β corresponde a la máxima transferencia de datos alcanzable, los tiempos estimados mediante su uso deberían ser inferiores a los tiempos de ejecución reales. Así pues, la conducta observada no es la esperada y concluimos que el modelo no predice correctamente para el protocolo UDP. Este fenómeno puede observarse para las restantes codificaciones bajo este protocolo.

	Real UDP			Modelo UDP		
	DataDefault	**DataRaw**	**DataInPlace**	**DataDefault**	**DataRaw**	**DataInPlace**
2	405,90	397,10	396,81	398,42	395,45	395,06
3	279,34	274,27	269,78	276,52	272,85	272,36
4	214,35	210,04	209,69	218,30	213,85	213,26
5	179,34	174,02	173,48	185,55	180,29	179,60
6	155,91	150,66	149,93	165,54	159,46	158,65
7	140,07	136,37	135,74	152,80	145,89	144,98
8	128,29	125,26	125,12	144,61	136,87	135,85

Tabla 2.8 Multiplicación de Matrices con el protocolo UDP.

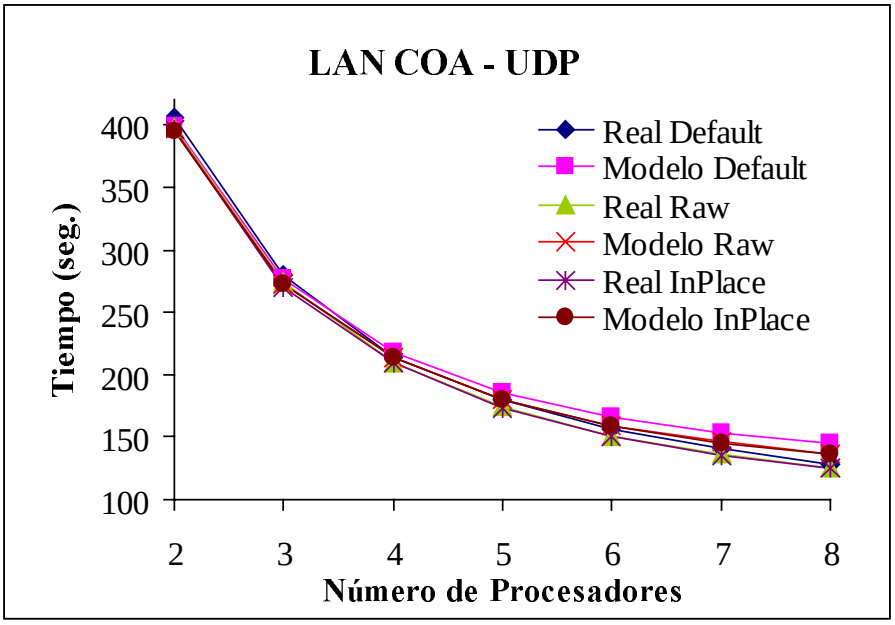

Figura 2.24 Valores estimados y reales de la Multiplicación de matrices.

Los errores calculados en este caso han aumentado y sobre todo el cambio de signo muestra que el modelo da peores tiempos que las ejecuciones reales. Aunque no son excesivamente grandes, cada vez son mayores lo que significa que el modelo se separa del tiempo real con aumento de procesadores.

Procesadores	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Error Default	1,84	1,01	-1,84	-3,46	-6,18	-9,09	-12,72
Error Raw	0,42	0,52	-1,81	-3,61	-5,84	-6,98	-9,27
Error InPlace	0,44	-0,96	-1,70	-3,53	-5,82	-6,81	-8,57

Tabla 2.9 Errores en UDP.

Este comportamiento no es asumible ya que los valores de β se han calculado de forma asintótica y por tanto son los mejores valores que podemos tener.

Un estudio exhaustivo de los cuatro sumandos que intervienen en la fórmula del modelo *(2.37)*, nos permitió concluir que la razón de este fenómeno está en las comunicaciones al enviar las matrices. La tabla 2.10 muestra los resultados de enviar a P procesadores la matriz A usando el protocolo UDP con codificación PvmDataRaw (el resto de los casos de codificación es idéntico). Los valores de la fila etiquetada "Modelo UDP" contienen los tiempos de envío estimados de acuerdo con la fórmula $P(\alpha + \beta Ni)$. Se observa que el tiempo teórico se mantiene por encima del tiempo real y que la separación entre ambos aumenta siguiendo aproximadamente una ley lineal en el número de procesadores. Este comportamiento es anómalo ya que los tiempos teóricos no deberían estar por encima de los tiempos reales porque β es un valor asintótico. Por tanto, el modelo para predecir el comportamiento de las comunicaciones en patrones de envío uno a muchos no es correcto en el protocolo UDP.

	2	**3**	**4**	**5**	**6**	**7**	**8**
Modelo UDP	9,21	13,81	18,42	23,02	27,62	32,23	36,83
Real UDP	7,44	10,75	13,12	15,99	19	23,04	26,12

Tabla 2.10 Tiempos de envío de la matriz *A*.

La figura 2.25 muestra los valores de la tabla 2.10 donde se aprecia claramente que el modelo queda por encima del tiempo real.

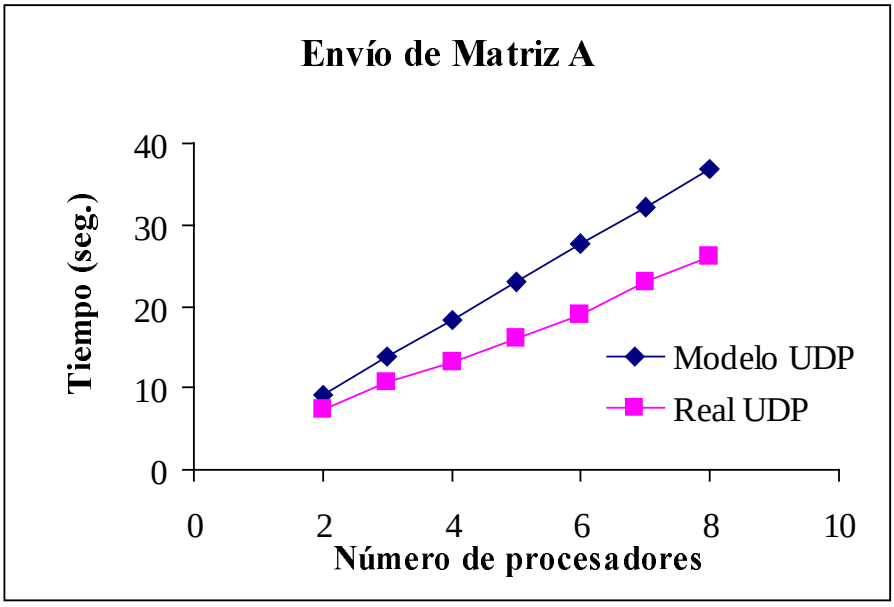

Figura 2.25 Tiempo modelo superior al tiempo real.

Es un hecho conocido que cuando se utiliza el protocolo UDP, los demonios de las distintas máquinas trabajan en paralelo. En el caso del experimento ping-pong el patrón de comunicaciones utilizado es uno a uno, lo que asume una serialización entre el cómputo y las comunicaciones. Por el contrario, en las comunicaciones uno a muchos el

tiempo de cómputo puede quedar solapado con el tiempo de las comunicaciones. Al no intervenir los demonios al utilizar TCP, este solapamiento desaparece.

Proponemos una experiencia alternativa que permite contemplar el paralelismo introducido por los demonios y estimar los parámetros α y β. En el experimento broadcast (BR) (figura 2.26), se plantea un patrón de comunicaciones uno a todos [Kum94]. El procesador origen envía un mensaje de tamaño N a cada uno de los procesadores destino. Bajo el protocolo UDP, las fases de empaquetado y envío en el procesador origen (líneas 1 a 5 de la figura 2.26), quedan solapadas con la recepción en los distintos demonios de las máquinas destino. El efecto es parecido al del modelo LogP. Existe un comportamiento de solape para este patrón de comunicaciones.

```
Procesador Origen:
        tiempo 1
1:      para cada procesador i
2:              inicializar buffer de envío
3:              empaquetar N octetos
4:              enviar N octetos al destino i
5:      fin para
6:      esperar por ACK del último
7:      desempaquetar ACK
        tiempo 2
        Total:(tiempo2-tiempo1)

Procesador Destino i:
        recibir N octetos
        desempaquetar N octetos recibidos
        si último procesador
                empaquetar ACK
                enviar ACK octetos
        fin si
```

Figura 2.26 Algoritmo Broadcast (BR)

La nueva experiencia se ha realizado 100 veces con valores de N comprendidos entre 4 y 1048576 octetos. La tabla 2.11 muestra los nuevos valores estimados para α y β. Puede observarse como los valores de α y β son mejores con respecto a los obtenidos en el experimento PingPong. Lógicamente, el valor de α sigue presentando dependencia respecto al tipo de protocolo, sin embargo, el valor de β es ahora similar con ambos protocolos. Se hace necesario para UDP distinguir entre estos nuevos valores para la latencia y la velocidad de transferencia de los anteriores.

Utilizaremos la notación α_{PP} y β_{PP} para los valores obtenidos mediante el algoritmo ping-pong y α_{BR} y β_{BR} para los valores correspondientes al algoritmo broadcast descrito en la figura 2.26. Mientras que para UDP el cociente β_{PP} / β_{BR} es de 1.50, el correspondiente cociente en TCP es de 0.92, por lo que esta distinción no es necesaria para el protocolo TCP. En [Rod96] se describe este experimento y otros que ofrecen

características importantes a tener en cuenta en las redes de área local tipo bus coaxial. Por ejemplo, la influencia distribuir los procesadores en diferentes segmentos para evitar las colisiones.

	α_{BR}		β_{BR}	
	UDP	**TCP**	**UDP**	**TCP**
PvmDataDefault	0,00236	0,00103	$1,19 \times 10^{-6}$	$1,16 \times 10^{-6}$
PvmDataRaw	0,00234	0,00102	$1,08 \times 10^{-6}$	$1,05 \times 10^{-6}$
PvmDataInPlace	0,00286	0,00148	$9,82 \times 10^{-7}$	$8,98 \times 10^{-7}$

Tabla 2.11 Nuevos valores de α y β para comunicaciones de uno a muchos.

Aplicando los nuevos valores de los parámetros del modelo, la fórmula *(2.37)* para el algoritmo paralelo de la multiplicación de matrices se reescribe de la siguiente forma:

$$P\ (\alpha_{BR} + \beta_{BR}\, Ni) + P\ (\alpha_{BR} + \beta_{BR}\, Ni/P\ ) + D\, N^3 / P + (\alpha_{PP} + \beta_{PP}\, Ni/P\ ) \qquad (2.38)$$

La tabla 2.12 muestra para todas las posibles codificaciones los tiempos reales considerando el protocolo UDP y los valores estimados según la ecuación *(2.38)*. Omitimos la tabla para el protocolo TCP. La similitud de los valores de α y β para el protocolo TCP para ambos patrones de comunicaciones (ping-pong y broadcast) hacen que los tiempos del modelo no difieran substancialmente de los obtenidos en la sección anterior.

	Real UDP			**Modelo UDP**		
	DataDefault	**DataRaw**	**DataInPlace**	**DataDefault**	**DataRaw**	**DataInPlace**
2	405,90	397,10	396,81	392,14	390,01	387,66
3	279,34	274,27	269,78	268,14	266,11	264,00
4	214,35	210,04	209,69	207,83	205,69	203,56
5	179,34	174,02	173,48	172,98	170,66	168,40
6	155,91	150,66	149,93	150,87	148,32	145,89
7	140,07	136,37	135,74	136,04	133,24	130,61
8	128,29	125,26	125,12	125,76	122,69	119,84

Tabla 2.12 Multiplicación de Matrices con el protocolo UDP con α_{BR} y β_{BR}.

El comportamiento anómalo desaparece al utilizar la fórmula *(2.38)*. El tiempo estimado por el modelo se conserva por debajo del tiempo real, independientemente del número de procesadores. La diferencia entre el tiempo de ejecución y el del modelo disminuye con respecto al anterior modelo *(2.37)*.

La tabla 2.13 muestra los errores en las tres codificaciones. El máximo se alcanza en 4,25% además de que ningún signo negativo aparece con lo cual aseguramos que para todos los casos, el modelo se mantiene por debajo del tiempo real. La gráfica 2.27 muestra como la curva del modelo queda por debajo de la curva real.

Procesadores	2	3	4	5	6	7	8
Error Default	3,65	4,25	3,28	3,77	3,44	3,07	2,16
Error Raw	1,79	2,97	2,07	1,93	1,55	2,30	2,05
Error InPlace	1,96	1,80	2,60	2,61	2,39	3,50	3,95

Tabla 2.13 Errores con el nuevo modelo.

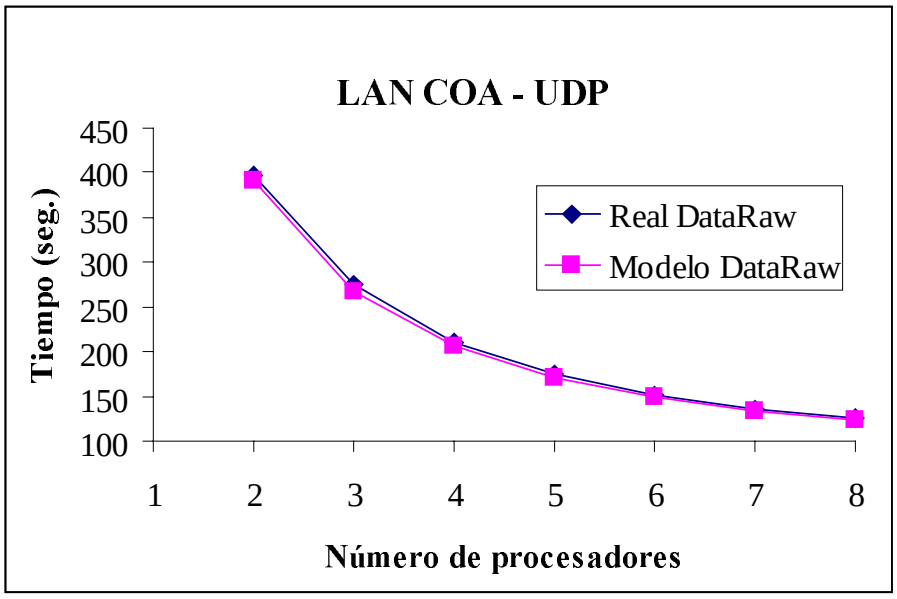

Figura 2.27 Valores estimados y reales de la Multiplicación de matrices.

Hemos descubierto que, en redes de área local bajo PVM, los parámetros proporcionados por el algoritmo ping-pong clásico no proporcionan un modelo adecuado para la estimación del rendimiento de las comunicaciones uno a muchos. Proponemos un experimento alternativo (broadcast) que permite estimar correctamente nuevos valores para tales parámetros. Estos nuevos valores reflejan el solapamiento entre el cómputo y las comunicaciones que aparece en estos patrones de comunicación. La detección de este fenómeno y la determinación de los nuevos parámetros facilita el diseño de algoritmos paralelos al aumentar la eficacia de predicción del modelo de patrones. Comprobamos experimentalmente que la propuesta que hacemos es válida, contrastándola con un algoritmo paralelo de multiplicación de matrices basado en el paradigma Maestro-Esclavo.

2.4.1.2 Transformada Rápida de Fourier

Otro ejemplo ampliamente utilizado es la paralelización de la Transformada Rápida de Fourier. En este caso las comunicaciones se realizan con patrones injectivos donde intervienen un número par de procesadores cada vez que ocurre una comunicación.

En este apartado introducimos una fórmula computacional para patrones injectivos que cubre tanto las multicomputadoras como las redes de área local. Los resultados computacionales que se muestran prueban la validez del modelo en ambos tipos de

arquitecturas. El modelo queda caracterizado por unos pocos parámetros bien determinados y fáciles de aplicar [Rod97].

Los experimentos fueron llevados a cabo en una red de área local tipo bus con coaxial y en la IBM SP2. Como en el caso anterior de la multiplicación de matrices aplicamos una función lineal [Sch94], [Don95], [Rod96] para estimar el coste de las comunicaciones basadas en paso de mensajes.

Esta fórmula es exacta cuando los procesadores están conectados directamente o con una red de altas prestaciones. Pero en el caso de redes más lentas aparecen fenómenos necesarios de estudiar. No conocemos estudios que resuelvan este tipo de cuestiones.

Existen muchos algoritmos que utilizan el patrón injectivo para sus comunicaciones. Por ejemplo, estos patrones aparecen en las fases de división y combinación de algoritmos del tipo divide y vencerás [Rodri97] [Alm95]. El algoritmo paralelo para resolver la FFT nos ilustrará de forma práctica nuestro objetivo.

Un patrón injectivo se caracteriza por la existencia de un subconjunto S de procesadores de un conjunto total H, tal que cada procesador r de S envía un mensaje a un procesador diferente $d(r)$ de H. Decimos que un patrón injectivo es por parejas cuando la función de destinos es involutiva: $d(d(r)) = r$ para cualquier procesador r en S.

Paralelización de la Transformada Rápida de Fourier

Dada una secuencia $A=(A[0], ..., A[N-1])$ de longitud N, la Transformada Rápida de Fourier de A es la secuencia $C=(C[0], ..., C[N-1])$, dada por:

$$C[i] = \sum_{k=0..N-1} A[k]\, w^{ki}, \qquad (2.39)$$

donde $w = e^{2\pi\sqrt{-1}/N}$ es la raíz n-ésima de la unidad en el plano complejo. *(2.39)* se puede descomponer en:

$$C[i] = \sum_{k=0..N/2-1} A[2k]\, w^{2ki} + w^{i} \sum_{k=0..N/2-1} A[2k+1]\, w^{2ki} \qquad (2.40)$$

De esta fórmula se deduce que la DFT C de A puede obtenerse por la combinación de la DFT B de las componentes pares y de la DFT D de las componentes impares de A. La figura 2.28 muestra el seudocódigo ejecutado en el procesador identificado por NAME. Al principio del cómputo, todos los procesadores contienen una copia del vector A.

```
1:    j  := 0;
2:    for (i = NAME; i < n; i+= p)
3:      A[j++] := A[i];
4:    n := n / p;
5:    SequentialFastFourierTransform( A, n, w);
6:    for (i = log(p); i >=0; i--)   {
7:      if (bit i of NAME is non zero) {
8:        Send to processor NAME XOR 2^i the computed vector A
9:        break;
10:   }
11:   Receive in C from p = NAME XOR 2^i its resulting vector
12:   for (j = 0; j < n; j++)  {
13:     A[j+n] = A[j] - w^j C[j];
14:     A[j] = A[j] + w^j C[j];
15:   }
16:   n = 2 * n;
17:   w = e^(2πi/n);
18: }
```

Figura 2.28 Seudo-código de la FFT para procesador NAME.

Entre las líneas 1 y 4, los procesadores calculan el subvector de tamaño N/p a partir de A, resultando el patrón de las componentes impar-par dado por los bits de su nombre en orden inverso. A partir de aquí, cada procesador procede a calcular la DFT secuencial que sobre los datos que cada uno posee (línea 5). Hasta este momento no es necesaria ninguna comunicación. Las líneas 12 a la 17 contienen las instrucciones de la fase de combinación. Aquellos procesadores con los mismos $log(p)-1-i$ bits comparten el mismo ancestro i en el árbol de división. De esta forma, un procesador activo con su bit i igual a 1 envía su vector con componentes impares al procesador que ha computado la parte del vector con las componentes pares. En este momento existen varias comunicaciones en el mismo instante con tamaño: $N*2^i/p$ (líneas 7-11). De esta forma se subiría en el árbol realizando todos los cómputos de combinaciones y las comunicaciones. El tiempo total invertido por el algoritmo corresponde a la suma de las correspondientes fases. La fase de división es del orden de $O(N/p)$, la fase de la DFT secuencial es de orden $O(N/p\ log(N/p))$ y las comunicaciones sucesivas cada una de coste $(\alpha + \beta\ N/p)$ con las consiguientes combinaciones de orden $O(N\ 2^i/p)$. La fórmula computacional completa sería la siguiente:

$$T_{log(p)} = D(N/p) + F\ N/p\ log(N/p) + \\ + \sum_{i=0..log\ p-1} (\alpha + \beta\ (N\ 2^i/p) + R\ N\ 2^i/p) \qquad (2.41)$$

La tabla 2.14 contiene los valores de las constantes computacionales, D, F y V de las tres fases.

Constants	D	F	R
LAN	6,04E-7	1,44E-6	2,65E-6
IBM SP2	5,51E-7	5,85E-7	8,72E-7

Tabla 2.14 Valores de las constantes computacionales.

Una vez computadas las constantes del algoritmo y utilizando los valores de los parámetros para α y β calculados anteriormente, pasamos a la comparación de los valores del modelo con los valores de los tiempos reales del algoritmo para problemas de 524.288 datos reales.

En este punto debemos conocer el rendimiento de la red de interconexión para definir el modelo. Si asumimos la propuesta realizada en el trabajo [Sch94], donde incluso una red de estaciones conectadas por un bus, se comporta como un multicomputador hablaremos de modelo paralelo. En cambio si la red es un medio compartido y sólo un mensaje puede estar ocupando el canal en un instante dado, diremos que estamos trabajando en un modelo secuencial.

En el modelo que denominamos "paralelo" asumimos que la red de interconexión realiza las comunicaciones de las diferentes parejas en paralelo, sin degradar apreciablemente el rendimiento del sistema. Por otro lado, nos encontramos con el modelo que hemos denominado "secuencial", en el que las comunicaciones no se realizan en paralelo sino que al existir un canal de comunicaciones compartido, las mismas se serializan, dando lugar a rendimientos bajos de la red de conexión.

En el caso de la IBM SP2, la tabla 2.15 y la figura 2.29 muestran los resultados tanto del modelo "paralelo" como de los tiempos reales de ejecución. La concordancia de los datos reales con el modelo es precisa, tanto en los valores absolutos como en su comportamiento.

#procesadores	**2**	**4**	**8**
Real	3,26	1,90	1,27
Modelo Paralelo	3,23	1,85	1,21

Tabla 2.15 Tiempos del modelo y reales de la DFT para la IBM SP2

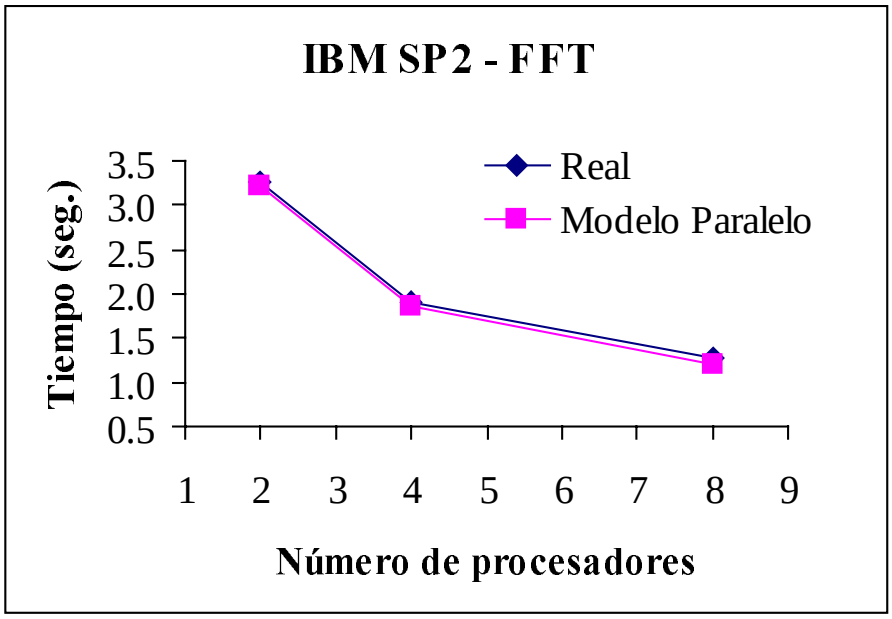

Figura 2.29 FFT en la IBM SP2, tiempos estimados y reales

Los errores que se producen son los mostrados en la tabla siguiente:

IBM SP2	2	4	8
Error	1,11	2,60	4,63

Tabla 2.16 Errores para la IBM SP2

En el caso de las estaciones de trabajo conectadas por un bus coaxial, se muestra el caso de asumir las comunicaciones secuenciales entre las parejas.

LAN	2	4	8
Real LAN	11,91	10,8	10,94
Modelo Secuencial	11,17	11,22	13,03

Tabla 2.17 Tiempos del modelo secuencial

En la tabla 2.17 se observa como el modelo utilizado en este caso predice incorrectamente los valores reales del algoritmo. Los errores mostrados en la tabla 2.18, aumentan con el número de procesadores, la curva del modelo se cruza con la curva de los tiempos reales y se queda por encima y el comportamiento de la curva del modelo es erróneo. La figura 2.30 muestra claramente esta inadecuación del modelo secuencial.

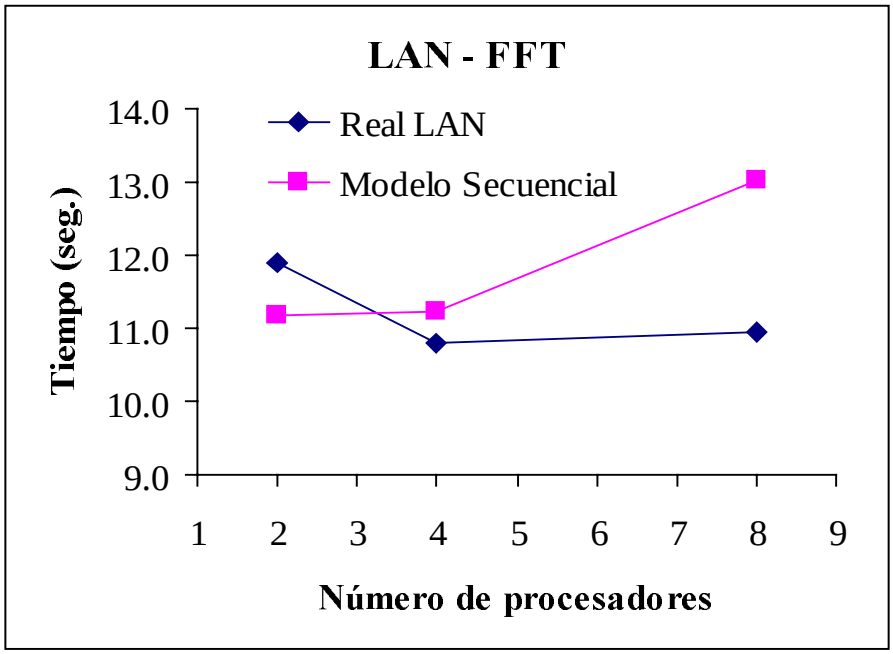

Figura 2.30 FFT en la LAN con modelo secuencial.

LAN	2	4	8
Error Secuencial	6,22	-3,87	-19,13

Tabla 2.18 Errores del modelo secuencial.

El modelo paralelo, como podría suponerse para este tipo de redes, tampoco es válido. La tabla 2.19 y la figura 2.31 muestran los valores de las predicciones del modelo paralelo, secuencial y los tiempos reales. Así, en el modelo secuencial, las predicciones provocan una curva creciente de los tiempos, en el modelo paralelo ocurre lo contrario,

los tiempos del modelo hacen decrecer la curva. Una red de área local no tiene ninguno de estos dos comportamientos bajo patrones injectivos.

LAN	2	4	8
Real	11,91	10,8	10,94
Modelo Secuencial	11,17	11,22	13,03
Modelo Paralelo	11,17	9,51	8,76

Tabla 2.19 Tiempos del modelo secuencial

La tabla 2.20 indica los errores que ahora aparecen en el modelo paralelo. Los errores introducidos son muy parecidos a los errores del modelo secuencial. Crece con el número de procesadores.

LAN	2	4	8
Error M. Secuencial	6,22	-3,87	-19,13
Error M. Paralelo	6,22	11,96	19,93

Tabla 2.20 Errores del modelo secuencial y paralelo.

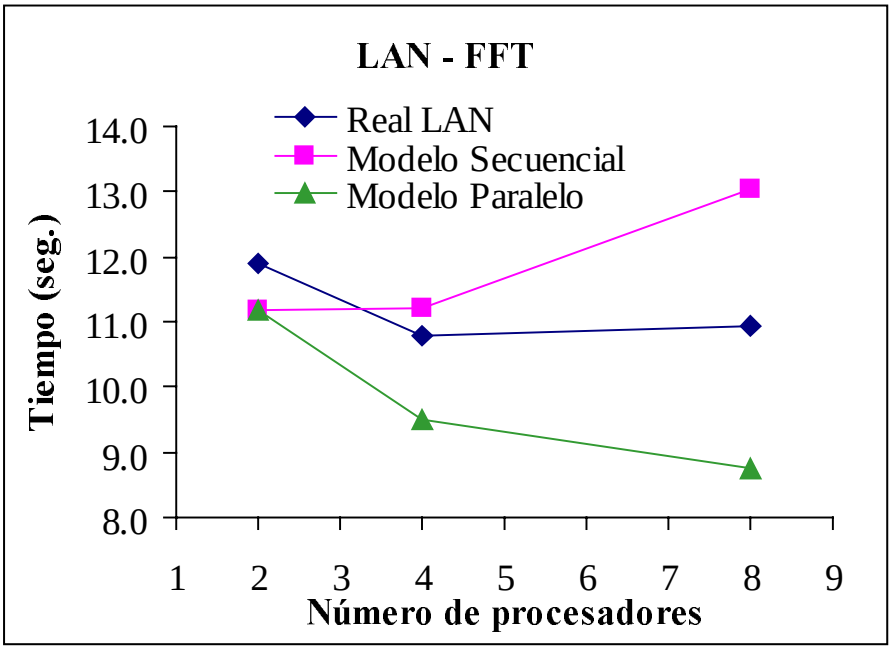

Figura 2.31 Modelo paralelo y secuencial para la LAN.

Ninguno de los modelos anteriores se ajusta adecuadamente a los tiempos ofrecidos por el algoritmo. Sobre la fórmula *(2.37)* se deben realizar ciertas consideraciones en cuanto a las comunicaciones se refiere. En esta fórmula, todos los envíos se realizan en paralelo por niveles como muestra la figura 2.28. Es decir en caso de 8 procesadores, en el primer caso existen 4 parejas de procesadores intercambiando información al mismo tiempo en el nivel más bajo. Subiendo un nivel, son dos las parejas que intercambian esta información pero con tamaños cada vez mayores, para concluir 1 sola pareja donde el procesador raíz tendrá toda la información: la DFT completa.

Al igual que en el caso anterior hemos elaborado un experimento para observar el comportamiento de las comunicaciones entre parejas de procesadores que dan lugar a los patrones injectivos de nuestros algoritmos. El experimento crea sucesivas parejas de procesadores que se envían datos de uno a otro. El rango de valores ha variado de 4 bytes a 4 Mbytes y cada ejecución se ha repetido 100 veces. A partir de los tiempos se han evaluado los valores de α y β con respecto al número de parejas involucradas.

Veamos en primer lugar los resultados que nos deparó la IBM SP2. La tabla 2.21 muestra los valores de α y β. La figura 2.32 nos ofrece la tendencia estable del parámetro β en este tipo de máquinas.

Parejas	1	2	3	4
α	0,000079	0,000079	0,000081	0,000079
β	4,20E-08	4,21E-08	4,21E-08	4,22E-08

Tabla 2.21 Valores de α y β para la IBM SP2.

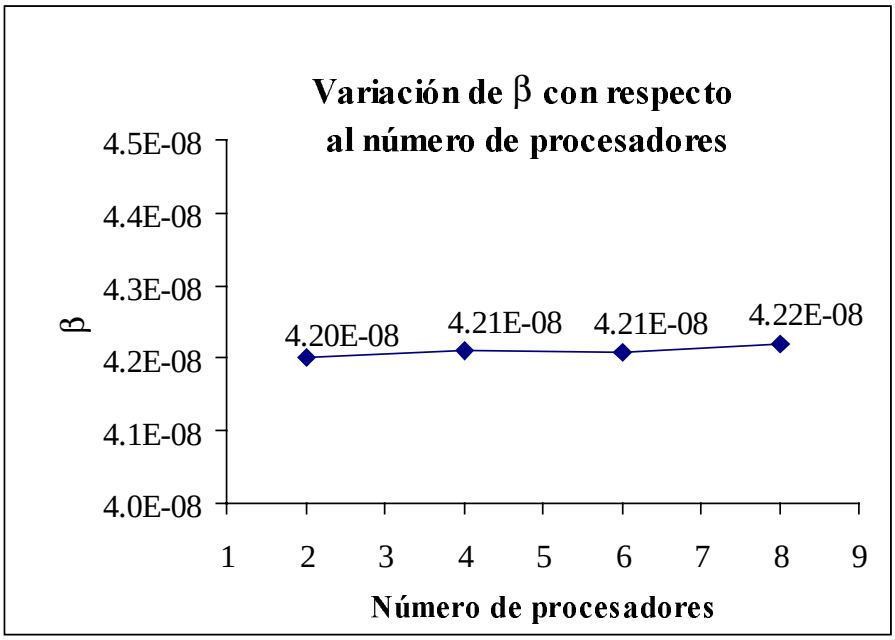

Figura 2.32 Variación de β respecto al número de parejas para la IBM SP2.

El experimento se realizó también para la red de estaciones de trabajos conectadas por el medio compartido y los resultados se exponen en la tabla 2.22 y figura 2.33. Ahora el fenómeno es diferente. Con el aumento de parejas, la red se va saturando hasta provocar una subida brusca de la pendiente de las β. La tabla 2.22 muestra como los valores de las β van desde 1,60E-06 con 2 procesadores hasta 3,29E-06 con 8 procesadores, se ha multiplicado por un factor de 2 aproximadamente. Observamos que la tendencia es lineal hasta con 6 procesadores. A partir de este momento la pendiente que ofrece la β hasta con 8 procesadores aumenta alcanzando un valor máximo.

El comportamiento de una red de este tipo no es ni secuencial ni paralelo, es lineal con respecto al número de procesadores.

Parejas	1	2	3	4
α	0,003278	0,003245	0,003322	0,003566
β	1,60E-06	2,00E-06	2,45E-06	3,29E-06

Tabla 2.22 Valores de ALFA y BETA para la LAN.

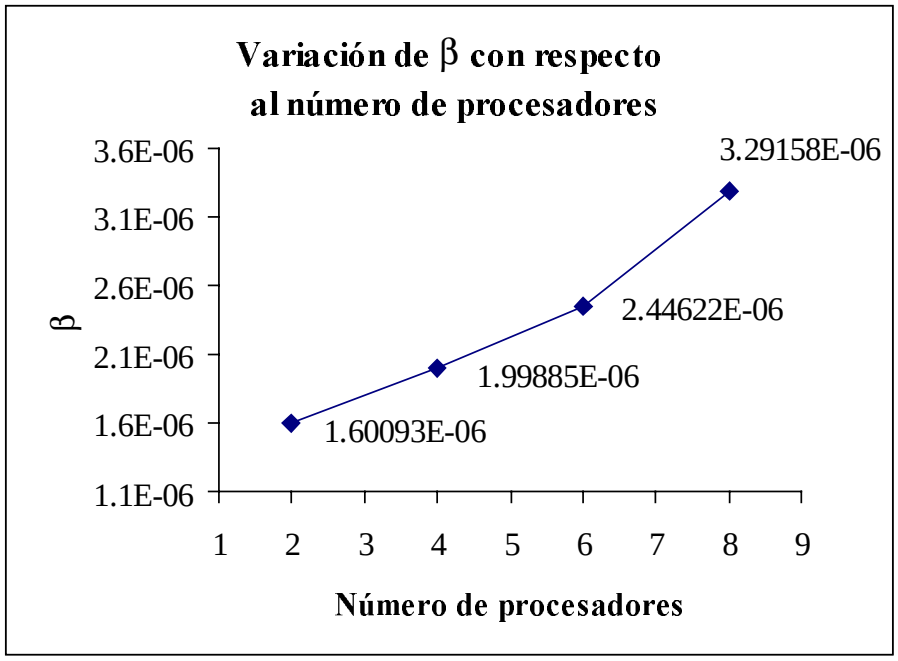

Figura 2.33 Variación de β respecto al número de parejas para la LAN.

Tenemos dos aspectos indeseables en los métodos de cálculo de la complejidad utilizados. Tenemos dos modelos en lugar de uno e incluso peor ninguno de los dos se ajusta a las estimaciones para redes locales.

Proponemos un modelo que unifique las propuestas anteriores y que sirva tanto para computadoras de altas prestaciones como para redes de área local. Caracterizamos el ancho de banda del sistema como una función del número de parejas involucradas. Denominamos modelo Injectivo cuando las comunicaciones pueden ser computadas con la siguiente fórmula:

$$Comm\ (|S| = p,\ size) = \alpha_p + \beta_p\ size \quad (2.42)$$

donde α_p y β_p corresponden con los valores mostrados en la tabla 2.20 anterior. Cuando el número de parejas involucradas *s*, saturan el canal de comunicaciones compartido, la contención queda definida por una función lineal. Para el caso donde aún no se alcanza la saturación ($p < s$, 1 a 3 parejas) se observa el mismo comportamiento lineal pero con una pendiente mucho menor. De esta forma los valores de β_p siguen la fórmula:

$$\beta_p = \delta_u\ p + \theta_u \text{ si sólo si } p < s \qquad (2.43)$$
$$\beta_p = \delta_u\ p + \theta_u \text{ si sólo si } p >= s$$

Denominaremos β_p el ancho de banda *p*-injección. El parámetro δ es la variación del ancho de banda *p*-injección. Los subíndices *u* y *s* indican respectivamente a los casos de saturación y no saturación. Para el caso de la IBM SP2, el modelo Injectivo propuesto

se adecua perfectamente ya que el valor de δ_u es igual a 0, el valor de θ_u es igual a 4,28E-8 y el valor teórico de *s* es ∞.

Por tanto la fórmula anterior tiene carácter general y como casos particulares aparecen los modelos considerados en un principio: el Modelo Paralelo ($\delta_u = 0$, $\theta_u = \beta$) y el Modelo Secuencial ($\delta_s = 1$, $\theta_u = 0$).

Aplicando el modelo Injectivo al ejemplo de la DFT, la fórmula de la complejidad computacional queda de la siguiente forma:

$$T = D(N/p) + F\,N/p\,log(N/p) + \sum_{i=0..log\,p-1} (\alpha_{p/2^i} + \beta_{p/2^i}\ N\ 2^i/p) + V\,N\,2^i/p) \quad (2.44)$$

Calculando de nuevo los valores estimados por la fórmula *(2.44)* comprobamos como el Modelo Injectivo predice correctamente los tiempos reales de ejecución. La distancia entre los valores estimados ahora son menores que en los casos anteriores. Y aún más importante, la variación de la curva real es idéntica a la que nos muestra el modelo Injectivo. Otra observación importante es que nos permite indicar el número óptimo de procesadores para ejecutar el problema.

LAN	2	4	8
Real LAN	11,91	10,8	10,94
Modelo Secuencial	11,17	11,22	13,03
Modelo Paralelo	11,17	9,51	8,76
Modelo Injectivo	11,17	10,01	10,24

Tabla 2.23 Resultados con el modelo Injectivo propuesto.

LAN	2	4	8
Error Secuencial	6,22	-3,87	-19,13
Error Paralelo	6,22	11,96	19,93
Error Injectivo	6,22	7,30	6,38

Tabla 2.24 Errores en los tres casos.

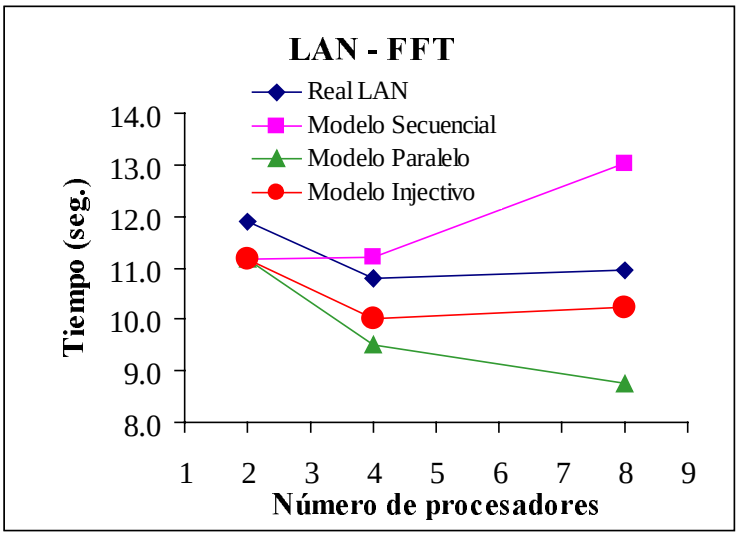

Figura 2.34 Comparación de Modelos y tiempo real en la LAN.

2.5 CONCLUSIONES

El modelo LogP es un modelo de computación paralela asíncrono basado en las nuevas tecnologías de las arquitecturas paralelas. Una de las características principales del modelo es que permite el solapamiento entre el cómputo y comunicaciones. Las predicciones realizadas bajo el modelo LogP funcionan correctamente en las redes actuales si el software que les da soporte es adecuado. Las plataformas software sobre las que trabaja el modelo LogP son Active Messages y Fast Messages. Desgraciadamente, el estándar de programación paralela MPI, no se adapta al modelo LogP.

El modelo C^3 es similar al modelo LogP, pero tiene en cuenta el tipo de encaminamiento, e introduce la congestión de los enlaces de la red y de los procesadores. Más que un modelo de propósito general con el que abordar el diseño y análisis del conjunto global de algoritmos paralelos, el modelo está pensado para el estudio a bajo nivel de las primitivas de comunicación. Este modelo es adecuado para el análisis y diseño de las funciones que constituyen el núcleo básico de cualquier librería de comunicaciones.

Proponemos el modelo de Patrones con el objetivo de ser un modelo de predicción donde existe un compromiso entre la eficiencia del rendimiento y la simplicidad para obtener la complejidad. El modelo de patrones permite el análisis de algoritmos escritos utilizando funciones colectivas. El modelo es más preciso que cualquiera de los considerados, si bien sufre de dos carencias: El conjunto de algoritmos a los que se aplica se restringe a aquellos que limitan las comunicaciones a funciones colectivas y la portabilidad del análisis es menor debida al mayor número de coeficientes en la descripción. Utilizando este modelo hemos unificado el caso de patrones inyectivos para multicomputadoras y para redes de área local. En el caso de los patrones uno a todos se ha observado como decrece el tiempo cuando aumentamos el número de procesadores. En comunicaciones por parejas, hemos mostrado el rendimiento de las multicomputadoras y se ha comparado con las redes de área local.

3.1 INTRODUCCIÓN

Desde los años 80 han surgido diferentes modelos de computación paralela con la idea de intentar cubrir la distancia existente entre las plataformas paralelas hardware y el software que permite su uso. Estos modelos han sido ya estudiados en los capítulos anteriores y como se destacó, ninguno ha sido lo suficientemente general como para poder ser un modelo de referencia. Cada uno de esos modelos se ajustaba con bastante precisión a la arquitectura considerada. En la actualidad las diferentes arquitecturas parecen seguir el mismo esquema interno: procesadores convencionales muy potentes conectados a través de una red de interconexión de muy altas prestaciones.

Así como en computación secuencial, el modelo de von Neumann ha marcado claramente las tendencias de software y hardware, en la computación en paralelo no ha podido crearse un modelo que contemplara un espectro suficientemente amplio que englobara las diferentes arquitecturas que en los últimos años han aparecido.

El modelo Bulk Synchronous Parallel (BSP) es un modelo de computación paralela propuesto en 1990 por Leslie Valiant [Val90] que intenta cubrir el hueco existente entre las plataformas hardware y software actuales. De los modelos que han aparecido los últimos años, el BSP es el que ha tenido más popularidad.

3.2 DESCRIPCIÓN DEL MODELO BSP

El modelo BSP establece un nuevo estilo de programación paralela para la realización de programas de propósito general cuyas características principales son:

- Los programas BSP son sencillos de escribir.
- Los programas BSP son independientes de la arquitectura destino.
- Se puede predecir su comportamiento en diferentes arquitecturas conocidos los parámetros que las caracterizan.

La idea fundamental del BSP es la división entre cómputo y las comunicaciones. Todo programa BSP consiste en un conjunto de pasos, denominados “SUPERPASOS”, en los cuales primero existe una fase de cómputo, le sigue una fase global de comunicaciones y finalmente ocurre una sincronización por barreras que permite separar los diferentes superpasos. Esta forma de programación paralela es válida en muchas computadoras, tanto de memoria distribuida, como memoria compartida y redes de estaciones de trabajo.

Una computadora BSP se define de la siguiente forma:

- Un conjunto de pares de procesador–memoria.
- Una red de comunicaciones que permite la entrega de mensajes punto a punto, y
- Un mecanismo de sincronización de los procesadores en los superpasos.

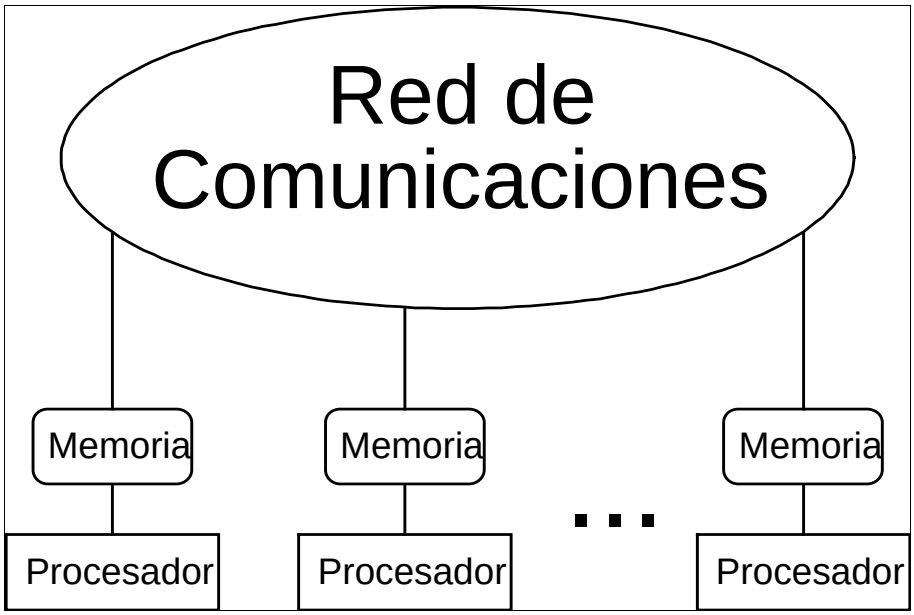

Figura 3. 1 Esquema actual de las multicomputadoras.

Entre las computadoras que conforman el modelo BSP podemos incluir las máquinas con un único procesador y con memoria fuera del chip, las redes de estaciones de trabajo u ordenadores personales conectadas con alguna librería de paso de mensajes como PVM o MPI, los procesadores con memoria distribuida como IBM SP2, Intel Paragon, Meiko, etc.; los procesadores con memoria compartida distribuida como Silicon Origin, Cray T3E, Convex Spp, Multiprocesadores Pentium.

Definimos un "*step*" o paso como una operación básica que se realiza sobre datos locales de un procesador. Un cómputo BSP consiste en una secuencia de "supersteps" o superpasos que se realizan en paralelo, donde cada uno consiste en una secuencia de *steps* que realizan operaciones sobre datos locales, seguidos por comunicaciones globales y una sincronización por barreras. Cualquier petición de datos remotos se puede realizar durante el superpaso pero no se podrán utilizar los mismos hasta que se entre en el siguiente superpaso, después de la sincronización. Todas estas peticiones son sin bloqueos, es decir no quedan en espera de los datos sino que continúan con su cómputo. Esta forma de proceder al realizar los programas permiten a los diseñadores de algoritmos paralelos tener un modelo de costes sencillo de computar. La figura 3.2 siguiente muestra gráficamente la forma de proceder del modelo.

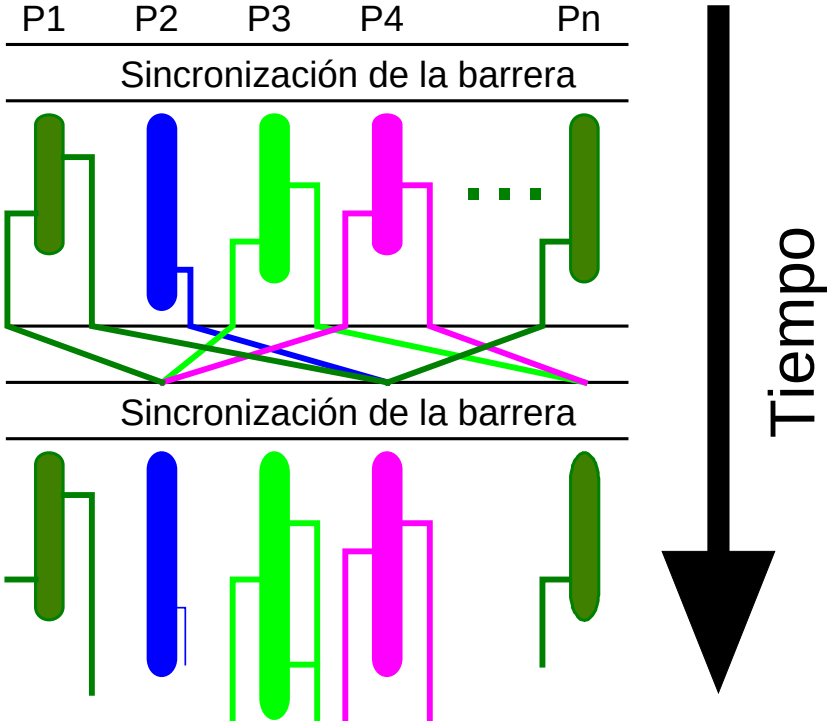

Figura 3. 2. Modelo de computación BSP.

La ejecución de un algoritmo BSP consiste en una secuencia de *superpasos*. Cada *superpaso* consta de tres fases:

1. Los procesadores computan localmente. Para ello utilizan los valores almacenados en su memoria privada.
2. En cualquier momento del superpaso, los procesadores pueden transmitir mensajes.
3. Al finalizar el superpaso tiene lugar una sincronización por barrera. **Se garantiza que al finalizar la sincronización por barrera todos los datos emitidos durante el superpaso han alcanzado su destino**.

El tiempo t_s de un superpaso s vendrá acotado por la suma del máximo tiempo de cómputo más el tiempo de las comunicaciones y la sincronización final.

$$t_s \leq W_s + CS_s \qquad (3.1)$$

Donde

$$W_s = max \{ w_{si} \ / \ i \in \{0,...,p-1\} \}, \qquad (3.2)$$

w_{si} denota el tiempo de cómputo invertido por el procesador i en el superpaso s y CS_s es el tiempo invertido en las comunicaciones y la sincronización en el superpaso s.

Para poder definir el coste de las comunicaciones, es necesario precisar como medimos la cantidad de datos, c_{si}, comunicada por un procesador i. Durante un superpaso s, cada procesador $i \in \{0,...,p-1\}$ envía un cierto número de mensajes out_{si} y recibe un cierto número in_{si} de mensajes. Si las entradas y las salidas tienen lugar en estricta secuencia, es natural que la medida de la cantidad comunicada por el procesador i sea definida como la suma $c_{si} = in_{si} + out_{si}$. Si, por el contrario las entradas y salidas pueden ocurrir en paralelo, la medida debería ser el máximo de ambos: $c_{si} = in_{si}\ max\ out_{si}$. La

elección del operador binario, @ = + ó *max*, más adecuado dependerá de la arquitectura concreta. Podemos entonces definir el procesador d_s "que más comunica" en el superpaso *s* como aquel que alcanza el siguiente máximo:

$$h_s = max\ \{in_{si}\ @\ out_{si} / i \in \{0,...,p-1\}\} \qquad (3.3)$$

Se dice que un superpaso *s* conlleva una *h*-relación (con h = 0,1,2,...) si el patrón de comunicaciones que tiene lugar en el superpaso *s* es tal que $h_s = h$.

El cómputo global de un programa BSP se calcula sumando los costes individuales de cada uno de los superpasos en que se ha dividido. Para una plataforma con el software *BSP* adecuado y una arquitectura de red *H* "suficientemente buena", existen constantes g_H y L_H tales que el tiempo $CS_{H,s}$ invertido en comunicación g_H más sincronización L_H en un superpaso *s* **es una función lineal** del tamaño h_s de la *h*-relación asociada al superpaso *s*:

$$CS_{H,s}(h_s) = g_H * h_s + L_H \qquad (3.4)$$

El tiempo t_s invertido por un algoritmo BSP en un superpaso *s* puede aproximarse por:

$$t_s \approx W_s + g_H * h_s + L_H \qquad (3.5)$$

3.3 PARAMETROS DE LA COMPUTADORA BSP

Una computadora BSP queda caracterizada por el ancho de banda de la red de interconexión y por el tiempo para realizar una sincronización de los procesadores. Obviamente el número de procesadores y su velocidad forman parte de los parámetros de una computadora BSP.

Normalmente se comparan los diferentes sistemas y para ello es necesario normalizar los parámetros respecto a la velocidad de los procesadores. De esta forma todo estaría en unidades de velocidad de cómputo. Si definimos un *step* como la unidad de cómputo (muchas veces se toma el tiempo en realizar una operación en coma flotante), podemos decir que la velocidad del procesador es de *k steps/sec*. De esta forma, el coste de una sincronización por barreras de los procesadores, puede medirse en términos del número de *steps*, que conlleva realizar esa tarea.

Una computadora BSP queda caracterizada por los siguientes parámetros:

- P, el número de procesadores de la máquina BSP.
- *s*, la velocidad del procesador,
- *L*, el coste, en *steps*, de realizar la sincronización,
- *g*, el coste, en *steps/word*, de entregar los mensajes.

De estos cuatro, *s* es un factor de normalización y por tanto el modelo queda caracterizado por 3 parámetros independientes: *L, g* y *P*.

La velocidad *s* se mide realizando un conjunto de operaciones reales sobre el sistema en cuestión, no se toman los valores de los fabricantes.

El valor de *g* se calcula del coste medio de transferir un mensaje por la red de comunicaciones en la que viajan otros paquetes. No se debe tomar el valor que indican los fabricantes de su ancho de banda. El valor de *g* normalizado se obtiene de:

g = número total de operaciones locales en un procesador / número de words comunicadas por el sistema de comunicaciones.

El valor de *g* nos permite obtener una estimación del tiempo que conlleva a dos procesadores intercambiar mensajes. Si el máximo número de *words* comunicadas por cualquier procesador es *h*, podemos estimar que *gh* es el número de *steps* de cómputo que el procesador puede realizar en ese mismo tiempo.

Cuanto más se acerque el valor de *g* a 1 y más pequeño sea el valor de la latencia *L*, es más fácil de lograr mayor escalabilidad sobre un sistema. Así, aumentando el número de procesadores de una computadora BSP tendremos menos cómputos locales en cada procesador y los parámetros dominantes serán *g* y *L*. Valores publicados [Ski96] nos permiten hacernos una idea del orden de magnitud de los parámetros sobre diferentes arquitecturas usando la librería BSPLib [Hil97].

3.4 CALCULO DEL COSTE DEL MODELO

La existencia de un modelo de coste sencillo en el modelo BSP, permite predecir el comportamiento de algoritmos existentes y nuevos en diferentes arquitecturas paralelas.

El coste de un superpaso en el modelo BSP es la suma de tres términos. El primero indica el valor máximo de cómputo local en los diferentes procesadores. El segundo coste representa las comunicaciones de las *h*-relaciones, y el tercero es el coste de la barrera de sincronización al final del superpaso.

La fórmula del coste del modelo es:

$$\textit{Coste de un superpaso} = MAX_i\ w_i + MAX_i\ h_i\, g + L \qquad (3.6)$$

Donde *i* varía en el número de procesadores y w_i es el tiempo de cómputo local de cada procesador.

La suma del coste de cada uno de los superpasos conforma el coste total de un programa BSP. La facilidad que implica la evaluación del coste es una de las características que han hecho del modelo BSP uno de los más populares.

El modelo de coste nos sugiere un conjunto de estrategias para hacer eficiente la ejecución de los programas BSP:

1. Se debe equilibrar el cómputo *w* entre los diferentes procesadores que intervienen. El superpaso no finalizará hasta que todos los procesadores se encuentren en la barrera de sincronización y por tanto si alguno de ellos se retrasa, todos se retrasan.
2. Se debe intentar equilibrar las comunicaciones existentes entre los diferentes procesadores siguiendo la hipótesis de la *h*-relación.
3. Debemos intentar minimizar el número de superpasos para evitar en la medida de lo posible las retenciones debido a la sincronización *L*.

Se han propuesto otros modelos de costes para el BSP, que lo hacen algo más detallado. Se podría considerar que el cómputo y las comunicaciones están solapadas, ofreciendo una fórmula de coste igual a:

$$MAX\,(w,\ hg) + L \qquad (3.7)$$

Otros modelos de coste sugieren que debe existir una diferencia entre los paquetes entrantes y salientes en un procesador:

$$(h_{in} + h_{out})\ g \qquad (3.8)$$

Estas propuestas no cambian considerablemente la original. En todo caso, modifican la complejidad de los algoritmos en un factor máximo constante de 2.

3.5 ENTORNOS BSP DE PROGRAMACION

El BSP es un modelo de computación paralela que proporciona mecanismos para predecir las comunicaciones. El modelo no entra en detalles de cómo se deben realizar los cómputos en los procesadores ni cómo se deben llevar a cabo las comunicaciones. El modelo define que un programa debe dividirse en superpasos y que al final de cada uno debe existir una sincronización de los procesadores. Partiendo de esta base, el modelo BSP puede expresarse en muchos lenguajes y librerías como PVM, MPI, MPL, Cray SHMEM, etc. El único requerimiento es que todas ellas proporcionen mecanismos de comunicaciones sin bloqueos y una forma de implementar sincronismos por barrera. Cuando utilizamos estas librerías, los valores de los parámetros L y g, no sólo dependen de la plataforma hardware, sino también depende de los *overheads* debidos a las plataformas software.

Existen implementaciones como la librería Oxford BSP [Hil97] donde las comunicaciones que se produce en un superpaso, se retrasan hasta el final del mismo. Al finalizar su fase de cómputo y producirse la llamada a la correspondiente barrera se producen todas las comunicaciones en bloque. Aunque retrasar las comunicaciones al final del superpaso puede acarrear un incremento de la memoria *buffer* necesitada y, en el peor caso, un aumento al doble del tiempo de ejecución, también ofrece oportunidades para mejorar el rendimiento de las comunicaciones. Por ejemplo, se disminuye el número de veces que tienen efecto las latencias de arranque al reunir los mensajes con el mismo destino y se pueden reordenar los mensajes con vistas a evitar posibles congestiones. También se unifican los procesos de comunicación y sincronización y no es necesario considerar su costo por separado. Si se sigue esta estrategia los patrones de comunicación desaparecen y las comunicaciones ocurren según el esquema aplicado por la herramienta BSP utilizada.

La librería BSPLib [Hil97] es la que más popularidad ha tenido. También se han desarrollado algunos lenguajes BSP entre los que podemos mencionar GPL [Mco95], un lenguaje MIMD que permite sincronizaciones entre conjuntos de procesos; y Opal [Sim94] que está basado en objetos.

3.6 EJEMPLO DE PROGRAMA BSP

El siguiente ejemplo de programa BSP ha sido escrito utilizando la librería BSPLib anteriormente referenciada. Para entender bien el programa introduciremos brevemente las funciones de la librería. La librería contiene solamente 20 funciones agrupadas por

características en seis bloques diferentes. El primer bloque consiste en las funciones de inicialización de un programa BSP. En este existen las funciones *bsp_init()*, que crea los procesos solicitados, *bsp_begin()*, que inicia nuestro programa SPMD y *bsp_end()*, que finaliza la ejecución de nuestro programa.

Un segundo conjunto incluye tres funciones que permiten obtener información del sistema. *bsp_pid()* permite obtener el nombre lógico del proceso. *bsp_nprocs()* nos da el número de procesos actualmente en ejecución. *bsp_time()* nos da el tiempo local de un procesador en un momento dado.

Existe una rutina que se encarga de la sincronización de todos los procesos: *bsp_sync()*. Y otra rutina se encarga de parar la ejecución de todos los procesos de un programa BSP: *bsp_abort()*.

El cuarto grupo de funciones nos ofrece la posibilidad de acceso a memoria remota (denominado Direct Remote Memory Access: DRMA). La función *bsp_pushregister()* inicializa un espacio de memoria para que sea visible globalmente. *bsp_popregister()* libera la región anterior. *bsp_put()* permite insertar esta información en la memoria remota y *bsp_get()* permite leer de la memoria remota.

La otra forma de comunicarse es a través de mensajes (denominado Bulk Synchronous Message Passing: BSMP). A diferencia del caso anterior, los datos son copiados en una cola del proceso receptor que serán obtenidos por este al principio del siguiente superpaso. En este caso, la información no tiene porqué estar tan estructurada, puede depender del algoritmo. En este caso, es necesario definir un tamaño de mensaje con *bsp_set_tag_size()* antes de enviar un mensaje con *bsp_send()*. Para leer una etiqueta de un mensaje tenemos *bsp_get_tag()* y para buscar mensajes en las colas *bsp_move()*.

Por último existen unas rutinas que corresponden a versiones de alto rendimiento pero sin *buffers,* de las anteriores DRMA: *bsp_hpput()*, *bsp_hpget()*, *bsp_hpmove()*.

El programa BSP que se expone es la suma de prefijos. Consiste en realizar las sumas parciales de P números almacenados en los P procesadores. El algoritmo tiene un total de $\lceil \log P \rceil + 1$ superpasos separados por sincronismos por barreras.

A la función *bsp_suma_prefijos()* se le pasa el argumento *x* que es el número de valores que hay que sumar. La función comienza con la inicialización de los procesos a través de la función *bsp_begin()*. La función *bsp_nprocs()* nos permite conocer el número de procesos que se han inicializado y la función *bsp_pid()* inicializa la variable *name* con el nombre virtual de cada proceso.

La llamada a *bsp_pushregister(&other, sizeof(int))* declara como perteneciente al espacio de direcciones compartido a la variable *other* de tamaño *sizeof(int)*. ¿Cómo se consigue este espacio de direcciones compartido? Obsérvese que no tiene por que ocurrir que las diferentes copias de la variable *other* estén en la misma dirección en los distintos procesadores, ya que la librería se encarga de crear la correspondencia entre la variable *other* y las direcciones físicas en cada procesador. Después de esta función se debe realizar un sincronismo por barreras para que todos los procesadores registren la nueva zona compartida.

Cada procesador *name*, dentro del bucle *i*, realiza *log P* llamadas a la función DRMA *bsp_put(name+i, &partial, &other, o, sizeof(int))* copiando remotamente en la dirección de *&other*, el entero *partial* en la memoria del procesador *name+i*. La llamada a

bsp_popregister(&other) saca la variable *&other* del espacio de direcciones compartido. Sin embargo, esta extracción del espacio de direcciones compartido no se hace efectiva hasta la siguiente sincronización por barrera. El coste del algoritmo es

$$Tprefix = ceil(log(p))*(C+g+L) \qquad (3.9)$$

Siendo p el número de procesadores y C la correspondiente constante de cómputo.

```
main()
{ int i;
  bsp_begin();
  i = bsp_prefix(bsp_pid()+1);
  bsp_end();
 }
```

Figura 3. 3 Programa principal

```
int bsp_suma_prefijos(int x) {
  int i, other, partial, name, numprocs;

  bsp_begin();
  numprocs = bsp_nprocs();
  name = bsp_pid();

  bsp_pushregister(&other, sizeof(int));
  bsp_sync();

  partial= x;
  for(i=1; i<numprocs;  i *=2) {
    if  (name+i <  numprocs)
      bsp_put(name+i, &partial, &other, 0, sizeof(int));
    bsp_sync();
    if (name >= i) partial = partial + other;
  }
  bsp_popregister(&other);
  bsp_end();
  return partial;
}
```

Figura 3. 4 Sumas parciales utilizando funciones BSPLib.

3.7 EFECTO DEL TAMAÑO DEL MENSAJE EN EL VALOR DE *g*

El tamaño de los mensajes puede afectar al valor de *g* y por tanto al modelo de costes. En el caso de librerías como la BSPLib el tamaño del mensaje no afecta en el modelo ya que todas las comunicaciones se retrasan hasta el final del superpaso y se realizan en bloque incluso reordenando los envíos para evitar la congestión de la red.

El modelo de costes original no hace distinción ninguna entre enviar *h* mensajes de tamaño 1 o 1 mensaje de tamaño *h*. En ambos casos la *h*-relación es *h g*. Sin embargo en el caso de un superpaso con un número total de comunicaciones pequeño, existirá una diferencia importante respecto al modelo de costes original, debido al efecto de grandes valores de *startup* en la transmisión de pocos mensajes.

Miller afinó el modelo de costes [Mil94] utilizando la técnica de Hockney [Hoc91] para modelar el efecto del tamaño del mensaje en el coste de las comunicaciones. En este nuevo modelo, *g* se define como una función del tamaño del mensaje *x*:

$$g(x)=(n_{1/2} / x + 1) \qquad (3.10)$$

donde g_∞ es el coste asintótico de las comunicaciones para mensajes grandes, y $n_{1/2}$ es el tamaño del mensaje con el que se alcanza la mitad del ancho de banda óptimo de la máquina tal que $g(n_{1/2}) = 2\, g_\infty$.

El valor de $n_{1/2}$ se calcula experimentalmente para cada máquina realizando los ajustes a las curvas que dan los valores de $g(x)$. Este valor, $n_{1/2}$, se puede utilizar para obtener el tamaño mínimo de mensaje para el cual el modelo de costes estándar coincide en un porcentaje dado con respecto al modelo detallado propuesto por Miller. Para que el modelo estándar difiera en un porcentaje de *y%*, respecto al modelo que incluye la granularidad del mensaje en las comunicaciones, se debe cumplir lo siguiente:

$$(100+y) / 100 \;\; h_0\, g_\infty = h_0\, g_\infty = (n_{1/2} / h_0 + 1)\, h_0\, g_\infty \qquad (3.11)$$

donde h_0 words es el parámetro de Valiant [Val90] que mide el tamaño mínimo tamaño de la *h*-relación para alcanzar el ancho de banda $n_{1/2}$. De esta forma, el porcentaje de error cometido en el modelo de costes $h_0\, g_\infty$ es:

$$y = (100\, n_{1/2} / h_0)\ \% \qquad (3.12)$$

Como se aprecia, al aumentar el tamaño de la *h*-relación, el error del modelo BSP estándar disminuye.

3.8 OTROS MODELOS

3.8.1 Modelo E-BSP

El modelo E-BSP de Juurlink et al. [Juu95] se caracteriza principalmente por describir las comunicaciones como una (M, h_1, h_2)-relación, donde cada procesador envía como máximo h_1 mensajes, recibe como máximo h_2 *y* donde el número total de mensajes encaminados no sobrepasa *M*. E-BSP extiende el Modelo BSP de dos formas diferentes. Por un lado, ofrece la posibilidad de tratar con patrones de comunicaciones no

balanceados (cada procesador recibe o envía diferente cantidad de mensajes). Por otro, permite considerar la localidad del lugar donde están los datos a recibir o donde enviar. Bajo este nuevo modelo, sus autores proponen mejoras en los algoritmos diseñados para el modelo BSP.

En el BSP, el coste de las comunicaciones depende de la máxima cantidad de datos a enviar o recibir por cualquier procesador. En muchos casos, esto supone un sobrecoste. Por ejemplo, cuando se realiza un *broadcast* personalizado, donde un procesador envía $p-1$ mensajes a cada uno de los restantes p procesadores, el coste de las comunicaciones bajo el modelo BSP es del orden de $O(L+g\ p)$. En este caso, el sobrecoste podría venir provocado por la existencia de caminos Hamiltonianos en la red de interconexión, imponiendo un coste de tan sólo $O(p)$.

La segunda aportación en este modelo, ofrece la posibilidad de incorporar localidad de los datos en el modelo BSP. Este último sólo distingue entre accesos a memoria local y remota. En E-BSP se propone diferenciar en "localidad de vecinos" donde el retraso de los accesos depende del lugar relativo de los procesadores en la red de interconexión. Para entender mejor este problema, consideremos el problema de encaminamiento de permutaciones en un árbol binario. Bajo el modelo BSP, el coste de las comunicaciones es de $O(p)$, ya que se asume que todos los mensajes deben pasar por el procesador raíz del árbol. El rendimiento depende mucho de la localidad de las comunicaciones ya que algunos mensajes podrían ser encaminados por los diferentes subárboles y no pasar necesariamente por la raíz. De esta forma no limitados el ancho de banda del procesador raíz.

El modelo E-BSP está compuesto de elementos procesador/memoria, de un mecanismo de encaminamiento de mensajes y de un mecanismo de sincronismo. En este modelo se asume que todo programa se divide en superpasos y al final de cada superpaso existe una sincronización por barreras como en el BSP. En cada superpaso, se realizan operaciones sobre datos locales, y se envían o reciben datos de otros procesadores. La diferencia principal con el BSP es que en E-BSP el coste de las comunicaciones se computa de otra forma.

Denotemos por h-relación el patrón de comunicaciones que un procesador envía o recibe como máximo h mensajes. En el BSP, todas estas comunicaciones se ven como una h-relación y su coste queda definido por la fórmula: $g\ h\ +\ L$. Estas h-relaciones se consideran que están llenas en el sentido que un procesador envía y recibe exactamente h mensajes. Pero existen muchas situaciones donde realizar los cómputos de esta forma provocan una sobre estimación de los costes. Para intentar solucionar estos casos, de comunicaciones descompensadas, Juurlink et. al. proponen que las comunicaciones en cada superpaso se definan por una terna (M, k_1, k_2)-relación. Esta nueva situación define mejor muchos casos prácticos y no sobrecarga el coste del modelo que obliga a cargar con una h-relación.

Consideremos que $[P]$ denota el conjunto de identificadores de los procesadores $\{0, 1, ..., p-1\}$. Un problema de encaminamiento de mensajes se describe en término de una distribución de encaminamiento. Definen una distribución de encaminamiento R como una bolsa de parejas de procesadores tomados de $[P]x[P]$. Cada pareja especifica el procesador fuente y destino de un paquete. Definen un problema de encaminamiento como el encaminamiento simultáneo de paquetes de una distribución determinada. A

partir de estas definiciones se puede dar una descripción más completa de *(M, k_1, k_2)*-relación y de un problema de encaminamiento con localidad *L*.

Definición: Una distribución de encaminamiento *R=[(s,d) | s,d ∈ [P]]* es una *(M, k_1, k_2)*-relación si *#R <= M*, y para todo *t∈ [P]: #[(s,d) ∈ R | s = t]<= k_1* y *#[(s,d) ∈ R | d = t] <= k_2*. Una distribución de encaminamiento *R* tiene localidad *L* si para todo *(s, d) ∈ R*, *|d-s| <=L*.

Obsérvese que una relación *h* llena, modelo BSP, es un caso especial de *(M, k_1, k_2)*-relación, con *M = h P* y *k_1 = k_2 = h*.

Para una descripción más detallada del modelo podemos ver las referencias [Juu95][Juu96a][Juu96b]. En estas se encuentran ejemplos de algoritmos de comunicaciones sobre un array de procesadores y sobre una malla. En ambos casos se muestra la complejidad del modelo E-BSP y se compara con el BSP. También se encuentran aplicaciones como la multiplicación de matrices y el problema de las diferencias finitas.

Como resumen del modelo E-BSP, se puede decir que se ha complicado el cómputo de la complejidad de las comunicaciones respecto al BSP pero se ha conseguido mayor precisión de los costes de las comunicaciones. El modelo propuesto hace más difícil el diseño y análisis de los algoritmos que en el BSP debido a dos cuestiones principales: el modelo BSP no tiene en cuenta la localidad de los mensajes y el tiempo de ruteo de los mensajes no siempre puede expresarse como una función lineal de *h*.

3.8.2 Modelo COARSED GRAINED MULTICOMPUTER

El modelo *Coarsed Grain Multicomputer*, CGM [Cac97], utiliza únicamente dos parámetros, *N* y *P*, y se asume un conjunto de *P* procesadores con memoria local de tamaño *N/P*, conectados a una red de altas prestaciones que permite las comunicaciones punto a punto.

Un algoritmo CGM consiste en un conjunto alternativo de etapas de cómputo y de comunicaciones separados por una sincronización de barreras. Una etapa de computación es equivalente al superpaso de computación del modelo BSP y su coste se define de la misma forma. Una etapa de comunicaciones consiste en una única *h*-relación con *h<=N/P*. El coste $H_{N,p}$ de cada etapa de comunicación es el mismo. El coste total de las comunicaciones es $Tcomm = \lambda H_{N,p}$.

El CGM ayuda al diseño y análisis de algoritmos teóricos eficientes para sistemas paralelos de grano grueso. Este modelo ofrece especial interés cuando los algoritmos tienen mucho cómputo en relación con las comunicaciones. Otro punto interesante es que dado que el tamaño del mensaje es maximal, el modelo minimiza el *overhead* asociado cuando se envía un único mensaje.

La principal ventaja del modelo CGM es que permite caracterizar el coste de las comunicaciones de un algoritmo paralelo por un único parámetro: el número de etapas de comunicación.

3.9 EVALUACIÓN DE LOS PARÁMETROS *g* Y *L*

En esta sección se detallan los estudios presentados en [Rod98c] y [Rod98d] para la evaluación de los parámetros del modelo BSP. Los experimentos muestran la variación del tiempo de las comunicaciones de *h*-relaciones con respecto al número de procesadores y diferentes patrones de comunicaciones. Esto no es más que mostrar el grado de cumplimiento de la hipótesis de Valiant sobre plataformas hardware actuales y con software estándar actual.

El cómputo de los parámetros del modelo g y L se ha realizado tomando los patrones de comunicaciones más utilizados en la mayoría de los algoritmos. En el caso del BSP, los trabajos como [Hil97] muestran que el estudio del coste lo han realizado para *h*-relaciones arbitrarias entre diferentes procesadores. Nuestro estudio se basa en utilizar aquellos patrones más usuales en los algoritmos paralelos más que relaciones aleatorias entre los diferentes procesadores. Aún más, son patrones que en sí mismos están implementados en las librerías estándar como MPI o PVM.

El número de procesadores varía según la disponibilidad de cada máquina, y respecto a los patrones de comunicaciones se consideran un conjunto $\Pi = \{E, PP, OA, OAP, AO, AA\}$ con seis de los patrones de comunicación más comunes. Estos seis patrones son: el patrón de intercambio o "*Exchange"* denotado en lo que sigue por *E,* el *"PingPong"* que denotamos *PP*, el uno a todos o *"OneToAll" (OA*), el uno a todos personalizado *"OneToAll Personalizado"* (*OAP*), el patrón todos a uno *"AllToOne" (AO*) y el patrón todos a todos *"AllToAll" (AA)*.

En un patrón *Exchange*, un cierto número p de procesadores simultáneamente envían y reciben mensajes de longitud m_E. Difiere del patrón de comunicaciones *PingPong (PP)* en el que los *p/2* procesadores de una mitad envían un mensaje de tamaño m_{PP} a los *p/2* procesadores de la otra mitad. Observe que, a diferencia de la definición habitual de *PingPong*, en esta definición no hay devolución del mensaje por parte del receptor (esto daría lugar a un nuevo superpaso) y es, por tanto, una comunicación más asimétrica que el *Exchange*. En el patrón *OneToall (OA)* uno de los procesadores envía el mismo mensaje de tamaño m_{OA} a los restantes *p-1* procesadores. En el patrón *OneToAll Personalizado* (*OAP*) el mensaje que envía el emisor a cada uno de los destinatarios es diferente. Recíprocamente, en el patrón *AllToOne* (*AO*), uno de los procesadores recibe *p-1* mensajes de tamaño m_{AO} de los otros *p-1* procesadores. Bajo el patrón *AllToAll (AA),* cada uno de los p procesadores envía *p-1* mensajes de tamaño m_{AA} a los otros *p-1* procesadores. Si medimos la masa de las comunicaciones mediante el operador $@ = +$, tenemos que los cinco patrones $\Pi - \{OA\} = \{E, PP, OAP, AO, AA\}$ dan lugar a la misma *h*-relación si se cumplen las siguientes igualdades:

$$h = m_{PP} = 2 * m_E = (p-1) * m_{OAP} = (p-1) * m_{AO} = 2*(p-1)*m_{AA} \qquad (3.13)$$

Si tomamos el operador $@ = \max$, las igualdades que deben cumplirse para obtener la misma *h*-relación son:

$$h = m_{PP} = m_E = (p-1) * m_{OAP} = (p-1) * m_{AO} = (p-1)*m_{AA} \qquad (3.14)$$

En lo que sigue la definición de *h*-relación corresponde al operador $@ = +$. Una vez fijado el operador para la arquitectura de red específica, la hipótesis de la *h*-relación establece que si la **secuencia de igualdades anterior se cumple, los tiempos de comunicación invertidos en los cinco patrones** *{E, PP, OAP, AO, AA}* **deberían ser**

similares. El patrón *OneToall* constituye una excepción a esta afirmación. En general, la emisión de un *OneToAll* tiene lugar según un árbol (dependiente de la implementación sobre una librería) y aunque el procesador que más mensajes envía es el procesador raíz de la emisión, este suele enviar menos de *p-1* mensajes y por tanto la *h*-relación no es necesariamente $(p-1)*m_{OA}$. El número máximo de mensajes dependerá de las optimizaciones realizadas en la implementación de las funciones PVM o MPI utilizadas. Por tanto, la mejora de la conducta del *OneToAll* con respecto al patrón *OneToAll Personalizado* para una *h*-relación fija nos da una medida de las optimizaciones introducidas en la implementación. Es por esta razón que resulta de interés estudiar su comportamiento con $m_{OA} = h/(p-1)$.

Los resultados se han estructurado en varias tablas para cada patrón y arquitectura. Las primeras tablas contienen el tiempo medio obtenido para los mensajes cuyo tamaño se refleja en la primera columna. En la primera fila aparecen en orden de izquierda a derecha, el número de procesadores utilizados en ese patrón, el tiempo medio (*MEDIA)* observado de las 100 ejecuciones para cada tamaño de mensaje y número de procesadores según:

$$T_{MEDIO}(h) = \sum_{\rho \in \Pi} (\sum_{i \in H\rho} Tiempo_{\rho,i}(h) / |H_\rho|) / |\Pi| \qquad (3.15)$$

donde H_ρ es el conjunto de procesadores considerados para el patrón de comunicaciones ρ. Aplicando regresión lineal en *h* a los valores $T_{MEDIO}(h)$ se obtienen los valores de la pendiente *g* y el término independiente *L* de la recta.

La etiqueta "MODELO" presenta el valor ofrecido por el modelo de costes según la fórmula:

$$Time_{\rho,i}(h) = g_\rho(i) * h + L_\rho \qquad (3.16)$$

y los errores medio (*ErrMed*) y máximo (*ErrMax*) quedan definidos por:

$$ErrMed = 100*((\sum_{i=1,|H|}(Tiempo_{\rho i}(h) - (L_\Pi + g_\Pi * h))/|H| / (\sum_{i=1,|H|} Tiempo_{\rho i}(h)/|H|)) \quad (3.17)$$

$$ErrMax = 100*(max_i \{| Tiempo_{\rho i}(h) - (L_\Pi + g_\Pi h)| , i \in H_\Pi\} / min\{ Tiempo_{\rho i}(h)\}) \quad (3.18)$$

donde $Tiempo_{\Pi,i}(h)$ denota el tiempo invertido cuando *i* procesadores se comunican siguiendo el patrón $\rho \in \Pi$. El índice *i* varía entre $H_\Pi = \{2,4,6,8,16,24\}$ para los patrones *Exchange y PingPong, y* entre $H_\Pi = \{4,6,8,16,24\}$ para el resto de patrones: *OneToAll*, *OneToAll Personalizado*, *AllToOne* y *AllToAll*.

Utilizando ajuste lineal a partir de los datos de tamaños de mensajes y tiempos en concluir el patrón, obtenemos los valores de los parámetros *L* y *g* de la fórmula *(3.16)*. La última tabla de cada sección muestra los valores de *L* y *g* para cada uno de los patrones y diferentes arquitecturas.

3.9.1 Estudio de la *h*-relación para multicomputadoras y redes estaciones de trabajo, usando PVM

En este experimento [Rod98c] se utilizaron cuatro arquitecturas de redes diferentes: IBM SP2, ORIGIN 2000, estaciones de trabajo conectadas por par trenzado a un switch de altas prestaciones que hemos denominado LAN UTP y estaciones de trabajo conectadas por cable coaxial, que hemos denominado LAN COA.

3.9.1.1 Patrón de comunicaciones *Exchange*

El patrón de comunicaciones Exchange o intercambio se caracteriza por la existencia de un conjunto S de procesadores del conjunto total existente H, tal que cada procesador $p \in S$, envía su mensaje a un procesador diferente $d(p) \in H$ y además $d(d(p)) = p$ para cualquier procesador en H. El patrón se realiza entre parejas de procesadores, donde ambos envían y reciben en paralelo mensajes del otro procesador.

La función *pvm_send()* se ha utilizado para enviar los datos y la función *pvm_recv()* para recibir los mismos. En este caso la *h*-relación existente en este patrón de comunicaciones es $h = 2^* m_E$.

La tabla 3.1 muestra los valores obtenidos para la IBM SP2. La tabla 3.2 muestra los valores obtenidos para la ORIGIN 2000. La tabla 3.3 muestra los valores obtenidos para la LAN UTP. La tabla 3.4 muestra los valores obtenidos para la LAN COA.

Exchange	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000102	0.000117	0.000116	0.000120	0.000114	0.000088	22.65	31.39
420	0.000112	0.000119	0.000111	0.000115	0.000114	0.000094	17.60	22.40
840	0.000124	0.000120	0.000118	0.000121	0.000121	0.000106	11.84	14.87
1680	0.000151	0.000150	0.000150	0.000160	0.000153	0.000131	14.19	19.28
3360	0.000223	0.000210	0.000212	0.000221	0.000217	0.000180	16.71	20.32
6720	0.000303	0.000302	0.000307	0.000310	0.000306	0.000279	8.73	10.32
13440	0.000472	0.000474	0.000464	0.000471	0.000470	0.000476	1.19	2.55
26880	0.000834	0.000913	0.000833	0.000839	0.000855	0.000870	4.29	5.18
53760	0.001545	0.001546	0.001556	0.001565	0.001553	0.001658	6.75	7.30
107520	0.003091	0.003131	0.003094	0.003103	0.003105	0.003234	4.16	4.62
215040	0.006410	0.006437	0.006412	0.006412	0.006418	0.006386	0.50	0.80
430080	0.012678	0.012639	0.012679	0.012738	0.012684	0.012690	0.24	0.40
860160	0.025107	0.025261	0.025275	0.026148	0.025448	0.025298	1.08	3.39
1720320	0.050487	0.050225	0.050209	0.050868	0.050447	0.050513	0.48	0.71

Tabla 3.1 Tiempos del patrón de comunicaciones Exchange para la IBM SP2.

Exchange	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000042	0.000126	0.000154	0.000158	0.000120	0.000021	82.37	325.81
420	0.000043	0.000042	0.000041	0.000043	0.000042	0.000023	44.84	48.03
840	0.000048	0.000046	0.000045	0.000047	0.000047	0.000028	40.65	45.34
1680	0.000056	0.000054	0.000053	0.000055	0.000055	0.000036	33.62	37.40
3360	0.000062	0.000060	0.000058	0.000061	0.000060	0.000053	11.46	14.92
6720	0.000081	0.000078	0.000076	0.000080	0.000079	0.000088	11.33	15.36
13440	0.000134	0.000130	0.000123	0.000127	0.000129	0.000156	21.66	27.10
26880	0.000245	0.000237	0.000228	0.000227	0.000234	0.000294	25.36	29.36
53760	0.000572	0.000563	0.000546	0.000543	0.000556	0.000568	2.54	4.66
107520	0.001143	0.001104	0.001078	0.001058	0.001096	0.001118	3.15	5.63
215040	0.002322	0.002208	0.002143	0.002099	0.002193	0.002216	3.47	5.58
430080	0.004579	0.004443	0.004295	0.004230	0.004387	0.004413	2.83	4.33
860160	0.009206	0.008930	0.008654	0.008485	0.008819	0.008807	2.83	4.70
1720320	0.018332	0.017733	0.017325	0.017021	0.017603	0.017596	2.44	4.32

Tabla 3.2 Tiempos del patrón de comunicaciones Exchange para la ORIGIN 2000.

Exchange	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.004083	0.004039	0.004050	0.005753	0.004481	0.002275	49.23	86.10
420	0.004328	0.004290	0.004318	0.005126	0.004581	0.002589	42.05	59.14
840	0.004757	0.004675	0.004688	0.006340	0.005257	0.003216	36.12	66.82
1680	0.005767	0.005613	0.005636	0.007742	0.006374	0.004471	26.97	58.28
3360	0.007461	0.007410	0.007475	0.009226	0.008032	0.006980	11.37	30.32
6720	0.010673	0.010720	0.010668	0.013278	0.011557	0.011997	11.27	12.46
13440	0.020793	0.020692	0.020772	0.024936	0.022140	0.022033	7.62	14.03
26880	0.041995	0.041505	0.041693	0.046496	0.043332	0.042105	3.18	10.58
53760	0.079268	0.078629	0.078360	0.086996	0.081631	0.082248	4.67	6.06
107520	0.158583	0.157626	0.158151	0.173081	0.163097	0.162534	3.65	6.69
215040	0.313408	0.310319	0.310306	0.335242	0.319656	0.323107	3.71	4.13
430080	0.622516	0.618837	0.620371	0.674435	0.638596	0.644253	3.96	4.88
860160	1.247704	1.236556	1.236180	1.358974	1.281078	1.286544	4.13	5.86
1720320	2.500369	2.480668	2.487641	2.745955	2.575664	2.571126	4.07	7.05

Tabla 3.3 Tiempos del patrón de comunicaciones Exchange para la LAN UTP.

Exchange	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.004241	0.004572	0.004535	0.004563	0.004478	0.002109	52.90	58.08
420	0.004725	0.004694	0.004852	0.004732	0.004751	0.002754	42.03	42.13
840	0.004489	0.004926	0.006264	0.005417	0.005274	0.004045	23.30	30.56
1680	0.004983	0.005575	0.006321	0.008458	0.006334	0.006627	19.07	36.75
3360	0.006177	0.007666	0.012017	0.011317	0.009294	0.011790	28.07	90.87
6720	0.008298	0.014550	0.020641	0.033285	0.019194	0.022116	44.32	166.52
13440	0.017394	0.026241	0.042975	0.068369	0.038745	0.042769	43.69	147.18
26880	0.036500	0.058154	0.091423	0.132507	0.079646	0.084074	40.58	132.69
53760	0.068403	0.111387	0.186412	0.284509	0.162678	0.166685	44.74	172.25
107520	0.140615	0.226422	0.367615	0.573673	0.327081	0.331906	43.89	171.94
215040	0.275291	0.460248	0.760399	1.150537	0.661619	0.662349	44.41	177.34
430080	0.557598	0.936357	1.528648	2.323600	1.336551	1.323234	44.11	179.41
860160	1.120757	1.843690	3.007001	4.668633	2.660020	2.645004	44.28	180.56
1720320	2.216631	3.671321	5.998695	9.226720	5.278342	5.288545	44.23	177.66

Tabla 3.4 Tiempos del patrón de comunicaciones Exchange para la LAN COA.

Exchange	ORIGIN	IBM SP2	LAN UTP	LAN COA
L_E	1.90E-05	8.18E-05	1.96E-03	1.46E-03
g_E	1.02E-08	2.93E-08	1.49E-06	3.07E-06

Tabla 3.5 Valores de g_E y L_E para el patrón Exchange.

3.9.1.2 Patrón de comunicaciones *PingPong*.-

El patrón "PingPong" se realiza entre parejas de procesadores, donde el procesador fuente envía un mensaje de tamaño m, el procesador fuente lo recibe y lo vuelve a enviar al procesador origen. En este caso, siguiendo la notación dada en el punto anterior, cada procesador $p \in S$, envía su mensaje a un procesador diferente $d(p) \in H$ y $S \cap d(S) = \varnothing$.

El procesador fuente ejecuta primero la función *pvm_send()* seguida de la función *pvm_recv()*. El procesador destino ejecuta las funciones de forma inversa, es decir, primero recibe los datos del procesador fuente con la función *pvm_recv()* y luego devuelve los datos al procesador fuente con la función *pvm_send()*. El patrón de comunicaciones PingPong da lugar a una h-relación de $h = m_{pp}$.

PingPong	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000108	0.000110	0.000112	0.000113	0.000111	0.000122	35.18	13.17
420	0.000113	0.000114	0.000115	0.000115	0.000114	0.000131	49.05	16.21
840	0.000140	0.000142	0.000139	0.000139	0.000140	0.000150	21.56	7.57
1680	0.000190	0.000188	0.000189	0.000190	0.000189	0.000186	5.44	2.17
3360	0.000268	0.000262	0.000262	0.000259	0.000263	0.000259	6.08	3.59
6720	0.000381	0.000378	0.000377	0.000386	0.000381	0.000404	21.40	7.24
13440	0.000646	0.000650	0.000650	0.000648	0.000649	0.000695	23.47	7.66
26880	0.001157	0.001168	0.001167	0.001197	0.001172	0.001278	30.84	10.44
53760	0.002234	0.002244	0.002231	0.002248	0.002239	0.002442	29.79	9.48
107520	0.004816	0.004796	0.004818	0.004811	0.004810	0.004772	2.59	0.96
215040	0.009411	0.009443	0.009429	0.009415	0.009425	0.009430	0.40	0.21
430080	0.018712	0.018795	0.018750	0.018752	0.018752	0.018748	0.46	0.25
860160	0.038978	0.037263	0.037303	0.038926	0.038118	0.037382	5.72	4.28
1720320	0.074322	0.074281	0.074265	0.074292	0.074290	0.074652	1.58	0.52

Tabla 3.6 Tiempos del patrón de comunicaciones PingPong para la IBM SP2.

PingPong	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000035	0.000112	0.000139	0.000140	0.000107	-0.000019	118.17	455.28
420	0.000037	0.000038	0.000039	0.000037	0.000038	-0.000015	140.63	146.85
840	0.000046	0.000047	0.000048	0.000047	0.000047	-0.000007	115.56	120.24
1680	0.000048	0.000049	0.000051	0.000049	0.000049	0.000009	82.26	88.04
3360	0.000069	0.000071	0.000072	0.000069	0.000070	0.000041	41.87	45.16
6720	0.000109	0.000110	0.000113	0.000107	0.000110	0.000105	4.29	7.44
13440	0.000196	0.000196	0.000200	0.000189	0.000195	0.000233	19.56	23.51
26880	0.000451	0.000464	0.000475	0.000454	0.000461	0.000490	6.34	8.70
53760	0.000920	0.000958	0.000982	0.000947	0.000952	0.001004	5.47	9.11
107520	0.001945	0.002004	0.002030	0.001945	0.001981	0.002031	2.53	4.43
215040	0.003953	0.004082	0.004188	0.004038	0.004065	0.004086	1.76	3.35
430080	0.007933	0.008167	0.008397	0.008132	0.008157	0.008194	1.70	3.29
860160	0.015649	0.016203	0.016815	0.016197	0.016216	0.016412	2.45	4.88
1720320	0.031708	0.033155	0.034191	0.032797	0.032963	0.032848	2.15	4.24

Tabla 3.7 Tiempos del patrón de comunicaciones PingPong para la ORIGIN.

PingPong	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.003615	0.003850	0.003799	0.003595	0.003715	0.001551	58.26	63.96
420	0.004029	0.004240	0.004334	0.004374	0.004244	0.001967	53.67	59.75
840	0.004886	0.005107	0.005193	0.004810	0.004999	0.002798	44.02	49.79
1680	0.006343	0.006609	0.006597	0.006699	0.006562	0.004462	32.01	35.27
3360	0.008242	0.008568	0.008422	0.008661	0.008473	0.007788	8.08	10.59
6720	0.015308	0.015956	0.015958	0.015953	0.015794	0.014442	8.56	9.91
13440	0.030016	0.030228	0.030463	0.030949	0.030414	0.027749	8.76	10.66
26880	0.055888	0.055809	0.056544	0.056722	0.056241	0.054362	3.34	4.23
53760	0.106237	0.110417	0.111248	0.111951	0.109963	0.107590	2.77	4.11
107520	0.213340	0.215580	0.217377	0.220280	0.216644	0.214045	1.36	2.92
215040	0.416027	0.423370	0.428908	0.429014	0.424330	0.426954	1.09	2.63
430080	0.832076	0.842766	0.853856	0.850265	0.844741	0.852774	1.01	2.49
860160	1.640725	1.680611	1.704721	1.698164	1.681055	1.704412	1.40	3.88
1720320	3.624398	3.352483	3.332171	3.376577	3.421407	3.407690	2.77	6.50

Tabla 3.8 Tiempos del patrón de comunicaciones PingPong para la LAN UTP.

PingPong	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.004208	0.003325	0.003147	0.003732	0.004804	0.006523	60.78	96.18
420	0.004410	0.003508	0.003287	0.004052	0.005086	0.007190	66.37	104.95
840	0.004894	0.004019	0.003633	0.004624	0.005723	0.008523	73.92	112.07
1680	0.005911	0.005133	0.004414	0.008037	0.007832	0.011190	67.88	118.00
3360	0.007825	0.007392	0.005805	0.015361	0.012128	0.016524	61.25	123.54
6720	0.014246	0.015063	0.012238	0.027919	0.023155	0.027192	44.00	90.87
13440	0.026713	0.030760	0.024600	0.057951	0.046675	0.048527	39.06	81.66
26880	0.049722	0.057975	0.045565	0.112579	0.088614	0.091198	39.98	83.42
53760	0.099180	0.115685	0.094405	0.224273	0.177848	0.176540	37.68	78.00
107520	0.193503	0.226168	0.184138	0.443787	0.349199	0.347224	38.26	79.44
215040	0.382452	0.448265	0.364396	0.884982	0.693365	0.688591	38.47	80.05
430080	0.762119	0.890976	0.725900	1.773765	1.384253	1.371326	38.60	79.94
860160	1.513886	1.775002	1.437657	3.518486	2.748344	2.736796	38.80	80.78
1720320	3.020657	3.532737	2.797255	7.023510	5.458053	5.467735	39.43	81.01

Tabla 3.9 Tiempos del patrón de comunicaciones PingPong para la LAN COA.

PingPong	ORIGIN	IBM SP2	LAN UTP	LAN COA
L_{pp}	-2.34E-05	1.13E-04	1.13E-03	5.86E-03
g_{pp}	1.91E-08	4.33E-08	1.98E-06	3.17E-06

Tabla 3.10. Valores medios de g y L para el patrón injectivo PingPong.

El valor negativo del término independiente para la arquitectura ORIGIN es debido a la influencia de los tamaños grandes.

3.9.1.3 Patrón de comunicaciones *OneToAll*

Este experimento mide la capacidad de un procesador para enviar datos a p procesadores. El procesador fuente realiza p llamadas a la función *pvm_send()* y cada procesador destino realiza una llamada a la función *pvm_recv()*. Como se observa en las tablas, para obtener un ajuste más preciso, se podría considerar que en lugar de una sola recta, ahora la aproximación es lineal a trozos. Por razones de simplicidad y de utilización de valores asintóticos, hemos considerado sólo el ajuste a una única recta.

Esta función como se ha descrito es en realidad una emisión de uno a todos personalizada. En PVM no existe una función eficiente de OneToAll como se demostró en [Rod96].

Menos en el caso de la LAN UTP, en el resto de arquitecturas se aprecia una disminución del tiempo según aumentamos el número de procesadores. Este fenómeno es debido al incremento de paralelismo al aumentar el número de receptores y al efecto descrito en el capítulo 2 y mostrado en [Rod98a]. En la LAN UTP, el switch de interconexión por el que todo mensaje debe pasar es el cuello de botella.

En este caso, la h-relación queda establecida por $h = (p-1) * m_{OA}$.

OneToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000134	0.000184	0.000259	0.000192	0.000077	60.08	135.98
420	0.000138	0.000185	0.000234	0.000186	0.000084	54.69	108.60
840	0.000166	0.000193	0.000251	0.000203	0.000099	51.40	91.67
1680	0.000186	0.000222	0.000282	0.000230	0.000128	44.26	82.68
3360	0.000230	0.000265	0.000323	0.000273	0.000187	31.43	59.14
6720	0.000344	0.000349	0.000383	0.000359	0.000305	15.10	22.81
13440	0.000555	0.000559	0.000615	0.000576	0.000540	6.37	13.58
26880	0.000998	0.000975	0.001022	0.000998	0.001010	1.96	3.57
53760	0.001871	0.001837	0.001861	0.001856	0.001950	5.05	6.16
107520	0.003752	0.003636	0.003597	0.003662	0.003831	4.62	6.50
215040	0.007648	0.007405	0.007223	0.007425	0.007592	2.75	5.11
430080	0.015343	0.014755	0.014700	0.014933	0.015115	2.24	2.82
860160	0.030850	0.02979	0.029505	0.030048	0.030160	1.90	2.34
1720320	0.061907	0.060126	0.059125	0.060386	0.060251	1.60	2.80

Tabla 3.11 Tiempos del patrón de comunicaciones OneToAll para la IBM SP2.

OneToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000099	0.000140	0.000172	0.000137	0.000037	73.20	136.65
420	0.000102	0.000146	0.000171	0.000140	0.000039	71.80	129.04
840	0.000108	0.000147	0.000181	0.000145	0.000045	69.23	126.18
1680	0.000119	0.000163	0.000192	0.000158	0.000055	64.94	114.79
3360	0.000126	0.000182	0.000213	0.000174	0.000077	55.80	108.13
6720	0.000149	0.000189	0.000248	0.000195	0.000119	38.83	86.26
13440	0.000197	0.000235	0.000267	0.000233	0.000205	14.32	31.52
26880	0.000296	0.000330	0.000354	0.000327	0.000376	15.03	26.95
53760	0.000628	0.000528	0.000552	0.000569	0.000718	26.03	35.89
107520	0.001470	0.001261	0.000962	0.001231	0.001401	17.55	45.63
215040	0.002986	0.002870	0.002551	0.002802	0.002768	6.39	8.55
430080	0.005897	0.005425	0.005203	0.005508	0.005502	4.66	7.60
860160	0.011792	0.010042	0.009349	0.010394	0.010969	10.81	17.33
1720320	0.023795	0.022412	0.020398	0.022202	0.021905	5.86	9.27

Tabla 3.12 Tiempos del patrón de comunicaciones OneToAll para la ORIGIN.

OneToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.009154	0.012591	0.019256	0.013667	0.009897	31.21	102.24
420	0.009309	0.012655	0.019340	0.013768	0.010194	30.25	98.25
840	0.009620	0.013619	0.019706	0.014315	0.010787	30.08	92.71
1680	0.010178	0.013586	0.018657	0.014140	0.011974	23.79	65.66
3360	0.011852	0.013577	0.021539	0.015656	0.014348	22.27	60.68
6720	0.015540	0.016785	0.022926	0.018417	0.019095	17.55	24.65
13440	0.025221	0.023448	0.029586	0.026085	0.028590	12.15	21.93
26880	0.040654	0.043918	0.046733	0.043768	0.047580	8.71	17.04
53760	0.071688	0.076549	0.090096	0.079444	0.085559	11.50	19.35
107520	0.135787	0.167558	0.170056	0.157800	0.161518	8.51	18.95
215040	0.267081	0.285985	0.355494	0.302853	0.313436	12.75	17.36
430080	0.542378	0.588276	0.744115	0.624923	0.617271	12.31	23.39
860160	1.065558	1.160343	1.481935	1.235945	1.224942	12.97	24.12
1720320	2.166692	2.318882	2.818493	2.434689	2.440284	10.59	17.46

Tabla 3.13 Tiempos del patrón de comunicaciones OneToAll para la LAN UTP.

OneToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.009452	0.012552	0.017791	0.013265	0.009653	28.24	86.10
420	0.009474	0.012360	0.017297	0.013044	0.009894	26.29	78.14
840	0.009833	0.012515	0.017379	0.013242	0.010375	24.38	71.23
1680	0.010140	0.012705	0.017780	0.013542	0.011337	22.17	63.54
3360	0.011198	0.012994	0.018309	0.014167	0.013260	17.36	45.09
6720	0.014586	0.015100	0.018732	0.016139	0.017108	12.71	17.29
13440	0.024415	0.021381	0.023605	0.023134	0.024803	7.22	16.01
26880	0.039085	0.037542	0.033738	0.036788	0.040194	9.26	19.14
53760	0.069256	0.066290	0.063088	0.066211	0.070975	7.20	12.50
107520	0.131975	0.128281	0.118535	0.126264	0.132538	4.97	11.81
215040	0.265853	0.246512	0.232158	0.248174	0.255663	5.75	10.12
430080	0.548680	0.505375	0.471055	0.508370	0.501914	5.32	9.93
860160	1.070764	1.000478	0.944682	1.005308	0.994414	4.38	8.08
1720320	2.091705	1.983914	1.846068	1.973896	1.979416	4.22	7.22

Tabla 3.14. Tiempos del patrón de comunicaciones OneToAll para la LAN COA.

OneToAll	ORIGIN	IBM SP2	LAN UTP	LAN COA
L_{OA}	3.40E-05	6.94E-05	9.60E-03	9.41E-03
g_{OA}	1.27E-08	3.50E-08	1.41E-06	1.15E-06

Tabla 3.15 Valores medios de *g* y *L* para el patrón OneToAll.

3.9.1.4 Patrón de comunicaciones *AllToOne*

En este experimento se mide la capacidad del procesador raíz, para recibir *p-1* mensajes de tamaño m_{AO} de *p-1* procesadores. En este caso, la *h*-relación queda establecida por $h = (p-1) * m_{AO}$. El procesador destino ejecuta la rutina *pvm_recv() p-1* veces recibiendo de los *p-1* procesadores sin orden establecido. En este caso, de nuevo aproximamos por un ajuste lineal para obtener los valores de *g* y *L*.

AllToOne	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000109	0.00016	0.000219	0.000163	0.000086	47.06	121.91
420	0.000099	0.000154	0.000218	0.000157	0.000093	40.69	126.14
840	0.000119	0.000165	0.000224	0.000169	0.000107	36.75	98.23
1680	0.000141	0.000204	0.000272	0.000206	0.000135	34.32	97.10
3360	0.000193	0.000274	0.000328	0.000265	0.000191	27.90	70.96
6720	0.000308	0.000375	0.000449	0.000377	0.000303	19.70	47.41
13440	0.000497	0.000607	0.000683	0.000596	0.000527	14.89	31.42
26880	0.000888	0.000958	0.001133	0.000993	0.000975	8.78	17.84
53760	0.001779	0.001797	0.001891	0.001822	0.001870	3.38	5.12
107520	0.003430	0.003536	0.003719	0.003562	0.003661	3.87	6.73
215040	0.007122	0.007052	0.007067	0.007080	0.007243	2.29	2.70
430080	0.014015	0.014511	0.014139	0.014222	0.014406	1.79	2.79
860160	0.028319	0.02864	0.028715	0.028558	0.028733	0.61	1.46
1720320	0.057415	0.05756	0.057669	0.057548	0.057388	0.28	0.49

Tabla 3.16 Tiempos del patrón de comunicaciones AllToOne para la IBM SP2.

AllToOne	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000081	0.000133	0.000213	0.000142	0.000092	40.58	149.70
420	0.000041	0.000093	0.000219	0.000118	0.000093	50.52	306.49
840	0.000045	0.000088	0.000212	0.000115	0.000097	50.88	256.57
1680	0.000052	0.000099	0.000191	0.000114	0.000103	41.80	169.33
3360	0.000059	0.000108	0.000193	0.000120	0.000116	39.38	130.91
6720	0.000076	0.000109	0.000232	0.000139	0.000141	45.18	119.23
13440	0.000119	0.000148	0.000204	0.000157	0.000193	27.52	61.88
26880	0.000202	0.000219	0.000281	0.000234	0.000295	26.13	46.11
53760	0.000445	0.000377	0.000402	0.000408	0.000500	22.58	32.66
107520	0.000992	0.000886	0.000745	0.000874	0.000910	10.34	22.17
215040	0.001724	0.001813	0.001911	0.001816	0.001730	4.95	10.49
430080	0.003163	0.003514	0.003445	0.003374	0.003370	4.21	6.55
860160	0.006159	0.006936	0.007060	0.006718	0.006650	5.89	7.97
1720320	0.012041	0.013373	0.014098	0.013171	0.013210	5.62	9.71

Tabla 3.17 Tiempos del patrón de comunicaciones AllToOne para la ORIGIN.

AllToOne	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.004198	0.009990	0.014752	0.009647	0.004913	54.01	234.36
420	0.006053	0.009834	0.014074	0.009987	0.005144	48.49	147.53
840	0.006334	0.010380	0.014733	0.010482	0.005605	46.52	144.10
1680	0.006471	0.010221	0.017883	0.011525	0.006528	43.69	175.47
3360	0.007169	0.010601	0.016460	0.011410	0.008374	33.65	112.80
6720	0.010228	0.011458	0.015997	0.012561	0.012064	16.92	38.45
13440	0.019678	0.016877	0.018224	0.018260	0.019446	7.34	15.22
26880	0.034186	0.030182	0.027941	0.030770	0.034208	11.18	22.43
53760	0.063907	0.058090	0.059176	0.060391	0.063734	5.73	9.72
107520	0.126072	0.116088	0.114966	0.119042	0.122785	4.98	6.80
215040	0.252930	0.230256	0.222802	0.235329	0.240887	5.77	8.12
430080	0.501054	0.455216	0.469320	0.475197	0.477090	3.76	5.26
860160	1.001814	0.913264	0.901167	0.938748	0.949498	4.86	5.81
1720320	2.023263	1.865602	1.814855	1.901240	1.894313	4.16	7.11

Tabla 3.18 Tiempos del patrón de comunicaciones AllToOne para la LAN UTP.

AllToOne	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.008623	0.014195	0.027502	0.016773	0.010178	45.50	200.91
420	0.008649	0.01438	0.022048	0.015026	0.010410	38.53	134.56
840	0.008994	0.014364	0.021218	0.014859	0.010875	35.25	115.00
1680	0.009238	0.014856	0.020161	0.014752	0.011804	31.58	90.46
3360	0.009346	0.018381	0.020772	0.016166	0.013663	33.29	76.07
6720	0.011719	0.015849	0.021609	0.016392	0.017380	23.22	48.31
13440	0.02261	0.019574	0.025929	0.022704	0.024815	12.57	26.78
26880	0.037742	0.038519	0.031973	0.036078	0.039685	10.00	24.12
53760	0.062933	0.063412	0.071883	0.066076	0.069426	7.55	10.32
107520	0.116365	0.129227	0.125891	0.123828	0.128906	4.27	10.78
215040	0.237088	0.244328	0.252914	0.244777	0.247867	2.64	4.55
430080	0.469879	0.470588	0.512352	0.484273	0.485788	3.97	5.65
860160	0.942344	0.926134	1.003458	0.957312	0.961631	3.36	4.52
1720320	1.880168	1.863216	2.006803	1.916729	1.913317	3.07	5.02

Tabla 3.19. Tiempos del patrón de comunicaciones AllToOne la LAN COA.

AllToOne	ORIGIN	IBM SP2	LAN UTP	LAN COA
L_{AO}	9.01E-05	7.91E-05	4.68E-03	9.95E-03
g_{AO}	7.63E-09	3.33E-08	1.10E-06	1.11E-06

Tabla 3.20 Valores medios de *g* y *L* para el patrón AllToOne.

3.9.1.5 Patrón de comunicaciones *AllToAll*

En este caso generalizamos los dos casos anteriores, saturando la red de interconexión. Este experimento da lugar a una gran cantidad de mensajes que viajan simultáneamente por la red. En este patrón cada procesador envía al resto *p-1* de procesadores *p-1* mensajes diferentes de tamaño m_{AA}. El procesador *i* envía de forma cíclica los mensajes al procesador *i+1, i+2,..., i-1*.

Ajustando los valores de tiempo/tamaño de mensaje a una recta se calculan los parámetros *g* y *L*.

AllToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000268	0.000481	0.000502	0.000417	0.000284	34.46	81.38
420	0.000215	0.000357	0.000484	0.000352	0.000290	31.78	90.03
840	0.000229	0.000364	0.000492	0.000362	0.000304	29.81	82.31
1680	0.000256	0.000377	0.000507	0.000380	0.000330	26.17	69.27
3360	0.000293	0.000428	0.000597	0.000439	0.000382	26.56	73.38
6720	0.000366	0.000522	0.000694	0.000527	0.000487	22.97	56.66
13440	0.000566	0.000676	0.000824	0.000689	0.000696	13.45	22.95
26880	0.000914	0.001062	0.001287	0.001088	0.001114	13.04	21.93
53760	0.001633	0.001736	0.002292	0.001887	0.001952	15.45	20.85
107520	0.003326	0.003544	0.003605	0.003492	0.003626	3.84	9.01
215040	0.006503	0.006785	0.007360	0.006883	0.006974	5.07	7.24
430080	0.013319	0.013701	0.014036	0.013685	0.013671	1.82	2.74
860160	0.026582	0.027143	0.026809	0.026845	0.027064	1.01	1.81
1720320	0.053176	0.053709	0.055054	0.053980	0.053851	1.25	2.26

Tabla 3.21 Tiempos del patrón de comunicaciones AllToAll para la IBM SP2.

AllToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.001111	0.002674	0.006063	0.003283	0.000374	88.61	512.07
420	0.000132	0.000233	0.000338	0.000234	0.000376	60.54	185.00
840	0.000145	0.000243	0.000332	0.000240	0.000381	58.64	162.58
1680	0.000146	0.000248	0.000339	0.000244	0.000390	59.54	166.99
3360	0.000175	0.000267	0.000365	0.000269	0.000408	51.66	133.12
6720	0.000195	0.000312	0.000411	0.000306	0.000444	45.18	127.82
13440	0.000232	0.000334	0.000496	0.000354	0.000517	45.99	122.77
26880	0.000322	0.000427	0.000529	0.000426	0.000662	55.39	105.58
53760	0.000517	0.000607	0.000709	0.000611	0.000952	55.86	84.20
107520	0.001024	0.001010	0.001091	0.001042	0.001533	47.16	51.77
215040	0.002201	0.002167	0.001898	0.002089	0.002694	28.99	41.95
430080	0.004750	0.004411	0.004102	0.004421	0.005017	13.47	22.30
860160	0.009604	0.009483	0.009643	0.009577	0.009662	0.89	1.88
1720320	0.019845	0.018900	0.019053	0.019266	0.018952	1.81	4.73

Tabla 3.22 Tiempos del patrón de comunicaciones AllToAll para la ORIGIN.

AllToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.012975	0.026789	0.037090	0.025618	0.017211	43.84	153.21
420	0.012967	0.023407	0.038707	0.025027	0.017538	42.10	163.25
840	0.013324	0.026954	0.039384	0.026554	0.018191	43.71	159.06
1680	0.012974	0.027740	0.038389	0.026368	0.019497	42.55	145.61
3360	0.015466	0.025159	0.039424	0.026683	0.022110	33.74	111.95
6720	0.018659	0.031400	0.040892	0.030317	0.027336	28.91	72.65
13440	0.026070	0.033565	0.044774	0.034803	0.037787	21.96	44.94
26880	0.051033	0.049182	0.059168	0.053128	0.058688	11.07	19.33
53760	0.090768	0.100254	0.080411	0.090478	0.100492	11.07	24.97
107520	0.165549	0.174256	0.170425	0.170077	0.184099	8.24	11.21
215040	0.324657	0.354342	0.340514	0.339838	0.351314	3.97	8.21
430080	0.642591	0.689675	0.700719	0.677662	0.685744	3.05	6.72
860160	1.321982	1.387265	1.406978	1.372075	1.354602	2.86	3.96
1720320	2.585067	2.730222	2.749624	2.688304	2.692320	2.51	4.15

Tabla 3.23 Tiempos del patrón de comunicaciones AllToAll para la LAN UTP.

AllToAll	2 parejas	3 parejas	4 parejas	MEDIA	MODELO	ErrMed	ErrMax
210	0.014022	0.025534	0.036291	0.025282	0.034064	40.61	142.93
420	0.012799	0.026175	0.039322	0.026099	0.034834	44.93	172.16
840	0.013797	0.026075	0.040812	0.026895	0.036373	46.25	163.63
1680	0.013317	0.027096	0.036821	0.025745	0.039451	53.24	196.25
3360	0.014838	0.027654	0.040188	0.027560	0.045607	65.48	207.37
6720	0.017505	0.030106	0.046908	0.031506	0.057921	83.84	230.88
13440	0.032661	0.064332	0.103367	0.066787	0.082547	44.38	152.74
26880	0.066498	0.155142	0.221194	0.147611	0.131799	40.20	134.43
53760	0.119690	0.218024	0.474655	0.270790	0.230303	45.21	204.15
107520	0.234295	0.420306	0.655296	0.436632	0.427311	32.67	97.31
215040	0.460304	0.833109	1.216841	0.836751	0.821328	30.61	85.92
430080	0.934033	1.589743	2.348122	1.623966	1.609361	29.43	79.09
860160	1.831158	3.216080	4.585032	3.210757	3.185428	28.91	76.43
1720320	3.662985	6.248297	9.041233	6.317505	6.337561	28.85	73.81

Tabla 3.24 Tiempos del patrón de comunicaciones AllToAll la LAN COA.

AllToAll	ORIGIN	IBM SP2	LAN UTP	LAN COA
L_{AA}	3.72E-04	2.77E-04	1.69E-02	3.33E-02
g_{AA}	1.08E-08	3.11E-08	1.56E-06	3.66E-06

Tabla 3.25 Valores medios de *g* y *L* para el patrón AllToAll.

3.9.1.6 Gráficas de variaciones de las *g*

En las siguientes gráficas 3.5, 3.6, 3.7 y 3.8 se muestran las variaciones del parámetro *g* respecto al número de procesadores. Cada gráfica representa una arquitectura diferente y para todas ellas se han representado todos los patrones (*Exchange, PingPong, OneToAll, AllToOne* y *AllToAll*) para 4, 6 y 8 procesadores. El valor de *g* viene expresado en segundos por cada byte enviado. Cada uno de los patrones ha sido dibujado con una etiqueta diferente que se respeta en cada arquitectura.

En la primera figura, se muestran los valores obtenidos para la IBM SP2. En ella se aprecia una alta invarianza de los patrones al aumentar el número de procesadores. La diferencia entre el patrón más rápido, el Exchange, y el más lento, el PingPong es de casi el doble. El resto de patrones se mantiene entre los dos anteriores, con una tendencia mayor hacia el Exchange.

Un efecto importante a destacar es el del patrón OneToAll ya estudiado en el capítulo anterior, donde según aumentamos el número de procesadores, disminuye el tiempo para la *h*-relación observada. Este fenómeno es debido principalmente al paralelismo introducido en este patrón así como a las eficientes implementaciones del mismo sobre las diferentes arquitecturas.

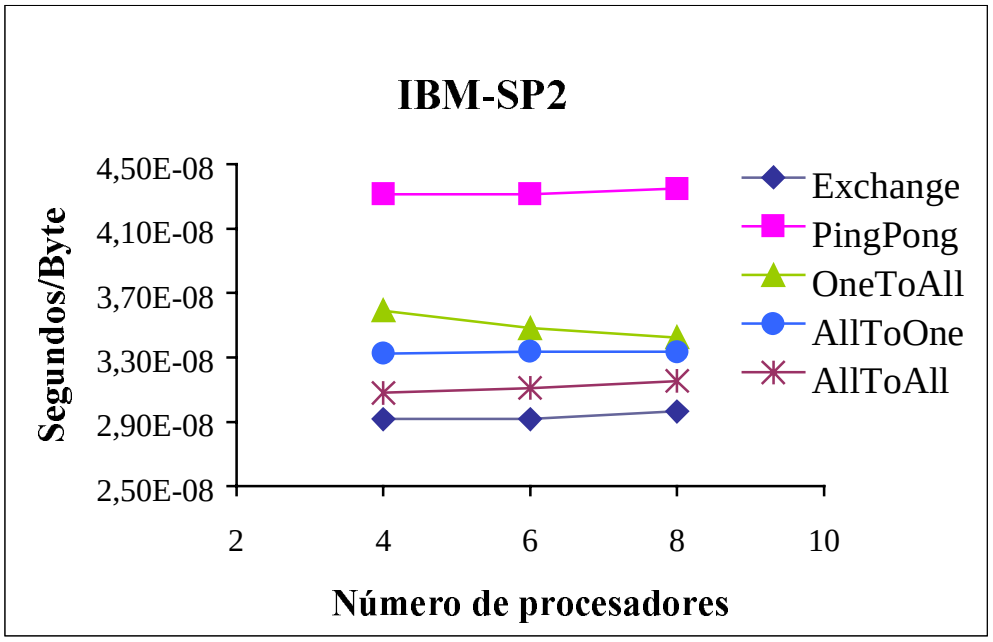

Figura 3. 5 Variaciones de *g* para la IBM SP2.

En la figura 3.6, se muestran los valores para la ORIGIN 2000. De nuevo los patrones más lento y más rápido son el PingPong y Exchange respectivamente. En este caso el rango entre uno y otro varía aproximadamente entre 7E-09 a 2E-8 segundos por cada entero enviado.

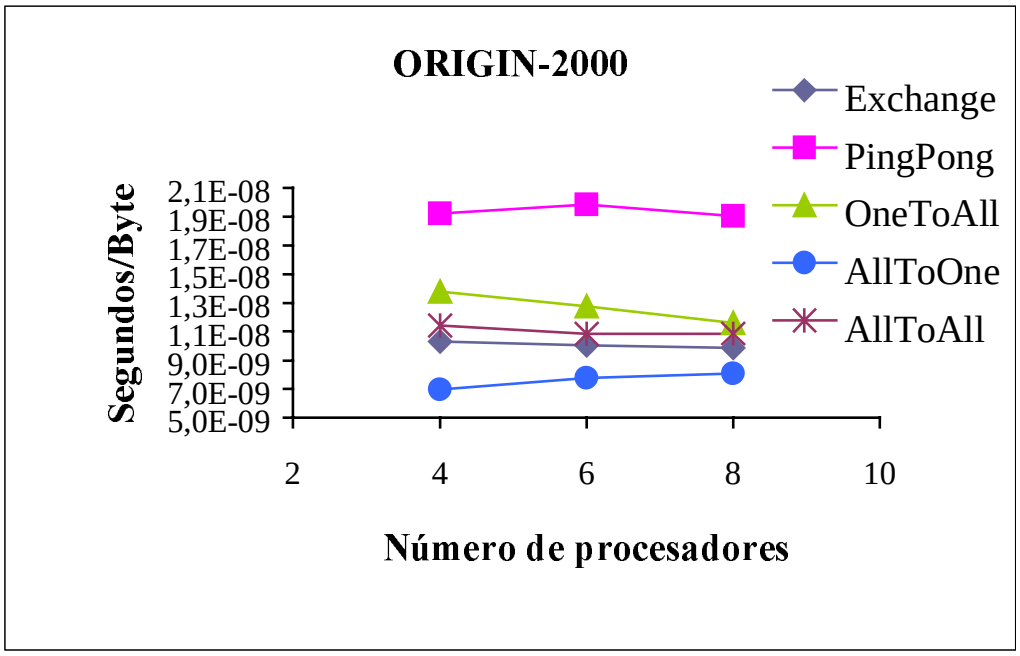

Figura 3. 6 Variaciones de *g* para la ORIGIN 2000.

En las figuras 3.7 y 3.8 comparamos las dos redes de área local Ethernet, una con las computadoras conectadas a un conmutador a través de par trenzado (LAN UTP) y la otra conecta las máquinas a través de cable coaxial (LAN COA). En el primer caso,

podemos observar el buen comportamiento de las curvas para todos los patrones excepto para el OneToAll. En este caso, se provoca una leve saturación de la red cuando aumentamos el número de procesadores. En el caso de la LAN COA, se satura de forma brusca al pasar de 4 a 6 procesadores y aún más de 6 a 8 procesadores para los patrones Exchange, PingPong y AllToAll, los más estresantes. Para los patrones OneToAll y AllToOne, la red se comporta de manera estable incluso disminuyendo mínimamente para el caso del OneToall.

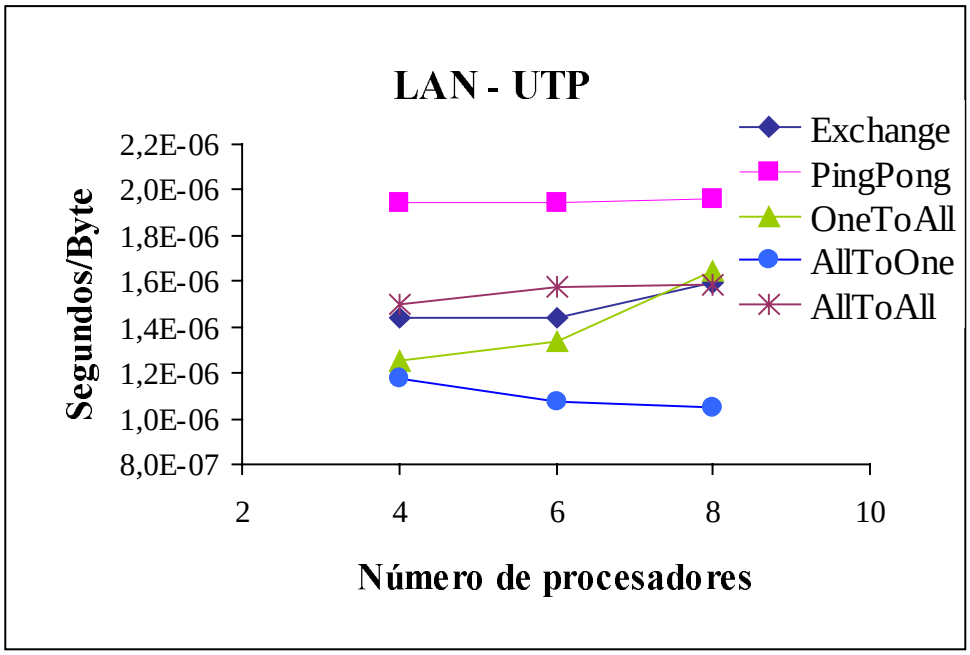

Figura 3. 7 Variaciones de *g* para la LAN UTP.

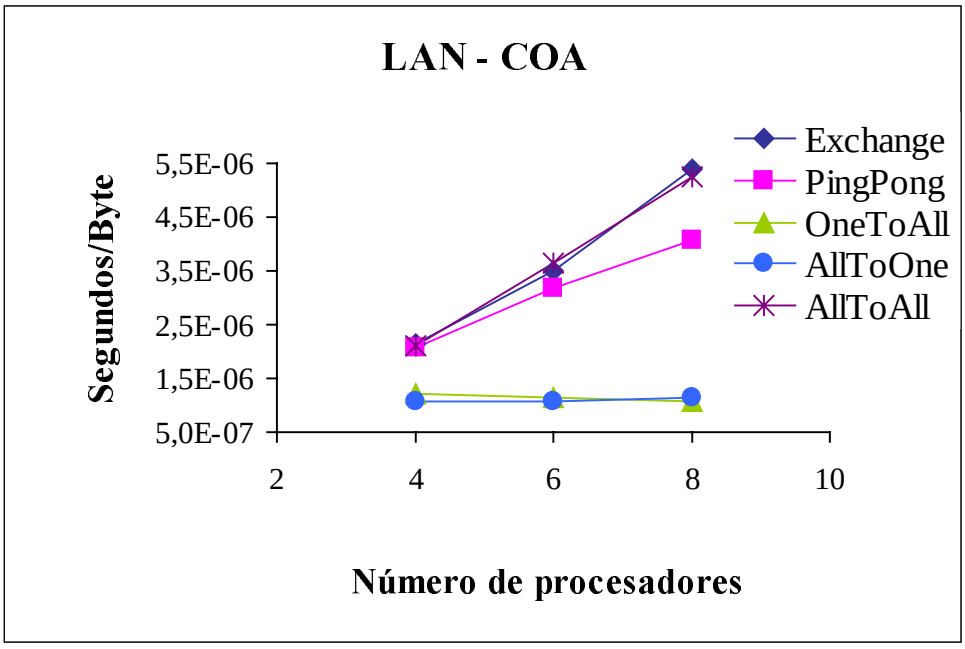

Figura 3. 8 Variaciones de *g* para la LAN COA.

El estudio del parámetro *g* sobre las diferentes arquitecturas a través de las *h*-relaciones nos proporciona una medida de saturación de la red, ya que el número de mensajes y la cantidad enviada en los mismos, es siempre constante.

3.9.1.7 Gráficas según tamaños y patrones

El estudio del apartado anterior nos ofrece una variación del parámetro *g*, según los patrones y el número de procesadores. En ese caso, los valores de *g* han sido calculados de manera asintótica y no ofrecen cuanto influye el tamaño del mensaje. Estudiamos la influencia del tamaño del mensaje en los diferentes patrones, es decir cómo varía el tiempo en función del tamaño utilizado para transmitir dicha *h*-relación.

La tabla 3.26 muestra para la IBM SP2 los valores observados para cada uno de los patrones, los valores medios de todos los patrones y los valores estimados a partir del modelo.

	Exchange	PingPong	OneToAll	AllToOne	AllToAll	MEDIA	MODELO
210	0.000114	0.000111	0.000192	0.000163	0.000417	0.000199	0.000131
420	0.000114	0.000114	0.000186	0.000157	0.000352	0.000185	0.000139
840	0.000121	0.000140	0.000203	0.000169	0.000362	0.000199	0.000153
1680	0.000153	0.000189	0.000230	0.000206	0.000380	0.000232	0.000182
3360	0.000217	0.000263	0.000273	0.000265	0.000439	0.000291	0.000240
6720	0.000306	0.000381	0.000359	0.000377	0.000527	0.000390	0.000355
13440	0.000470	0.000649	0.000576	0.000596	0.000689	0.000596	0.000587
26880	0.000855	0.001172	0.000998	0.000993	0.001088	0.001021	0.001049
53760	0.001553	0.002239	0.001856	0.001822	0.001887	0.001872	0.001974
107520	0.003105	0.004810	0.003662	0.003562	0.003492	0.003726	0.003825
215040	0.006418	0.009425	0.007425	0.007080	0.006883	0.007446	0.007525
430080	0.012684	0.018752	0.014933	0.014222	0.013685	0.014855	0.014926
860160	0.025448	0.038118	0.030048	0.028558	0.026845	0.029803	0.029728
1720320	0.050447	0.074290	0.060386	0.057548	0.053980	0.059330	0.059331

Tabla 3.26 Variación en los patrones para IBM SP2.

Las gráficas 3.9 y 3.10 muestran para la IBM SP2 los valores observados para los casos de pequeños y grandes tamaños. La linealidad para los valores de tamaños grandes es evidente, pero para tamaños pequeños de hasta 420 bytes de *h*-relación comienzan por disminuir y luego crece linealmente. La pendiente media de todos los patrones para valores grandes es de 3.44E-08 y para valores pequeños es de 3.27E-8. El término independiente es de 1.24E-04 y de 1.79E-04 para valores grandes y pequeños respectivamente.

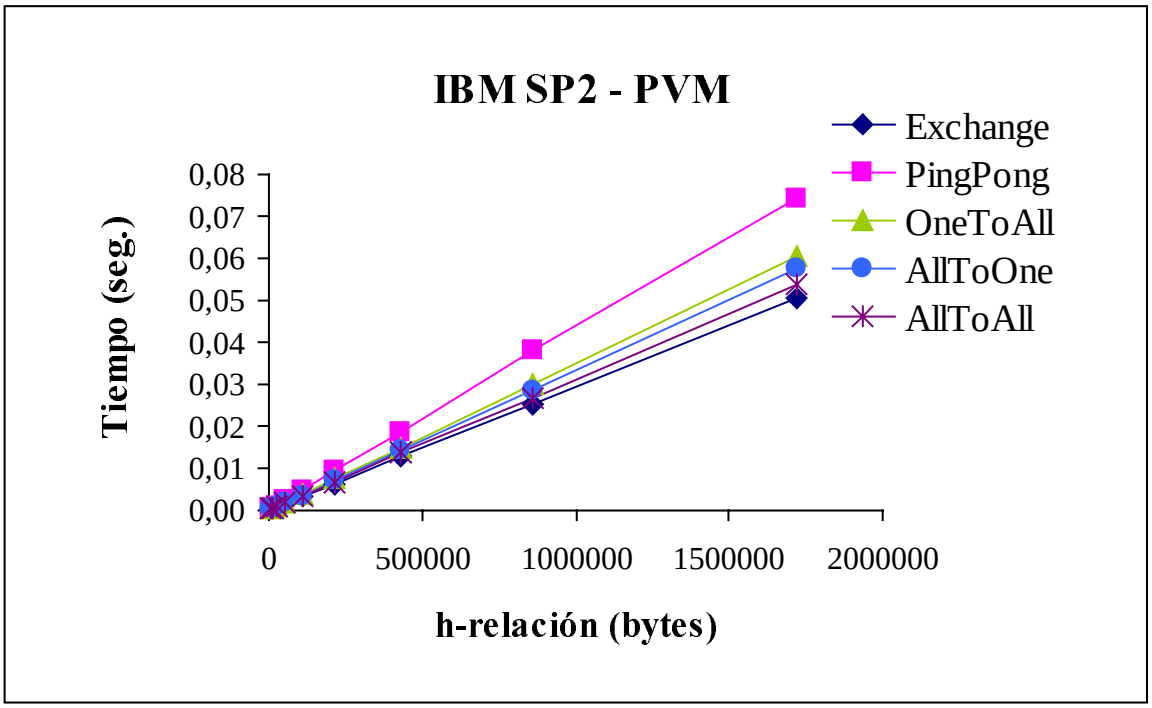

Figura 3. 9 Valores para los diferentes patrones con tamaños grandes.

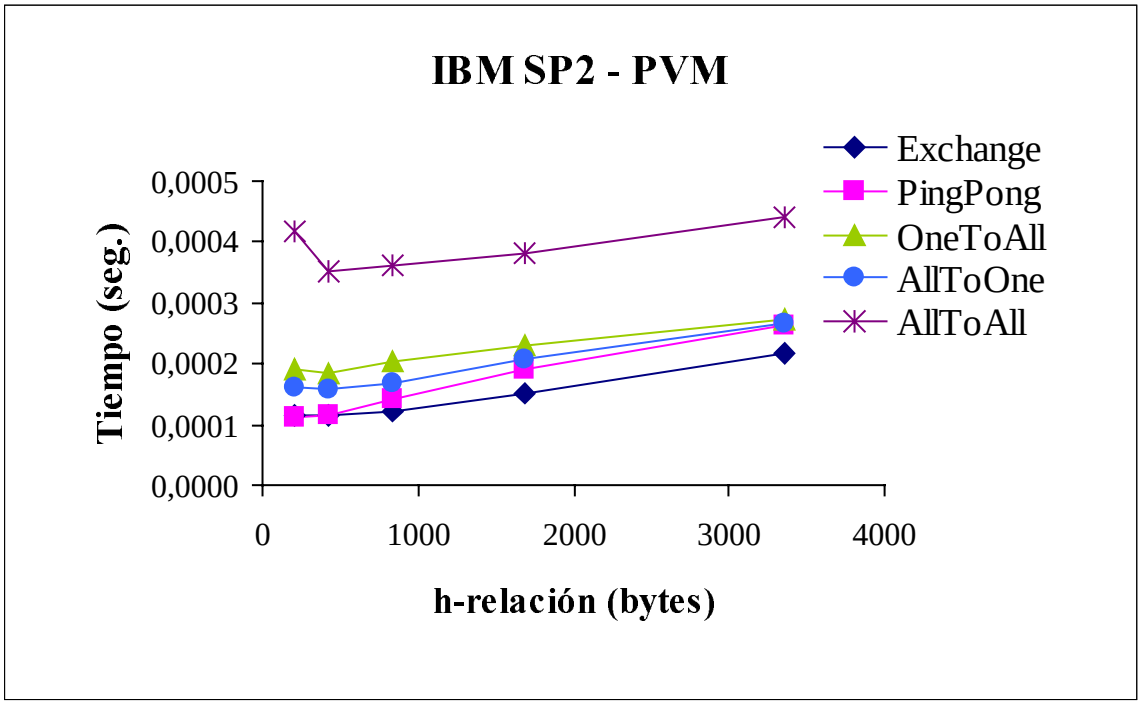

Figura 3. 10 Valores para los diferentes patrones con tamaños pequeños.

En el caso de la ORIGIN, se puede observar que para todo el rango de valores todos los patrones tienen un comportamiento creciente respecto a la *h*-relación. La tabla 3.27 muestra para la ORIGIN los valores observados para cada uno de los patrones, los valores medios de todos los patrones y los valores estimados a partir del modelo.

	Exchange	PingPong	OneToAll	AllToOne	AllToAll	MEDIA	MODELO
210	0.000120	0.000107	0.000137	0.000142	0.003283	0.000758	0.000101
420	0.000042	0.000038	0.000140	0.000118	0.000234	0.000114	0.000103
840	0.000047	0.000047	0.000145	0.000115	0.000240	0.000119	0.000108
1680	0.000055	0.000049	0.000158	0.000114	0.000244	0.000124	0.000119
3360	0.000060	0.000070	0.000174	0.000120	0.000269	0.000139	0.000139
6720	0.000079	0.000110	0.000195	0.000139	0.000306	0.000166	0.000180
13440	0.000129	0.000195	0.000233	0.000157	0.000354	0.000214	0.000261
26880	0.000234	0.000461	0.000327	0.000234	0.000426	0.000336	0.000423
53760	0.000556	0.000952	0.000569	0.000408	0.000611	0.000619	0.000748
107520	0.001096	0.001981	0.001231	0.000874	0.001042	0.001245	0.001399
215040	0.002193	0.004065	0.002802	0.001816	0.002089	0.002593	0.002699
430080	0.004387	0.008157	0.005508	0.003374	0.004421	0.005169	0.005299
860160	0.008819	0.016216	0.010394	0.006718	0.009577	0.010345	0.010500
1720320	0.017603	0.032963	0.022202	0.013171	0.019266	0.021041	0.020902

Tabla 3.27 Variación en los patrones para ORIGIN.

De nuevo la linealidad observada es evidente y las pendientes para el caso de tamaños grandes es 1.21E-08 y para valores pequeños es de 7.63E-09. El término independiente da lugar a un valor de 3.26E-05 para tamaños grandes y 1.12E-04 para *h*-relaciones pequeñas.

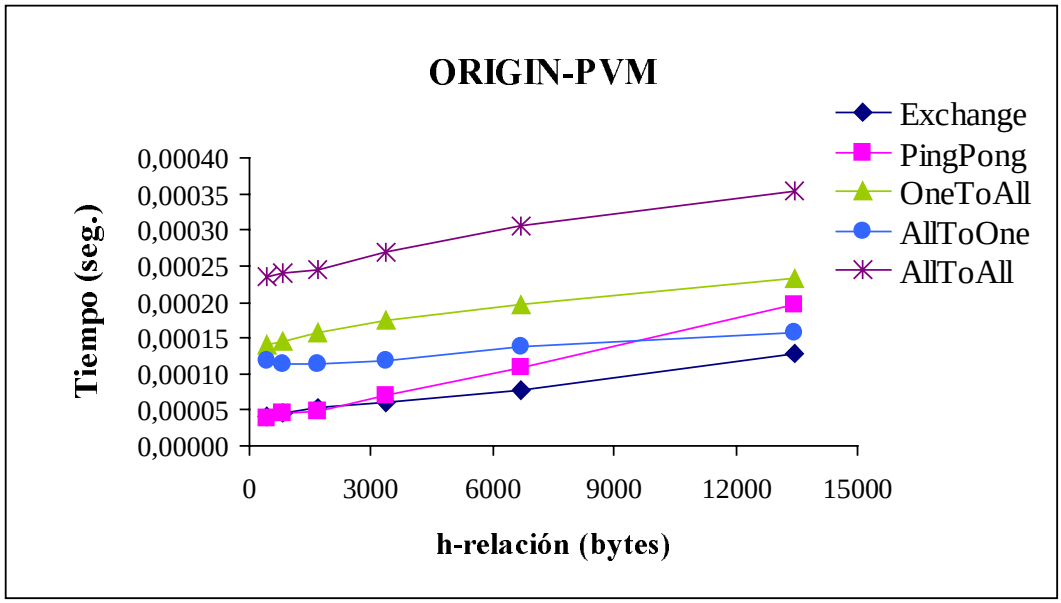

Figura 3. 11 Valores para los diferentes patrones con tamaños pequeños.

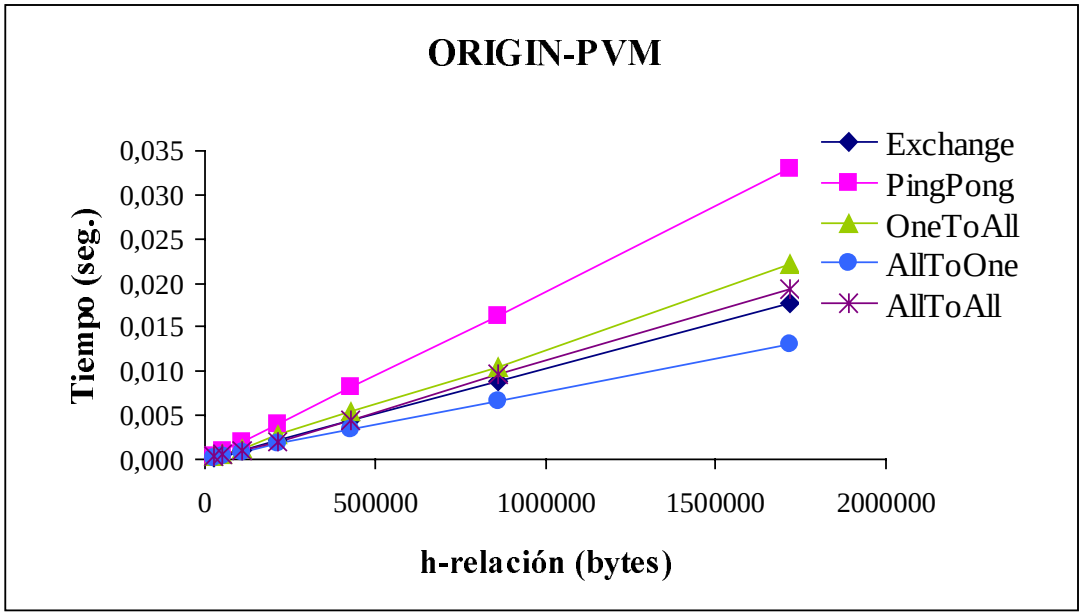

Figura 3. 12 Valores para los diferentes patrones con tamaños grandes.

La tabla 3.28 muestra para la LAN UTP los valores observados para cada uno de los patrones, los valores medios de todos los patrones y los valores estimados a partir del modelo. Las figuras 3.13 y 3.14 muestran los tiempos para la red de área local LAN UTP.

	Exchange	PingPong	OneToAll	AllToOne	AllToAll	MEDIA	MODELO
210	0.004481	0.003715	0.013667	0.009647	0.025618	0.011426	0.004140
420	0.004581	0.004244	0.013768	0.009987	0.025027	0.011522	0.004457
840	0.005257	0.004999	0.014315	0.010482	0.026554	0.012322	0.005092
1680	0.006374	0.006562	0.014140	0.011525	0.026368	0.012994	0.006361
3360	0.008032	0.008473	0.015656	0.011410	0.026683	0.014051	0.008899
6720	0.011557	0.015794	0.018417	0.012561	0.030317	0.017729	0.013975
13440	0.022140	0.030414	0.026085	0.018260	0.034803	0.026340	0.024126
26880	0.043332	0.056241	0.043768	0.030770	0.053128	0.045448	0.044430
53760	0.081631	0.109963	0.079444	0.060391	0.090478	0.084381	0.085036
107520	0.163097	0.216644	0.157800	0.119042	0.170077	0.165332	0.166249
215040	0.319656	0.424330	0.302853	0.235329	0.339838	0.324401	0.328676
430080	0.638596	0.844741	0.624923	0.475197	0.677662	0.652224	0.653528
860160	1.281078	1.681055	1.235945	0.938748	1.372075	1.301780	1.303233
1720320	2.575664	3.421407	2.434689	1.901240	2.688304	2.604261	2.602644

Tabla 3.28 Variación en los patrones para LAN UTP.

Podemos observar como la mayor saturación se alcanza con el patrón AllToAll. En los patrones Exchange y PingPong, se aprecia el beneficio de utilizar un conmutador

que paraleliza las comunicaciones existentes en las parejas de procesadores. En ambos casos, los tiempos para *h*-relaciones pequeñas, son aproximadamente iguales, pero cuando aumentamos el tamaño del mensaje, el patrón PingPong supera en tiempo al patrón Exchange al tener un comportamiento más secuencial. Los patrones OneToAll y AllToOne quedan definidos en el rango de los tres anteriores. El valor de *g* para tamaños grandes es de 1.51E-06 y de 8.38E-07 para tamaños pequeños. En este caso, el término

independiente es 3.82E-03 y de 1.11E-02 para valores grandes y pequeños respectivamente.

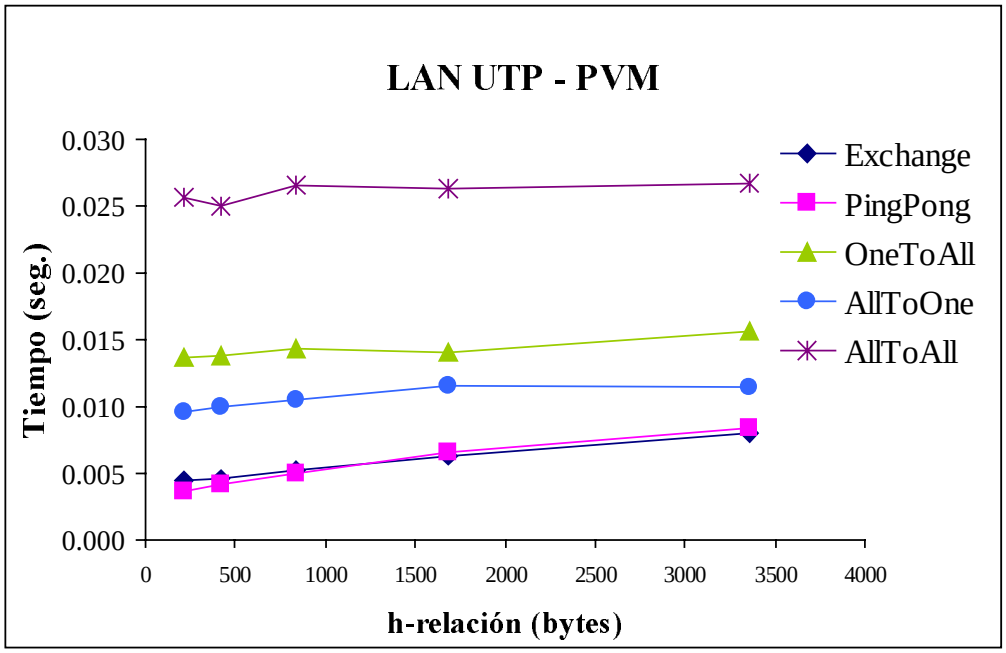

Figura 3. 13 Valores para los diferentes patrones con tamaños pequeños.

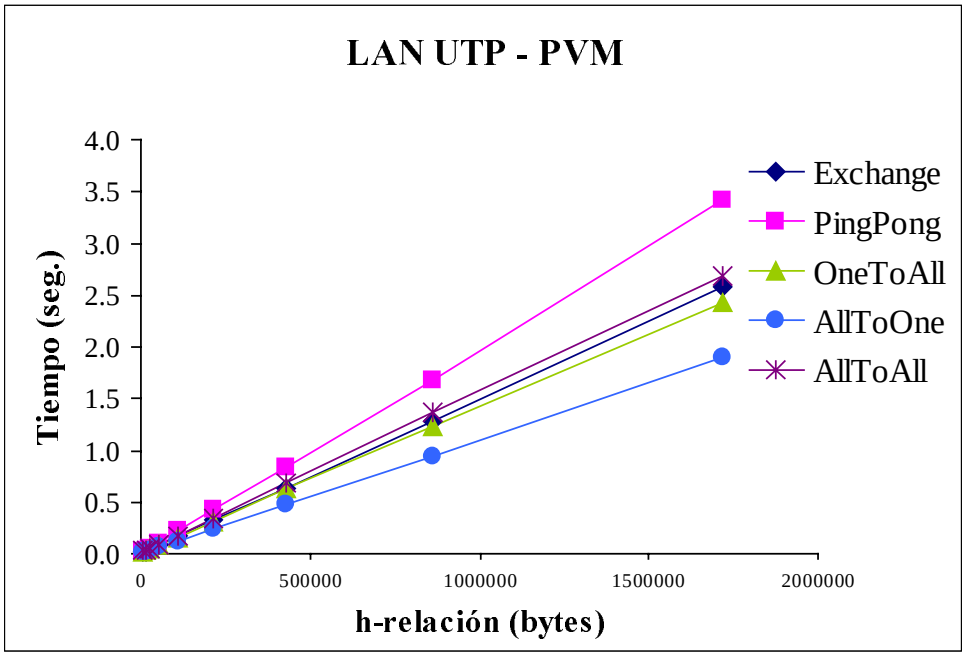

Figura 3. 14 Valores para los diferentes patrones con tamaños grandes.

La red de área local LAN COA ofrece un comportamiento parecido a la LAN UTP, pero los patrones Exchange y PingPong comienzan a separarse antes siendo en PingPong el que más satura la red de los dos. Respecto al AllToAll, sigue siendo el más castigado

de todos. Los patrones OneToAll y OneToAll tienen un comportamiento muy similar, existiendo algunas diferencias para valores pequeños de *h*-relación y prácticamente ninguno para valores grandes. Para tamaños grandes de nuevo la linealidad es clara ofreciendo un valor de *g* como media de los patrones de 2.43E-06 y un valor de 1.32E-02 para el término independiente. Para tamaños de *h*-relación pequeños, el valor queda en 9.61E-07 y el término independiente en 1.24E-02. La tabla 3.29 muestra para la LAN COA los valores observados para cada uno de los patrones, los valores medios de todos los patrones y los valores estimados a partir del modelo. Las figuras 3.15 y 3.16 muestran los casos de valores pequeños y grandes respectivamente.

	Exchange	PingPong	OneToAll	AllToOne	AllToAll	MEDIA	MODELO
210	0.004478	0.004804	0.013265	0.016773	0.025282	0.012920	0.013731
420	0.004751	0.005086	0.013044	0.015026	0.026099	0.012801	0.014242
840	0.005274	0.005723	0.013242	0.014859	0.026895	0.013199	0.015263
1680	0.006334	0.007832	0.013542	0.014752	0.025745	0.013641	0.017306
3360	0.009294	0.012128	0.014167	0.016166	0.027560	0.015863	0.021391
6720	0.019194	0.023155	0.016139	0.016392	0.031506	0.021277	0.029562
13440	0.038745	0.046675	0.023134	0.022704	0.066787	0.039609	0.045904
26880	0.079646	0.088614	0.036788	0.036078	0.147611	0.077747	0.078587
53760	0.162678	0.177848	0.066211	0.066076	0.270790	0.148720	0.143954
107520	0.327081	0.349199	0.126264	0.123828	0.436632	0.272601	0.274689
215040	0.661619	0.693365	0.248174	0.244777	0.836751	0.536937	0.536157
430080	1.336551	1.384253	0.508370	0.484273	1.623966	1.067483	1.059094
860160	2.660020	2.748344	1.005308	0.957312	3.210757	2.116348	2.104967
1720320	5.278342	5.458053	1.973896	1.916729	6.317505	4.188905	4.196714

Tabla 3.29 Variación en los patrones para LAN COA.

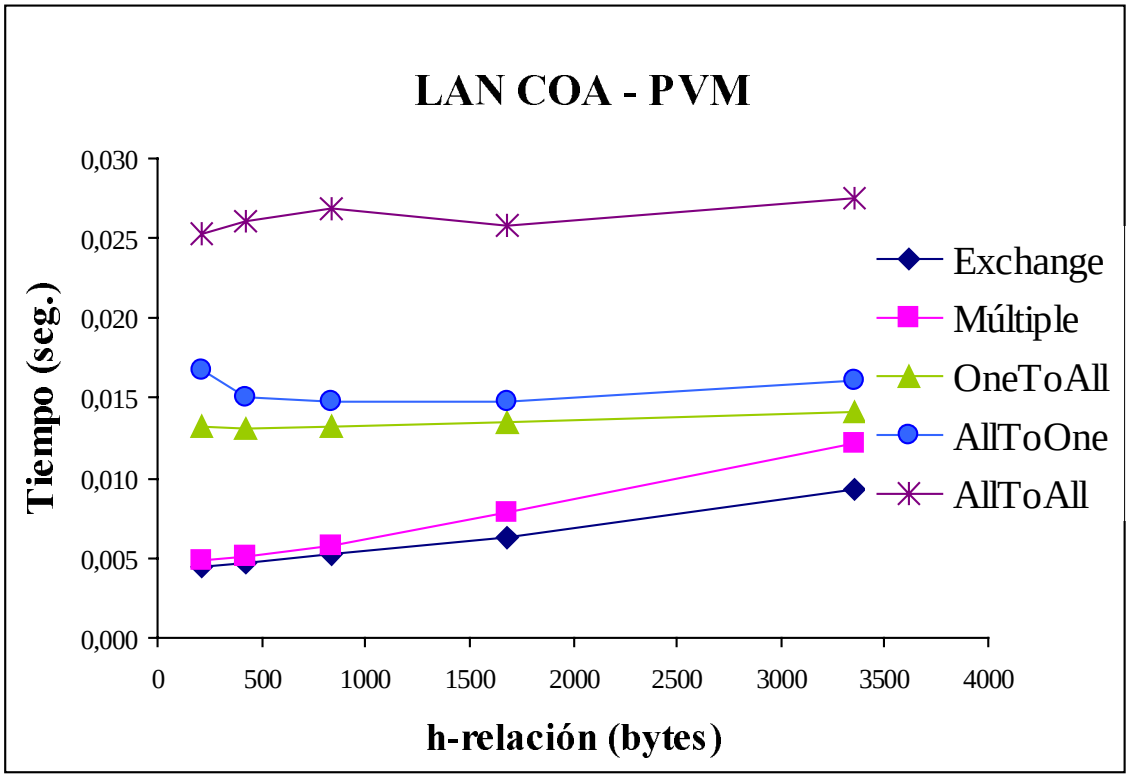

Figura 3.15 Valores para los diferentes patrones con tamaños pequeños.

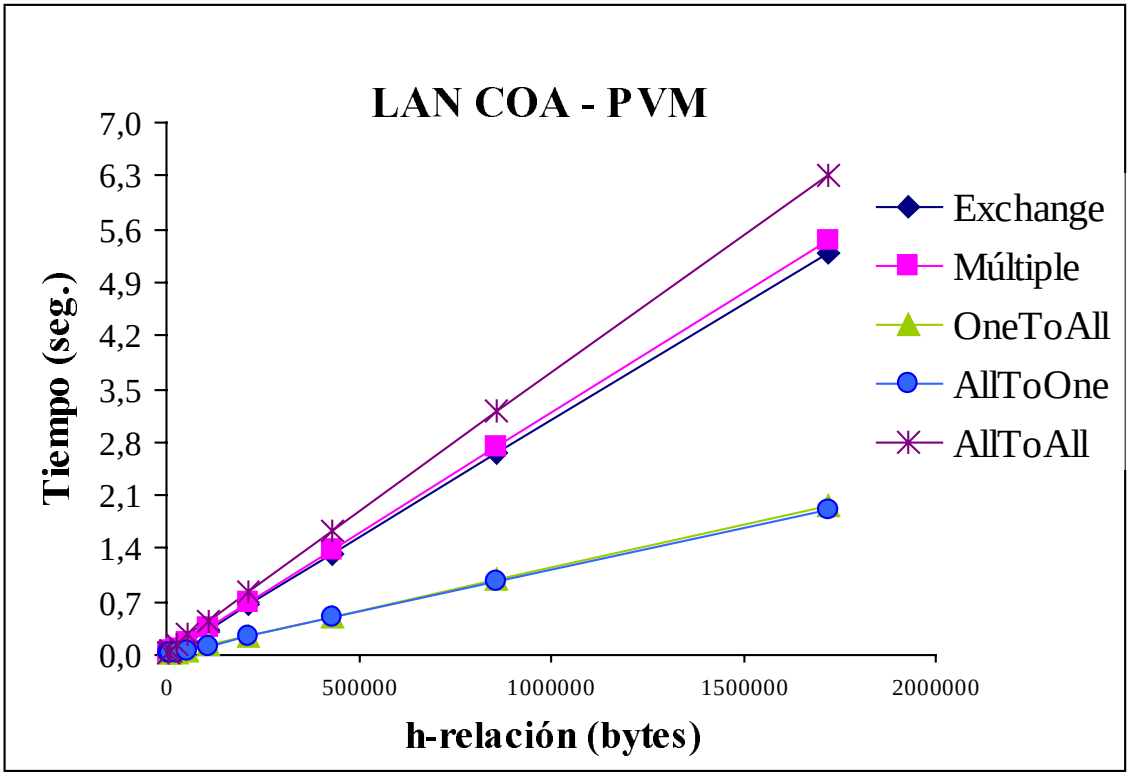

Figura 3. 16 Valores para los diferentes patrones con tamaños grandes.

3.9.1.8 *L* y *g* para las cuatro arquitecturas, tablas de patrones y media de procesadores

En la tabla 3.30 se presentan los valores de los parámetros *g* y *L* del modelo BSP. La relación en el parámetro *g* de la IBM SP2 y la ORIGIN es de 2.84 y de 1.61 entre la LAN COA y LAN UTP.

	ORIGIN	IBM SP2	LAN UTP	LAN COA
L	9.83E-05	1.24E-04	3.82E-03	1.32E-02
g	1.21E-08	3.44E-08	1.51E-06	2.43E-06

Tabla 3.30 Valores medios de g y L en PVM.

3.9.2 Estudio de la *h*-relación para multicomputadoras usando MPI

En este segundo conjunto de experimentos, las pruebas se realizaron bajo la plataforma software MPI y sobre las arquitecturas Cray T3E, Digital Alpha Server 8400, IBM SP2 y Silicon Origin 2000.

3.9.2.1 Exchange

La descripción del patrón Exchange se ha realizado en el apartado anterior. En este caso se utilizan las funciones *MPI_SendRecv()* para el envío y recepción de mensajes. De nuevo, la *h*-relación existente es $h = 2^* m_E$.

La tabla 3.31 muestra los valores obtenidos para el CRAY T3E. La tabla 3.32 muestra los valores obtenidos para la DIGITAL. La tabla 3.33 muestra los valores obtenidos para la IBM SP2. La tabla 3.34 muestra los valores obtenidos para la ORIGIN.

Exchange	1 pareja	2 parejas	3 parejas	4 parejas	MEDIA	MODELO	ErrMed	ErrMax
210	0.000027	0.000028	0.000028	0.000029	0.000028	0.000081	190.01	200.75
420	0.000036	0.000036	0.000037	0.000036	0.000036	0.000087	139.42	141.09
840	0.000058	0.000061	0.000059	0.000060	0.000060	0.000098	64.65	68.91
1680	0.000090	0.000090	0.000100	0.000118	0.000100	0.000120	20.93	33.69
3360	0.000121	0.000121	0.000134	0.000132	0.000127	0.000165	29.95	36.39
6720	0.000214	0.000217	0.000238	0.000253	0.000231	0.000254	10.39	18.90
13440	0.000395	0.000419	0.000432	0.000459	0.000426	0.000433	4.67	9.69
26880	0.000755	0.000792	0.000874	0.000830	0.000813	0.000791	4.89	11.00
53760	0.001440	0.001487	0.001574	0.001539	0.001510	0.001506	3.08	4.70
107520	0.002900	0.002916	0.002976	0.002988	0.002945	0.002937	1.26	1.76
215040	0.005564	0.005717	0.005765	0.005865	0.005728	0.005798	1.81	4.21
430080	0.011039	0.011405	0.011348	0.011446	0.011310	0.011521	1.87	4.37
860160	0.022673	0.022985	0.023237	0.023109	0.023001	0.022966	0.79	1.29
1720320	0.044956	0.047466	0.046717	0.046073	0.046303	0.045857	1.94	3.58
3440640	0.089726	0.092431	0.091839	0.091753	0.091437	0.091638	0.83	2.13

Tabla 3.31 Tiempos del patrón de comunicaciones Exchange para el Cray T3E.

Exchange	1 pareja	2 parejas	3 parejas	4 parejas	MEDIA	MODELO	ErrMed	ErrMax
210	0.000039	0.000049	0.000107	0.000107	0.000076	-0.002511	3425.60	6712.38
420	0.000037	0.000056	0.000069	0.000052	0.000054	-0.002491	4756.24	6919.16
840	0.000038	0.000057	0.000108	0.000096	0.000075	-0.002452	3379.75	6735.83
1680	0.000097	0.000116	0.000168	0.000153	0.000134	-0.002373	1877.27	2619.24
3360	0.000155	0.000233	0.000337	0.000270	0.000249	-0.002215	990.35	1646.29
6720	0.000309	0.000445	0.000542	0.000674	0.000493	-0.001899	485.57	832.66
13440	0.000641	0.000860	0.001022	0.001198	0.000930	-0.001267	236.23	384.60
26880	0.000951	0.001937	0.002617	0.004148	0.002413	-0.000004	100.17	436.60
53760	0.001673	0.002668	0.004446	0.007025	0.003953	0.002523	46.93	269.13
107520	0.003118	0.006260	0.008881	0.014239	0.008125	0.007576	42.29	213.71
215040	0.006058	0.011343	0.018782	0.030081	0.016566	0.017682	47.48	204.67
430080	0.011949	0.027887	0.036366	0.060436	0.034160	0.037894	43.93	217.13
860160	0.023711	0.046411	0.068981	0.121932	0.065259	0.078319	53.43	230.31
1720320	0.047201	0.129167	0.153915	0.246973	0.144314	0.159168	40.71	237.21
3440640	0.132246	0.277607	0.365316	0.552910	0.332020	0.320867	38.28	175.46

Tabla 3.32 Tiempos del patrón de comunicaciones Exchange para la Digital.

Exchange	1 pareja	2 parejas	3 parejas	4 parejas	MEDIA	MODELO	ErrMed	ErrMax
210	0.000105	0.000099	0.000102	0.000099	0.000101	0.000462	356.32	366.69
420	0.000118	0.000122	0.000132	0.000129	0.000125	0.000481	284.12	307.72
840	0.000170	0.000170	0.000176	0.000174	0.000173	0.000519	201.03	205.46
1680	0.000248	0.000266	0.000263	0.000296	0.000268	0.000596	122.04	140.17
3360	0.000508	0.000576	0.000549	0.000648	0.000570	0.000748	31.22	47.30
6720	0.000855	0.000957	0.000929	0.000933	0.000919	0.001054	14.72	23.24
13440	0.001492	0.001614	0.001587	0.001601	0.001574	0.001664	5.78	11.56
26880	0.003326	0.002894	0.003004	0.002834	0.003014	0.002886	5.12	15.52
53760	0.005092	0.005265	0.005383	0.005379	0.005280	0.005329	1.92	4.65
107520	0.009970	0.010115	0.010214	0.010396	0.010174	0.010215	1.29	2.46
215040	0.019767	0.020175	0.020192	0.020253	0.020097	0.019987	1.09	1.35
430080	0.038775	0.039264	0.039950	0.040271	0.039565	0.039531	1.38	1.95
860160	0.077944	0.078183	0.079300	0.083618	0.079761	0.078618	2.13	6.41
1720320	0.154454	0.159407	0.160832	0.158598	0.158323	0.156794	1.70	2.61
3440640	0.307048	0.311609	0.312714	0.316973	0.312086	0.313145	0.95	1.99

Tabla 3.33 Tiempos del patrón de comunicaciones Exchange para el IBM SP2.

Exchange	1 pareja	2 parejas	3 parejas	4 parejas	MEDIA	MODELO	ErrMed	ErrMax
210	0.000046	0.000059	0.000075	0.000086	0.000067	-0.004636	7071.34	10265.09
420	0.000054	0.000068	0.000068	0.000072	0.000066	-0.004619	7152.18	8687.37
840	0.000070	0.000080	0.000081	0.000082	0.000078	-0.004586	5960.26	6668.08
1680	0.000103	0.000114	0.000122	0.000120	0.000115	-0.004519	4037.78	4505.44
3360	0.000161	0.000185	0.000197	0.000201	0.000186	-0.004384	2457.25	2848.13
6720	0.000276	0.000322	0.000324	0.000325	0.000312	-0.004116	1420.38	1609.16
13440	0.000626	0.000674	0.000719	0.000727	0.000687	-0.003580	621.46	688.00
26880	0.001135	0.001281	0.001323	0.001367	0.001277	-0.002507	296.40	341.32
53760	0.002139	0.002398	0.002585	0.002612	0.002434	-0.000361	114.85	139.00
107520	0.003892	0.004608	0.004792	0.005175	0.004617	0.003930	15.29	31.99
215040	0.007480	0.008757	0.009888	0.010140	0.009066	0.012513	38.02	67.29
430080	0.015072	0.017392	0.019548	0.020130	0.018036	0.029679	64.56	96.91
860160	0.029815	0.035376	0.039433	0.040810	0.036359	0.064010	76.05	114.69
1720320	0.081479	0.109623	0.139149	0.161015	0.122817	0.132672	22.20	62.83
3440640	0.178931	0.248810	0.323232	0.382549	0.283381	0.269998	24.53	62.90

Tabla 3.34 Tiempos del patrón de comunicaciones Exchange para el Origin.

Exchange	CRAY T3E	DIGITAL	IBM SP2	ORIGIN
L_E	7.56E-05	-2.53E-03	4.43E-04	-4.65E-03
g_E	2.66E-08	9.40E-08	9.09E-08	7.98E-08

Tabla 3.35 Valores medios de *g* y *L* para el patrón Exchange.

3.9.2.2 PingPong

El procesador fuente ejecuta primero la función *MPI_Send()* seguida de la función *MPI_Recv()*. El procesador destino ejecuta las funciones de forma inversa, es decir, primero recibe los datos del procesador fuente con la función *MPI_Recv()* y luego devuelve los datos al procesador fuente con la función *MPI_Send()*. El patrón de comunicaciones PingPong da lugar a una *h*-relación de $h = m_{pp}$.

PingPong	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000041	0.000047	0.000043	0.000043	0.000044	-0.000002	103.49	118.34
420	0.000063	0.000065	0.000065	0.000065	0.000065	0.000009	86.40	89.25
840	0.000097	0.000100	0.000100	0.000099	0.000099	0.000029	70.34	72.83
1680	0.000121	0.000130	0.000127	0.000130	0.000127	0.000071	44.46	49.15
3360	0.000206	0.000204	0.000203	0.000209	0.000206	0.000153	25.61	27.65
6720	0.000369	0.000373	0.000369	0.000390	0.000375	0.000318	15.37	19.63
13440	0.000694	0.000698	0.000743	0.000738	0.000718	0.000647	9.93	13.84
26880	0.001320	0.001327	0.001399	0.001387	0.001358	0.001306	3.87	7.07
53760	0.002678	0.002682	0.002687	0.002676	0.002681	0.002623	2.15	2.38
107520	0.005267	0.005241	0.005227	0.005234	0.005242	0.005258	0.39	0.60
215040	0.010169	0.010176	0.010173	0.010190	0.010177	0.010528	3.45	3.53
430080	0.020847	0.020851	0.020851	0.020999	0.020887	0.021068	0.87	1.06
860160	0.042224	0.042229	0.042217	0.041795	0.042116	0.042148	0.34	0.85
1720320	0.084312	0.084313	0.084316	0.084378	0.084330	0.084308	0.03	0.08
3440640	0.168686	0.168685	0.168675	0.168632	0.168670	0.168629	0.02	0.03

Tabla 3.36 Tiempos del patrón de comunicaciones PingPong para el Cray T3E.

PingPong	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000024	0.000042	0.000059	0.000016	0.000035	-0.000260	838.99	1996.84
420	0.000029	0.000039	0.000036	0.000033	0.000034	-0.000234	781.89	939.81
840	0.000058	0.000065	0.000077	0.000068	0.000067	-0.000180	368.13	442.50
1680	0.000117	0.000117	0.000136	0.000112	0.000121	-0.000072	159.63	185.59
3360	0.000223	0.000245	0.000261	0.000258	0.000247	0.000144	41.75	52.59
6720	0.000408	0.000434	0.000509	0.000491	0.000461	0.000575	24.84	40.90
13440	0.001257	0.001464	0.001909	0.002781	0.001853	0.001437	27.29	106.90
26880	0.002282	0.002763	0.003561	0.005304	0.003478	0.003162	27.46	93.87
53760	0.004386	0.005193	0.006866	0.010185	0.006658	0.006611	28.06	81.48
107520	0.008680	0.010368	0.013564	0.020654	0.013317	0.013510	28.48	82.31
215040	0.016900	0.021069	0.027476	0.039048	0.026123	0.027307	27.33	69.47
430080	0.033525	0.042102	0.054099	0.077378	0.051776	0.054902	27.74	67.04
860160	0.066982	0.094384	0.114583	0.158258	0.108552	0.110091	25.67	71.91
1720320	0.145427	0.198081	0.244772	0.320988	0.227317	0.220470	24.44	69.12
3440640	0.278491	0.374472	0.470794	0.630862	0.438655	0.441227	25.57	68.09

Tabla 3.37 Tiempos del patrón de comunicaciones PingPong para la Digital.

PingPong	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000097	0.000154	0.000173	0.000124	0.000137	0.000346	489.19	256.92
420	0.000173	0.000135	0.000206	0.000166	0.000170	0.000372	383.49	175.20
840	0.000182	0.000226	0.000216	0.000226	0.000213	0.000422	325.37	131.94
1680	0.000369	0.000434	0.000380	0.000375	0.000390	0.000523	108.89	41.83
3360	0.000574	0.000586	0.000586	0.000660	0.000602	0.000726	74.45	26.44
6720	0.001010	0.001057	0.001054	0.001467	0.001147	0.001131	30.96	33.30
13440	0.001809	0.001833	0.001863	0.002095	0.001900	0.001940	18.68	8.54
26880	0.003436	0.003497	0.003494	0.003503	0.003483	0.003560	7.67	3.61
53760	0.007086	0.006720	0.006788	0.006778	0.006843	0.006799	5.59	4.27
107520	0.013281	0.013236	0.013349	0.013985	0.013463	0.013277	2.18	5.35
215040	0.025967	0.026785	0.026839	0.026887	0.026620	0.026233	5.96	2.52
430080	0.051588	0.051921	0.052856	0.052896	0.052315	0.052146	3.21	1.45
860160	0.103599	0.104219	0.104262	0.104467	0.104137	0.103970	0.99	0.48
1720320	0.206207	0.207541	0.208352	0.209340	0.207860	0.207619	1.28	0.83
3440640	0.411834	0.413864	0.414955	0.418168	0.414705	0.414918	1.20	0.79

Tabla 3.38 Tiempos del patrón de comunicaciones PingPong para el IBM SP2.

PingPong	2	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000043	0.000044	0.000057	0.000058	0.000051	-0.004447	8905.54	10476.28
420	0.000042	0.000040	0.000049	0.000047	0.000045	-0.004430	10054.70	11197.10
840	0.000059	0.000060	0.000068	0.000067	0.000064	-0.004396	7022.72	7565.97
1680	0.000094	0.000097	0.000112	0.000108	0.000103	-0.004328	4312.25	4723.50
3360	0.000199	0.000193	0.000221	0.000213	0.000207	-0.004192	2130.23	2286.75
6720	0.000357	0.000371	0.000421	0.000418	0.000392	-0.003921	1100.91	1216.27
13440	0.000569	0.000634	0.000729	0.000703	0.000659	-0.003378	612.85	721.86
26880	0.000962	0.001149	0.001307	0.001291	0.001177	-0.002293	294.78	374.23
53760	0.001776	0.002136	0.002482	0.002396	0.002198	-0.000122	105.57	146.64
107520	0.003295	0.004062	0.004883	0.004836	0.004269	0.004219	13.83	28.04
215040	0.006486	0.008045	0.009665	0.009520	0.008429	0.012902	53.07	98.92
430080	0.013315	0.016107	0.019432	0.019109	0.016991	0.030268	78.14	127.32
860160	0.035454	0.034118	0.041118	0.040695	0.037846	0.064999	71.74	90.51
1720320	0.079814	0.120194	0.154250	0.182159	0.134104	0.134461	25.43	68.47
3440640	0.176279	0.249556	0.321263	0.381681	0.282195	0.273386	24.55	61.43

Tabla 3.39 Tiempos del patrón de comunicaciones PingPong para el Origin.

PingPong	CRAY T3E	DIGITAL	IBM SP2	ORIGIN
L_{PP}	-1.18E-05	-2.87E-04	3.21E-04	-4.46E-03
g_{PP}	4.90E-08	1.28E-07	1.20E-07	8.08E-08

Tabla 3.40 Valores medios de *g* y *L* para el patrón PingPong.

3.9.2.3 OneToAll

A diferencia del caso del patrón OneToAll en PVM, aquí hacemos uso de las funciones colectivas de MPI. Todos los procesadores hacen la llamada a la función *MPI_Bcast()* y sólo aquel procesador cuyo nombre aparezca en el parámetro *root* de la función, será el que envíe a los demás. En este caso, la *h*-relación queda establecida por $h = (p-1) * m_{OA}$. De nuevo se aprecia para todos los casos una disminución del tiempo según aumentamos el número de procesadores.

OneToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000025	0.000028	0.000028	0.000027	0.000026	5.58	6.09
420	0.000027	0.000032	0.000032	0.000030	0.000029	8.31	9.49
840	0.000035	0.000037	0.000035	0.000036	0.000035	2.21	4.69
1680	0.000047	0.000055	0.000046	0.000049	0.000047	6.22	16.95
3360	0.000073	0.000075	0.000066	0.000071	0.000071	5.19	7.41
6720	0.000128	0.000123	0.000109	0.000120	0.000118	6.59	8.94
13440	0.000235	0.000219	0.000192	0.000215	0.000213	7.59	11.46
26880	0.000453	0.000411	0.000358	0.000407	0.000402	8.47	14.11
53760	0.000882	0.000796	0.000695	0.000791	0.000781	8.49	14.47
107520	0.001727	0.001558	0.001358	0.001548	0.001539	8.35	13.82
215040	0.003416	0.003078	0.002683	0.003059	0.003055	8.24	13.87
430080	0.006804	0.006117	0.005321	0.006081	0.006087	8.29	14.39
860160	0.01362	0.012193	0.010594	0.012136	0.012150	8.43	14.69
1720320	0.027126	0.024407	0.021193	0.024242	0.024277	8.34	14.55
3440640	0.054323	0.048907	0.042427	0.048552	0.048531	8.43	14.39

Tabla 3.41 Tiempos del patrón de comunicaciones OneToAll para el Cray T3E.

OneToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000043	0.000027	0.000045	0.000038	0.000158	310.92	483.40
420	0.000042	0.000034	0.000034	0.000037	0.000167	354.44	390.08
840	0.000059	0.000049	0.000055	0.000054	0.000185	240.20	277.23
1680	0.000105	0.000081	0.000082	0.000089	0.000221	147.70	173.18
3360	0.000181	0.000138	0.000132	0.000150	0.000294	95.66	122.84
6720	0.000314	0.000260	0.000244	0.000273	0.000440	61.33	80.28
13440	0.000627	0.000480	0.000455	0.000521	0.000731	40.47	60.74
26880	0.001603	0.000929	0.000906	0.001146	0.001314	31.48	45.07
53760	0.002903	0.003013	0.001574	0.002497	0.002480	24.86	57.57
107520	0.005377	0.005389	0.005080	0.005282	0.004812	8.90	11.36
215040	0.010420	0.010126	0.009290	0.009945	0.009476	5.97	10.17
430080	0.020318	0.019555	0.018016	0.019296	0.018803	5.28	8.41
860160	0.040176	0.038324	0.034157	0.037552	0.037457	6.11	9.66
1720320	0.079541	0.075808	0.067656	0.074335	0.074766	5.80	10.51
3440640	0.166829	0.151626	0.129965	0.149473	0.149385	8.72	14.94

Tabla 3.42 Tiempos del patrón de comunicaciones OneToAll para la Digital.

OneToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.00021	0.00021	0.00024	0.000221	0.000481	118.13	129.21
420	0.00023	0.00024	0.00028	0.000248	0.000496	100.09	113.61
840	0.00028	0.00029	0.00033	0.000297	0.000524	76.64	89.87
1680	0.00036	0.00035	0.00039	0.000366	0.000581	58.88	66.95
3360	0.00068	0.00049	0.00050	0.000556	0.000695	24.97	43.27
6720	0.00101	0.00109	0.00067	0.000923	0.000923	18.17	37.29
13440	0.00151	0.00154	0.00129	0.001443	0.001378	8.75	12.43
26880	0.00262	0.00261	0.00206	0.002428	0.002289	11.97	15.80
53760	0.00479	0.00445	0.00355	0.004263	0.004111	12.31	18.97
107520	0.00903	0.00836	0.00632	0.007901	0.007755	13.99	22.79
215040	0.01760	0.01607	0.01175	0.015139	0.015044	15.14	28.05
430080	0.03477	0.03187	0.02268	0.029772	0.029620	16.05	30.60
860160	0.06998	0.06229	0.04468	0.058981	0.058774	16.28	31.55
1720320	0.13760	0.12500	0.08995	0.117515	0.117080	15.76	30.17
3440640	0.27420	0.24703	0.17895	0.233391	0.233693	15.51	30.59

Tabla 4.43 Tiempos del patrón de comunicaciones OneToAll para el IBM SP2.

OneToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000083	0.00008	0.000104	0.000089	-0.000124	239.70	285.42
420	0.000088	0.000081	0.000106	0.000092	-0.000120	231.33	279.49
840	0.000098	0.000112	0.000132	0.000114	-0.000112	198.67	249.48
1680	0.000117	0.000119	0.000179	0.000138	-0.000097	169.89	235.63
3360	0.000163	0.000160	0.000180	0.000168	-0.000065	138.82	153.18
6720	0.000252	0.000237	0.000245	0.000245	-0.000002	100.77	107.12
13440	0.000517	0.000395	0.000389	0.000434	0.000125	71.29	100.89
26880	0.000791	0.000785	0.000765	0.000780	0.000377	51.65	54.07
53760	0.001345	0.001268	0.001213	0.001275	0.000883	30.77	38.09
107520	0.002407	0.002297	0.001894	0.002199	0.001894	13.88	27.08
215040	0.004307	0.003968	0.003248	0.003841	0.003917	9.64	20.59
430080	0.008225	0.007769	0.006292	0.007429	0.007962	9.54	26.53
860160	0.016506	0.015229	0.012552	0.014762	0.016051	10.79	27.88
1720320	0.033968	0.030709	0.024510	0.029729	0.032231	12.31	31.50
3440640	0.084656	0.064877	0.049109	0.066214	0.064590	18.04	40.86

Tabla 3.44 Tiempos del patrón de comunicaciones OneToAll para el Origin.

OneToAll	CRAY T3E	DIGITAL	IBM SP2	ORIGIN
L_{OA}	2.352E-05	1.48E-04	4.67E-04	-1.28E-04
G_{OA}	1.410E-08	4.34E-08	6.78E-08	1.88E-08

Tabla 3.45 Valores medios de *g* y *L* para el patrón OneToAll.

3.9.2.4 AllToOne

En este caso también utilizamos la función colectiva de MPI para realizar los *p-1* envíos a un procesador sumidero. Todos los procesadores realizan una llamada a la función *MPI_Gather()* y sólo aquel procesador cuyo nombre aparezca en el parámetro *root* de la función, será el que reciba de todos los demás. En este caso, la *h*-relación queda establecida por $h = (p-1) * m_{AO}$.

AllToOne	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000026	0.000029	0.000034	0.000030	0.000052	75.99	100.81
420	0.000032	0.000033	0.000037	0.000034	0.000058	71.68	82.41
840	0.000044	0.000044	0.000047	0.000045	0.000071	57.10	60.68
1680	0.000072	0.000072	0.000074	0.000073	0.000095	31.21	32.43
3360	0.000119	0.000111	0.000123	0.000118	0.000145	22.93	30.31
6720	0.000224	0.000209	0.000227	0.000220	0.000243	10.56	16.38
13440	0.000432	0.000390	0.000440	0.000421	0.000440	4.70	12.93
26880	0.000839	0.000755	0.000842	0.000812	0.000835	3.74	10.57
53760	0.001662	0.001497	0.001719	0.001626	0.001624	5.34	8.46
107520	0.003258	0.002971	0.003402	0.003210	0.003201	5.07	7.75
215040	0.006488	0.005907	0.006704	0.006366	0.006356	4.86	7.61
430080	0.013008	0.011897	0.013252	0.012719	0.012666	4.45	6.47
860160	0.026270	0.023819	0.026180	0.025423	0.025287	4.38	6.16
1720320	0.052188	0.047691	0.051819	0.050566	0.050528	3.82	5.95
3440640	0.104754	0.095159	0.102935	0.100949	0.101010	3.80	6.15

Tabla 3.46 Tiempos del patrón de comunicaciones AllToOne para el Cray T3E.

AllToOne	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000117	0.000180	0.000263	0.000187	0.002687	1339.47	2196.59
420	0.000084	0.000147	0.000125	0.000119	0.002713	2186.01	3129.44
840	0.000109	0.000146	0.000185	0.000147	0.002764	1784.67	2435.94
1680	0.000160	0.000211	0.000221	0.000197	0.002867	1352.91	1691.92
3360	0.000266	0.000320	0.000318	0.000301	0.003073	919.75	1055.21
6720	0.000508	0.000566	0.000546	0.000540	0.003484	545.26	585.91
13440	0.000981	0.001020	0.000996	0.000999	0.004308	331.18	339.09
26880	0.002341	0.001905	0.001919	0.002055	0.005954	189.72	212.53
53760	0.004606	0.008668	0.003734	0.005669	0.009246	63.09	147.62
107520	0.008940	0.017426	0.024140	0.016835	0.015831	33.25	92.94
215040	0.018064	0.010420	0.028161	0.018882	0.029001	53.59	178.32
430080	0.030626	0.022079	0.088998	0.047234	0.055341	64.67	152.44
860160	0.094020	0.071807	0.317916	0.161248	0.108020	53.77	292.31
1720320	0.213171	0.165478	0.276239	0.218296	0.213379	16.94	37.99
3440640	0.436450	0.280703	0.513017	0.410057	0.424097	19.89	51.08

Tabla 3.47 Tiempos del patrón de comunicaciones AllToOne para la Digital.

AllToOne	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000266	0.000235	0.000369	0.000290	0.000481	65.81	104.61
420	0.000303	0.000265	0.000402	0.000323	0.000506	56.34	90.76
840	0.000376	0.000333	0.000493	0.000401	0.000555	38.48	66.62
1680	0.000595	0.000446	0.000632	0.000558	0.000654	17.19	46.54
3360	0.000635	0.000955	0.000965	0.000852	0.000851	16.99	34.01
6720	0.001024	0.001097	0.001620	0.001247	0.001246	19.91	36.55
13440	0.001854	0.001949	0.002021	0.001941	0.002035	4.84	9.78
26880	0.003459	0.003603	0.003846	0.003636	0.003614	3.65	6.70
53760	0.006728	0.006653	0.006751	0.006711	0.006773	0.92	1.80
107520	0.013182	0.012819	0.012792	0.012931	0.013089	1.70	2.32
215040	0.026200	0.025299	0.025584	0.025694	0.025722	1.35	1.89
430080	0.052484	0.049867	0.049539	0.050630	0.050987	2.68	3.02
860160	0.104296	0.099445	0.099030	0.100924	0.101519	2.42	2.80
1720320	0.207017	0.198613	0.212264	0.205965	0.202581	2.93	4.88
3440640	0.416083	0.396847	0.396717	0.403216	0.404706	2.25	2.87

Tabla 3.48 Tiempos del patrón de comunicaciones AllToOne para el IBM SP2.

AllToOne	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000087	0.000088	0.000128	0.000101	-0.000687	780.01	936.56
420	0.000095	0.000101	0.000134	0.000110	-0.000676	714.95	853.10
840	0.000102	0.000133	0.000158	0.000131	-0.000656	600.55	797.77
1680	0.000130	0.000141	0.000205	0.000159	-0.000614	487.14	630.21
3360	0.000200	0.000189	0.000213	0.000201	-0.000531	364.80	393.84
6720	0.000291	0.000294	0.000297	0.000294	-0.000366	224.34	227.68
13440	0.000620	0.000485	0.000508	0.000538	-0.000034	106.31	134.83
26880	0.001168	0.001123	0.001025	0.001105	0.000629	43.07	52.56
53760	0.002710	0.001677	0.002416	0.002268	0.001956	21.95	44.97
107520	0.004869	0.003275	0.004501	0.004215	0.004609	13.46	40.72
215040	0.010716	0.006759	0.008682	0.008719	0.009915	19.84	46.69
430080	0.022207	0.016931	0.015036	0.018058	0.020526	19.87	36.52
860160	0.045784	0.041731	0.032610	0.040042	0.041750	10.98	28.03
1720320	0.092012	0.086370	0.068643	0.082342	0.084197	10.34	22.66
3440640	0.193972	0.179730	0.138785	0.170829	0.169092	12.84	21.84

Tabla 3.49 Tiempos del patrón de comunicaciones AllToOne para el Origin.

AllToOne	CRAY T3E	DIGITAL	IBM SP2	ORIGIN
L_{AO}	4.60E-05	2.66E-03	4.56E-04	-6.97E-04
g_{AO}	2.93E-08	1.22E-07	1.17E-07	4.93E-08

Tabla 3.50 Valores medios de g y L para el patrón AllToOne.

3.9.2.5 AllToAll

La función *MPI_Alltoall()* se utiliza para observar el patrón que provoca un mayor número de mensajes. Cada procesador envía al resto *p-1* mensajes. En este caso, la *h*-relación queda establecida por $h = 2\,(p-1) * m_{AA}$.

AllToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000041	0.000057	0.000074	0.000057	0.000046	25.66	68.61
420	0.000046	0.000061	0.000079	0.000062	0.000051	23.14	60.95
840	0.000056	0.000072	0.000088	0.000072	0.000061	19.84	47.94
1680	0.000078	0.000088	0.000103	0.000090	0.000082	11.70	27.52
3360	0.000113	0.000142	0.000147	0.000134	0.000122	13.36	21.86
6720	0.000198	0.000207	0.000208	0.000204	0.000204	2.15	2.93
13440	0.000378	0.000368	0.000347	0.000364	0.000367	2.94	5.72
26880	0.000724	0.000679	0.000636	0.000680	0.000693	5.00	8.95
53760	0.001437	0.001357	0.001227	0.001340	0.001345	5.52	9.62
107520	0.002811	0.002674	0.002436	0.002640	0.002649	5.05	8.76
215040	0.005731	0.005044	0.004773	0.005183	0.005258	7.54	10.16
430080	0.011415	0.010537	0.009497	0.010483	0.010475	6.30	10.30
860160	0.022875	0.020595	0.019174	0.020881	0.020909	6.41	10.25
1720320	0.045622	0.041819	0.038251	0.041897	0.041778	5.90	10.05
3440640	0.091432	0.082792	0.076174	0.083466	0.083515	6.38	10.39

Tabla 3.51 Tiempos del patrón de comunicaciones AllToAll para el Cray T3E.

AllToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000366	0.000533	0.000729	0.000543	0.021271	3819.66	5711.66
420	0.000126	0.000133	0.000165	0.000141	0.021324	14987.48	16823.52
840	0.000157	0.000211	0.000208	0.000192	0.021430	11061.24	13549.41
1680	0.000181	0.000260	0.000259	0.000233	0.021641	9174.90	11856.59
3360	0.000249	0.000358	0.000364	0.000324	0.022065	6717.25	8761.51
6720	0.000435	0.000520	0.000633	0.000529	0.022913	4228.58	5167.27
13440	0.000800	0.000932	0.000969	0.000900	0.024608	2633.15	2975.94
26880	0.001570	0.001898	0.001987	0.001818	0.027997	1439.72	1683.27
53760	0.003654	0.003905	0.003879	0.003813	0.034777	812.14	851.75
107520	0.007155	0.054287	0.008141	0.023194	0.048336	125.50	575.56
215040	0.014157	0.127136	0.313970	0.151754	0.075454	77.21	1684.79
430080	0.029156	0.167332	0.468651	0.221713	0.129691	71.73	1162.57
860160	0.051780	0.300269	0.557227	0.303092	0.238165	62.42	616.19
1720320	0.126295	0.558402	0.874293	0.519663	0.455111	54.61	331.91
3440640	0.345932	0.763690	1.368009	0.825877	0.889005	46.31	156.99

Tabla 3.52 Tiempos del patrón de comunicaciones AllToAll para la Digital.

AllToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000265	0.000320	0.000385	0.000323	0.000775	139.74	192.51
420	0.000283	0.000360	0.000416	0.000353	0.000795	125.22	180.92
840	0.000343	0.000464	0.000474	0.000427	0.000835	95.49	143.36
1680	0.000320	0.000394	0.000746	0.000487	0.000914	87.85	185.68
3360	0.000479	0.000545	0.000606	0.000543	0.001073	97.50	124.02
6720	0.000944	0.000843	0.000941	0.000909	0.001391	52.95	64.99
13440	0.001665	0.001726	0.001665	0.001685	0.002026	20.24	21.71
26880	0.003135	0.003202	0.003346	0.003228	0.003298	3.17	5.19
53760	0.006158	0.006681	0.006336	0.006392	0.005840	8.63	13.66
107520	0.011241	0.011927	0.012619	0.011929	0.010924	8.42	15.08
215040	0.021408	0.021853	0.022345	0.021869	0.021094	3.54	5.85
430080	0.041623	0.042100	0.042533	0.042085	0.041432	1.55	2.65
860160	0.082310	0.082211	0.082740	0.082420	0.082108	0.38	0.77
1720320	0.163498	0.163428	0.164049	0.163658	0.163461	0.13	0.36
3440640	0.331059	0.323711	0.322702	0.325824	0.326167	1.11	1.52

Tabla 3.53 Tiempos del patrón de comunicaciones AllToAll para el IBM SP2.

AllToAll	4	6	8	MEDIA	MODELO	ErrMed	ErrMax
210	0.000137	0.000224	0.000235	0.000199	-0.006013	3126.72	4560.65
420	0.000140	0.000232	0.000278	0.000217	-0.005991	2865.23	4478.09
840	0.000162	0.000255	0.000286	0.000234	-0.005948	2638.18	3848.02
1680	0.000189	0.000294	0.000332	0.000272	-0.005861	2257.32	3276.58
3360	0.000251	0.000330	0.000440	0.000340	-0.005687	1770.89	2440.88
6720	0.000410	0.000468	0.000533	0.000470	-0.005338	1235.01	1432.04
13440	0.000668	0.000764	0.000843	0.000758	-0.004642	712.11	821.09
26880	0.001436	0.001357	0.001414	0.001402	-0.003249	331.67	345.24
53760	0.002680	0.002719	0.002702	0.002700	-0.000463	117.14	118.73
107520	0.004989	0.005059	0.005274	0.005107	0.005109	2.19	3.30
215040	0.009812	0.010219	0.010346	0.010126	0.016253	60.51	65.65
430080	0.019806	0.020453	0.021468	0.020576	0.038541	87.31	94.59
860160	0.047599	0.046725	0.053201	0.049175	0.083117	69.02	77.89
1720320	0.140750	0.167841	0.191828	0.166806	0.172270	11.09	22.39
3440640	0.299999	0.365493	0.427379	0.364290	0.350574	13.02	25.60

Tabla 3.54 Tiempos del patrón de comunicaciones AllToAll para el Origin.

AllToAll	CRAY T3E	DIGITAL	IBM SP2	ORIGIN
L_{AA}	4.08E-05	2.12E-02	7.55E-04	-6.03E-03
g_{AA}	2.43E-08	2.52E-07	9.46E-08	1.04E-07

Tabla 3.55 Valores medios de g y L para el patrón AllToAll.

3.9.2.6 Gráficas de variaciones de las *g*

En esta sección hemos utilizado MPI sobre cuatro arquitecturas de multicomputadoras diferentes: Cray T3E, Digital, IBM SP2 y Origin 2000. En las gráficas 3.17, 3.18, 3.19 y 3.20 se muestran las variaciones del parámetro *g* respecto al número de procesadores. Cada gráfica representa una arquitectura diferente y para todas ellas se han representado todos los patrones (Exchange, PingPong, OneToAll, AllToOne y AllToAll) para 4, 6 y 8 procesadores. El valor de *g* viene expresado en segundos por cada entero enviado. Cada uno de los patrones ha sido dibujado con una etiqueta diferente que se respeta en todas las arquitecturas.

En la primera figura, se muestran los valores obtenidos para el Cray T3E. En ella se aprecia una alta invarianza de los patrones al aumentar el número de procesadores. La diferencia entre el patrón más rápido, el Exchange, y el más lento, el PingPong es de casi el doble. El resto de patrones se mantiene entre los dos anteriores, con una tendencia mayor hacia el Exchange.

El menor valor se encuentra en 1,23E-08 para el caso del patrón OneToAll y el caso peor se encuentra con el patrón PingPong con 4,90E-08. Destacar la gran estabilidad de cada patrón respecto al número de procesadores. La influencia del incremento del número de procesadores es menor que la diferencia de los patrones.

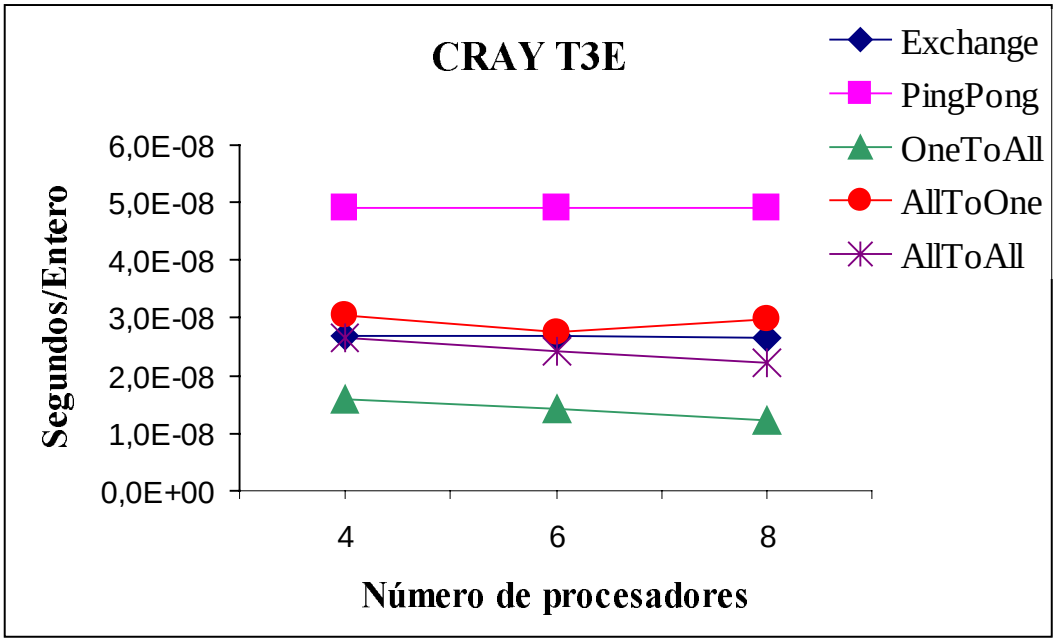

Figura 3. 17 Variaciones de *g* para el CRAY T3E

En la figura 3.18, se muestran los valores para la Digital. El comportamiento de los patrones respecto al número de procesadores es muy dependiente. Todos los patrones comienzan con unos valores muy cercanos para cuatro procesadores y según aumentamos su número, la diferencia es mayor. El patrón AllToAll satura el bus de comunicaciones de esta máquina. El resto de patrones menos el OneToAll va incrementando su valor ligeramente con el número de procesadores. En el caso del OneToAll, se repite el fenómeno observado en el punto anterior.

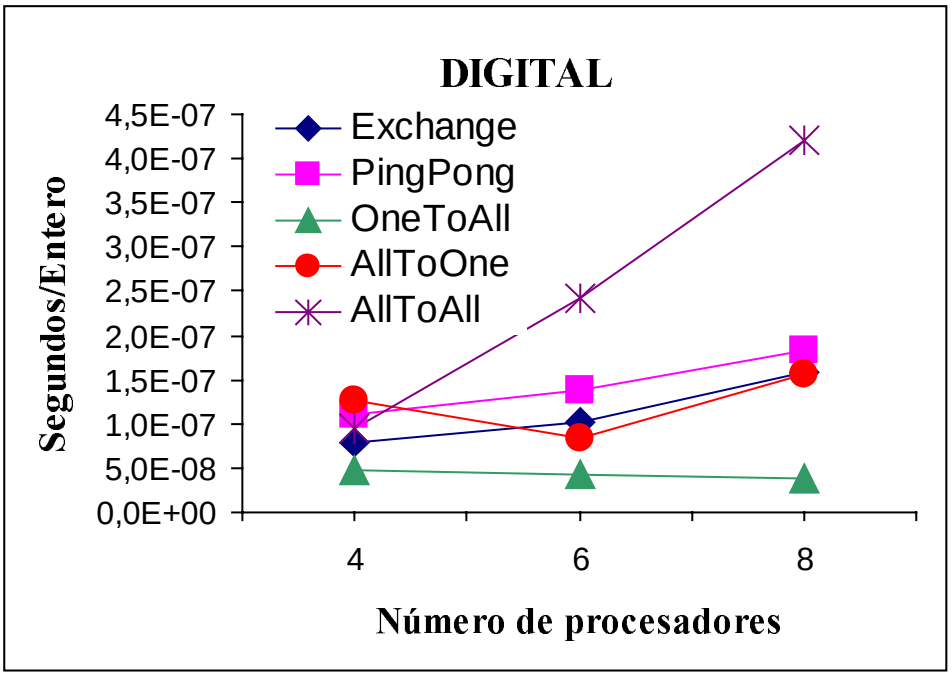

Figura 3. 18 Variaciones de *g* para la DIGITAL

En la figura 3.19 observamos los valores de g para la IBM SP2. Los patrones PingPong y AllToOne siguen una tendencia muy similar, siendo los que más saturan la red. Los patrones Exchange y AllToAll siguen también un comportamiento muy parecido y el OneToAll se aprovecha de las implementaciones internas de MPI para un patrón típico de emisión de uno a varios procesadores el mismo mensaje.

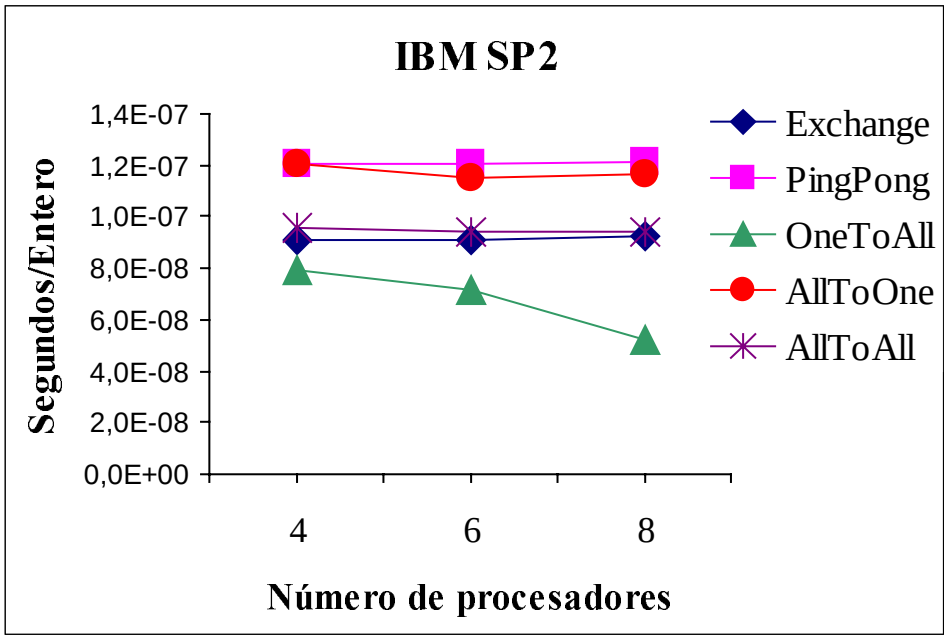

Figura 3. 19 Variaciones de *g* para la IBM SP2.

Los valores para la Origin 2000 se muestran en la figura 3.20. El patrón que más satura la red es el AllToAll seguido del Exchange y PingPong que muestran una conducta

muy similar. En el caso del OneToAll y el AllToOne, se aprecia un decrecimiento de los valores según aumentamos el número de procesadores.

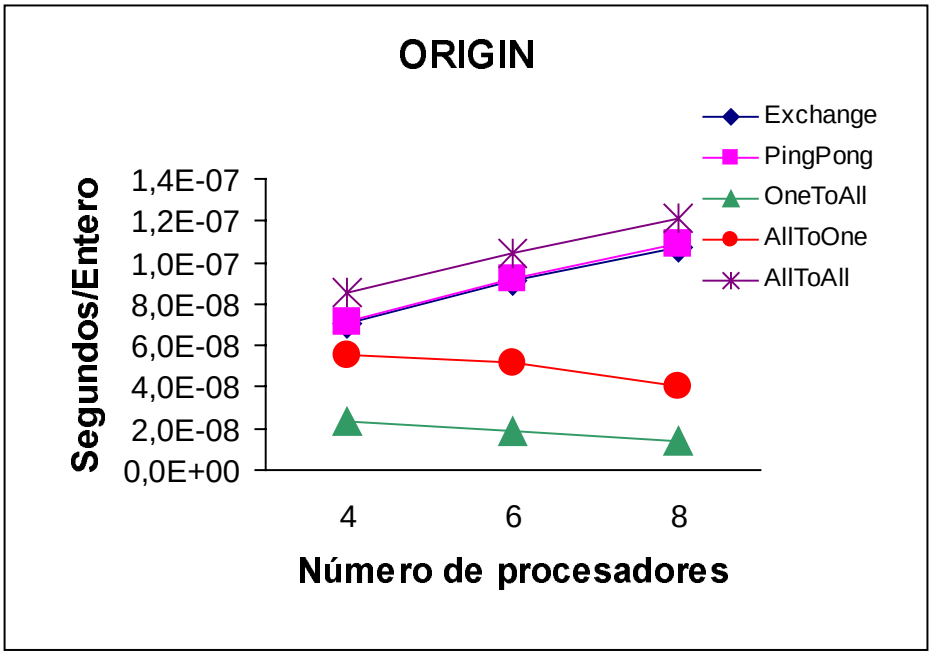

Figura 3. 20 Variaciones de *g* para la ORIGIN.

3.9.2.7 Gráficas según tamaños y patrones

En la tabla 3.56 se muestran los valores medios según el número de procesadores, para los diferentes patrones y la media de todos los patrones. De esta última columna se calcularán los parámetros del modelo BSP.

	Exchange	PingPong	OneToAll	AllToOne	AllToAll	MEDIA	MODELO
210	0.000028	0.000044	0.000027	0.000030	0.000057	0.000037	0.000037
420	0.000036	0.000065	0.000030	0.000034	0.000062	0.000045	0.000044
840	0.000060	0.000099	0.000036	0.000045	0.000072	0.000062	0.000056
1680	0.000100	0.000127	0.000049	0.000073	0.000090	0.000088	0.000080
3360	0.000127	0.000206	0.000071	0.000118	0.000134	0.000131	0.000128
6720	0.000231	0.000375	0.000120	0.000220	0.000204	0.000230	0.000224
13440	0.000426	0.000718	0.000215	0.000421	0.000364	0.000429	0.000417
26880	0.000813	0.001358	0.000407	0.000812	0.000680	0.000814	0.000802
53760	0.001510	0.002681	0.000791	0.001626	0.001340	0.001590	0.001573
107520	0.002945	0.005242	0.001548	0.003210	0.002640	0.003117	0.003114
215040	0.005728	0.010177	0.003059	0.006366	0.005183	0.006103	0.006196
430080	0.011310	0.020887	0.006081	0.012719	0.010483	0.012296	0.012361
860160	0.023001	0.042116	0.012136	0.025423	0.020881	0.024711	0.024690
1720320	0.046303	0.084330	0.024242	0.050566	0.041897	0.049468	0.049349
3440640	0.091437	0.168670	0.048552	0.100949	0.083466	0.098615	0.098666

Tabla 3.56 Variación en los patrones para el Cray T3E.

En las figuras 3.21 y 3.22 mostramos los tiempos para tamaños pequeños y grandes de la *h*-relación. Se puede observar como el patrón OneToAll es el que menores tiempos

medios ofrece para todos los tamaños. El PingPong desde 420 enteros se convierte en el que más tiempo tarda. El valor del parámetro g es de 2,87E-08 para valores grandes y 2,95E-08 para valores pequeños. El término independiente L es de 3,15E-05 para valores grandes y de 3,43E-05 para valores pequeños.

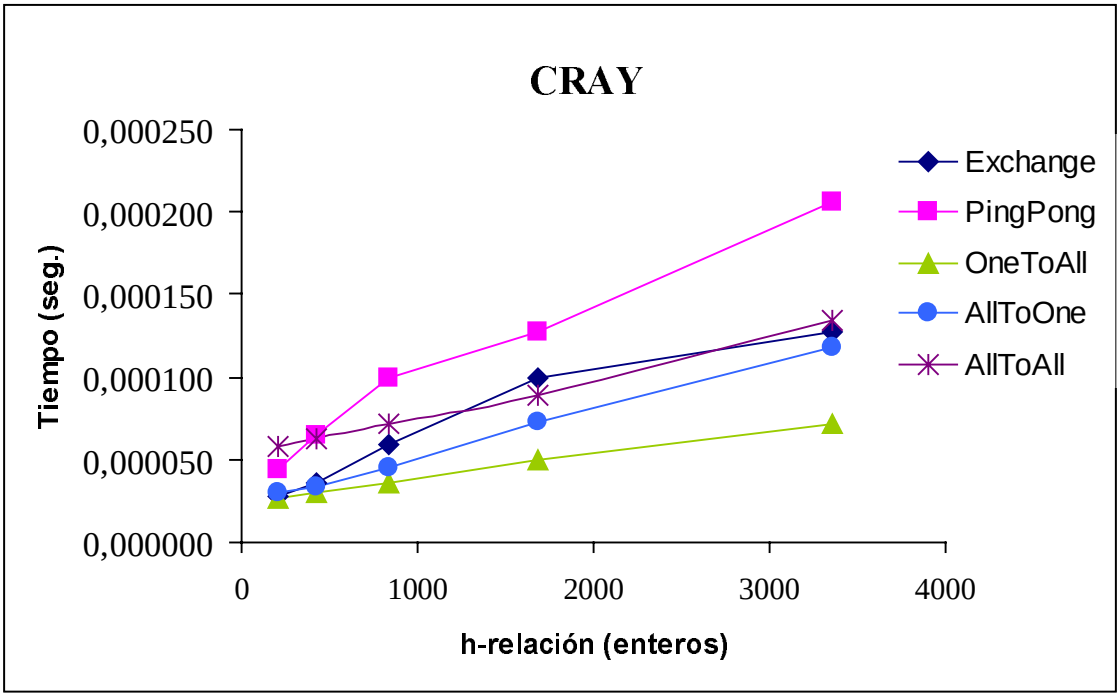

Figura 3. 21 Influencia de los patrones para tamaños pequeños.

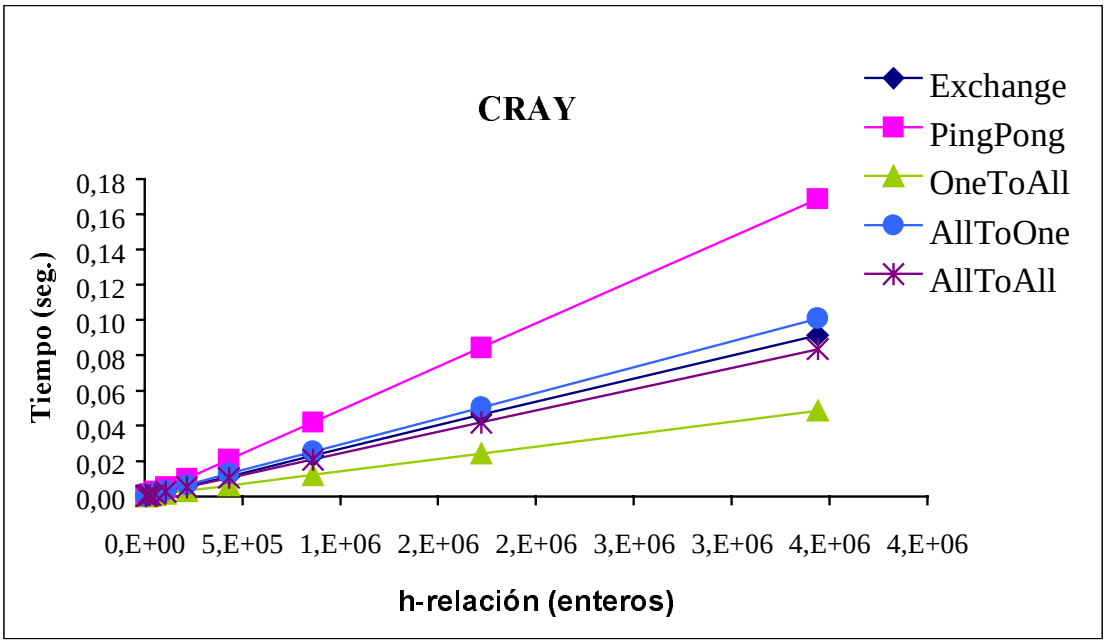

Figura 3. 22 Influencia de los patrones para tamaños grandes.

La siguiente tabla 3.57, ofrece los resultados para la Digital. Todos los patrones ofrecen tiempos crecientes con el tamaño de la *h*-relación siendo el patrón AllToAll el que peor comportamiento ofrece para todos los casos. El valor de *g* para tamaños grandes es 1,28E-07 y de 5,97E-08 para pequeños. El término independiente tiene más diferencias entre los tamaños que para el Cray. Su valor es de 4,24E-03 para valores grandes y de 5,42E-05.

	Exchange	PingPong	OneToAll	AllToOne	AllToAll	MEDIA	MODELO
210	0.000076	0.000035	0.000038	0.000187	0.000543	0.000176	0.004269
420	0.000054	0.000034	0.000037	0.000119	0.000141	0.000077	0.004296
840	0.000075	0.000067	0.000054	0.000147	0.000192	0.000107	0.004349
1680	0.000134	0.000121	0.000089	0.000197	0.000233	0.000155	0.004457
3360	0.000249	0.000247	0.000150	0.000301	0.000324	0.000254	0.004672
6720	0.000493	0.000461	0.000273	0.000540	0.000529	0.000459	0.005103
13440	0.000930	0.001853	0.000521	0.000999	0.000900	0.001041	0.005963
26880	0.002413	0.003478	0.001146	0.002055	0.001818	0.002182	0.007685
53760	0.003953	0.006658	0.002497	0.005669	0.003813	0.004518	0.011127
107520	0.008125	0.013317	0.005282	0.016835	0.023194	0.013351	0.018013
215040	0.016566	0.026123	0.009945	0.018882	0.151754	0.044654	0.031784
430080	0.034160	0.051776	0.019296	0.047234	0.221713	0.074836	0.059326
860160	0.065259	0.108552	0.037552	0.161248	0.303092	0.135141	0.114411
1720320	0.144314	0.227317	0.074335	0.218296	0.519663	0.236785	0.224579
3440640	0.332020	0.438655	0.149473	0.410057	0.825877	0.431216	0.444916

Tabla 3.57 Variación en los patrones para Digital.

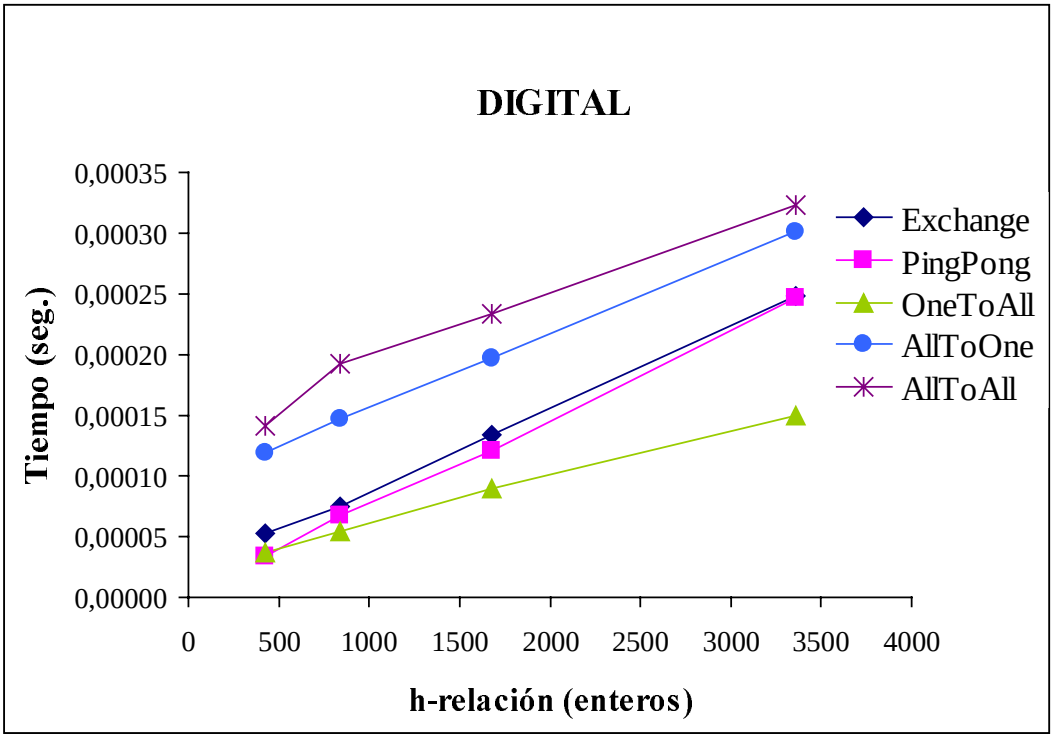

Figura 3. 23 Influencia de los patrones para tamaños pequeños.

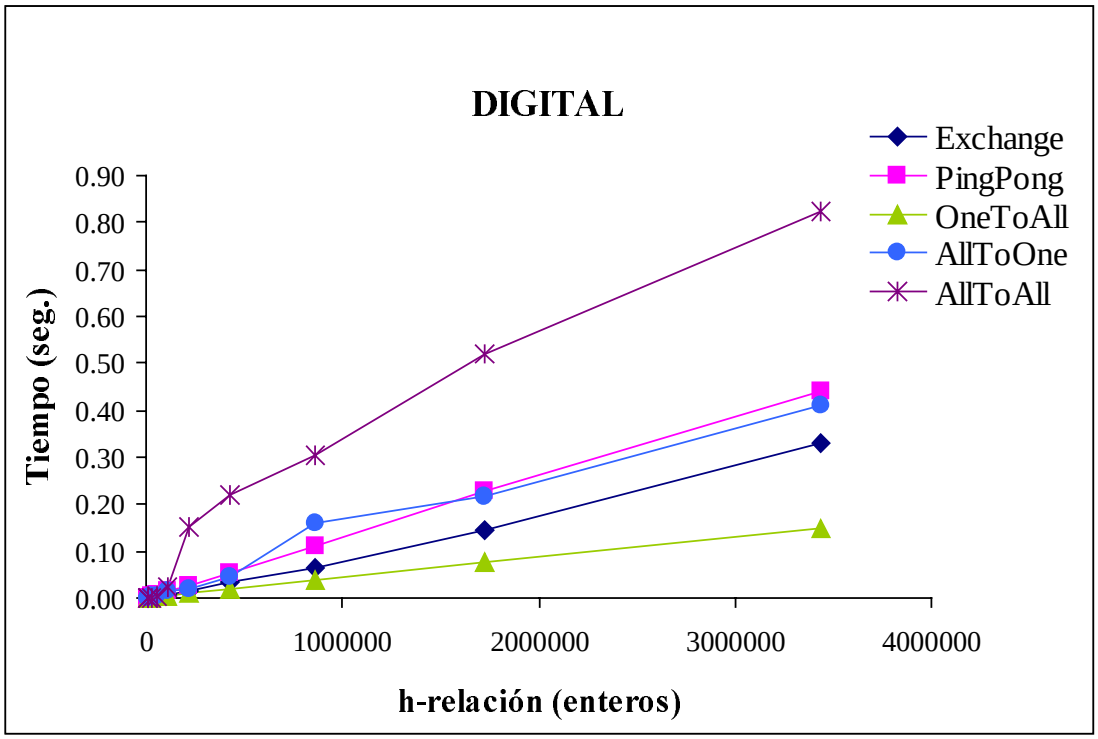

Figura 3. 24 Influencia de los patrones para tamaños grandes.

En el caso de la IBM SP2, los patrones siguen una conducta lineal para tamaños grandes dando un valor de *g* de 9,79E-08 y de 1,25E-07 para valores pequeños. El término independiente es de 5,51E-04 y de 1,95E-04 respectivamente. El patrón que más satura la red es el PingPong y el AllToOne, y el que menos el OneToAll.

	Exchange	PingPong	OneToAll	AllToOne	AllToAll	MEDIA	MODELO
210	0.000106	0.000137	0.000221	0.000290	0.000323	0.000215	0.000472
420	0.000134	0.000170	0.000248	0.000323	0.000353	0.000246	0.000492
840	0.000192	0.000213	0.000297	0.000401	0.000427	0.000306	0.000533
1680	0.000260	0.000390	0.000366	0.000558	0.000487	0.000412	0.000616
3360	0.000507	0.000602	0.000556	0.000852	0.000543	0.000612	0.000780
6720	0.000852	0.001147	0.000923	0.001247	0.000909	0.001016	0.001109
13440	0.001519	0.001900	0.001443	0.001941	0.001685	0.001698	0.001767
26880	0.002725	0.003483	0.002428	0.003636	0.003228	0.003100	0.003084
53760	0.005119	0.006843	0.004263	0.006711	0.006392	0.005866	0.005716
107520	0.009895	0.013463	0.007901	0.012931	0.011929	0.011224	0.010982
215040	0.019547	0.026620	0.015139	0.025694	0.021869	0.021774	0.021512
430080	0.038664	0.052315	0.029772	0.050630	0.042085	0.042693	0.042574
860160	0.077091	0.104137	0.058981	0.100924	0.082420	0.084711	0.084696
1720320	0.154786	0.207860	0.117515	0.205965	0.163658	0.169957	0.168942
3440640	0.307268	0.414705	0.233391	0.403216	0.325824	0.336881	0.337432

Tabla 3.58 Variación en los patrones para IBM SP2.

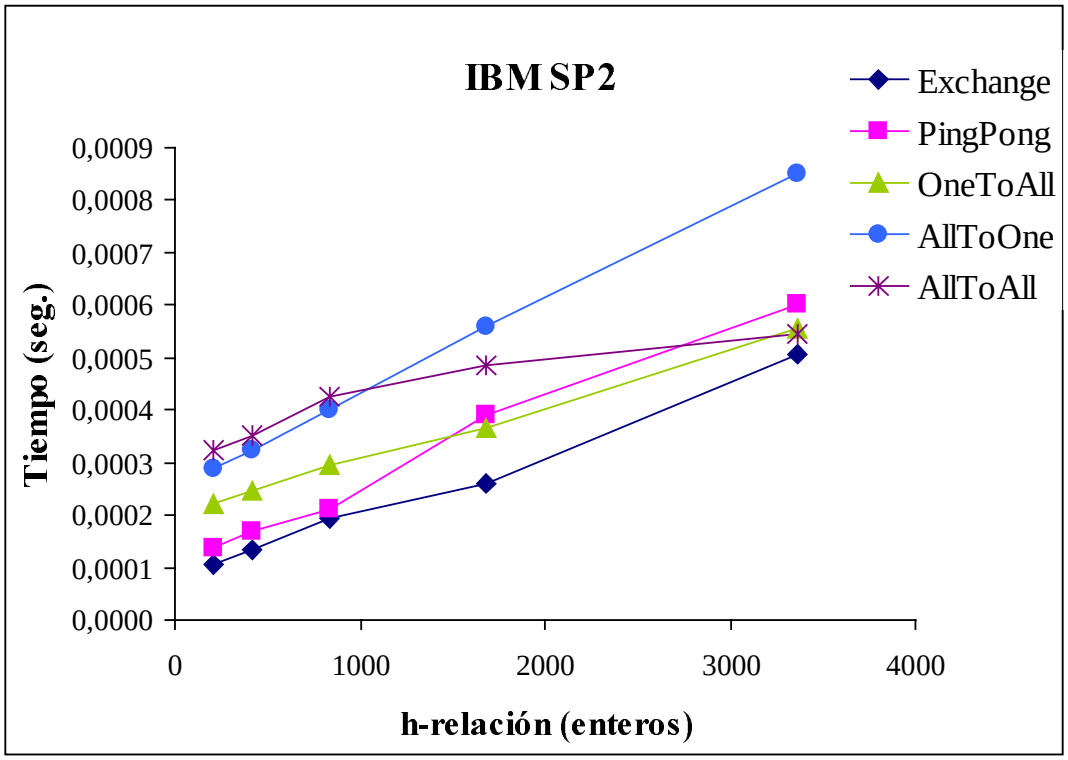

Figura 3. 25 Influencia de los patrones para tamaños pequeños.

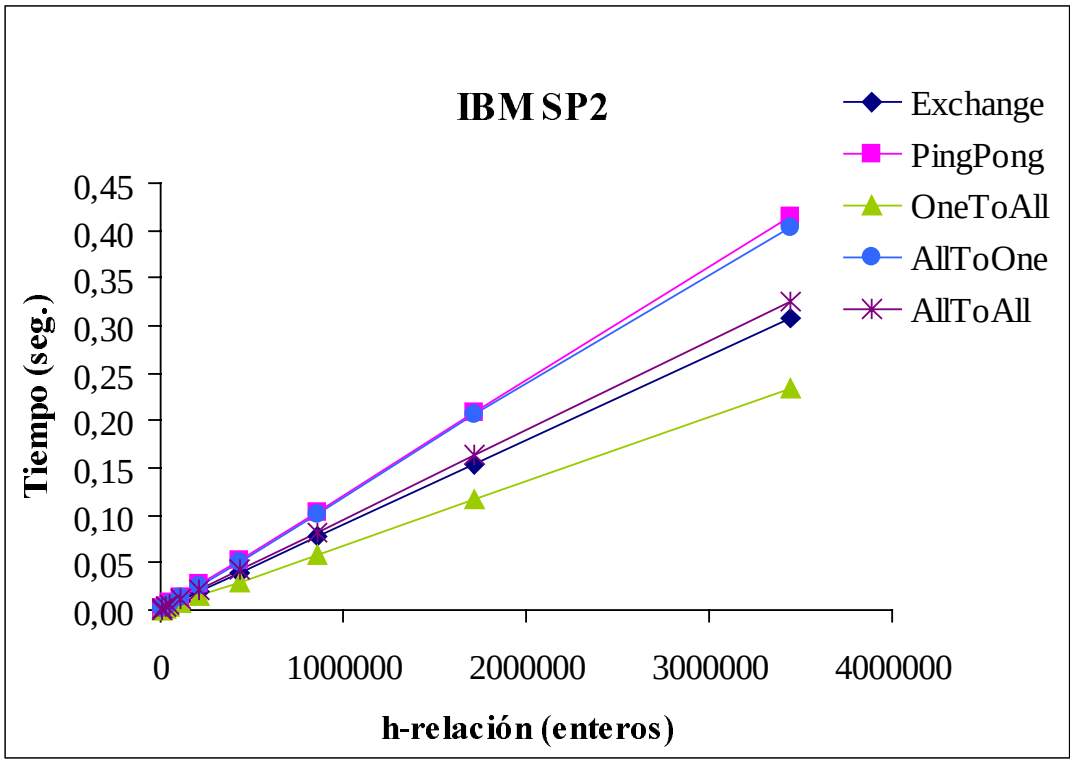

Figura 3. 26 Influencia de los patrones para tamaños grandes.

Los valores para la arquitectura de la Origin ofrecen resultados de *g* de 6,65E-8 para valores grandes y de 3,87E-08 para tamaños pequeños. El término independiente es de –3,20E-03 y 9,09E-05 para valores grandes y pequeños respectivamente. Una observación importante es que en las dos arquitecturas de memoria compartida, el patrón que más satura la red es el AllToAll y el que menos el OneToAll.

	Exchange	PingPong	OneToAll	AllToOne	AllToAll	MEDIA	MODELO
210	0.000067	0.000051	0.000089	0.000101	0.000199	0.000101	-0.003181
420	0.000066	0.000045	0.000092	0.000110	0.000217	0.000106	-0.003167
840	0.000078	0.000064	0.000114	0.000131	0.000234	0.000124	-0.003140
1680	0.000115	0.000103	0.000138	0.000159	0.000272	0.000157	-0.003084
3360	0.000186	0.000207	0.000168	0.000201	0.000340	0.000220	-0.002972
6720	0.000312	0.000392	0.000245	0.000294	0.000470	0.000343	-0.002749
13440	0.000687	0.000659	0.000434	0.000538	0.000758	0.000615	-0.002302
26880	0.001277	0.001177	0.000780	0.001105	0.001402	0.001148	-0.001408
53760	0.002434	0.002198	0.001275	0.002268	0.002700	0.002175	0.000378
107520	0.004617	0.004269	0.002199	0.004215	0.005107	0.004081	0.003952
215040	0.009066	0.008429	0.003841	0.008719	0.010126	0.008036	0.011100
430080	0.018036	0.016991	0.007429	0.018058	0.020576	0.016218	0.025395
860160	0.036359	0.037846	0.014762	0.040042	0.049175	0.035637	0.053986
1720320	0.122817	0.134104	0.029729	0.082342	0.166806	0.107160	0.111166
3440640	0.283381	0.282195	0.066214	0.170829	0.364290	0.233382	0.225528

Tabla 3.59 Variación en los patrones para Origin.

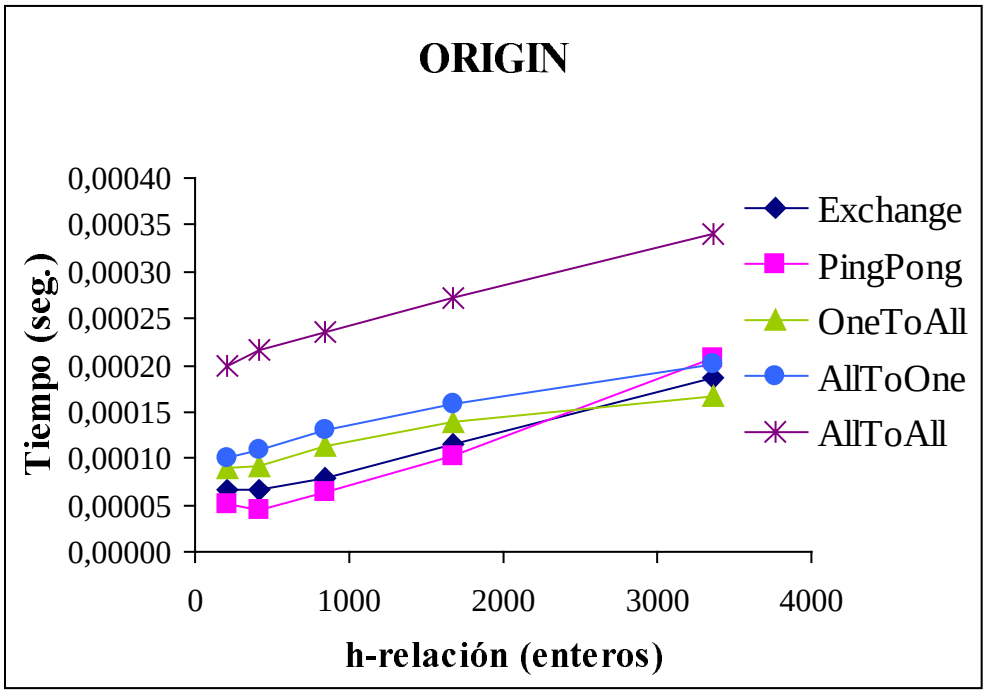

Figura 3. 27 Influencia de los patrones para tamaños pequeños.

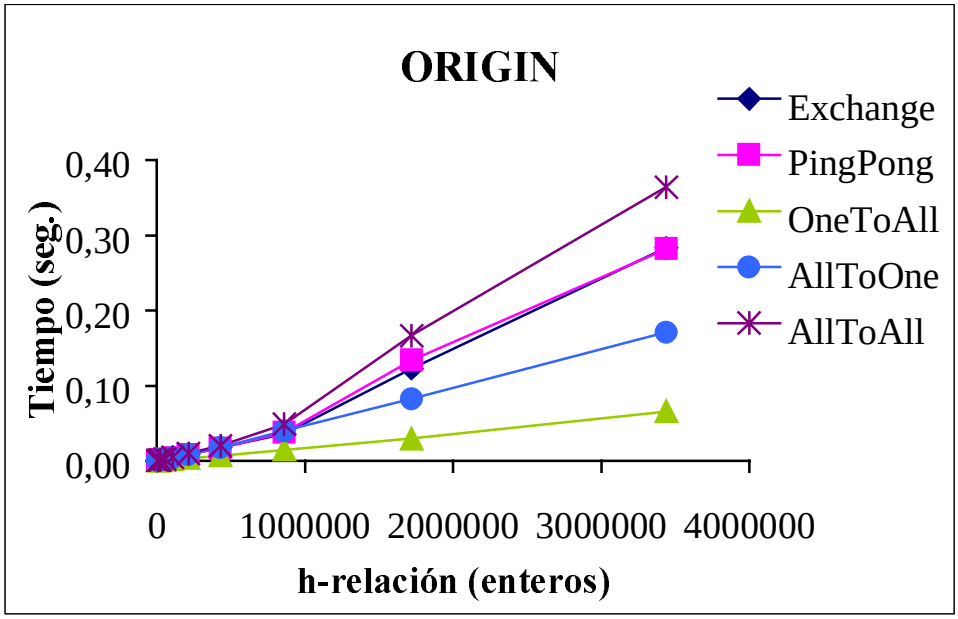

Figura 3. 28 Influencia de los patrones para tamaños pequeños.

3.9.2.8 L y g de todas las arquitecturas

Todos	CRAY T3E	DIGITAL	IBM SP2	ORIGIN
L	3.15E-05	4.24E-03	4.51E-04	-3.20E-03
g	2.87E-08	1.28E-07	9.79E-08	6.65E-08

Tabla 3.60 Valores medios de *g* y *L*.

3.9.3 Estudio de la *h*-relación para un número grande de procesadores en MPI

En este caso hemos considerado los patrones anteriormente vistos más el OneToAll personalizado (OAP). Las ejecuciones se han realizado sobre el Cray T3E y la Origin 2000 y el número de procesadores es de 2, 4, 6, 8, 16 y 24. De nuevo se utiliza MPI como plataforma software para implementar los algoritmos.

3.9.3.1 Exchange

La tabla 3.61 muestra los resultados obtenidos al ejecutar el patrón Exchange con 1, 2, 3, 4, 8 y 12 parejas procesadores en el Cray T3E. Los resultados muestran el espectacular rendimiento de esta máquina al no degradarse para este patrón ni en *h*-relaciones grandes.

Exchange	2	4	6	8	16	24	MEDIA	MODELO
4830	0.000161	0.000166	0.000175	0.000162	0.000179	0.000176	0.000170	0.000322
9660	0.000298	0.000332	0.000336	0.000331	0.000348	0.000369	0.000336	0.000449
19320	0.000541	0.000628	0.000618	0.000653	0.000655	0.000673	0.000628	0.000704
38640	0.001034	0.001146	0.001136	0.001136	0.001140	0.001164	0.001126	0.001213
77280	0.002066	0.002109	0.002142	0.002150	0.002201	0.002166	0.002139	0.002231
154560	0.004084	0.004206	0.004210	0.004232	0.004251	0.004250	0.004206	0.004268
309120	0.008095	0.008208	0.008200	0.008257	0.008261	0.008308	0.008222	0.008342
618240	0.016054	0.016340	0.016374	0.016346	0.016383	0.016444	0.016324	0.016488
1236480	0.033206	0.033388	0.033272	0.033153	0.033183	0.033331	0.033256	0.032782
2472960	0.065735	0.067265	0.065976	0.066071	0.066380	0.066557	0.066331	0.065370
4945920	0.126939	0.131415	0.130547	0.130489	0.130186	0.130306	0.129980	0.130546

Tabla 3.61 Tiempos del patrón de comunicaciones Exchange para el CRAY.

En cuanto a la Origin, el comportamiento no es similar. El incremento del número de procesadores degrada los tiempos de envío en este patrón entre 0,24 y 0,39 para una *h*-relación de 2472960 enteros. A diferencia del Cray los errores en el modelo son significativos y dependientes del número de procesadores.

Exchange	2	4	6	8	16	24	MEDIA	MODELO
4830	0.000186	0.000223	0.000250	0.000262	0.000345	0.000466	0.000289	-0.002746
9660	0.000426	0.000467	0.000497	0.000518	0.000567	0.000643	0.000520	-0.002430
19320	0.000715	0.000860	0.000925	0.000942	0.000977	0.001154	0.000929	-0.001798
38640	0.001418	0.001666	0.001826	0.001885	0.001931	0.002331	0.001843	-0.000532
77280	0.002783	0.003296	0.003733	0.003854	0.003915	0.004489	0.003678	0.001998
154560	0.005282	0.006346	0.006944	0.007056	0.007696	0.008424	0.006958	0.007059
309120	0.010779	0.012625	0.014119	0.014096	0.015109	0.017300	0.014005	0.017180
618240	0.021773	0.025490	0.028752	0.028768	0.030785	0.036074	0.028607	0.037423
1236480	0.048873	0.059734	0.073582	0.080980	0.071714	0.078159	0.068840	0.077909
2472960	0.120693	0.134742	0.160659	0.251653	0.170714	0.173487	0.168658	0.158881
4945920	0.246560	0.277256	0.300495	0.349465	0.350587	0.392299	0.319444	0.320825

Tabla 3.62 Tiempos del patrón de comunicaciones Exchange para ORIGIN.

Los valores de los parámetros del modelo para ambas arquitecturas se muestran en la tabla *E.2*.

Exchange	CRAY T3E	ORIGIN
L_E	1.95E-04	-3.06E-03
g_E	2.64E-08	6.55E-08

Tabla 3.63 Valores medios de *g* y *L* para el patrón Exchange.

3.9.3.2 PingPong

Los resultados son similares que en el Exchange para el caso del Cray. Los valores no dependen del número de procesadores que intervienen y son casi invariantes en el valor aproximado de 0.23 segundos para la mayor *h*-relación. Para valores pequeños, se aprecia cierta variación motivada por cuestiones de cachés, etc.

PingPong	2	4	6	8	16	24	MEDIA	MODELO
4830	0.000277	0.000284	0.000293	0.000297	0.000298	0.000299	0.000291	0.000134
9660	0.000504	0.000528	0.000512	0.000531	0.000526	0.000537	0.000523	0.000367
19320	0.000967	0.001038	0.000963	0.001034	0.001040	0.001040	0.001014	0.000835
38640	0.001897	0.001872	0.001884	0.001883	0.001931	0.001924	0.001899	0.001771
77280	0.003780	0.003795	0.003783	0.003788	0.003794	0.003793	0.003789	0.003642
154560	0.007431	0.007450	0.007446	0.007439	0.007436	0.007523	0.007454	0.007385
309120	0.014639	0.014649	0.014639	0.014651	0.014632	0.014658	0.014645	0.014871
618240	0.029903	0.030025	0.029910	0.029971	0.030069	0.030074	0.029992	0.029842
1236480	0.059875	0.059511	0.059879	0.059519	0.059237	0.059877	0.059650	0.059785
2472960	0.118476	0.118320	0.118479	0.118342	0.118202	0.118459	0.118380	0.119670
4945920	0.232791	0.232722	0.232792	0.232756	0.232665	0.276930	0.240109	0.239441

Tabla 3.64 Tiempos del patrón de comunicaciones PingPong para el CRAY.

En la Origin 2000 de nuevo se aprecia una gran variación que oscila entre 0.25 para 2 procesadores y 0.43 para 24 procesadores.

PingPong	2	4	6	8	16	24	MEDIA	MODELO
4830	0.000280	0.000287	0.000321	0.000332	0.000358	0.000384	0.000327	-0.001285
9660	0.000433	0.000482	0.000526	0.000522	0.000543	0.000564	0.000512	-0.000916
19320	0.000717	0.000839	0.000923	0.000915	0.000958	0.001001	0.000892	-0.000178
38640	0.001296	0.001609	0.001838	0.001796	0.001875	0.001954	0.001728	0.001298
77280	0.002273	0.002982	0.003576	0.003392	0.003608	0.003823	0.003276	0.004249
154560	0.004733	0.005901	0.007068	0.006844	0.007418	0.007992	0.006659	0.010151
309120	0.009387	0.011831	0.014146	0.013636	0.014455	0.015273	0.013121	0.021956
618240	0.019029	0.023789	0.029204	0.028012	0.039919	0.051826	0.031963	0.045565
1236480	0.049539	0.066659	0.082865	0.090001	0.136246	0.182490	0.101300	0.092784
2472960	0.123455	0.146086	0.182047	0.265950	0.278354	0.290758	0.214442	0.187221
4945920	0.255876	0.296786	0.329614	0.426937	0.431353	0.435768	0.362722	0.376097

Tabla 3.65 Tiempos del patrón de comunicaciones PingPong para ORIGIN.

La tabla 3.66 muestra los valores medios de los parámetros para las dos arquitecturas.

PingPong	CRAY T3E	ORIGIN
L_{PP}	-1.00E-04	-1.65E-03
g_{PP}	4.84E-08	7.64E-08

Tabla 3.66 Valores medios de *g* y *L* para el patrón PingPong.

3.9.3.3 OneToAll

La tabla 3.67 muestra los valores para el caso del Cray y patrón OneToAll. El tiempo decrece según aumentamos el número de procesadores en ambas arquitecturas.

OneToAll	4	6	8	16	24	MEDIA	MODELO
4830	0.000095	0.000096	0.000085	0.000076	0.000067	0.000084	0.000069
9660	0.000174	0.000163	0.000145	0.000111	0.000116	0.000142	0.000125
19320	0.000330	0.000298	0.000264	0.000190	0.000165	0.000249	0.000236
38640	0.000638	0.000573	0.000503	0.000357	0.000295	0.000473	0.000458
77280	0.001248	0.001113	0.000979	0.000681	0.000560	0.000916	0.000903
154560	0.002449	0.002207	0.001929	0.001332	0.001085	0.001800	0.001792
309120	0.004883	0.004386	0.003825	0.002630	0.002134	0.003572	0.003571
618240	0.009731	0.008731	0.007614	0.005229	0.004224	0.007106	0.007128
1236480	0.019469	0.017429	0.015209	0.010419	0.008423	0.014190	0.014241
2472960	0.039024	0.034923	0.030536	0.020812	0.016810	0.028421	0.028469
4945920	0.078087	0.070508	0.060965	0.041721	0.033537	0.056964	0.056925

Tabla 3.67 Tiempos del patrón de comunicaciones OneToAll para el Cray.

OneToAll	4	6	8	16	24	MEDIA	MODELO
4830	0.000223	0.000219	0.000206	0.000223	0.000331	0.000240	-0.000525
9660	0.000362	0.000320	0.000323	0.000251	0.000252	0.000302	-0.000446
19320	0.000654	0.000566	0.000572	0.000381	0.000345	0.000504	-0.000289
38640	0.001069	0.001036	0.001025	0.000769	0.000587	0.000897	0.000026
77280	0.001842	0.001755	0.001453	0.001331	0.001118	0.001500	0.000655
154560	0.003253	0.003047	0.002536	0.001860	0.001766	0.002492	0.001914
309120	0.006028	0.005629	0.004560	0.003261	0.002877	0.004471	0.004431
618240	0.011903	0.010947	0.008908	0.006262	0.005215	0.008647	0.009465
1236480	0.023522	0.021823	0.017754	0.012377	0.010164	0.017128	0.019533
2472960	0.053642	0.043814	0.035578	0.024626	0.019925	0.035517	0.039669
4945920	0.152268	0.100044	0.073750	0.048266	0.039045	0.082675	0.079941

Tabla 3.68 Tiempos del patrón de comunicaciones OneToAll para ORIGIN.

OneToAll	CRAY T3E	ORIGIN
L_{OA}	1.37E-05	-6.03E-04
g_{OA}	1.151E-08	1.63E-08

Tabla 3.69 Valores medios de *g* y *L* para el patrón OneToAll.

3.9.3.4 OneToAll Personalizado

En el patrón de comunicaciones uno a muchos (OneToAll) el procesador raíz envía el mismo mensaje a *p-1* procesadores. Las optimizaciones realizadas en este patrón pueden provocar una importante disminución del tiempo de envío. En muchos casos, las implementaciones del OneToAll se realizan siguiendo una distribución de los datos según un árbol. Es necesario realizar el estudio de enviar diferentes mensajes a *p-1* procesadores. Para ello utilizamos la función *MPI_Scatter()* donde uno de los parámetros,

root, indica el procesador que realiza la emisión y por tanto el resto, al ejecutar esta función, están pendientes de recibir los diferentes mensajes.

La tabla 3.70 muestra los valores obtenidos con el patrón OneToAll personalizado. Se aprecia un considerable aumento del tiempo respecto al patrón anterior. Se observa la optimización llevada a cabo en el OneToAll (función *MPI_Bcast()*). La relación existente entre ambos patrones es de 2.28 para 2 procesadores, aumentando hasta 4 con 24 procesadores.

OneToAll_P	4	6	8	16	24	MEDIA	MODELO
4830	0.000178	0.000182	0.000178	0.000246	0.000272	0.000211	0.000117
9660	0.000341	0.000315	0.000307	0.000340	0.000459	0.000352	0.000251
19320	0.000646	0.000590	0.000571	0.000582	0.000614	0.000601	0.000520
38640	0.001268	0.001135	0.001090	0.001058	0.001072	0.001125	0.001059
77280	0.002512	0.002250	0.002169	0.002048	0.002038	0.002203	0.002136
154560	0.004936	0.004485	0.004273	0.004003	0.003954	0.004330	0.004291
309120	0.009875	0.008894	0.008448	0.007869	0.007768	0.008571	0.008600
618240	0.019906	0.017697	0.016948	0.015628	0.015328	0.017101	0.017217
1236480	0.039414	0.035553	0.033310	0.031152	0.030721	0.034030	0.034453
2472960	0.080589	0.071291	0.068624	0.062872	0.061255	0.068926	0.068924
4945920	0.162157	0.144201	0.135641	0.125380	0.122541	0.137984	0.137866

Tabla 3.70 Tiempos del patrón de comunicaciones OneToAll_P para el CRAY.

El patrón OneToAll personalizado en la ORIGIN es 1.86 veces más lento que el OneToAll antes descrito para 2 procesadores y con 24 procesadores es 4.82 veces más lento.

OneToAll_P	4	6	8	16	24	MEDIA	MODELO
4830	0.000213	0.000252	0.000277	0.000416	0.000713	0.000374	-0.001115
9660	0.000468	0.000384	0.000462	0.000530	0.000720	0.000513	-0.000902
19320	0.000787	0.000759	0.000824	0.000870	0.001092	0.000866	-0.000476
38640	0.001581	0.001426	0.001599	0.001455	0.001678	0.001548	0.000376
77280	0.002957	0.002678	0.002835	0.002818	0.003107	0.002879	0.002081
154560	0.005885	0.005452	0.005213	0.005294	0.005570	0.005483	0.005490
309120	0.011115	0.010546	0.009804	0.010632	0.010832	0.010586	0.012309
618240	0.026139	0.023010	0.022496	0.021931	0.020435	0.022802	0.025946
1236480	0.061686	0.053268	0.048206	0.048245	0.046532	0.051587	0.053220
2472960	0.132714	0.109837	0.096564	0.098397	0.095463	0.106595	0.107768
4945920	0.282360	0.224285	0.200542	0.196126	0.188336	0.218330	0.216865

Tabla 3.71 Tiempos del patrón de comunicaciones OneToAll_P para Origin.

OneToAll_P	CRAY T3E	ORIGIN
L_{OAP}	-1.81E-05	-1.33E-03
g_{OAP}	2.78E-08	4.41E-08

Tabla 3.72 Valores medios de *g* y *L* para el patrón OneToAll_P.

3.9.3.5 AllToOne

El tiempo del patrón AllToOne es creciente respecto al número de procesadores, llegando casi a doblarse.

AllToOne	4	6	8	16	24	MEDIA	MODELO
4830	0.000180	0.000177	0.000171	0.000178	0.000179	0.000177	-0.000136
9660	0.000325	0.000320	0.00033	0.000325	0.000342	0.000328	0.000021
19320	0.000610	0.000620	0.000616	0.000607	0.000611	0.000613	0.000335
38640	0.001209	0.001210	0.001208	0.001202	0.001202	0.001206	0.000963
77280	0.002392	0.002405	0.002467	0.00234	0.002364	0.002394	0.002218
154560	0.004696	0.004718	0.004773	0.004692	0.004698	0.004715	0.004729
309120	0.009337	0.009404	0.009606	0.009321	0.009256	0.009385	0.009750
618240	0.018601	0.018685	0.018942	0.018969	0.018623	0.018764	0.019792
1236480	0.037083	0.036763	0.038203	0.03778	0.045587	0.039083	0.039876
2472960	0.074407	0.072840	0.074844	0.075354	0.108167	0.081122	0.080044
4945920	0.148312	0.143310	0.145668	0.150647	0.212988	0.160185	0.160381

Tabla 3.73 Tiempos del patrón de comunicaciones AllToOne para el Cray.

AllToOne	4	6	8	16	24	MEDIA	MODELO
4830	0.000261	0.000266	0.000281	0.000524	0.001196	0.000506	-0.001460
9660	0.000474	0.000391	0.00041	0.000532	0.0007	0.000501	-0.001218
19320	0.000998	0.000825	0.000794	0.00083	0.000969	0.000883	-0.000734
38640	0.001595	0.001522	0.001754	0.001592	0.00155	0.001603	0.000233
77280	0.002842	0.003274	0.00326	0.003147	0.003012	0.003107	0.002169
154560	0.005841	0.005531	0.005635	0.006487	0.004825	0.005664	0.006041
309120	0.013091	0.011850	0.009316	0.012652	0.009465	0.011275	0.013784
618240	0.028721	0.027999	0.020599	0.025152	0.020706	0.024635	0.029270
1236480	0.066026	0.063009	0.052017	0.057901	0.051945	0.058180	0.060243
2472960	0.138630	0.124968	0.112196	0.128849	0.113774	0.123683	0.122189
4945920	0.271995	0.253498	0.231756	0.244853	0.230692	0.246559	0.246079

Tabla 3.74 Tiempos del patrón de comunicaciones AllToOne para Origin 2000.

AllToOne	CRAY T3E	ORIGIN
L_{AO}	-2.92E-04	-1.70E-03
g_{AO}	3.25E-08	5.01E-08

Tabla 3.75 Valores medios de *g* y *L* para el patrón AllToOne.

3.9.3.6 AllToAll

El patrón AllToAll es el que más satura la red de comunicaciones. El envío de *p-1* mensajes de cada procesador al resto provoca los mayores tiempos. Sin embargo, los valores crecen lentamente con el aumento de procesadores en el Cray. Para la Origin el incremento es superior aunque los valores del modelo están en el orden de las centésimas.

AllToAll	4	6	8	16	24	MEDIA	MODELO
4830	0.000153	0.000167	0.000223	0.000271	0.000320	0.000227	0.000213
9660	0.000287	0.000292	0.000345	0.000427	0.000481	0.000366	0.000350
19320	0.000508	0.000530	0.000597	0.000722	0.000835	0.000638	0.000624
38640	0.000982	0.001023	0.001186	0.001313	0.001397	0.001180	0.001171
77280	0.001903	0.002021	0.002252	0.002511	0.002628	0.002263	0.002266
154560	0.003775	0.003903	0.004489	0.004795	0.005024	0.004397	0.004456
309120	0.007533	0.007945	0.009056	0.009484	0.010234	0.008850	0.008835
618240	0.014683	0.015919	0.018115	0.019259	0.019750	0.017545	0.017594
1236480	0.029411	0.031922	0.037036	0.036304	0.042824	0.035499	0.035112
2472960	0.058047	0.062810	0.072102	0.071952	0.083264	0.069635	0.070148
4945920	0.118132	0.124710	0.144456	0.145889	0.168746	0.140387	0.140220

Tabla 3.76 Tiempos del patrón de comunicaciones AllToAll para el Cray.

AllToAll	4	6	8	16	24	MEDIA	MODELO
4830	0.000324	0.000477	0.000569	0.001278	0.003033	0.001136	-0.001739
9660	0.000526	0.000663	0.000773	0.001372	0.001839	0.001035	-0.001355
19320	0.000923	0.001071	0.001194	0.001895	0.003397	0.001696	-0.000587
38640	0.001908	0.001955	0.002141	0.003778	0.005663	0.003089	0.000948
77280	0.003949	0.003906	0.004255	0.005307	0.009896	0.005463	0.004019
154560	0.007107	0.007492	0.007754	0.010220	0.011158	0.008746	0.010162
309120	0.014168	0.014925	0.016023	0.018404	0.020366	0.016777	0.022447
618240	0.029981	0.032894	0.035228	0.042609	0.045898	0.037322	0.047018
1236480	0.092766	0.102708	0.083456	0.091966	0.097507	0.093681	0.096158
2472960	0.214617	0.254851	0.167267	0.184455	0.210528	0.206344	0.194439
4945920	0.431310	0.374154	0.333578	0.355493	0.441586	0.387224	0.391002

Tabla 3.77 Tiempos del patrón de comunicaciones AllToAll para Origin.

En la tabla 3.78 se presentan los valores del patrón AllToAll para las dos arquitecturas utilizadas. El término independiente es negativo al haber calculado la recta sobre todos los valores de tamaños y tiempos.

AllToAll	CRAY T3E	ORIGIN
L_{AA}	7.62E-05	-2.12E-03
g_{AA}	2.83E-08	7.95E-08

Tabla 3.78 Valores medios de *g* y *L* para el patrón AllToAll.

3.9.3.7 Gráficas de variaciones de las *g*

Al igual que en las secciones anteriores, estudiamos la variación de *g* con respecto al número de procesadores en los distintos patrones. En las gráficas 3.29, 3.30, 3.31 y 3.32 se muestran las variaciones del parámetro *g* respecto al número de procesadores en el Cray y en la Origin. Cada gráfica representa una arquitectura diferente y para todas ellas se han representado todos los patrones (Exchange, PingPong, OneToAll, OneToAll Personalizado, AllToOne y AllToAll) para 4, 6, 8, 16 y 24 procesadores. El valor de *g* viene expresado en segundos por cada entero enviado.

La figura 3.29 muestra los valores obtenidos para el Cray con los diferentes patrones y número de procesadores variando entre 2 y 24. La variación respecto al número de procesadores es menor que respecto a los patrones. El patrón más costoso es el PingPong y el menos es el OneToAll. El OneToAll Personalizado queda muy aproximado al Exchange. Sólo con 24 procesadores se observa un ligero incremento de los valores de *g* para casi todos los patrones. La mayor diferencia entre los patrones está para el caso de 24 entre el PingPong y el OneToAll, con 4.79E-08.

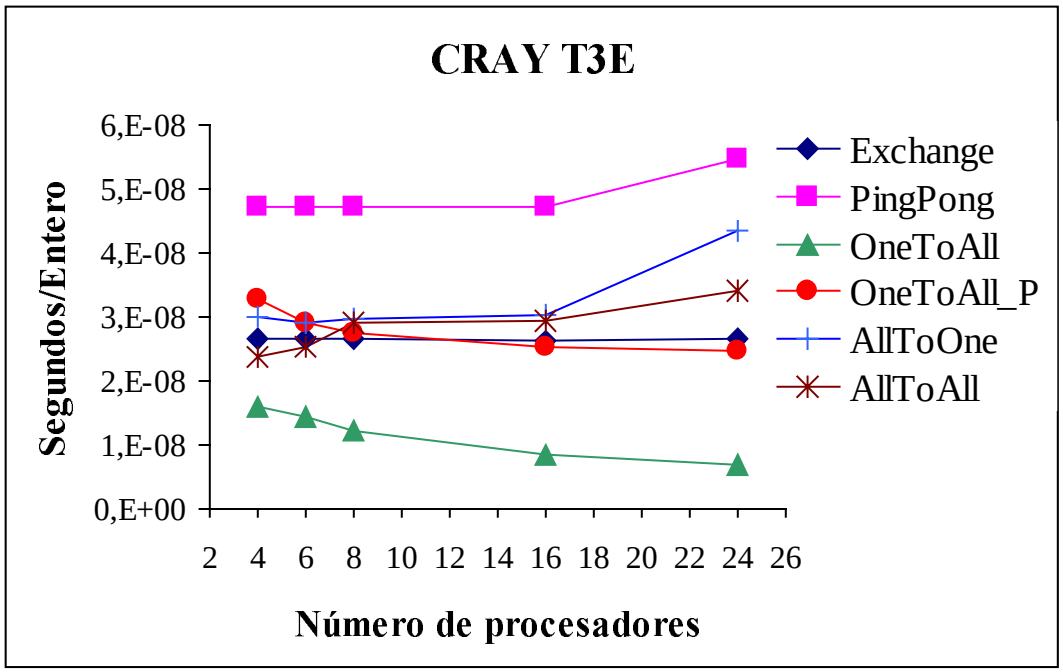

Figura 3. 29 Variación de *g* respecto a los patrones en el Cray T3E.

En la figura 3.30, se muestran los valores para la Origin. Podemos distinguir entre dos grupos de curvas bien diferenciadas que coinciden con los patrones colectivos y los inyectivos. Los patrones colectivos comienzan con un fuerte descenso hasta con 8 procesadores, para luego crecer de forma continua menos el OneToAll. De nuevo el fenómeno de optimización del OneToAll es claro con respecto al OneToAll Personalizado. Otro aspecto importante que para la misma *h*-relación, los patrones AllToOne y OneToAll personalizado siguen la misma conducta. La máxima diferencia entre los patrones es de 8.68E-08 entre el PingPong y el OneToAll.

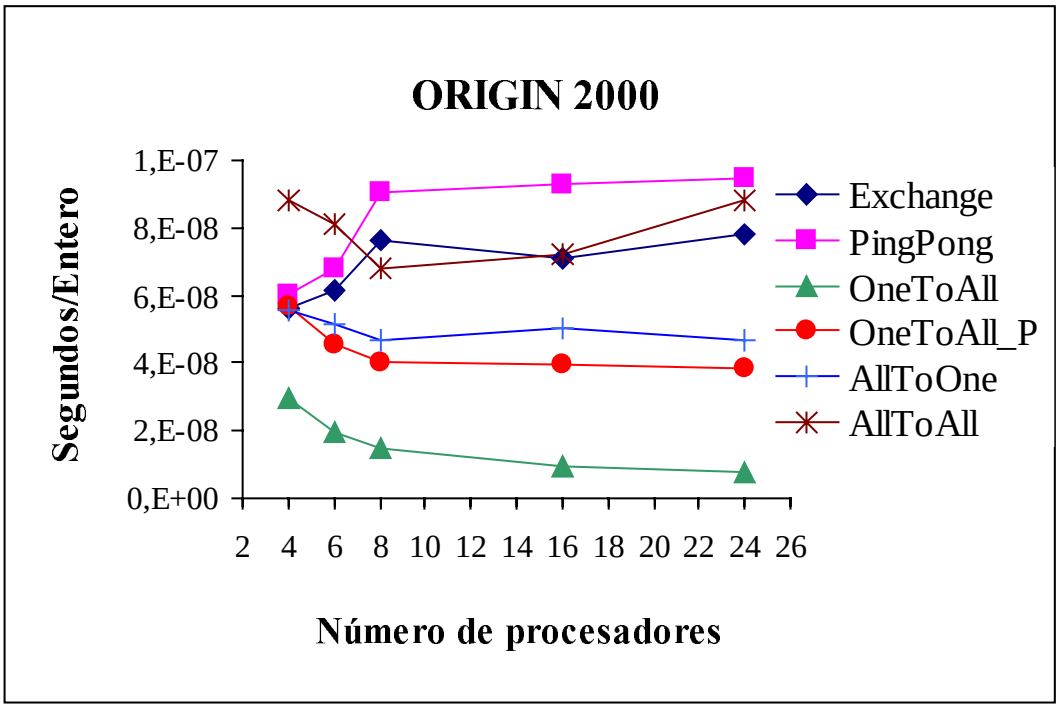

Figura 3. 30 Variación de *g* respecto a los patrones en la Origin 2000.

3.9.3.8 Gráficas por tamaños

En este apartado comparamos los valores de *h*-relaciones de tamaños grandes y pequeños. Las tablas que se presentan muestran los valores medios de cada patrón en el número de procesadores, los valores medios de los patrones y los valores de los parámetros del modelo.

La tabla 3.79 ofrece los valores para el Cray T3E. La diferencia más grande está en los patrones PingPong y OneToAll con un valor de 0.18 para una *h*-relación de 4945920 enteros, incluso superando el valor de algunos patrones. Esto es debido una vez más a las optimizaciones de MPI que nos llevarían a pensar que la *h*-relación en este caso debe ser menor ya que el mismo mensaje se envía a muchos procesadores según un esquema de árbol u otro.

	Exchange	PingPong	OneToAll	OneToAll_P	AllToOne	AllToAll	MEDIA	MODELO
4830	0.000170	0.000291	0.000084	0.000211	0.000177	0.000227	0.000193	0.000120
9660	0.000336	0.000523	0.000142	0.000352	0.000328	0.000366	0.000341	0.000261
19320	0.000628	0.001014	0.000249	0.000601	0.000613	0.000638	0.000624	0.000542
38640	0.001126	0.001899	0.000473	0.001125	0.001206	0.001180	0.001168	0.001106
77280	0.002139	0.003789	0.000916	0.002203	0.002394	0.002263	0.002284	0.002233
154560	0.004206	0.007454	0.001800	0.004330	0.004715	0.004397	0.004484	0.004487
309120	0.008222	0.014645	0.003572	0.008571	0.009385	0.008850	0.008874	0.008995
618240	0.016324	0.029992	0.007106	0.017101	0.018764	0.017545	0.017805	0.018010
1236480	0.033256	0.059650	0.014190	0.034030	0.039083	0.035499	0.035951	0.036042
2472960	0.066331	0.118380	0.028421	0.068926	0.081122	0.069635	0.072136	0.072104
4945920	0.129980	0.240109	0.056964	0.137984	0.160185	0.140387	0.144268	0.144230

Tabla 3.79 Variación en los patrones para Cray.

Las gráficas 3.31 y 3.32 muestran los valores para tamaños grandes y pequeños respectivamente. Las curvas de los patrones PingPong y OneToAll son los límites superior e inferior de los demás patrones. Para tamaños pequeños las curvas de los patrones OneToAll Personalizado, AllToOne, Exchange y AllToAll quedan prácticamente solapadas verificando la hipótesis de la *h*-relación. Para tamaños grandes siguen siendo muy similares aunque con alguna diferencia. El valor de *g* para valores grandes es de 2.92E-08 y el término independiente *L* es –2.11E-05. Para tamaños pequeños *g* da un valor de 2.96E-08 y *L* de 5.21E-05.

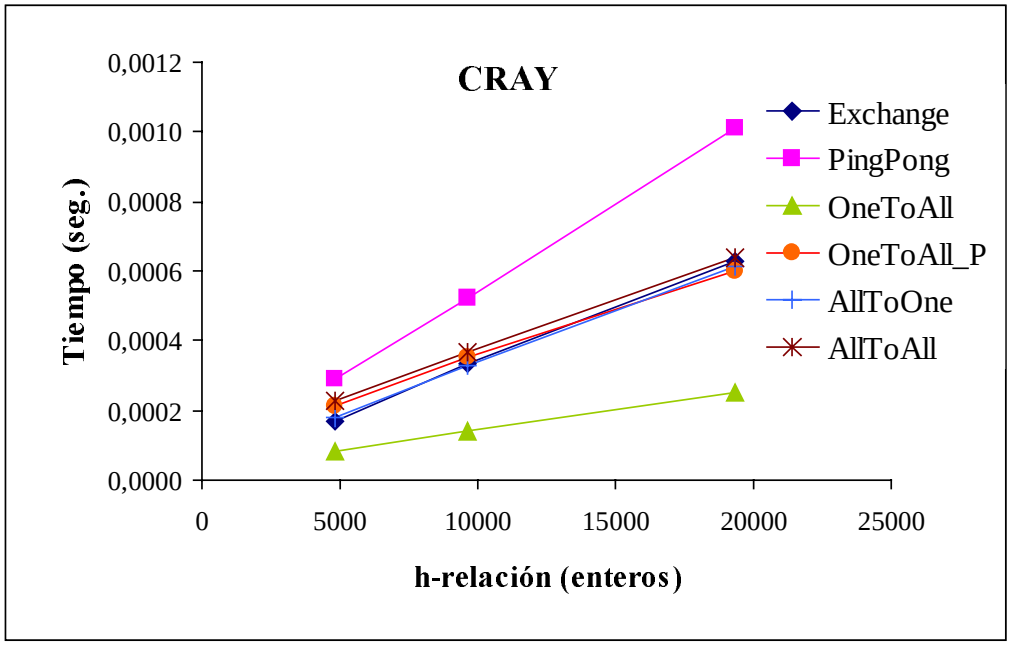

Figura 3. 31 Influencia de los patrones para tamaños pequeños.

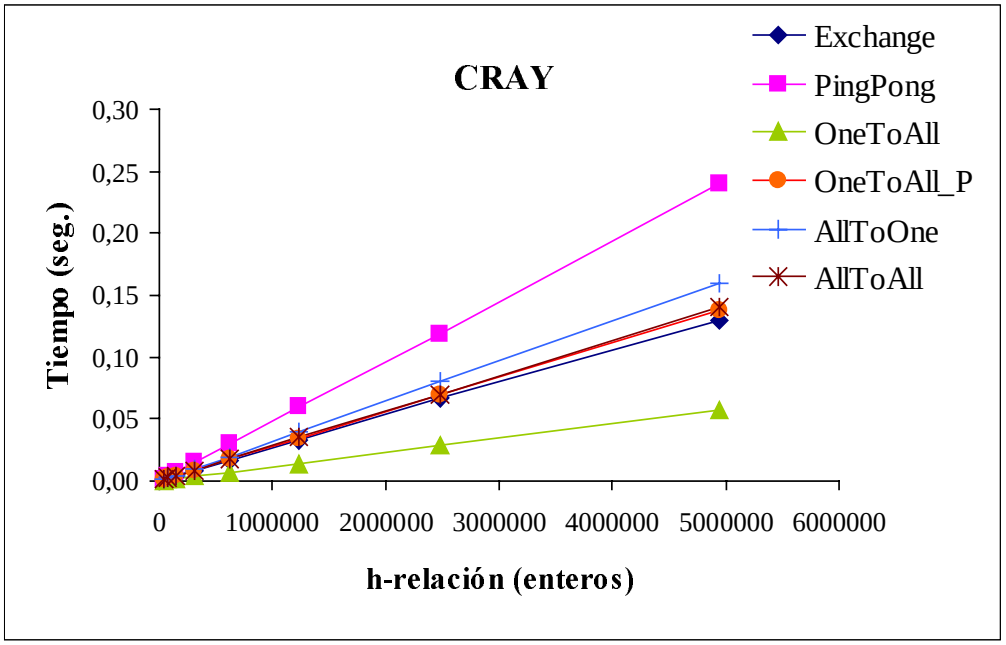

Figura 3. 32 Influencia de los patrones para tamaños grandes.

En el caso de la Origin 2000, la tabla 3.80 nos ofrece los valores medios para cada patrón, la media de los patrones y el valor del modelo según los parámetros estimados. En este caso, para valores pequeños los patrones extremos son el AllToAll para el límite superior y el OneToAll para el inferior. El resto de curvas quedan muy pegadas entre las otras dos. Se observa de nuevo el comportamiento de los patrones colectivos al comenzar con una disminución importante en el tiempo para luego crecer continuamente. Para tamaños grandes la diferencia entre los patrones es mayor que en el caso del Cray. El patrón que ofrece unos tiempos más grandes es el AllToAll seguido de cerca por el PingPong y Exchange. El AllToOne y el OneToAll Personalizado tienen un comportamiento muy similar. El valor de *g* para tamaños grandes es de 5.52E-08 y de 3.98E-08 para valores pequeños. El término independiente vale –1.75E-03 y de 2.27E-04 para valores grandes y pequeños respectivamente.

	Exchange	PingPong	OneToAll	OneToAll_P	AllToOne	AllToAll	MEDIA	MODELO
4830	0.000289	0.000327	0.000240	0.000374	0.000506	0.001136	0.000479	-0.001478
9660	0.000520	0.000512	0.000302	0.000513	0.000501	0.001035	0.000564	-0.001211
19320	0.000929	0.000892	0.000504	0.000866	0.000883	0.001696	0.000962	-0.000677
38640	0.001843	0.001728	0.000897	0.001548	0.001603	0.003089	0.001785	0.000392
77280	0.003678	0.003276	0.001500	0.002879	0.003107	0.005463	0.003317	0.002529
154560	0.006958	0.006659	0.002492	0.005483	0.005664	0.008746	0.006000	0.006803
309120	0.014005	0.013121	0.004471	0.010586	0.011275	0.016777	0.011706	0.015351
618240	0.028607	0.031963	0.008647	0.022802	0.024635	0.037322	0.025663	0.032448
1236480	0.068840	0.101300	0.017128	0.051587	0.058180	0.093681	0.065119	0.066641
2472960	0.168658	0.214442	0.035517	0.106595	0.123683	0.206344	0.142540	0.135028
4945920	0.319444	0.362722	0.082675	0.218330	0.246559	0.387224	0.269492	0.271802

Tabla 3.80 Variación en los patrones para la Origin 2000.

Las figuras 3.33 y 3.34 muestran las curvas de los diferentes patrones para tamaños pequeños y grandes.

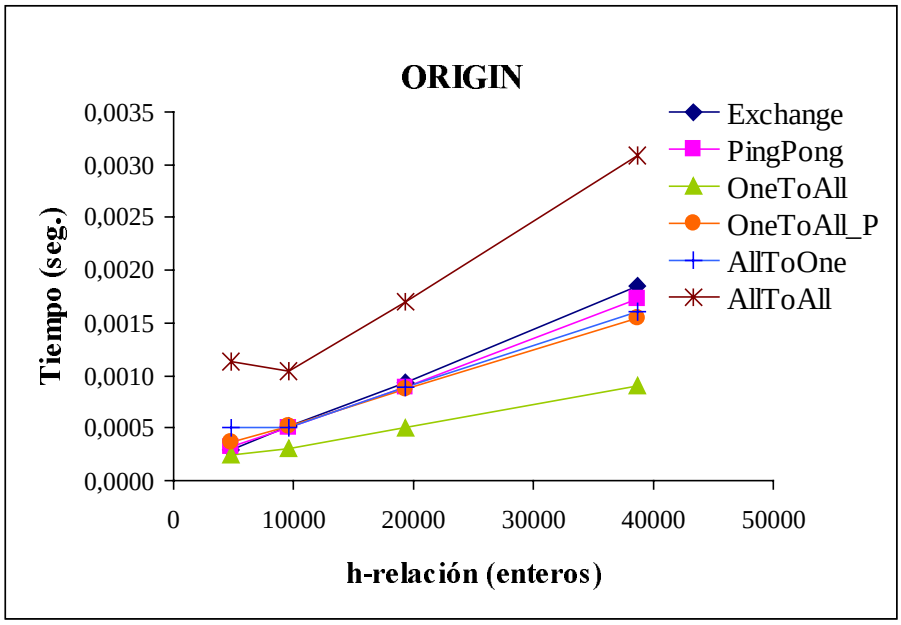

Figura 3. 33 Influencia de los patrones para tamaños pequeños.

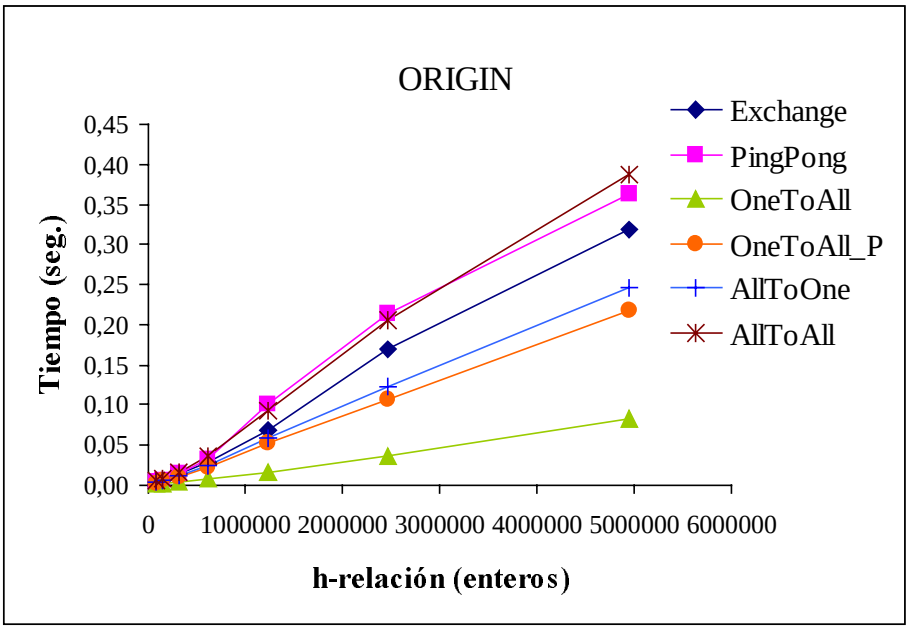

Figura 3. 34 Influencia de los patrones para tamaños grandes.

3.9.3.9 *L* y *g* para todas las arquitecturas

La tabla 3.81 muestra los valores de los parámetros del modelo obtenidos a partir de las medias de los diferentes patrones para cada una de las máquinas.

BSPWB	CRAY T3E	ORIGIN
L	-2.11E-05	-1.75E-03
g	2.92E-08	5.53E-08

Tabla 3.81 Valores medios de *g* y *L*.

3.10 CONCLUSIONES

El modelo BSP es más fácil de usar que el modelo LogP, pero la clase general de algoritmos BSP es más restringida que la de algoritmos LogP. El modelo BSP establece restricciones al imponer el sincronismo por barrera. El modelo BSP funciona de manera precisa si el software que les da soporte es adecuado. Este es el caso de librerías como la Oxford BSPLib.

En este capítulo hemos obtenido los valores de los parámetros del modelo BSP para PVM y MPI. Nuestra propuesta es calcularlos utilizando los patrones de comunicaciones más comunes sobre las máquinas citadas en el capítulo 1. Los resultados nos muestran que en todas las arquitecturas contempladas el error cometido al aceptar la hipótesis de la h-relación es menor cuando varía el número de procesadores que cuando variamos los patrones de comunicación.

El modelo BSP de Valiant nos ha servido como punto de referencia para proponer el modelo que a continuación presentamos. El nuevo modelo permite realizar el diseño y análisis de algoritmos paralelos asíncronos así como obtener el coste computacional de los programas utilizando librerías estándar como PVM o MPI.

4.1 El Concepto M-paso

La ejecución de un programa MPI o PVM consiste en *etapas* de computación seguidas de las comunicaciones necesarias para proporcionar y/o obtener los datos requeridos para la siguiente etapa. Aquí la palabra *comunicación* significa un *flujo continuo* de mensajes. El procesador que hace una o varias recepciones se ve obligado a esperar hasta que se produzcan los envíos deseados desde aquellos procesadores que disponen de los datos que necesita. Cada procesador *i* ejecuta un número variable R_i de tales etapas. Además, estando en una de esas etapas *s*, puede recibir mensajes de procesadores que se encuentran en distintas etapas. Este intercalado de etapas es arbitrario y dificulta el análisis del tiempo de ejecución. El concepto de *M-step* o *M-paso* intenta poner orden en este entramado de etapas. Las etapas de computación pueden ser subdivididas de manera arbitraria por el diseñador del algoritmo en varias etapas más pequeñas.

En el modelo **BSP Sin Barreras** o en inglés **BSP Without Barriers** (**BSPWB**) [Rod98c] la ejecución de un algoritmo MPI/PVM se divide, en cada procesador, en M-pasos. En un paso siempre hay dos fases: una fase de computación seguida de una fase de comunicaciones. Una división en *M-pasos* de un algoritmo MPI es *correcta* en el sentido **BSPWB**, si es una partición en pasos compuestos de fases computación/comunicación (donde una o las dos fases pueden ser vacías) de manera que se cumplan las siguientes dos propiedades:

BSPWB-I. **El número total *R* de *M-pasos* ejecutados por cada uno de los procesadores es el mismo.**

BSPWB-II. **Las comunicaciones siempre tienen lugar entre procesadores que están en el mismo *M-paso s*.**

La primera labor de un diseñador de algoritmos MPI al hacer el análisis **BSPWB** es determinar una partición en M-pasos correcta.

4.2 Modelo Bulk Synchronous Parallel Sin Barreras (BSPWB)

El tiempo $\Phi_{s,i}$ en que un procesador *i* termina un cierto M-paso *s* es posterior al tiempo $\Phi_{s-1,i} + w_{s,i}$, en que termina el paso anterior $\Phi_{s-1,i}$ más el tiempo que invierte $w_{s,i}$ en la fase de cómputo correspondiente al M-paso *s*. Después de pasado este tiempo, debe realizar sus envíos y esperar por la llegada de los mensajes que recibe en este M-paso *s* de los procesadores *j* en el conjunto $\Omega_{s,i}$ de procesadores que le envían mensajes en el paso *s*:

$$\Omega_{s,i} = \{\, j \in \{0, \ldots, P-1\} / \text{El procesador } j \text{ envía un mensaje al procesador } i \text{ en el paso } s\} \cup \{i\} \qquad (4.1)$$

La inclusión del propio procesador *i* en el conjunto $\Omega_{s,i}$, generaliza el concepto de envío a las lecturas en memoria local. Diremos que los procesadores en $\Omega_{s,i}$ son los "socios de entrada" del procesador *i* en el paso *s*.

Denotemos por $h_{s,i}$ el volumen de datos movidos por el conjunto de procesadores en $\Omega_{s,i}$ en el paso *s*:

$$h_{s,i} = max \{ in_{s,j} @ out_{s,j} / j \in \Omega_{s,i} \} \quad i = 0,1,...,P-1 \qquad (4.2)$$

Donde $out_{s,j}$ representa el número de paquetes enviados por el procesador *j* en el paso *s* e $in_{s,j}$ es el número de paquetes recibidos por el procesador *j* en el paso *s*. Como se discutió en la sección del capítulo anterior, dependiendo de la arquitectura específica, la operación @ puede tomarse como la suma o el máximo. El modelo BSP Sin Barreras o BSPWB establece que una cota del tiempo real $t_{s,i}$ en que el procesador *i* finaliza el M-paso *s*, viene dado por el valor $\Phi_{s,i}$ calculado mediante las fórmulas recursivas:

$$\Phi_{s,i} = max \{\Phi_{s-1,j} + w_{s,j} / j \in \Omega_{s,i}\} + (g*h_{s,i} + L),\ i = 0,..., P-1 \qquad (4.3)$$
$$\Phi_{1i} = max \{ w_{1,j} / j \in \Omega_{1,i}\} + (g*h_{1,i} + L), \quad i = 0,1,..., P-1 \qquad (4.4)$$

La diferencia principal del modelo BSPWB con el modelo BSP es que esta fórmula no "castiga" al procesador *i* con una sincronización global, sino que considera una sincronización con sus socios de entrada. El primer sumando de la fórmula *(4.3)* dice que el procesador *i* no termina el paso *s* hasta que todos los procesadores con los que comunica en ese paso han finalizado su cómputo. El segundo sumando no considera los solapamientos que pudieran ocurrir y penaliza el tiempo de comunicaciones del paso *s* con una cota superior conservadora, igual al tiempo invertido por el procesador socio *j* en $\Omega_{s,i}$ con mayor volumen de comunicaciones. Si el conjunto de socios es igual al conjunto total de procesadores *{0, 1,..., P-1}* la sincronización local se convierte en una sincronización global, el M-paso se convierte en un superpaso y el tiempo $\Phi_{s,i}$ se convierte en el tiempo T_s "equivalente" al computado utilizando el modelo BSP:

$$T_s = T_{s-1} + max \{w_{s,i}\} + max \{g*h_{s,i} + L\}, \quad s = 2,...,R \quad i=0,..., P-1 \qquad (4.5)$$
$$T_1 = max \{w_{1,i}\} + max \{g*h_{1,i} + L\}, \qquad i = 0,1,..., P-1 \qquad (4.6)$$

La figura 4.1 nos muestra las diferencias entre el modelo de las fórmulas *(4.5)* y *(4.6)*, al que nos referiremos de aquí en adelante como modelo "BSP-like" y el modelo BSPWB. El diagrama ilustra los tiempos de una aplicación en una máquina con 4 procesadores. La ejecución toma dos M-pasos. Los segmentos blancos de las barras se corresponden con fases de cómputo, mientras que los negros corresponden a las fases de comunicación. Durante el primer paso, los procesadores *0* y *1* realizan una tarea más pesada, que les lleva *4* segundos, que la tarea que ocupa a los procesadores *2* y *3* (*2* segundos). Los números dentro de los segmentos indican tiempo. Después tiene lugar un intercambio según un patrón *Exchange*. Las flechas indican las fuentes y destinos de las comunicaciones. La situación se invierte en el segundo M-paso y ahora los procesadores *0* y *1* realizan una tarea más ligera que *2* y *3*, compensando el desequilibrio anterior. Por último, ocurre un intercambio entre los procesadores *0* y *2* de un lado y *1* y *3* del otro. Mientras que el tiempo obtenido al aplicar las fórmulas *(4.3)* y *(4.4)* del modelo BSPWB es *10* segundos, la cota de tiempo utilizando las fórmulas BSP-like *(4.5)* y *(4.6)* es mayor:

12 segundos (barra etiquetada *Like* en la figura 4.1). En general, la cota del tiempo de computación proporcionada por el modelo BSPWB será mas precisa que la cota que obtengamos utilizando las fórmulas BSP-like *(4.5)* y *(4.6)*.

Puesto que el modelo BSPWB asume que todos los procesadores realizan sus envíos tan pronto cómo finalizan su fase de cómputo, debemos suponer que, o bien el programa ha sido escrito de manera que los envíos y recepciones ocurren en paralelo (a través del uso de *MPI_Sendrecv()*), o bien que las capacidades de "buffering" del sistema de comunicaciones son suficientes (lo que puede lograrse a través del uso de funciones como *MPI_Buffer_attach(), MPI_Bsend()*, etc.).

El tiempo total de un programa MPI en el modelo BSPWB viene dado por la suma del cómputo y comunicaciones realizados en los *R* pasos:

$$\Psi = max\ \{\ \Phi_{R,j} / j \in \{0,...,P-1\}\} \qquad (4.7)$$

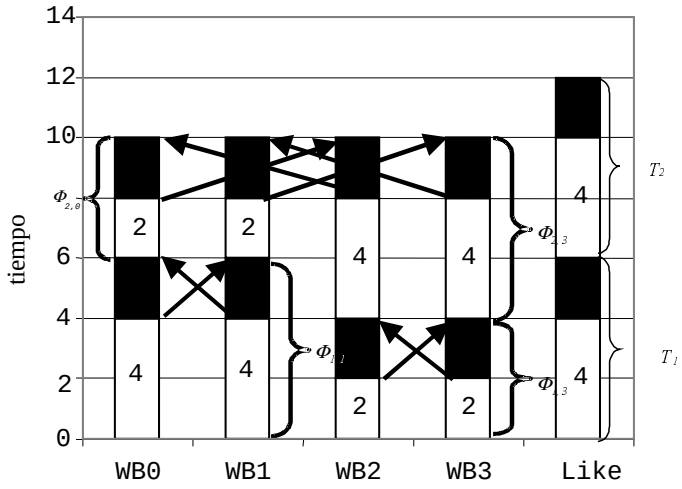

Figura 4. 1 Comparación BSPWB vs. BSP

4.3 ESTUDIO PRÁCTICO DEL MODELO UTILIZANDO PVM

4.3.1 La Transformada Rápida de Fourier

Dada una secuencia *A=(A[0], ..., A[N-1])* de números complejos de longitud *N*, la Transformada Discreta de Fourier (*DFT*) de *A* se define como la secuencia *C=(C[0], ..., C[N-1])* de números complejos, dada por:

$$DFT^N : \mathbf{C}^N \longrightarrow \mathbf{C}^N$$
$$A \longrightarrow DFT(A) = C$$
$$DFT^N{}_j(A) = C[j] = \sum_{k=0..N-1} A[k]\, w^{kj}, \qquad (4.8)$$

Donde $w = e^{2\pi\sqrt{-1}/N}$ es la raíz n-ésima de la unidad en el plano complejo $\boldsymbol{C}$. La ecuación *(4.8)* se puede descomponer en:

$$DFT^N{}_j(A) = C[j] = \sum_{k=0..N/2-1} A[2*k]\, w^{2kj} + w^j \sum_{k=0..N/2-1} A[2*k+1]\, w^{2kj}$$

$$\text{para } 0 \leq j < N \qquad (4.9)$$

Para los valores de j en la segunda mitad, $j = N/2+s$ con $s = 0,\ldots,N/2-1$ sustituyendo en *(4.9)*, tenemos:

$$DFT^N{}_j(A) = \sum_{k=0..N/2-1} A[2*k]\, (w^2)^{ks} - w^s \sum_{k=0..N/2-1} A[2*k+1]\, (w^2)^{ks}$$

Puesto que w^2 es la raíz $N/2$-ésima de la unidad, las fórmulas *(4.8)* y *(4.9)* nos indican que la Transformada de Fourier Discreta C de A puede obtenerse por la combinación de la Transformada de Fourier Discreta de B de las componentes pares y de la Transformada de Fourier Discreta D de las componentes impares de A:

$$DFT^N{}_j(A) = DFT^{N/2}{}_j(B) + w^j * DFT^N{}_j(D) \quad 0 \leq j < N/2 \qquad (4.10)$$

$$DFT^N{}_j(A) = DFT^{N/2}{}_s(B) - w^j * DFT^N{}_s(D) \quad j = N/2+s \qquad (4.11)$$

Estas fórmulas inducen de manera natural un algoritmo divide y vencerás. La versión secuencial de ese algoritmo se conoce con el nombre de Transformada Rápida de Fourier o más abreviadamente, FFT.

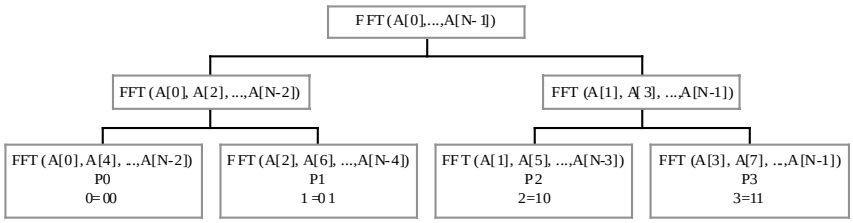

Figura 4. 2 Esquema de división

Sea $P = 2^{log(P)}$ el número de procesadores. Suponemos que, al comienzo del cómputo, todos los procesadores contienen una copia del vector A. La figura 4.2 muestra como se realiza el reparto de tareas entre los procesadores. El correspondiente código se muestra en la figura 4.3.

Entre las líneas 2 y 5 los procesadores calculan en tiempo $D*N/P$ el subvector de A de tamaño N/P que tiene como componentes los elementos resultantes del patrón par impar dado por los bits del nombre *NAME* del procesador (leídos de más significativo a menos significativo). En este momento cada procesador emplea un tiempo $F*N/P*log(N/P)$ en calcular de forma secuencial la Transformada de Fourier sobre sus datos (línea 6). Al entrar por primera vez en el bucle con $i = log(P)-1$ los procesadores con el último bit igual a *1* envían su transformada de las componentes impares al procesador que ha computado la correspondiente transformada de las componentes pares (líneas 8-13). Se tiene, por tanto una h-relación con $h = N/P$. En ese punto termina el primer M-paso:

$$\Phi_{1,NAME} = D*N/P+F*N/P\ *log(N/P)+L+g*N/P \qquad (4.12)$$

y los socios entrantes en este primer M-paso vienen dados por:

$$\Omega_{1,NAME} = \{NAME,\ NAME\ XOR\ 1\}\ si\ NAME\ XOR\ P/2 = 0 \qquad (4.13)$$
$$\Omega_{1,NAME} = \{NAME\}\ si\ NAME\ XOR\ P/2 \neq 0$$

En las sucesivas iteraciones del bucle $i = log(P) - 2, \ldots, 0$, tienen lugar los M-pasos $s = 2,\ldots,log(P)-1$. Los tiempos de estos subsiguientes M-pasos s, se descomponen en el tiempo $V*N*2^{(s-2)}/P$ de la fase de combinación de las líneas 15 a la 20, con valor del bucle $i = log(P) - s + 1$ y las comunicaciones de la mitad de un PingPong de las líneas 7-11 con valor del bucle $i = log(P)-s$, que dan lugar a un tamaño de h-relación $h = N*2^{(s-1)}/P$.

$$\Phi_{s,NAME} = \Phi_{s-1,NAME} + V*N\ 2^{s-2}/P + L+g*N*\ 2^{s-1}/P \quad (4.14)$$

y $\Omega_{s,NAME} = \{NAME,\ NAME\ XOR\ 2^{log(P)-s}\}$ si el procesador $NAME$ participa en el paso. En caso contrario, suponemos que el procesador realiza un M-paso vacío. En el último M-paso $R = log(P)$ sólo trabaja el procesador $NAME = 0$. Este último M-paso únicamente incluye una fase de combinación:

$$\Phi_{log(P),0} = \Phi_{log(P)-1,0} + V*N\ /2 \qquad (4.15)$$

Sustituyendo recursivamente, obtenemos el tiempo total:

$$\Psi = D*N/P+F*N/P*log(N/P)+\sum_{k=0..log(P)-1} L+g*N*2^k/P+ V*N*2^k/P \qquad (4.16)$$

```
1.  /* M_step = 1 */
2.  j := 0;
3.  for (i = NAME; i < N; i+= p)
4.    A[j++] := A[i];
5.  N := N / P;
6.  SequentialFastFourierTransform( B, A, N);
7.  for (i = log(P)-1; i >= 0; i--)   {
8.    if ((2^i  XOR NAME) ) ≠ 0) {
9.      Send B to  NAME XOR 2^i
10.     /* dummy M-steps up to log(P)*/
11.     break;
12.   }  /* if */
13.   Receive D from NAME XOR 2^i
14.   /* M_step ++ */
15.   for (j = 0; j < N; j++)  {
16.     A[j+N] = B[j] - w^j D[j];
17.     A[j]     = B[j] + w^j D[j];
18.   }
19.   N = 2 * N;
20.   w = e^(2π√-1/N)  ;
21. } /* for i = log(P) ... */
```

Figura 4. 3: Código de la FFT para el procesador NAME.

El algoritmo de la FFT se ha ejecutado bajo una Origin 2000, una IBM SP2, una red de estaciones de trabajo conectadas tanto por UTP como por un cable COAXIAL. En la tabla siguiente, mostramos los valores reales y los valores que estimados por el modelo, para 2, 4 y 8 procesadores y un tamaño de problema de 524.288 reales. Los valores del modelo se han calculado a partir de los parámetros L y g, obtenidos en el capítulo anterior. Comprobamos que el modelo es correcto, no sólo en la pequeña diferencia respecto a los valores reales sino también en su comportamiento.

FFT	2	4	8
ORIGIN Real	1.61	0.96	0.85
ORIGIN Modelo	1.55	0.86	0.56
IBM SP2 Real	3.26	1.90	1.27
IBM SP2 Modelo	3.21	1.83	1.18
LAN UTP Real	12.36	11.31	11.03
LAN UTP Modelo	11.00	8.41	7.27
LAN COA Real	11.91	10.80	10.94
LAN COA Modelo	12.76	12.01	11.73

Tabla 4.1 Valores reales y estimados de la FFT para 2, 4 y 8 procesadores.

Las constantes computacionales aparecen en la tabla 4.2. Los errores que se muestran en la tabla 4.3 han sido calculados con la siguiente fórmula: 100*(REAL-MODELO)/REAL. Los valores son muy pequeños en el caso de la IBM SP2. En la Origin 2000, como ya se estudió en la sección anterior, tiene una dependencia importante en el número de procesadores y que aquí se refleja. En la LAN UTP ocurre exactamente lo mismo que para la Origin. En cambio, es asombroso el resultado obtenido para la LAN COA, donde el estudio sobre los patrones y número de procesadores indicaba que sería el que peor se comportara, resulta tener un error muy pequeño. Esto es debido a un efecto de compensación en las sucesivas fases del algoritmo. En el caso de 8 procesadores tenemos $log\ p$ fases, en cada una de las cuales ocurre una comunicación en paralelo entre cada par de procesadores. Hemos estudiado cuidadosamente cada una de las fases y hemos observado que el modelo difiere en gran medida del tiempo real, pero en una fase el modelo es superior en valor que el tiempo real y en otras este valor queda por debajo, apareciendo esta compensación.

CONSTANTES	D	F	V
ORIGIN	2.39E-7	2.84E-7	4.81E-7
IBM SP2	5.51E-7	5.86E-8	8.69E-7
LAN COA-UTP	6.04E-7	1.44E-6	2.65E-6

Tabla 4.2 Constantes computacionales para la FFT.

ERRORES FFT	2	4	8
ORIGIN	3.68	10.85	34.10
IBM SP2	1.59	3.85	6.82
LAN UTP	11.01	25.65	34.13
LAN COA	7.14	11.17	7.22

Tabla 4.3 Errores en las cuatro arquitecturas según el número procesadores.

Las figuras 4.4 y 4.5 muestran gráficamente los resultados de la FFT para las cuatro arquitecturas.

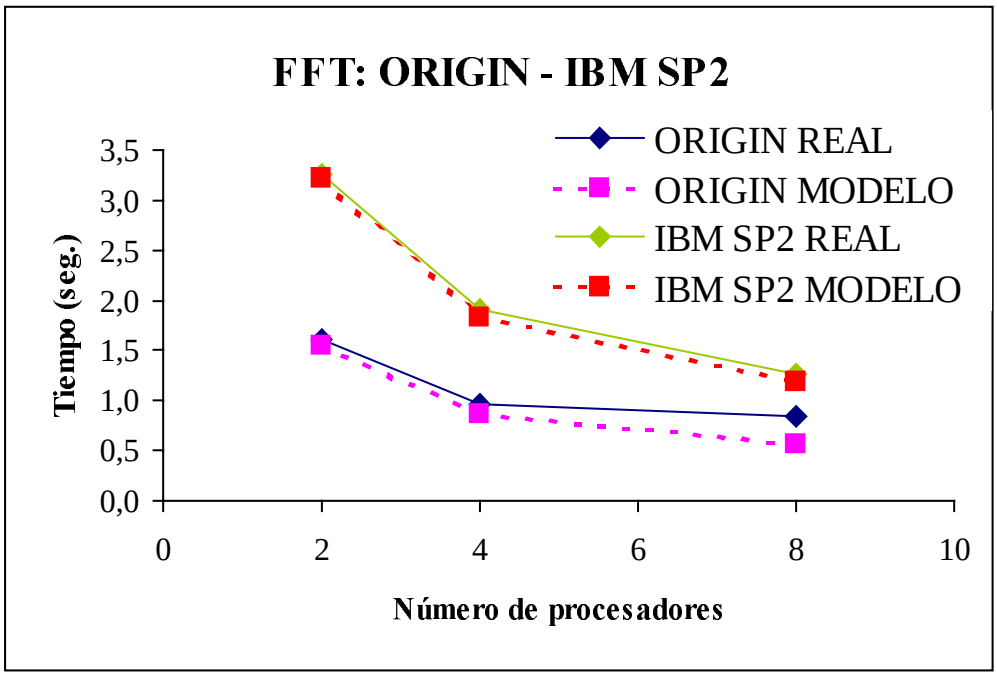

Figura 4. 4 FFT para las multicomputadoras.

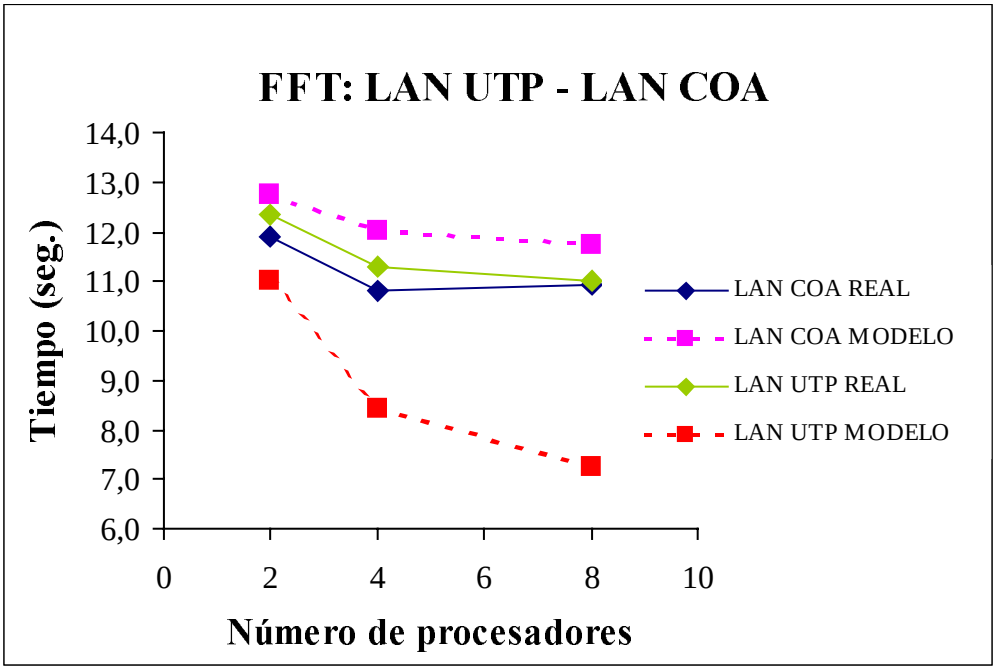

Figura 4. 5 FFT para las redes de área local.

4.4 ESTUDIO PRÁCTICO DEL MODELO UTILIZANDO MPI

El segundo conjunto de experimentos fue realizado sobre multicomputadoras de memoria distribuida como la Cray T3E y la IBM SP2 y sobre multicomputadoras de memoria compartida como la Digital y la Origin 2000. La plataforma software utilizada es MPI. En este caso el algoritmo considerado es la ordenación paralela por el método del QuickSort.

4.4.1 Algoritmo de Ordenación: Quicksort

El código de figura 4.6 muestra una paralelización *natural* del clásico algoritmo de ordenación quicksort. El problema consiste en ordenar un vector A de tamaño N que esta replicado en cada uno de los procesadores y obtener el vector ordenado en cada uno de los procesadores. El ejemplo es un caso particular del paradigma que denominamos "computación replicada". Llamamos computación replicada a aquella en que todos los procesadores del conjunto actual disponen de una replica de los datos de entrada (*input*) del algoritmo al comienzo de la ejecución y se requiere disponer de una replica del resultado (*output*) en todos los procesadores del conjunto actual al final de la ejecución del algoritmo.

```
1  #define TYPE int
2  void qs(int first, int last) {
3    int s1, s2, i, j;
4    if (NUMPROCESSORS > 1)
5      if (first < last) {
6        partition(&i, &j, &s1, &s2, first, last);
7        PAR(qs(first, j), A+first, s1, qs(i, last), A+i, s2);
8        /* M_step++ ;*/
9      }
10   else
11     quicksortseq(first, last);
12 }
```

Figura 4. 6 Quicksort con paralelismo anidado

El código de la Figura 4.6 está escrito usando un conjunto de macros y funciones que extienden MPI y PVM con capacidades para el paralelismo anidado denominado *La Laguna C* [Rod98]. La Figura 4.7 contiene el correspondiente código del cuerpo principal. Aunque facilita las labores de escritura, el uso de *La Laguna C* no es esencial. La metodología utilizada para proporcionar paralelismo anidado puede realizarse sin grandes dificultades en MPI o PVM.

```
1   main(void) {
2    clock_t itime, ftime;
3    int first, last;
4
5    INITIALIZE;
6    initialize(A);   /* Read array A */
7    itime = clock();
8    first = 0; last = SIZE - 1;
9    qs(first, last);
10   ftime = clock();
11   GPRINTF("\n%d: time: (%lf)\n",
12   NAME, difftime(ftime,itime));
13   EXIT;
14  } /* main */
```

Figura 4. 7 Programa principal

El clásico procedimiento *partition*, debido a Hoare, invierte tiempo *B*N* en la división del vector *A* en dos intervalos. Todos los elementos en el intervalo que va desde *i* hasta *last* son mayores que aquellos en el segmento desde *first* hasta *j*. Al comienzo de la computación todos los procesadores están en el mismo "conjunto" y replican el código. Durante ese periodo, la variable *NUMPROCESSORS* guarda el número total *P* de procesadores solicitados al sistema. La llamada a la macro *PAR* de *La Laguna C* utilizada en la línea 7 divide el conjunto inicial en dos conjuntos de procesadores, de manera que uno de los conjuntos realiza la llamada correspondiente a la ordenación del primer intervalo (*qs(first, j)*) y el otro la llamada a la ordenación del segundo intervalo (*qs(i, last)*). Los parámetros adicionales segundo y tercero, *A + first* y *s1*, sirven para indicar que el resultado de la llamada a la función *qs(first, j)* esta constituido por los *s1* enteros apuntados a partir de *A + first*. Análogamente los dos parámetros *A+i* y *s2* describen que los resultados de la segunda llamada están constituidos por los *s2* enteros que siguen a la dirección apuntada por *A+i*. Supuesto que los dos intervalos resultantes de la partición sean del mismo tamaño, los dos conjuntos de procesadores en los que la macro *PAR* escinde el conjunto actual serán también del mismo tamaño. La macro *PAR* se encarga de actualizar el valor de la variable *NUMPROCESSORS*. Cada procesador *NAME* en un grupo *G* elige un procesador su “socio" en el otro grupo *G'* con el que intercambiará los resultados de la computación de su rutina.

Cuando no hay más procesadores disponibles se llama en la línea 11 al algoritmo de ordenación secuencial *quicksortseq(first, last)*. Si la elección de los pivots es correcta, el equilibrado será casi perfecto y el tiempo de la ordenación se aproximará a *C*(n/P*log(n/P))*. Al finalizar las llamadas, los "socios" en los dos subconjuntos de procesadores intercambian los resultados de sus segmentos ordenados dando lugar a una *h*-relación con *h = s1+s2 = (last-first+1)* enteros. Después del intercambio, los dos grupos se reunen en el grupo original. Así pues, el intercambio (según un patrón *Exchange*) dentro de la macro *PAR* crea un nuevo M-paso. Tenemos *log(P)* M-pasos. Puesto que la asignación de procesos a procesadores que realiza la macro *PAR* sólo requiere tiempo constante, el tiempo invertido por un procesador arbitrario *NAME* en el primer M-paso se divide en las *log(P)* llamadas a *partition* mas el tiempo invertido e ordenar su trozo y el del primer intercambio:

$$\Phi_{1,NAME} = \sum_{i=0,\ log(P)-1} B*n/2^i + C*(n/P)*log(n/P) + g*2*n/P + L \qquad (4.17)$$

Los *log(P)-1* M-pasos posteriores sólo conllevan operaciones de intercambio (*Exchange*):

$$\Phi_{s,NAME} = \Phi_{s-1,NAME} + g*(2^s*n/P) + L \quad s = 2,...,log(P) \qquad (4.18)$$

Sustituyendo recursivamente, tenemos el tiempo total:

$$\Phi_{log(P),NAME} = \sum_{i=0,\ log(P)-1} B*n/2^i + C*(n/P)*log(n/P) + \sum_{s=1,\ log(P)} g*(2^s*n/P) + L \qquad (4.19)$$

La tabla 4.4 muestra los resultados del algoritmo QuickSort. Los tiempos de reales de ejecución aparecen con la etiqueta nombre de la máquina y la palabra "REAL". El tiempo del modelo obtenido de la fórmula *(4.19)* y de los parámetros obtenidos en el capítulo anterior, se han puesto con el nombre de la máquina y la etiqueta "MODELO". De la tabla podemos observar la validez del modelo. Los tiempos estimados y reales están muy próximos.

QuickSort	2	4	8
CRAY REAL	3.25	2.04	1.55
CRAY MODELO	3.25	2.05	1.49
DIGITAL REAL	2.58	2.38	3.62
DIGITAL MODELO	3.20	2.73	2.80
IBM SP2 REAL	10.03	6.26	4.41
IBM SP2 MODELO	10.01	6.09	4.18
ORIGIN REAL	3.31	2.34	1.79
ORIGIN MODELO	3.41	2.35	1.83

Tabla 4.4 Valores reales y estimados del QuickSort.

La tabla 4.5 contiene los valores de las constantes computacionales para las cuatro arquitecturas utilizadas.

CONSTANTES	B	C
CRAY	4.05E-08	3.35E-08
DIGITAL	3.41E-08	3.77E-08
IBM SP2	6.58E-08	1.14E-07
ORIGIN	3.07E-08	3.45E-08

Tabla 4.5 Constantes computacionales del QuickSort.

Los errores de este algoritmo se muestran en la tabla 4.6. Se ha calculado de con la fórmula: 100*(REAL-MODELO)/REAL. Menos para la DIGITAL, el resto de máquinas ofrecen un error menor del 6%.

Errores	2	4	8
CRAY T3E	0.03	0.12	4.15
DIGITAL	24.00	14.77	22.59
IBM SP2	0.29	2.82	5.32
ORIGIN	2.80	0.63	1.86

Tabla 4.6 Errores del tiempo real respecto al modelo.

La figura 4.8 muestra las curvas de los tiempos estimados y reales para las cuatro arquitecturas. En casi todos los casos, el solapamiento existente entre ambos casos no permiten diferenciarlos claramente.

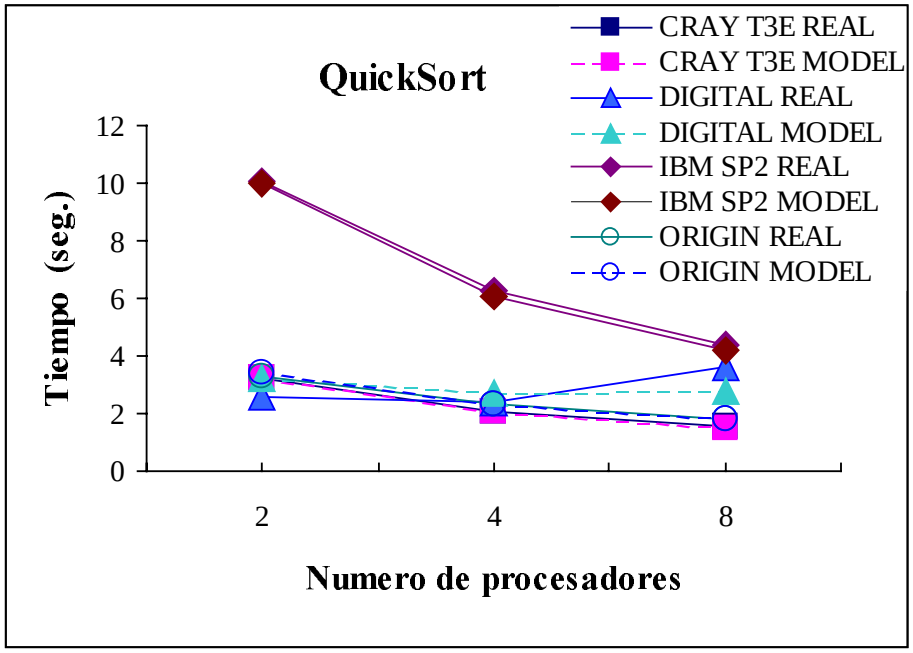

Figura 4. 8 Resultados del quicksort para las cuatro arquitecturas.

4.4.2 Paralelismo de Segmentación: SRAP

En esta sección el modelo BSPWB es aplicado a un algoritmo de programación dinámica paralelo en el que surgen un gran número de mensajes de pequeño tamaño. El problema es conocido con el nombre de *Problema de Asignación de un Unico Recurso* o en inglés *Single Resource Allocation Problem*, de donde vienen las siglas para denotarlo: *SRAP*. El problema consiste en encontrar la asignación óptima de unos recursos limitados del mismo tipo a un conjunto de tareas con vistas a maximizar su rendimiento. Supondremos que tenemos "*M*" unidades de un recurso indivisible y "*N*" tareas, y que por cada tarea $j \in \{ 0, ..., N-1 \}$ tenemos una función asociada $f_j(x)$, que nos da el beneficio obtenido cuando una cantidad x del recurso es asignada a la tarea j. Matemáticamente se puede formular de la siguiente manera:

$$max \quad z = \sum_{j=0}^{N-1} f_j(x_j) \qquad (4.20)$$

$$s.a. \quad \sum_{j=0}^{N-1} x_j \leq M \text{ donde } x_j \in N$$

donde ***N*** denota el conjunto de los números enteros no negativos. Denotemos por *Q[k][m]* el máximo beneficio para el subproblema de asignación de recursos constituido por las primeras "*k*" tareas y "*m*" unidades del recurso. El principio de programación dinámica se puede aplicar a este problema:

$$Q[k][m] = max \{ Q[k-1][m-x] + f_k(x) : 0 \leq x \leq m \} \qquad (4.21)$$

para $k > 0$ con:

$$Q[0][0] = 0 \qquad (4.22)$$
$$Q[1][m] = f_1(m),\ \ m = 0,\ 1,\ \ldots,\ M$$

La figura 4.9 contiene el código para el procesador que se encarga de una etapa genérica *NAME*. A través de la llamada a la macro *IN* en la línea 10, el procesador *NAME* recibe en *x* el valor óptimo *Q[NAME-1][m]* de su vecino izquierdo. En razón del crecimiento de las dependencias, es seguro que después de la actualización de la línea 13, el valor *Q[m] = Q[NAME][m]* esta ahora completamente calculado. Por ello es enviado en la línea 14 al vecino derecho *NAME+1*. Los parámetros segundo y tercero de la macro *OUT* indican el número de elementos a enviar y el tamaño en octetos de los elementos individuales. En las líneas 16 y 17 se actualizan los valores *Q[j] = Q[NAME][j]* con $j \geq m$ que dependen del valor *Q[m] = Q[NAME-1][m]* recién llegado.

```
1.  void transition ()
2.  {
3.    int m, j, x;
4.
5.    for( m = 0; m <= M; m++)
6.      Q[m] = 0;
7.    /* for(M_step=2; M_step<NAME;M_step++); */
8.    for (m = 0; m <= M; m++) {
9.      /* Ω_{M_step,NAME} = { NAME-1, NAME} */
10.     IN(&x);
11.     /*M_step ++ ;  Ω_{M_step,NAME} = { NAME} */
12.     Q[m] = max(Q[m], x + f(NAME-1, 0));
13.     OUT(&Q[m], 1, sizeof(int));
14.     /* M_step++  */
15.     for (j = m + 1; j <= M; j++)
16.       Q[j] = max(Q[j], x + f(NAME - 1, j - m));
17.   } /* for m ... */
18.   /* for (M_step=1; M_step<M+N; M_step++);*/
19. } /* transition */
```

Figura 4. 9 Código para la etapa genérica *NAME*.

El requerimiento **BSPWP-I** exige que las comunicaciones deben ocurrir entre procesadores en el mismo M-paso *s*. Ello implica que el procesador *NAME* debe ejecutar *NAME-1* pasos vacíos antes de la primera recepción. Por tanto, el último procesador *N-1* ejecuta *N-1* M-pasos vacíos además de los M-pasos dentro del bucle de las líneas 9-18. Cada iteración del bucle contiene dos nuevos M-pasos, producidos por las dos comunicaciones que tienen lugar en las líneas 10 y 13. Tenemos por tanto un total de $R = N-1+2*(M+1) = N + 2*M+1$ M-pasos. Para cualquier paso *s* y para cada procesador $N-1 > NAME > 0$ se cumple que:

$$\Omega_{s,NAME} = \{NAME\},\ h_{s,NAME} = 0, \quad \text{para } s < NAME \qquad (4.23)$$
$$\Omega_{s,NAME} = \{NAME\},\ \ h_{s,NAME} = 1 \qquad s = NAME+2*m+1,\ \ m = 0,\ldots,M$$
$$\Omega_{s,NAME} = \{NAME-1,\ NAME\},\ \ h_{s,NAME} = 1,\ s = NAME+2*m,\ m=0,\ldots,M$$

Puesto que la longitud de los mensajes que surgen en el algoritmo es de un entero, utilizaremos en nuestro análisis el valor de la latencia L_0 en vez de los valores asintóticos de L y g.

Tenemos $\Phi_{s,NAME} = A*(M+1)$ para $s < NAME$, donde A es la constante asociada con el bucle que inicializa Q en las líneas 6 y 7. El primer paso no vacío ocurre en el M-paso $NAME$, cuando tiene lugar la primera recepción desde el vecino izquierdo. Aplicando las fórmulas BSPWB *(4.3)* y *(4.4)* para $s = NAME$, se tiene:

$$\Phi_{NAME,\ NAME} = max\ \{\ \Phi_{NAME-1,\ NAME-1} + w_{NAME,NAME-1},\ \Phi_{NAME-1,\ NAME} + w_{NAME,NAME}\ \} + L_0 =$$
$$= \Phi_{NAME-1,\ NAME-1} + w_{NAME,NAME-1} + L_0 = \Phi_{NAME-1,\ NAME-1} + B + L_0 \qquad (4.24)$$

donde B es el tiempo constante invertido en la línea 9. Sustituyendo recursivamente en la fórmula anterior *(4.24)*, vemos que el procesador $NAME$ comienza su primer paso no vacío en el instante:

$$\Phi_{NAME\ NAME} = (NAME-1)*(B + L_0) + \Phi_{1,1} = (NAME-1)*B + NAME*L_0 + A*(M+1) \quad (4.25)$$

Desde este M-paso en adelante el procesador entra en el bucle de las líneas 9-18. Se tiene:

$$\Phi_{s,NAME} = \Phi_{s-1,\ NAME} + B + L_0\ , \quad \text{para}\ \ s = NAME + 2*m + 1,\ m = 0..M \qquad (4.26)$$

para el M-paso de las líneas 12-14 y para el otro M-paso de las líneas 15-18 y 9-11, se tiene:

$$\Phi_{s,NAME} = max\ \{\ \Phi_{s-1,\ NAME-1} + B,\ \Phi_{s-1,\ NAME} + C*(M-m+1)\} + L_0$$
$$\Phi_{s,NAME} = \Phi_{s-1,\ NAME} + C*(M-m+1) + L_0 \qquad (4.27)$$

con $s = NAME + 2*m,\ m = 1..M$, y donde C es el factor constante asociado con el bucle de las líneas 16-17. El tiempo del algoritmo $\Phi_{R,N-1}$ alcanzado cuando el último procesador del "pipe" ejecuta el último paso se obtiene sustituyendo iterativa y alternativamente en las dos anteriores ecuaciones *(4.27)*:

$$\Phi_{R,N-1} = \Phi_{N,-1\ N-1} + \sum_{i=0,M-1} \{\ (B + L_0) + (C*(M-i) + L_0)\} + B \qquad (4.28)$$

$$\Phi_{R,N-1} = \Phi_{N,-1\ N-1} + (B + L_0)*M + (C*(M+1)/2 + L_0)*M + B \qquad (4.29)$$

La fórmula *(4.29)* nos da el tiempo BSPWB en una máquina con N procesadores. Cuando sólo se dispone de P procesadores, debemos tener en cuenta que el último procesador *P-1* comienza en el instante $\Phi_{P-1,\ P-1}$ en vez de en el instante $\Phi_{N-1,\ N-1}$. La complejidad BSPWB de una máquina con P procesadores $\Phi_{R,P-1}$ se obtiene multiplicando el segundo sumando de la fórmula *(4.29)* por N/P.

$$\Phi_{R,P-1} = \Phi_{P-1\ P-1} + ((B + L_0)*M + (C*(M+1)/2 + L_0)*M + B\ )*N/P \qquad (4.30)$$
$$\Phi_{R,P-1} = (P-1)*B + P*L_0 + A*(M+1) + ((B + L_0)*M + (C*(M+1)/2 + L_0)*M + B)*N/P \qquad (4.31)$$

En la tabla 4.7 se exponen los valores estimados y los valores reales obtenidos para el Cray T3E y para la Origin 2000 con 2, 4 y 8 procesadores. Se puede apreciar claramente la bondad del modelo no sólo en precisión sino en comportamiento.

SRAP: 500x500	2	4	8
CRAY REAL	4.88	2.64	1.22
CRAY MODELO	4.54	2.27	1.14
DIGITAL REAL	1.35	0.71	0.41
DIGITAL MODELO	1.20	0.60	0.30
IBM SP2 REAL	12.41	6.21	3.61
IBM SP2 MODELO	12.42	6.21	3.10
ORIGIN REAL	4.35	2.57	1.35
ORIGIN MODELO	4.06	2.03	1.02

Tabla 4.7 Valores reales y estimados del SRAP.

	CRAY	DIGITAL	IBM SP2	ORIGIN
A	1.96E-08	6.3E-09	3.43E-08	5.51E-08
B	1.18E-07	1.6E-08	2.73E-07	1.08E-08
C	6.48E-08	1.8E-08	1.96E-07	6.19E-08

Tabla 4.8 Constantes computacionales del SRAP.

La tabla 4.8 ofrece las constantes computacionales para el SRAP. La tabla 4.9 muestra los errores del modelo con respecto al tiempo real. De nuevo los resultados muestran que aunque la Digital ofrece los mejores resultados en tiempo de ejecución del algoritmo, la predicción del tiempo de ejecución es más difícil de realizar. Este efecto ya se observó en el estudio de los patrones y los errores que introducían.

	2	4	8
CRAY	6.96	14.00	7.07
DIGITAL	11.34	15.82	26.50
IBM SP2	0.07	0.01	13.99
ORIGIN	6.52	20.95	24.56

Tabla 4.9 Errores del SRAP en las diferentes arquitecturas.

La figura 4.10 muestra las curvas reales y estimadas para la IBM SP2 y la Origin y la figura 4.11 muestra las de la Digital y Cray.

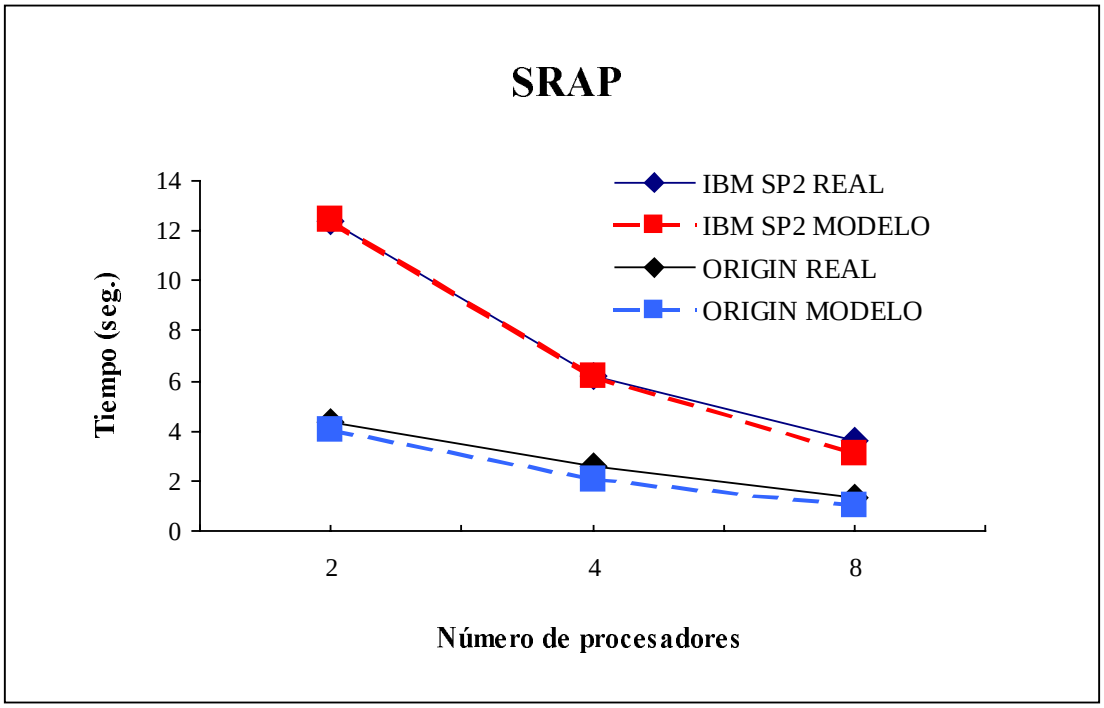

Figura 4. 10 SRAP para la IBM y ORIGIN.

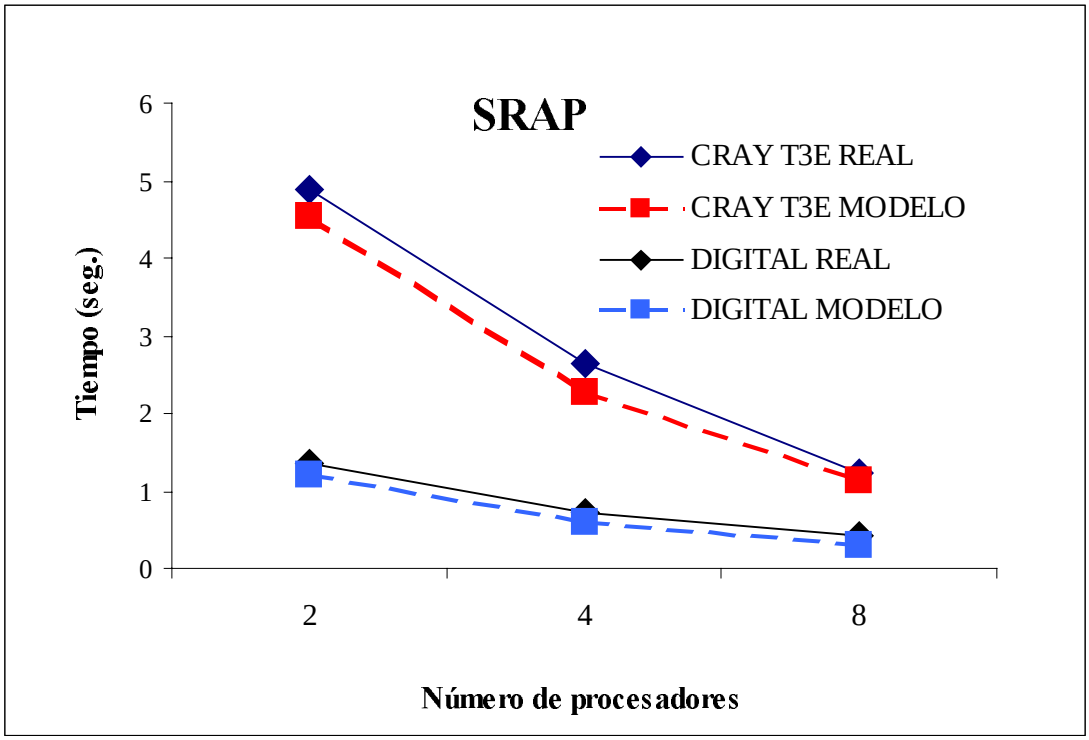

Figura 4. 11 SRAP para la Digital y Cray.

4.5.1 Ordenación Paralela por Muestreo Regular

El algoritmo de ordenación "Parallel Sort by Regular Sampling" o PSRS, debido a Li y otros [Li93] es un ejemplo sencillo de algoritmo síncrono que se adapta al estilo BSP. Se basa en dividir el array total a ordenar y tomar unos elementos como pivotes de referencia. Estos servirán para realizar una nueva partición de los elementos que cada procesador tiene inicialmente, a partir de la cual, se enviarán entre todos ellos los que le pertenecen a cada uno.

Como se puede ver en el correspondiente código MPI de la figura 4.12, el algoritmo hace uso de un conjunto amplio de operaciones colectivas de MPI. El algoritmo en sus siete pasos procede de la siguiente forma. Se comienza por una emisión personalizada, desde el procesador *0* hasta los otros *P-1* procesadores, de los diferentes segmentos de tamaño *N/P* del vector *A* a ordenar (una llamada a la función MPI *MPI_Scatter()* en la línea 2). La constante *MPI_COMM_WORLD* ha sido abreviada a *MCW*). El tiempo que invierte el procesador *NAME* en el primer M-paso $\Phi_{1,NAME}$ es:

$$\Phi_{1,NAME} = g^* (P-1)^*N/P+L \quad (4.32)$$

En el segundo M-paso, cada uno de los *P* procesadores tarda *B*N/P*log(N/P)* en ordenar su segmento (llamada a *SequentialQuicksort()* en la línea 4). Después cada uno de los procesadores elige una muestra de tamaño *P* (línea 6). Las muestras son recolectadas por el procesador *0* con la función *MPI_Gather()* en la línea 7. Ello da lugar a una *h*-relación con *h = P*(P-1)*.

$$\Phi_{2,NAME} = B * N/P * log(N/P) + C * P + g * P * (P-1) + L + \Phi_{1,NAME} \quad (4.33)$$

En el tercer M-paso, el procesador *0* mezcla las *P* muestras (línea 10), selecciona los *P-1* pivotes (línea 11) y los envía a los otros *P-1* procesadores (línea 13):

$$\Phi_{3,NAME} = D^*P^2+E^*(P-1)+g^*(P-1)^*log(P)+L+\Phi_{2,NAME} \quad (4.34)$$

El factor *log(P-1)* en el sumando de comunicaciones proviene de asumir una implementación eficiente de la rutina *MPI_Bcast()*. Puesto que la emisión se realiza internamente según un esquema de comunicaciones en árbol, da internamente lugar a un número de orden logarítmico de M-pasos. No haremos sin embargo, un análisis BSPWB detallado y optaremos por tratarla como un sólo M-paso con una *(P-1)*log(P)* relación. El procedimiento *ComputeFragments* calcula el vector *MySzs* que contiene los tamaños de los segmentos formados por los elementos que están entre pivotes consecutivos (línea 15). Esto se logra mediante una búsqueda binaria de cada uno de los *P-1* pivotes en los *N/P* elementos del correspondiente segmento ordenado del vector *A*. Los tamaños de los fragmentos se intercambian en el *AllToAll personalizado* de la línea 16. De acuerdo con los requerimientos de la función *MPI_Alltoallv()*, los *P* elementos del vector *Ons* contienen un *1* y el *i*-ésimo elemento del vector denominado *Lin* vale *i*. La *h*-relación es *h*

```
1   /* M_step = 1 */
2   MPI_Scatter(A, N/P,MPI_INT,A,N/P,MPI_INT,0,MCW);
3   /* M_step ++ */
4   SequentialQuickSort(A,N/P);
5   Rate = N/(P*P);
6   for(i=0;i<P;i++) Sample[i] = A[i*Rate];
7   MPI_Gather(Sample,P,MPI_INT,S,P,MPI_INT,0,MCW);
8   /* M_step ++ */
9   if (NAME == 0) {
10     PMerge(Dest, S,P1);
11     for(i=1;i<P;i++) Pivots[i-1] = Dest[i*P+P/2-1];
12  }
13  MPI_Bcast(Pivots,P-1,MPI_INT,0,MCW);
14  /* M_step ++ */
15  ComputeFragments(MySzs,A,Pivots);
16  MPI_Alltoallv(MySzs,Ons,Lin,MPI_INT,Szs,Ons,Lin,MPI_INT,MCW);
17  /* M_step ++ */
18  ComputeOffsets(MySize,OfIn, OfOu, Szs);
19  MPI_Alltoallv(A,MySzs,OfOu,MPI_INT,Seg,Szs,OfIn,MPI_INT,MCW);
20  /* M_step ++ */
21  PMerge(Dest,Seg,Szs);
22  MPI_Gather(&MySize,1,MPI_INT,Cnt,1,MPI_INT,0,MCW);
23  /* M_step ++ */
24  if (NAME == 0) ComputeDisplacements(OfIn, Cnt);
25  MPI_Gatherv(Dest, MySize,MPI_INT,A,Cnt,OfIn,MPI_INT,0,MCW);
```

Figura 4. 12 Código MPI para el PSRS.

$= 2*(P-1)$, ya que cada procesador envía sus $P-1$ tamaños y recibe los $P-1$ tamaños de los fragmentos que le corresponden desde los otros procesadores:

$$\Phi_{4,NAME} = F * (P-1) * log(N/P) + g * 2 * (P-1) + L + \Phi_{3,NAME} \qquad (4.35)$$

Una vez que los procesadores han obtenido en el vector *Szs* sus correspondientes tamaños, computan las desviaciones, *OfIn,* donde deben ser almacenados los segmentos que van a ser recibidos y las desviaciones, *OfOu*, donde comienzan los segmentos a enviar. También se computa el número final, *MySize,* de elementos que terminarán en el procesador (llamada a *computeOffsets()* en la línea 18). Los resultados teóricos y experimentales [Li93] demuestran que puede esperarse un comportamiento equilibrado y que el tamaño del segmento dirigido a cada procesador esté próximo a N/P^2. Al final del quinto M-paso, cada procesador envía $P-1$ segmentos de aproximadamente N/P^2 elementos y recibe $P-1$ segmentos de alrededor de N/P^2 elementos (línea 19).

$$\Phi_{5,NAME} = H*(P-1)+g*2*(P-1)*N/P^2+L+\Phi_{4,NAME} \qquad (4.36)$$

Los P segmentos ordenados se mezclan en la línea 21. Para preparar la recolección de los vectores por el procesador *0*, los $P-1$ procesadores envían al procesador *0* los tamaños *MySize* de sus vectores (línea 22).

$$\Phi_{6,NAME} = I*N/P+g*(P-1)+L+\Phi_{5,NAME} \qquad (4.37)$$

El procesador *0* calcula los desplazamientos *OfIn* para los segmentos ordenados en la línea 24. En la línea 25 los *P-1* intervalos recolectados se ubican directamente en su posición final en el vector *A*.

$$\Phi_{7,NAME} = J*(P-1) +g*(P-1)*N/P+L+\Phi_{6,NAME} \qquad (4.38)$$

Sustituyendo iterativamente se obtiene la fórmula para el tiempo total del algoritmo.

Una vez computado la complejidad del algoritmo PSRS, mostramos los resultados utilizando el Cray T3E y la Origin 2000 para 2, 4, 8 16 y 24 procesadores. La tabla 4.10 muestra también los valores estimados con la fórmula *(4.38)* utilizando los parámetros del modelo, *g* y *L*.

PSRS	2	4	8	16	24
CRAY REAL	0.7954	0.4645	0.2867	0.2038	0.1838
CRAY MODELO	0.7720	0.4618	0.2910	0.2069	0.1845
ORIGIN REAL	1.0251	0.7096	0.5291	0.4328	0.4640
ORIGIN MODELO	1.0021	0.6928	0.5242	0.4336	0.4082

Tabla 4.10 Valores de tiempos reales y estimados para las dos arquitecturas.

Los errores se muestran en la tabla 4.11. Han sido calculados con la siguiente fórmula: 100*(REAL-MODELO)/REAL.

Errores	2	4	8	16	24
CRAY	2.94	0.59	1.53	1.52	0.36
ORIGIN	2.24	2.37	0.94	0.19	12.01

Tabla 4.11 Errores cometidos en el modelo.

Las tablas 4.12 y 4.13 muestran los valores de las constantes computacionales del algoritmo tanto del Cray como de la Origin.

CONSTANTES	2	4	8	16	24
B	5.31E-08	5.39E-08	5.48E-08	5.55E-08	6.26E-08
C	2.03E-05	1.03E-05	1.07E-05	2.94E-06	2.09E-06
D	1.25E-05	3.35E-06	1.28E-06	1.13E-06	1.33E-06
E	2.54E-05	8.60E-06	3.87E-06	1.79E-06	1.26E-06
F	1.55E-06	5.59E-07	2.66E-07	1.42E-07	1.03E-07
H	2.73E-05	8.73E-06	4.23E-06	2.09E-06	1.39E-06
I	3.62E-07	5.51E-07	8.04E-07	1.31E-06	1.81E-06
J	2.88E-05	9.13E-06	4.04E-06	1.97E-06	1.23E-06

Tabla 4.12 Constantes computacionales del PSRS para el CRAY.

CONSTANTES	2	4	8	16	24
B	4.47E-08	4.62E-08	4.93E-08	5.11E-08	6.05E-08
C	3.95E-06	2.08E-06	1.30E-06	6.13E-07	4.75E-07
D	9.78E-06	3.48E-06	1.80E-06	2.19E-06	3.20E-06
E	5.80E-06	2.13E-06	9.29E-07	4.27E-07	3.52E-07
F	7.11E-07	3.11E-07	7.43E-07	1.97E-07	1.14E-06
H	1.05E-05	4.87E-06	5.73E-06	1.03E-06	6.08E-06
I	8.92E-07	1.40E-06	2.33E-06	4.16E-06	5.93E-06
J	6.20E-06	2.23E-06	9.14E-07	4.67E-07	3.30E-07

Tabla 4. 13 Constante computacionales del PSRS para ORIGIN

La figura 4.13 muestra las curvas del tiempo ofrecido por el modelo y los tiempos reales.

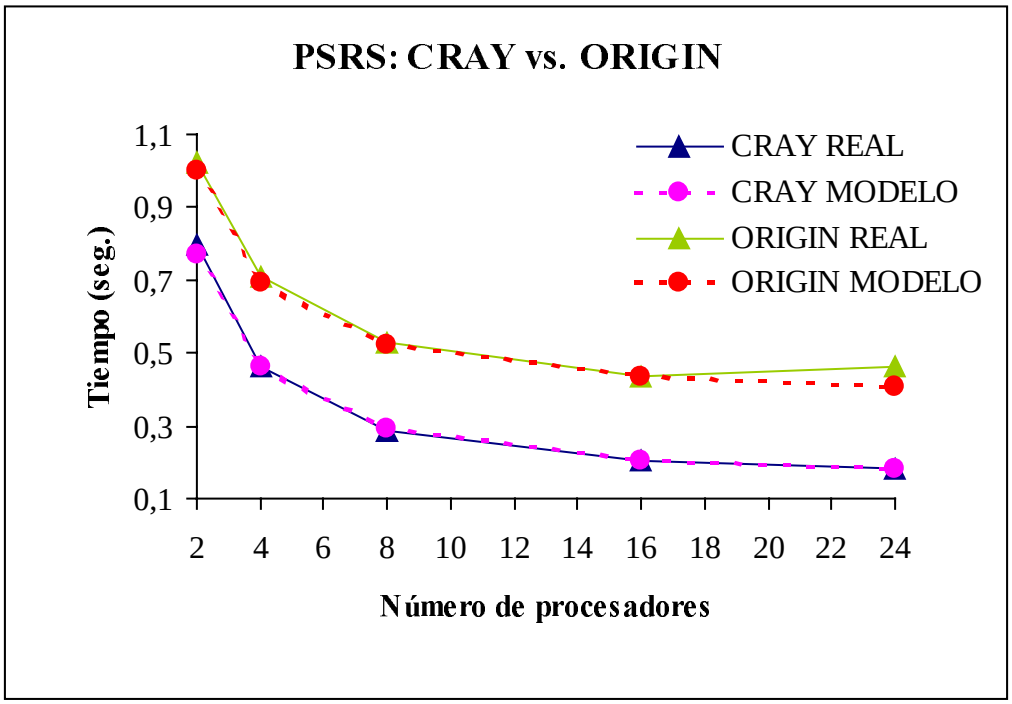

Figura 4. 13 Tiempos estimados y reales para el PSRS.

4.5.2 Problema de la Asignación de Recursos: SRAP

El problema de programación dinámica de la asignación de recursos lo hemos utilizado de nuevo para observar el comportamiento con un número grande de procesadores, 24. La descripción del algoritmo y su complejidad ya fueron realizadas en la sección anterior por lo que pasamos directamente a los valores reales de ejecución y los estimados.

La tabla 4.14 muestra los valores del SRAP para 2, 4, 8, 16 y 24 procesadores. En este caso, el aumento de procesadores sigue validando el modelo con unos valores estimados muy próximos a los reales. De nuevo destacar el uso de L_0 ya que el envío de

mensajes de un entero no nos permite utilizar el término independiente L obtenido del modelo.

	4.88	2.64	1.22	0.63	0.42
	4.54	2.27	1.14	0.57	0.38
N	4.35	2.57	1.34	0.72	0.49
N	4.06	2.03	1.02	0.51	0.34

Tabla 4.14 Resultados estimados y reales.

La tabla 4.15 muestra los valores de las constantes computacionales utilizadas en el SRAP.

N N		**N**
	1.96E-08	5.51E-08
	1.18E-07	1.08E-08
	6.48E-08	6.19E-08

Tabla 4.15 Constantes computacionales del SRAP.

La tabla 4.16 muestra los errores para las dos máquinas. El error en el Cray T3E es menor del 10% para 24 procesadores. Para la Origin los errores llegan a ser casi del 32%. En el estudio realizado sobre la influencia de los patrones y número de procesadores, la Origin ofrecía unos errores mayores que el Cray.

rrores					
	6.96	14.00	7.07	9.82	9.79
N	6.52	20.95	24.22	29.72	31.47

Tabla 4.16 Errores del modelo respecto al real.

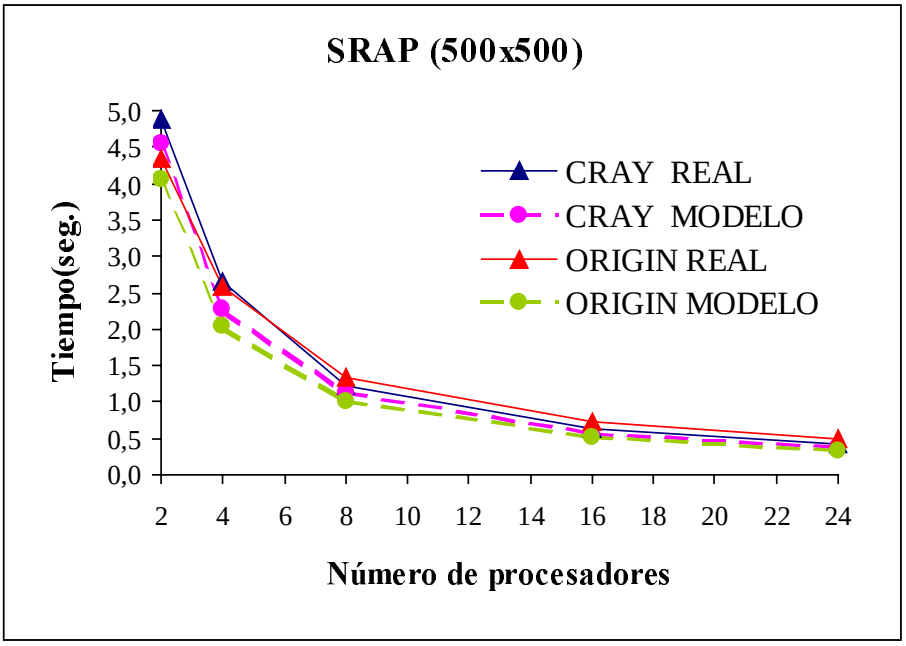

Figura 4. 14 Tiempos estimados y reales para el SRAP.

Hemos estudiado los modelos de computación paralela actuales, ofreciendo las ventajas e inconvenientes de cada uno de ellos. El modelo LogP, aún teniendo en cuenta muchas características de las computadoras actuales, es complejo de aplicar sobre algoritmos reales. La posibilidad de solapar cómputo y comunicaciones es realista respecto a la tecnología actual, pero introduce grandes dificultades a la hora del diseño y análisis de sus algoritmos. Los propios autores realizan simplificaciones para poder obtener soluciones eficientes de compromiso. El modelo C^3 es similar al modelo LogP, pero tiene en cuenta el tipo de encaminamiento, e introduce la congestión de los enlaces de la red y de los procesadores. El modelo BSP establece fuertes restricciones con la imposición del sincronismo por barrera, además de ofrecer computación síncrona.

Los modelos BSP y LogP funcionan de manera aceptable en las redes actuales si el software que les da soporte es adecuado. Desgraciadamente, el estándar de programación paralela, MPI, no se adapta totalmente a ninguno de los dos. El modelo BSP es más fácil de usar que el modelo LogP, pero la clase general de algoritmos BSP es más restringida que la de los algoritmos LogP.

Proponemos el modelo de Patrones con el objetivo de ser un modelo de predicción donde existe un compromiso entre la eficiencia del rendimiento y la simplicidad para obtener la complejidad. El modelo de patrones permite el análisis de algoritmos escritos utilizando funciones colectivas. El modelo es más preciso que cualquiera de los considerados pero tiene la desventaja de que el conjunto de algoritmos a los que se aplica se restringe a aquellos que utilizan las funciones colectivas para llevar a cabo las comunicaciones. Además la portabilidad del análisis es menor debida al mayor número de coeficientes en la descripción de las diferentes funciones. Utilizando este modelo hemos unificado el caso de patrones inyectivos para multicomputadoras y para redes de área local. En el caso de los patrones uno a todos se ha observado como decrece el tiempo cuando aumentamos el número de procesadores. En comunicaciones por parejas, hemos mostrado el rendimiento de las multicomputadoras y se ha comparado con las redes de área local.

Hemos obtenido los valores de los parámetros del modelo BSP para PVM y MPI sobre un conjunto amplio de máquinas: desde estaciones de trabajo conectadas tanto por cable coaxial compartido como por un conmutador de altas prestaciones, pasando por multicomputadoras de memoria distribuida como la IBM SP2 y el Cray T3E, multicomputadoras de memoria compartida como la Digital Alpha Server y multicomputadoras de memoria compartida distribuida como la Origin 2000. Nuestra propuesta consiste en obtener los valores de los parámetros utilizando los patrones de comunicaciones más comunes. Como mostramos en el capítulo 3, la hipótesis de la *h*-relación se cumple con mayor precisión respecto a la variación del número de procesadores que respecto a los diferentes patrones utilizados.

En el capítulo 4 proponemos el modelo BSP Sin Barreras, o BSP Without Barriers (BSPWB). Este permite el uso de librerías estándar para estudiar y analizar algoritmos asíncronos. El modelo introduce el concepto de "M-paso" que generaliza el concepto de superpaso de BSP y permite un análisis sencillo y preciso de la computación asíncrona. Para demostrar la validez del problema, se expone con cuatro ejemplos diferentes como trabaja el modelo. Se presentan las estimaciones ofrecidas por el BSPWB y se comparan con las ejecuciones sobre seis plataformas diferentes. También se comprueba como el cálculo de los parámetros L y g ha resultado ser válido para el modelo BSPWB.

El trabajo realizado en estos años nos permite tener nuevas perspectivas de problemas a estudiar. Entre las cuestiones más importantes podemos destacar:

1. Estudiar la validez del modelo LogP para librerías estándar y obtener una metodología adecuada para evaluar los parámetros del LogP en MPI/PVM sobre diferentes máquinas sin necesidad de grandes cambios.
2. Analizar el comportamiento del modelo LogP sobre redes de área local.
3. Formalización del modelo de Patrones y su relación con la Computación Colectiva.
4. Estudiar nuevas aproximaciones del modelo BSPWB, validándolas con un mayor número de aplicaciones y en un rango mayor de máquinas.
5. Realizar un estudio más detallado de los parámetros del modelo para el caso de mensajes de tamaño pequeño.

ÍNDICE de Tablas

ÍNDICE DE FIGURAS

Erratas

Página	donde dice	debe decir
41	o_s	o_r
75	Syncronous	Synchronous
89	Time	Tiempo
89	$Tiempo_{\Pi,i}(h)$	$Tiempo_{p,i}(h)$
152	del algoritmo QuickSort.	del algoritmo QuickSort para un problema de tamaño 7M-enteros
154	línea 14	línea 13
154	línea 13	línea 12
154	líneas 16 y 17	líneas 15 y 16
155	líneas 6 y 7	líneas 5 y 6
155	L y g.	L y g. L_0 vale 1.0E-5 para el CRAY T3E, 2.5E-6 para la Digital, 8.5E-6 para la Origin y 2.5E-5 para la IBM SP2.
155	línea 9	línea 12
155	líneas 16 y 17	líneas 15 y 16
156	Cray T3E y para la Origin 2000	Cray T3E, la Origin 2000, la IBM SP2 y la Digital