# Capítulo 2 El modelo BSP

## 2.1 Breve introducción a la computación paralela

En los años 70 comenzaron a aparecer algunos computadores con instrucciones de *hardware* para operar sobre vectores y algunos computadores con diversos procesadores que operaban en paralelo. Los primeros recibieron el nombre de **computadores vectoriales** y a los segundos se los llamó **computadores paralelos**.

La idea en la que se basa un computador paralelo es que un determinado número de procesadores trabajan en equipo para desarrollar una única tarea. La motivación de esta idea es que si un único procesador necesita un tiempo de $t$ segundos para llevar a cabo una cierta tarea, entonces $p$ procesadores trabajando en conjunto para realizar la misma tarea deberían de tardar un tiempo de $t/p$ segundos. Esta constituye una situación óptima que en muy pocas ocasiones puede alcanzarse. Sin embargo, nuestro objetivo es proponer algoritmos que resulten cada vez más rápidos, aprovechando las características de estos ordenadores que disponen de diversos procesadores para ejecutar las tareas computacionales que cada proceso conlleva.

En los años ochenta se construyeron un buen número de ordenadores paralelos en los que se obtenía un rendimiento aceptable de aquellas aplicaciones que tenían en cuenta la arquitectura específica de la máquina en la que se ejecutaba la aplicación. Esto resultaba lento

y costoso debido a la gran diversidad de arquitecturas que surgían. Desde principios de los noventa hasta ahora se advierte una evolución más estándar en los modelos arquitectónicos que se desarrollan, consistentes básicamente en un conjunto de pares (procesador,memoria) inteconectados mediante una red de comunicación que permite el empleo de direcciones globales. Esta convergencia se basa en razones de tipo tecnológico y económico; los grandes fabricantes aprovechan los microprocesadores existentes para construir máquinas paralelas en lugar de crear nuevos microprocesadores y, por otro lado, la memoria distribuida presenta la ventaja de la rapidez de acceso a la misma.

En las máquinas paralelas nos encontramos con algunas dicotomías importantes que analizamos muy brevemente. La primera de ellas está relacionada con la forma en la que se controla el trabajo de los procesadores. En un sistema de *simple instrucción, múltiples datos* (SIMD), todos los procesadores se encuentran bajo el control del procesador principal o maestro, ejecutando todos los procesadores la misma instrucción al mismo tiempo. El primer gran sistema paralelo desasrrollado a principios de los 70, el Illiac IV era una máquina de este tipo. La alternativa a estos sistemas la constituye los sistemas de *múltiple instrucción, múltiples datos* (MIMD), en el que los procesadores individuales trabajan bajo el control de sus propios programas. Aquí se presenta el problema de la sincronización, ya que no hay un procesador principal que pueda llevar a cabo esa tarea.

Otra importante dicotomía en los computadores paralelos es la de la *memoria compartida* frente a la *memoria local o distribuida*. En los computadores con memoria compartida todos los procesadores tienen acceso a una memoria común, lo cual presenta la ventaja de que todas las comunicaciones entre los procesadores se realiza a través de la memoria común, por lo que las comunicaciones en este sistema resultan muy rápidas. La desventaja es que muchos procesadores al mismo tiempo pueden acceder a esa memoria compartida por todos, lo que puede provocar un retraso hasta que la memoria está libre. La alternativa a estos sistemas son los sistemas con memoria local, en los que cada procesador accede a su propia memoria. Las comunicaciones entre los procesadores tienen lugar a través de un proceso de paso de mensajes, en el que los datos se transfieren de un procesador a otro.

Otro aspecto fundamental de los computadores paralelos es el modo en el que los diferentes procesadores se comunican unos con otros. Existen distintos esquemas de interconexión de los procesadores. Las conexiones más importantes son las de anillo, de malla, de hypercubo y clusters. Para una descripción más detallada de estos esquemas de conexión podemos ver Ortega [96].

Definimos a continuación algunos conceptos fundamentales relacionados con el rendimiento de los algoritmos paralelos. Supongamos que tenemos un computador paralelo que consta de $p$ procesadores.

**Definición 2.1** Llamamos **grado de paralelismo** de un algoritmo numérico al número de operaciones en el algoritmo que pueden realizarse en paralelo.

Con el fin de ilustrar el significado de este concepto, consideremos el problema de sumar dos vectores $\boldsymbol{a}$ y $\boldsymbol{b}$ de $n$ componentes. Las sumas de las distintas componentes $a_i + b_i$, para $i = 1, 2, \ldots, n$, son independientes y pueden realizarse en paralelo. Así, el grado de paralelismo de este algoritmo es $n$ y es independiente del número de procesadores de nuestro sistema; constituye una medida intrínseca del paralelismo del algoritmo. El número de procesadores sólo afectará al tiempo necesario para completar la computación. Consideramos ahora el problema de sumar $n$ números $a_1, a_2, \ldots, a_n$ mediante el algoritmo en serie

$$s = a_1, \qquad s = s + a_i, \qquad i = 2, \ldots, n.$$

Este algoritmo resulta poco adecuado para el cálculo en paralelo aunque el problema en sí mismo puede paralelizarse mediante un algoritmo distinto que a continuación describimos. Supongamos que $n = 8$ y que queremos sumar los números $a_1 + a_2 + \cdots + a_8$. La figura 2.1 nos muestra la adición de estos ocho números en tres etapas, cada una de ellas con distinto color. En la primera etapa se realizan cuatro sumas en paralelo, en la segunda se realizan dos sumas, mientras que en la tercera etapa se lleva a cabo una única suma. El algoritmo que muestra la figura 2.1 se conoce con el nombre de ***fan-in*** e ilustra perfectamente el principio *divide y vencerás* que ya se estudió en la sección 1.2. Dividimos el problema de la suma en problemas más pequeños que pueden ser resueltos independientemente unos de otros de forma sencilla.

Los algoritmos de tipo *fan-in* son muy utilizados en computación paralela para la ejecución en paralelo de sumas, productos, cálculos del máximo o mínimo de $n$ números, etc. Claramente vemos que si $n = 2^q$, entonces un algoritmo *fan-in* como el que muestra la figura 2.1 consta de $q = \log_2 n$ etapas donde se realizan $n/2$ sumas en la primera etapa, $n/4$ sumas en la segunda etapa y así sucesivamente. Esto significa que el grado de paralelismo en la primera etapa es de $n/2$, en la segunda es de $n/4$ y así sucesivamente.

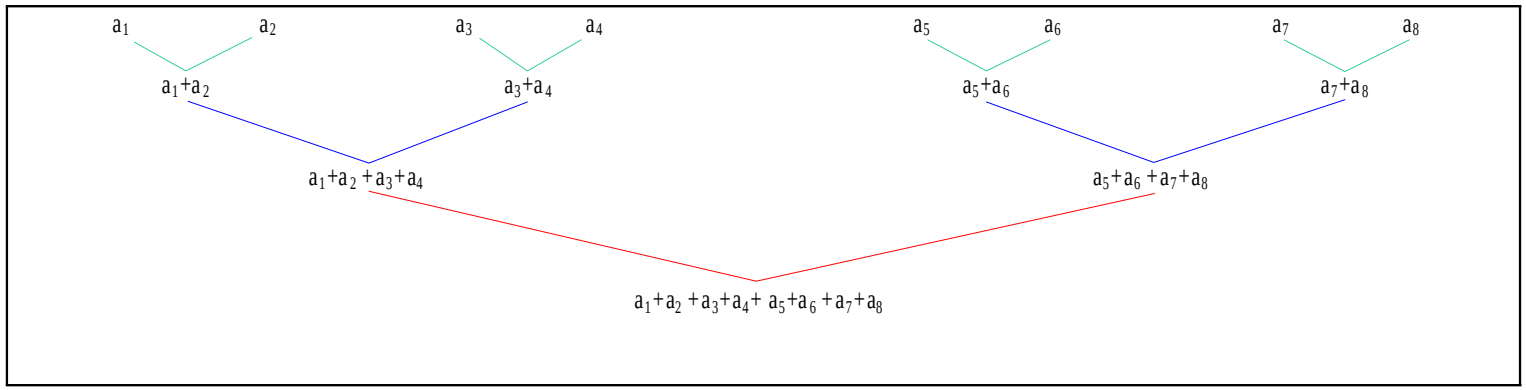


**Figura 2.1:** *Algoritmo fan-in para la suma de ocho números.*

Relacionado con el grado de paralelismo de un algoritmo está la idea de **granularidad**. Una granularidad a gran escala significa que pueden realizarse grandes tareas de forma independiente en paralelo; una granularidad a pequeña escala significa que pequeñas tareas pueden realizarse en paralelo. Un ejemplo simple de granularidad a pequeña escala puede ser la suma de dos vectores donde la tarea que puede realizarse en paralelo es la adición de dos escalares.

Los conceptos de *speedup* y *eficiencia* son los más utilizados para medir el paralelismo de un algoritmo.

**Definición 2.2** El *speedup* de un algoritmo paralelo, que denotamos por $S_p$, es $S_p = \dfrac{t_1}{t_p}$, donde $t_1$ es el tiempo de ejecución para un único procesador y $t_p$ es el tiempo de ejecución utilizando $p$ procesadores.

El speedup es una medida de comparación de un algoritmo cuando se utilizan 1 y $p$ procesadores. Relacionado con este concepto está el de eficiencia, $E_p$, que definimos a continuación.

**Definición 2.3** Definimos la eficiencia de un algoritmo paralelo como $E_p = \frac{S_p}{p}$.

Un objetivo fundamental en el desarrollo de algoritmos paralelos es conseguir el mayor speedup posible; idealmente, $S_p = p$. Los factores que más influyen en una reducción del speedup son la pérdida de paralelismo en el algoritmo unido al exceso de comunicaciones y el tiempo de sincronización.

Otro de los factores que puede degradar sensiblemente el grado de paralelismo de un algoritmo es el problema del equilibrado de la carga computacional. Por equilibrado de carga queremos significar el reparto de tareas entre los diferentes procesadores del sistema. Este reparto debe ser lo más equilibrado posible de manera que todos los procesadores estén activos el mayor tiempo posible.

## 2.2 El modelo de computación BSP

El modelo Bulk-Synchronous Parallel Computing BSP, inicialmente propuesto por Valiant [109] y desarrollado posteriormente por McColl [88, 89], abarca una amplia gama de arquitecturas ya que realiza una sencilla abstracción de una computación paralela. La sencillez de esta abstracción está en que un reducido número de parámetros recoge las variaciones más importantes entre las arquitecturas paralelas. El modelo no distingue ciertas diferencias cualitativas como por ejemplo la topología de comunicaciones de las redes de trabajo o las diferencias entre memoria compartida y distribuida que aparecen sólo en forma de diferentes costes para acceder a datos no locales.

Uno de los problemas clásicos en la computación paralela consiste en la escritura de algoritmos que sean válidos para las diferentes arquitecturas existentes. Una solución es la abstracción de los programas de la arquitectura de la máquina. El modelo BSP constituye un estilo de programación paralela desarrollado para paralelismo de propósito general, es decir, para un paralelismo que abarca todas las áreas de aplicación y un amplio rango de arquitecturas (véase McColl [89]). Por tanto, pretende cubrir unos objetivos no alcanzados por los tradicionales modelos de programación paralela que resultan adecuados para ciertos tipos concretos de aplicaciones o que trabajan

de forma eficiente en ciertos tipos particulares de máquinas.

Las propiedades más importantes de este modelo de computación se resumen en los siguientes puntos:

- Los programas BSP son fáciles de escribir. En cierto modo, esa simplicidad hace que sean muy similares a los programas secuenciales.
- El modelo es independiente del tipo de arquitectura. Esto supone que los programas sean independientes de la máquina en la que se estén ejecutando, lo que hace que el grado de portabilidad sea total.
- El rendimiento de un programa en una cierta arquitectura es predecible. Esto significa que el tiempo de ejecución de un algoritmo puede calcularse a partir del texto del programa, utilizando un reducido número de parámetros distintos en cada arquitectura.

Una primera versión de los componentes que integran una computadora BSP sería: un conjunto de procesadores secuenciales con memoria local, un sistema de comunicación que posibilita que los procesadores puedan acceder a datos no locales y un mecanismo para la sincronización global de todos los procesadores.

El término *bulk-synchronous* refleja la posición intermedia que ocupa el modelo BSP entre los extremos del paralelismo síncrono y asíncrono, conceptos que a continuación comentamos brevemente. En un sistema síncrono puro, los procesos se ejecutan en pasos fijos, sincronizados por un reloj global en cada instrucción. En un sistema asíncrono, los procesos pueden trabajar independientemente a diferentes ratios y se sincronizan implícitamente por parejas sólo cuando tiene lugar la comunicación. Los procesos de una computadora BSP se ejecutan concurrentemente a través de un programa, aunque se pueden realizar tareas de comunicación sin que lleven aparejadas una sincronización inmediata.

**Definición 2.4** Se define un **superpaso** como un segmento de la computación durante el cuál cada procesador puede realizar una secuencia de operaciones utilizando sólo sus propios datos locales y puede iniciar requerimientos de lectura y escritura de datos contenidos en otros procesadores. Al final de cada superpaso y antes de continuar con el siguiente, todos los procesadores esperan en un determinado punto, llamado **barrera de sincronización**, hasta que se completa el intercambio de datos no locales.

A partir de esta definición podemos subdividir cada superpaso en tres fases claramente diferenciadas: una primera fase de computación local en cada procesador, una segunda fase de comunicación y la última fase que coincide con la barrera de sincronización, que posibilita el movimiento de datos en las memorias locales de cada procesador.

Es importante resaltar que en el transcurso de un superpaso no tiene porqué producirse cálculos de tipo aritmético y comunicaciones. Puede haber superpasos donde únicamente se realizan comunicaciones y superpasos con cálculos aritméticos sólamente. La barrera de sincronización siempre se produce al final de un superpaso, por lo que el tamaño mínimo para un superpaso viene dado por la frecuencia con la que pueda ejecutarse cada barrera de sincronización. Este límite se cuantifica en el modelo BSP mediante un parámetro $l$ que es la periodicidad de sincronización. Al hablar de periodicidad de sincronización nos referimos a la mínima cantidad de tiempo que debe transcurrir entre sucesivas sincronizaciones globales. El tiempo transcurrido se mide en pasos de computación básicos, que es el tiempo que tarda un procesador en realizar una operación sobre datos locales, es decir, un *flop*.

El coste de las comunicaciones globales se expresa mediante el parámetro $g$ y se define a partir del concepto de $h$-relación, que a continuación se expone.

**Definición 2.5** Definimos una **$h$-relación** como un patrón de comunicación global en el que cada procesador puede enviar y recibir hasta un total de $h$ unidades de datos. A partir del concepto de $h$-relación, podemos definir el parámetro $g$ diciendo que una $h$-relación arbitraria puede comunicarse en $hg$ unidades de tiempo, imponiendo que $h$ es mayor que algún valor $h_0$.

Un modelo de comunicación en el que cada procesador envía un dato a otro procesador distinto representa una 1-relación. Sin embargo, un modelo de comunicación en el que un procesador envía un único dato al resto representa una $p$-relación. Este último modelo de comunicación se conoce con el nombre de **broadcast** y tiene gran importancia por tratarse de un modelo de comunicación muy utilizado en algoritmos de álgebra lineal. El modelo BSP no distingue entre un mensaje de longitud $m$ y $m$ mensajes de longitud 1. El coste es similar en ambos casos y viene determinado por $mg$. En cuanto a los procesos de comunicación cabe señalar que si en un superpaso la cantidad total de datos comunicados es muy pequeña, entonces los efectos del *start-up* de la máquina pueden jugar un papel importante en el rendimiento del algoritmo.

El parámetro $g$ representa la razón de la computación global por la capacidad de comunicación global y su cota depende de si se tienen suficientes datos para intercambiar, por lo que el sistema de comunicaciones puede alcanzar la capacidad indicada. Además, $g$ constituye una medida bastante precisa del coste de comunicar grandes cantidades de datos en una amplia gama de computadoras paralelas.

En resumen, podemos afirmar que una computadora BSP queda definida mediante los siguientes parámetros:

- $p$, que es el número de procesadores.
- $s$, que es la velocidad de computación de cada procesador, medida en megaflops por segundo, Mflop/seg (millones de operaciones aritméticas en coma flotante por segundo).
- $l$, que es el número de unidades de tiempo necesarias para realizar una sincronización entre los procesadores. El parámetro corresponde a la latencia de la red de comunicación.
- $g$, que representa una medida de la razón entre la capacidad total de la CPU por segundo y la capacidad total de comunicación por segundo, esto es, el cociente entre el número total de operaciones básicas que se pueden realizar en un segundo por todos los procesadores y el número total de unidades de datos que se pueden repartir por la red de comunicación en un segundo.

Notemos que con el fin de poder comparar el rendimiento paralelo en diversos sistemas, los parámetros de tiempo transcurrido se miden en pasos básicos de comunicación. Si la velocidad del procesador viene dada en flop/seg (flops por segundo), entonces el parámetro $l$ también se expresará en flops y $g$ vendría dada por flop/palabra (flops por palabra). La unidad flop/palabra representa el número de operaciones locales que cada procesador podría ejecutar durante el tiempo que tarda cada uno de ellos en comunicar una palabra en coma flotante.

## 2.3 El modelo de programación

El modelo propuesto por Valiant [109] no presupone ningún tipo de máquina específica que identifique dicho modelo. En la práctica la implementación se realiza con posterioridad y se organiza de acuerdo con el formato SPMD. En este modelo cada procesador paralelo ejecuta el mismo texto del programa, lo que significa que cada procesador ejecuta una y sólo una copia del programa original y dicha copia en ejecución se denomina proceso, por lo que hablar de proceso y procesador es equivalente. Los procesos se ejecutan a través de una secuencia de *superpasos*. Dentro de un superpaso cada procesador puede ejecutar cálculos distintos pero todos deben llegar al final de dicho superpaso antes de continuar con el siguiente. Cada procesador tiene su propio espacio de datos, pudiendo acceder de forma explícita al espacio de memoria del resto. Dicho acceso a datos remotos es asíncrono, lo que significa que no se puede garantizar que una operación de búsqueda de datos o almacenaje de un elemento se produzca hasta el final del superpaso actual, momento en el que se sincronizan todos los procesadores.

Como ya se mencionó en la sección 2.2 el concepto fundamental del modelo de programación es el de superpaso. Un algoritmo BSP es una serie de superpasos en los que se pueden desarrollar operaciones aritméticas y llamadas a datos no locales. Desde que este modelo fue propuesto por Valiant en 1990, se han venido desarrollando diversas librerías de funciones que han permitido la implementación de un buen número de algoritmos en computadores BSP. Algunas de estas librerías son extensiones de lenguajes de programación ya existentes como el C/C++ y FORTRAN. A continuación citamos algunas librerías que han sido implementadas en diferentes arquitecturas.

- **La librería BSP de Oxford**. Esta librería se desarrolló en Oxford en 1993 en el grupo de computación paralela dirigido por McColl y fue la primera librería que se implementó para este modelo utilizando los lenguajes C y FORTRAN. La versión original de esta librería era muy reducida, constando únicamente de seis funciones. Posteriores versiones han desarrollado notablemente las posibilidades de la misma a través de nuevas funciones que completan las existentes. Véase Donaldson, Hill y Skillicorn [37] para una descripción más detallada de la última versión de esta librería.

- **BSP++**. Constituye una aproximación a la programación orientada a objetos para computadores BSP. Es una extensión de la

librería de Oxford y aparece en el año 1994. Véase Lecomber [82] para un análisis más detallado de esta librería.

- **La librería Green BSP**. Esta librería data de 1995 y su característica más importante es que implementa como único método de comunicación el paso de mensajes. Su implementación es en lenguaje C. Véase Goudreau y otros [56] para una descripción más detallada de esta librería.
- **La librería BSP Worlwide Standard**. Es el resultado de la colaboración entre los grupos de Oxford y de la librería Green. Data de 1996 y se implementa tanto en lenguaje C como FORTRAN. Véase Goudreau y otros [55] para una descripción más detallada de esta librería.
- **Librería PUB**. Se trata de la librería desarrollada más recientemente, concretamete en 1999 en la Universidad de Paderborn (Alemania). Está implementada en C. Para profundizar en las características particulares de esta librería, véase Bonorden y otros [15].

Estas librerías tienen en común el haber sido desarrolladas y probadas en una amplia variedad de plataformas entre las que podemos citar IBM SP1 e IBM SP2, SGI *Power Challenge*, *Silicon Grafics Origin* 2000 CRAY T3D y CRAY T3E, *Convex* SPP, *clusters* de estaciones de trabajo Solaris (TCP/IP), *clusters* de Pentiums bajo Linux (TCP/IP), *clusters* de estaciones de trabajo IBM RS6000, *Hitachi* SR2001, *SunOS* 4.1.x y *Parsytec Explorer*.

## 2.4 El modelo de coste

Como ya se ha visto anteriormente, los parámetros $l$ y $g$ cuantifican el rendimiento del sistema. Podemos utilizar estos parámetros para predecir el rendimiento de un determinado programa o algoritmo que se esté ejecutando en el sistema. Un programa BSP está compuesto por una secuencia de superpasos que se ejecutan síncronamente, con lo que el tiempo total transcurrido será la suma de los tiempos invertidos en cada superpaso.

El modelo de coste, por tanto, se basa en el coste individual de cada superpaso del programa, que viene dado por la expresión

$$C_s = \max_{1 \leq i \leq p} w_i + \max_{1 \leq i \leq p} h_i g + l, \tag{2.1}$$

donde $C_s$ representa el coste total del superpaso, $w_i$ es el tiempo de computación aritmética del procesador $P_i$ en el transcurso del superpaso, $h_i$ es el número de unidades de datos que el procesador $P_i$ envía o recibe durante el superpaso y $g$ y $l$ son los valores de los parámetros definidos en la sección 2.2.

En consecuencia, el coste de un programa BSP es la suma del coste de todos sus superpasos. Las características de este modelo de coste nos sugieren diversas estrategias muy apropiadas para obtener programas BSP eficientes. Entre éstas podemos citar

- Equilibrar la computación en cada superpaso entre los diferentes procesadores, es decir, que cada procesador tenga asignado un trabajo similar, ya que la barrera de sincronización debe esperar a que el procesador más lento acabe las tareas encomendadas.
- Equilibrar las comunicaciones entre los procesadores, ya que el coste de comunicación es un máximo entre elementos enviados y recibidos.
- Minimizar el número de superpasos, con lo que estamos minimizando las barreras de sincronización.

Analizamos ahora, siguiendo el modelo de coste anteriormente expuesto, el coste de algunos modelos de comunicación habituales en los algoritmos del tipo *divide y vencerás*. Así, si tenemos en cuenta que una $h$-relación es un patrón de comunicación global en el que cada procesador puede enviar o recibir a lo sumo $h$ unidades de datos y su coste es de $hg$ flops, establecemos el coste BSP de los siguientes modelos de comunicación.

(i) Cada procesador envía un elemento al procesador siguiente. El coste de esta comunicación es $1g$ flop, ya que tenemos una 1-relación.

(ii) Cada procesador tiene un mensaje de longitud $h$ entrando y saliendo de él. El coste BSP de este patrón de comunicación es de $hg$ flops, de acuerdo con la expresión (2.1).

(iii) Desde $p$ procesadores se envía a otro procesador un mensaje de longitud $\frac{n}{p}$ unidades, suponiendo que $n$ es múltiplo de $p$. En este caso, el coste viene dado por el número de elementos recibidos por el procesador receptor de los mismos, ya que es el máximo. En este procesador entran $n$ unidades de datos lo que representa una $n$-relación, con un coste de $ng$ flops.

(iv) Los procesadores pares envían a los procesadores pares un mensaje de longitud $m$, suponiendo que $p$ es múltiplo de 2. Según este modelo de comunicación, en cada procesador par entran y salen $\frac{p}{2} - 1$ mensajes de longitud $m$, por lo que se trata de una $\left(\frac{p}{2} - 1\right)$-relación, y su coste viene dado por $\left(\frac{p}{2} - 1\right) g$ flops.

(v) Se considera el patrón de comunicación en el que desde un procesador se realiza un *broadcast* de $n$ elementos. El coste de este patrón de comunicación es de $n(p-1)g$ flops, ya que el procesador del que salen los datos envía un total de $n(p-1)$ datos al resto de procesadores, esto es, una $n(p-1)$-relación.

## 2.5 Valores de $s$, $g$ y $l$ en distintas máquinas

El modelo de coste nos permite la predicción en la ejecución de los programas BSP en diversas arquitecturas. Basta con calcular el coste computacional y de comunicación en cada superpaso y sustituir los valores de $p$, $s$, $g$ y $l$ en la expresión (2.1). Los valores de estos parámetros pueden medirse en cada máquina por medio de software. A lo largo de esta memoria se presentan resultados teóricos para todos los algoritmos estudiados con el fin de establecer comparaciones en los tiempos de ejecución de dichos algoritmos. Las máquinas elegidas para realizar un estudio comparativo de tiempos entre los diferentes algoritmos presentados son

- un IBM SP2, dotado con dos tipos de conexiones para efectuar las comunicaciones entre los procesadores: un switch de alto rendimiento y una conexión ethernet. Consta de ocho procesadores P2SC a 66.7 Mhz y 128 Mbytes de memoria principal por cada procesador,

- un CRAY T3D, que consta de 256 procesadores del tipo DECchip 21164 a 150 Mhz con 64 Mbytes de memoria local por cada procesador,
- un cluster de Pentiums formado por ocho Pentiums II a 400 Mhz con 128 Mbytes de memoria por procesador, conectados entre sí utilizando un switch ethernet Cisco 2916XL a 100 Mbytes por segundo.

Los valores de los parámetros $p$, $s$, $g$ y $l$ para las máquinas citadas anteriormente se muestran en la tabla 2.1 y son los que publica el grupo de BSP de la Universidad de Oxford. Las unidades utilizadas para medir estos parámetros son las siguientes: $s$ se mide en megaflops por segundo (Mflops), $p$ es el número de procesadores, $l$ se mide en flops y $g$ se mide en flops por palabra (flop/word), que en este caso son palabras de 32 bits. En la tabla 2.1 aparece $n_{1/2}$, que constituye una medida del tamaño del mensaje que produce una comunicación óptima en la red de comunicaciones de la computadora paralela.

Describimos brevemente a continuación la forma en que el grupo BSP de Oxford calcula los parámetros que aparecen en la tabla 2.1. El cálculo del parámetro $s$ depende fuertemente del tipo de cálculo que llevamos a cabo. Se realizan dos cálculos del parámetro y se toma el valor medio. El primero consiste en medir el coste de un producto de vectores de tamaño $n$ donde el número de operaciones que se realizan es de orden $n$. Se trabaja con estructuras de datos de tamaño $n$, siendo este tamaño superior al tamaño del caché de cada procesador. Este valor de $s$ constituye una cota inferior para la velocidad del procesador. Por otra parte, se calcula el parámetro $s$ para el coste de un producto de matrices densas donde el número de operaciones que se realizan es de orden $n^3$. Las estructuras de datos ahora son de tamaño $n^2$ y una buena parte de los cálculos se guardan en el caché de cada procesador. Este valor de $s$ constituye una cota superior de la velocidad del procesador.

Como ya se ha comentado en el transcurso de esta sección, la eficiencia de un algoritmo BSP depende en gran medida del diseño de un modelo de comunicación lo más equilibrado posible. Así, del mismo modo que para la medida de $s$ tomábamos el valor medio cuando se desarrollaban dos tipos distintos de cálculos, ahora medimos la capacidad de comunicación $g$ del sistema utilizando dos modelos distintos de comunicación equilibrada. El primero de ellos es el de una 1-relación, donde cada procesador comunica a su procesador vecino una unidad de datos. Como segundo modelo de comunicación se considera una $k$-relación en la que se produce un intercambio de datos entre

| $s$ | $p$ | $l$ | $g$ | $n_{1/2}$ words |
|---|---|---|---|---|
| 26 | 2 | 1903 | 6.3 | 6 |
| | 4 | 3583 | 6.4 | 7 |
| | 8 | 5412 | 6.9 | 6 |

**(a)** *IBM SP2 switch*

| $s$ | $p$ | $l$ | $g$ | $n_{1/2}$ words |
|---|---|---|---|---|
| 26 | 2 | 18759 | 182.1 | 3 |
| | 4 | 39025 | 388.2 | 5 |
| | 8 | 88795 | 1246.6 | 2 |

**(b)** *IBM SP2 ethernet*

| $s$ | $p$ | $l$ | $g$ | $n_{1/2}$ words |
|---|---|---|---|---|
| 12 | 2 | 164 | 0.7 | 71 |
| | 4 | 168 | 0.7 | 66 |
| | 8 | 175 | 0.8 | 59 |
| | 16 | 181 | 0.9 | 61 |
| | 32 | 201 | 1.1 | 28 |
| | 64 | 148 | 1.0 | 27 |
| | 128 | 301 | 1.1 | 20 |
| | 256 | 387 | 1.2 | 15 |

**(c)** *CRAY T3D*

| $s$ | $p$ | $l$ | $g$ | $n_{1/2}$ words |
|---|---|---|---|---|
| 88 | 2 | 5654 | 31.9 | 5 |
| | 4 | 11759 | 31.4 | 5 |
| | 8 | 18347 | 30.5 | 32 |

**(d)** *Cluster de Pentiums*

**Tabla 2.1:** *Valores de los parámetros $s$, $g$, $l$ y $n_{1/2}$.*

todos los procesadores (cada procesador realiza un broadcast de $k$ elementos). Esto nos proporciona dos medidas de $g$ distintas, una que llamamos **local**, correspondiente a la 1-relación y otra que llamamos **total**, correspondiente a la $pg$-relación. Los valores de $g$ que se muestran en la tabla 2.1 se refieren al valor de $g$ local y para comunicaciones de palabras de 32-bits.

La tabla 2.1 nos muestra un valor único del parámetro $g$ para cada valor de $p$, independientemente del tamaño de la comunicación que se realiza. En la predicción teórica de tiempos que se realiza para cada algoritmo a lo largo de los siguientes capítulos hemos utilizado el modelo propuesto por Hockney [64] y modificado por Miller [95]. En este modelo se debe tener en cuenta el efecto de la granularidad, por lo que se realizan comunicaciones aumentando el tamaño de los bloques comunicados, obteniéndose una gráfica que relaciona el cociente entre el tiempo $t$ (en flops) y el tamaño $n$ del bloque comunicado. Si suponemos que existe una relación lineal entre $t$ y $n$, tendremos que

$$t(n) = kn + l,$$

siendo $k$ el tiempo que tarda en comunicarse una palabra y $l$ el tiempo de una sincronización. La relación entre $n$ y $t$ se puede expresar como $g(n) = \frac{t}{n}$. Suponemos ahora que $g_\infty$ es el valor asintótico al que se aproxima $g(n)$ cuando $n$ tiende a $\infty$ y que $n_{1/2}$ es el tamaño del mensaje que produce la mitad del ancho de banda óptimo de la máquina, es decir, $g\left(n_{1/2}\right) = 2g_\infty$. Bajo estos supuestos, podemos expresar

$$t(n) = \left(n + n_{1/2}\right) g_\infty, \tag{2.2}$$

de donde

$$g(n) = \frac{t(n)}{n} = \left(1 + \frac{n_{1/2}}{n}\right) g_\infty. \tag{2.3}$$

La expresión (2.2) nos muestra que cuando $n = n_{1/2}$ entonces la mitad del tiempo se utiliza en la comunicación propiamente dicha y la otra mitad en el start-up. Se considera como valor BSP de la máquina $g = g(n_{1/2}) = 2g_\infty$.

Para calcular el coste $g$ de la comunicación de una palabra se utiliza la expresión (2.3). Sin embargo, distinguimos los casos siguientes:

- si $n \gg n_{1/2}$ entonces $g = g_\infty$.
- si $n \ll n_{1/2}$ entonces $\frac{n}{n_{1/2}} \cong 0$, por lo que podemos tomar $t(n) = n_{1/2} g_\infty$. Así, tendremos que

$$g = \frac{t(n)}{n} = \frac{n_{1/2} g_\infty}{n}.$$

El valor de $l$ representa el coste de una barrera de sincronización y puede ser calculado empíricamente en cada máquina. Una forma de calcular $l$ es mediante un programa que incluya una sucesión de superpasos vacíos para determinar el coste medio de una barrera de sincronización.

Analizando los parámetros que aparecen en la tabla 2.1, se observa que el CRAY T3D es una máquina donde la velocidad de los procesadores no es excesivamente alta, sin embargo, las comunicaciones son más rápidas que las operaciones aritméticas hasta $p = 16$ y prácticamente igual de rápidas para el resto de procesadores. Para $p = 256$ el valor de $g$ es 1.2, lo que nos da idea de la rapidez con que se comunican datos en una máquina de este tipo. Teniendo en cuenta estas características cabe esperar que aquellos algoritmos con una gran carga de comunicaciones y menor carga computacional sean los que mejor rendimiento ofrezcan. Hay que resaltar los valores de los parámetros que se consiguen para 64 procesadores. Se observa que el coste de realizar una barrera de sincronización desciende hasta 148 flop/seg., mientras que realizar la comunicación de una palabra de 32 bits cuesta exactamente lo mismo que realizar una operación en coma flotante. Esto significa que deberíamos obtener mejores tiempos en los algoritmos que estudiemos para $p = 64$ que para $p = 32$.

La máquina que presenta peores valores para los parámetros BSP es el IBM SP2 con conexión ethernet. Como muestra digamos que realizar una comunicación trabajando con $p = 8$ cuesta lo mismo que realizar 1246.6 operaciones aritméticas y una barrera de sincronización tarda un tiempo equivalente a realizar 88795 operaciones aritméticas.