Capítulo 5 Método divide y vencerás de Bondeli

5.1 Introducción

La idea en que se basa el método de Bondeli constituye una técnica del tipo *divide y vencerás* puesto que se particiona en bloques el sistema inicial y el vector de términos independiente para resolver en cada procesador un conjunto de subsistemas tridiagonales. Posteriormente, a partir de la definición de unas nuevas variables, se construye un sistema tridiagonal auxiliar que nos permite obtener la solución general.

Las diferencias básicas entre los distintos algoritmos se encuentran en la forma en que se construye y resuelve el sistema tridiagonal auxiliar y el procesador donde se obtienen las soluciones finales. En el último de los algoritmos estudiados se utiliza el método paralelo *recursive doubling* para la resolución del sistema tridiagonal auxiliar.

Se considera el problema general de obtener la solución del sistema

$$A\boldsymbol{x} = \boldsymbol{d}, \tag{5.1}$$

donde

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix} \quad \text{y} \quad \boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} \tag{5.2}$$

son, respectivamente, una matriz tridiagonal e irreducible y el vector de términos independiente. Supongamos que podemos encontrar dos números naturales k y p tales que $k = \frac{n}{p}$ y consideremos la siguiente partición por bloques, como en el capítulo 4,

$$A = \begin{bmatrix} A_0 & B_0 & & & \\ C_1 & A_1 & B_1 & & \\ & \ddots & \ddots & \ddots & \\ & & C_{p-2} & A_{p-2} & B_{p-2} \\ & & & C_{p-1} & A_{p-1} \end{bmatrix}, \tag{5.3}$$

de la matriz A donde cada uno de los bloques diagonales

$$A_i = \begin{bmatrix} a_{ik+1} & b_{ik+1} & & & \\ c_{ik+2} & a_{ik+2} & b_{ik+2} & & \\ & \ddots & \ddots & \ddots & \\ & & c_{(i+1)k-1} & a_{(i+1)k-1} & b_{(i+1)k-1} \\ & & & c_{(i+1)k} & a_{(i+1)k} \end{bmatrix}, \qquad i = 0, 1, \ldots, p-1,$$

es una matriz tridiagonal de tamaño $k \times k$ y, para $i = 0, 1, \ldots, p-2$, cada bloque subdiagonal

$$C_{i+1} = \left[\begin{array}{cccc|c} 0 & 0 & \cdots & 0 & c_{(i+1)k+1} \\ \hline & & & & 0 \\ & & O & & \vdots \\ & & & & 0 \\ & & & & 0 \end{array}\right] = c_{(i+1)k+1}\boldsymbol{e}_1\boldsymbol{e}_k^T \tag{5.4}$$

y superdiagonal

$$B_i = \left[\begin{array}{c|ccc} 0 & & & \\ 0 & & O & \\ \vdots & & & \\ 0 & & & \\ \hline b_{(i+1)k} & 0 & \cdots & 0 \quad 0 \end{array}\right] = b_{(i+1)k}\boldsymbol{e}_k\boldsymbol{e}_1^T \tag{5.5}$$

es de tamaño $k \times k$ con un único elemento no nulo. Los vectores $\boldsymbol{e}_1$ y $\boldsymbol{e}_k$, siguiendo la notación habitual, representan la primera y la última columna, respectivamente, de la matriz identidad I_k.

En los vectores $\boldsymbol{x}$ y $\boldsymbol{d}$ se considera una partición por bloques conforme con la partición en bloques de la matriz A dada por la expresión (5.3), es decir,

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{p-2} \\ \boldsymbol{x}_{p-1} \end{bmatrix} \quad \text{y} \quad \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_{p-2} \\ \boldsymbol{d}_{p-1} \end{bmatrix},$$

con

$$\boldsymbol{x}_i = \begin{bmatrix} x_{ik+1} \\ x_{ik+2} \\ \vdots \\ x_{(i+1)k-1} \\ x_{(i+1)k} \end{bmatrix} \quad \text{y} \quad \boldsymbol{d}_i = \begin{bmatrix} d_{ik+1} \\ d_{ik+2} \\ \vdots \\ d_{(i+1)k-1} \\ d_{(i+1)k} \end{bmatrix}, \quad i = 0, 1, \ldots, p-1.$$

5.2 Descripción del método

La notación introducida en la sección 5.1 nos permite escribir el sistema (5.1) como

$$\begin{bmatrix} A_0 & B_0 & & & \\ C_1 & A_1 & B_1 & & \\ & \ddots & \ddots & \ddots & \\ & & C_{p-2} & A_{p-2} & B_{p-2} \\ & & & C_{p-1} & A_{p-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{p-2} \\ \boldsymbol{x}_{p-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_{p-2} \\ \boldsymbol{d}_{p-1} \end{bmatrix}, \tag{5.6}$$

o en forma abreviada,

$$C_i \boldsymbol{x}_{i-1} + A_i \boldsymbol{x}_i + B_i \boldsymbol{x}_{i+1} = \boldsymbol{d}_i, \qquad i = 0, 1, \ldots, p-1, \tag{5.7}$$

teniendo en cuenta que $C_0 = B_{p-1} = O$.

Como los bloques diagonales A_i, para $i = 0, 1, \ldots, p-1$, son no singulares, sustituyendo las expresiones (5.4) y (5.5) en la expresión (5.7) y despejando el vector $\boldsymbol{x}_i$ obtenemos el sistema

$$\boldsymbol{x}_i = A_i^{-1}\boldsymbol{d}_i - c_{ik+1}A_i^{-1}\boldsymbol{e}_1\boldsymbol{e}_k^T\boldsymbol{x}_{i-1} - b_{(i+1)k}A_i^{-1}\boldsymbol{e}_k\boldsymbol{e}_1^T\boldsymbol{x}_{i+1}, \qquad i = 0, 1, \ldots, p-1. \tag{5.8}$$

Si definimos, para $i = 0, 1, \ldots, p-1$,

$$\boldsymbol{y}_i = A_i^{-1}\boldsymbol{d}_i, \tag{5.9}$$

$$\boldsymbol{z}_{2i-1} = A_i^{-1}\boldsymbol{e}_1, \quad \boldsymbol{z}_{2i} = A_i^{-1}\boldsymbol{e}_k, \tag{5.10}$$

$$\alpha_{2i-1} = -c_{ik+1}\boldsymbol{e}_k^T\boldsymbol{x}_{i-1}, \qquad \alpha_{2i} = -b_{(i+1)k}\boldsymbol{e}_1^T\boldsymbol{x}_{i+1}, \tag{5.11}$$

entonces podemos escribir la ecuación (5.8) como

$$\boldsymbol{x}_i = \boldsymbol{y}_i + \alpha_{2i-1}\boldsymbol{z}_{2i-1} + \alpha_{2i}\boldsymbol{z}_{2i}, \qquad i = 0, 1, \ldots, p-1. \tag{5.12}$$

Hemos formado, a partir del conjunto de ecuaciones (5.8), mediante las expresiones (5.9), (5.10) y (5.11), un nuevo sistema dado por la expresión (5.12) en el que ahora las incógnitas son los escalares α_0, α_1, $\alpha_2, \ldots,$ α_{2p-4}, α_{2p-3}. Estudiamos detalladamente la relación existente entre estos escalares y los vectores $\boldsymbol{z}_i$, para $i = 0, 1, \ldots, 2p-4, 2p-3$, por medio de las expresiones (5.8) y (5.12).

De

$$\alpha_0 = -b_k(\boldsymbol{e}_1^T\boldsymbol{x}_1) = -b_k\left[\boldsymbol{e}_1^T(\boldsymbol{y}_1 + \alpha_1\boldsymbol{z}_1 + \alpha_2\boldsymbol{z}_2)\right] = -b_k\left[\boldsymbol{e}_1^T\boldsymbol{y}_1 + (\boldsymbol{e}_1^T\boldsymbol{z}_1)\alpha_1 + (\boldsymbol{e}_1^T\boldsymbol{z}_2)\alpha_2\right],$$

tenemos que

$$\frac{1}{b_k}\alpha_0 + (\boldsymbol{e}_1^T\boldsymbol{z}_1)\alpha_1 + (\boldsymbol{e}_1^T\boldsymbol{z}_2)\alpha_2 = -\boldsymbol{e}_1^T\boldsymbol{y}_1,$$

mientras que

$$\alpha_1 = -c_{k+1}(\boldsymbol{e}_k^T \boldsymbol{x}_0) = -c_{k+1}\left[\boldsymbol{e}_k^T(\boldsymbol{y}_0 + \alpha_0 \boldsymbol{z}_0)\right] = -c_{k+1}\left[\boldsymbol{e}_k^T \boldsymbol{y}_0 + (\boldsymbol{e}_k^T \boldsymbol{z}_0)\alpha_0\right],$$

por lo que

$$(\boldsymbol{e}_k^T \boldsymbol{z}_0)\alpha_0 + \frac{1}{c_{k+1}}\alpha_1 = -\boldsymbol{e}_k^T \boldsymbol{y}_0.$$

Siguiendo un razonamiento análogo para $i = 2, 3, \ldots, 2p-4, 2p-3$, obtendremos el conjunto de ecuaciones

$$(\boldsymbol{e}_k^T \boldsymbol{z}_{2i-1})\alpha_{2i-1} + (\boldsymbol{e}_k^T \boldsymbol{z}_{2i})\alpha_{2i} + \frac{1}{c_{(i+1)k+1}}\alpha_{2i+1} = -(\boldsymbol{e}_k^T \boldsymbol{y}_i), \qquad i = 0, 1, \ldots, p-2, \tag{5.13}$$

$$\frac{1}{b_{ik}}\alpha_{2i-2} + (\boldsymbol{e}_1^T \boldsymbol{z}_{2i-1})\alpha_{2i-1} + (\boldsymbol{e}_1^T \boldsymbol{z}_{2i})\alpha_{2i} = -(\boldsymbol{e}_1^T \boldsymbol{y}_i), \qquad i = 1, 2, \ldots, p-1. \tag{5.14}$$

Las ecuaciones (5.13) y (5.14) constituyen un sistema tridiagonal de tamaño $(2p-2) \times (2p-2)$, que escribimos como

$$H\boldsymbol{\alpha} = \boldsymbol{\beta} \tag{5.15}$$

con incógnitas α_i, para $i = 0, 1, \ldots, 2p-3$, matriz de coeficientes

$$H = \begin{bmatrix} \delta_0 & \mu_0 & & & \\ \nu_1 & \delta_1 & \mu_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \nu_{2p-4} & \delta_{2p-4} & \mu_{2p-4} \\ & & & \nu_{2p-3} & \delta_{2p-3} \end{bmatrix} \tag{5.16}$$

con elementos diagonales, superdiagonales y subdiagonales, respectivamente

$$\delta_i = \begin{cases} \boldsymbol{e}_k^T \boldsymbol{z}_i & i = 0, 2, \ldots, 2p-4, \\ \boldsymbol{e}_1^T \boldsymbol{z}_i & i = 1, 3, \ldots, 2p-3, \end{cases}, \qquad \mu_i = \begin{cases} \frac{1}{c_{\left(\frac{i}{2}+1\right)k+1}} & i = 0, 2, \ldots, 2p-4, \\ \boldsymbol{e}_1^T \boldsymbol{z}_{i+1} & i = 1, 3, \ldots, 2p-5, \end{cases}, \qquad \nu_i = \begin{cases} \boldsymbol{e}_k^T \boldsymbol{z}_{i-1} & i = 2, 4, \ldots, 2p-4, \\ \frac{1}{b_{\frac{i+1}{2}k}} & i = 1, 3, \ldots, 2p-3, \end{cases} \tag{5.17}$$

y vector de términos independientes $\boldsymbol{\beta} = [\ \beta_0 \quad \beta_1 \quad \ldots \quad \beta_{2p-3} \quad \beta_{2p-3}\ ]^T$ con

$$\beta_i = \begin{cases} -\boldsymbol{e}_k^T \boldsymbol{y}_{\frac{i}{2}} & i = 0, 2, \ldots, 2p-4, \\ -\boldsymbol{e}_1^T \boldsymbol{y}_{\frac{i+1}{2}} & i = 1, 3, \ldots, 2p-3. \end{cases} \tag{5.18}$$

Una vez obtenida la solución $\boldsymbol{\alpha}$ del sistema (5.15), podemos obtener la solución $\boldsymbol{x}$ del sistema (4.1) sustituyendo las componentes de $\boldsymbol{\alpha}$ en la expresión (5.12).

Un algoritmo paralelo que describa las características de este método debería constar de las siguientes fases claramente diferenciadas:

- En una primera fase debemos distribuir la matriz A y los vectores $\boldsymbol{x}$ y $\boldsymbol{d}$ de acuerdo con la expresión (5.6).
- En la segunda fase cada procesador P_i, para $i = 0, 1, \ldots, p-1$, debe calcular en paralelo los vectores $\boldsymbol{y}_i$, $\boldsymbol{z}_{2i-1}$ y $\boldsymbol{z}_{2i}$, solución de los sistemas (5.9) y (5.10).
- En la tercera fase del algoritmo se construye y resuelve el sistema (5.15) mediante las expresiones (5.16), (5.17) y (5.18), cuya solución es $\boldsymbol{\alpha}$.
- En la última fase del algoritmo se sustituye $\boldsymbol{\alpha}$ en la expresión (5.12) para obtener la solución del sistema inicial.

Si se analizan detalladamente las fases anteriores del método, se observa que la fase donde se realizan la mayor parte de los cálculos aritméticos es la segunda, en la que se deben calcular las variables $\boldsymbol{y}_i$, para $i = 0, 1, \ldots, p-1$, y $\boldsymbol{z}_j$, para $j = 0, 1, \ldots, 2p-4, 2p-3$. Estas variables pueden ser calculadas en paralelo en los diferentes procesadores mediante las expresiones (5.9) y (5.10). Nótese que en estos sistemas la matriz de coeficientes no cambia, únicamente cambia el vector de términos independientes. Podemos resumir las tareas aritméticas que debe llevar a cabo cada procesador de la siguiente forma:

- En el procesador P_0 se resuelven los sistemas

$$A_0[\ \boldsymbol{y}_0 \quad \boldsymbol{z}_0\ ] = [\ \boldsymbol{d}_0 \quad \boldsymbol{e}_k\ ]. \tag{5.19}$$

- En el procesador P_i, para $i = 1, 2, \ldots, p-2$, se resuelven los sistemas

$$A_i[\ \boldsymbol{y}_i \quad \boldsymbol{z}_{2i-1} \quad \boldsymbol{z}_{2i}\ ] = [\ \boldsymbol{d}_i \quad \boldsymbol{e}_1 \quad \boldsymbol{e}_k\ ]. \tag{5.20}$$

- En el procesador P_{p-1} se resuelven los sistemas

$$A_{p-1}[\ \boldsymbol{y}_{p-1} \quad \boldsymbol{z}_{2p-3}\ ] = [\ \boldsymbol{d}_{p-1} \quad \boldsymbol{e}_1\ ]. \tag{5.21}$$

Observamos que el método en esta fase no está totalmente balanceado en cuanto a carga de trabajo computacional, ya que mientras que el procesador primero y último resuelven dos sistemas, los procesadores centrales resuelven tres sistemas, de la misma forma que ocurría en el capítulo 4 con el método basado en la fórmula de Sherman-Morrison-Woodbury.

Consideramos el siguiente ejemplo que resume los cálculos expuestos a lo largo de esta sección para un sistema con $n = 16$.

Ejemplo 5.1 *Consideramos la resolución del sistema (5.1), siendo*

$$A = \begin{bmatrix}
12 & 1 & & & & & & & & & & & & & \\
3 & 12 & 1 & & & & & & & & & & & & \\
 & 3 & 12 & 1 & & & & & & & & & & & \\
 & & 3 & 12 & 1 & & & & & & & & & & \\
 & & & 3 & 12 & 1 & & & & & & & & & \\
 & & & & 3 & 12 & 1 & & & & & & & & \\
 & & & & & 3 & 12 & 1 & & & & & & & \\
 & & & & & & 3 & 12 & 1 & & & & & & \\
 & & & & & & & 3 & 12 & 1 & & & & & \\
 & & & & & & & & 3 & 12 & 1 & & & & \\
 & & & & & & & & & 3 & 12 & 1 & & & \\
 & & & & & & & & & & 3 & 12 & 1 & & \\
 & & & & & & & & & & & 3 & 12 & 1 & \\
 & & & & & & & & & & & & 3 & 12 & 1 \\
 & & & & & & & & & & & & & 3 & 12
\end{bmatrix} \quad y \quad \boldsymbol{d} = \begin{bmatrix} 13 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 16 \\ 15 \end{bmatrix}.$$

Por la forma en que se han tomado la matriz A y el vector $\boldsymbol{d}$ resulta inmediato que la solución $\boldsymbol{x}$ del sistema $A\boldsymbol{x} = \boldsymbol{d}$ tiene todas sus componentes iguales a 1.

Supongamos que tomamos $p = 4$, por lo que $k = 4$. Entonces, el procesador P_0 tendrá los bloques

$$A_0 = \begin{bmatrix} 12 & 1 & & \\ 3 & 12 & 1 & \\ & 3 & 12 & 1 \\ & & 3 & 12 \end{bmatrix} \qquad y \qquad \boldsymbol{d} = \begin{bmatrix} 13 \\ 16 \\ 16 \\ 16 \end{bmatrix}.$$

Los procesadores no extremos P_i, para $i = 1, 2$ tendrán los bloques

$$A_i = \begin{bmatrix} 12 & 1 & & \\ 3 & 12 & 1 & \\ & 3 & 12 & 1 \\ & & 3 & 12 \end{bmatrix} \qquad y \qquad \boldsymbol{d} = \begin{bmatrix} 16 \\ 16 \\ 16 \\ 16 \end{bmatrix},$$

mientras que el procesador P_3 tendrá los bloques

$$A_3 = \begin{bmatrix} 12 & 1 & & \\ 3 & 12 & 1 & \\ & 3 & 12 & 1 \\ & & 3 & 12 \end{bmatrix} \qquad y \qquad \boldsymbol{d}_3 = \begin{bmatrix} 16 \\ 16 \\ 16 \\ 15 \end{bmatrix}.$$

Con estos bloques, el procesador P_0 resuelve los sistemas

$$A_0 \boldsymbol{y}_0 = \boldsymbol{d}_0, \qquad A_0 \boldsymbol{z}_0 = \boldsymbol{e}_k,$$

cuyas soluciones son

$$\boldsymbol{y}_0 = \begin{bmatrix} 1.000 \\ 1.001 \\ 0.993 \\ 1.085 \end{bmatrix}, \qquad \boldsymbol{z}_0 = \begin{bmatrix} -0.00005 \\ 0.00060 \\ -0.00720 \\ 0.08510 \end{bmatrix}.$$

El procesador P_1 resuelve los sistemas

$$A_1\boldsymbol{y}_1 = \boldsymbol{d}_1, \quad A_1\boldsymbol{z}_1 = \boldsymbol{e}_1, \quad A_1\boldsymbol{z}_2 = \boldsymbol{e}_k,$$

cuyas soluciones son

$$\boldsymbol{y}_1 = \begin{bmatrix} 1.2550 \\ 0.9354 \\ 1.0090 \\ 1.0810 \end{bmatrix}, \qquad \boldsymbol{z}_1 = \begin{bmatrix} 0.0850 \\ -0.2170 \\ 0.0056 \\ -0.0014 \end{bmatrix}, \qquad \boldsymbol{z}_2 = \begin{bmatrix} -0.00005 \\ 0.00060 \\ -0.00700 \\ 0.08510 \end{bmatrix}.$$

El procesador P_2 resuelve los sistemas

$$A_2\boldsymbol{y}_2 = \boldsymbol{d}_2, \quad A_2\boldsymbol{z}_3 = \boldsymbol{e}_1, \quad A_2\boldsymbol{z}_4 = \boldsymbol{e}_k.$$

Nótese que, debido a la forma de la matriz de coeficientes de este ejemplo, los bloques de los procesadores P_1 y P_2 son iguales, por lo

que las soluciones de los sistemas intermedios que resuelven son las mismas; así,

$$\boldsymbol{y}_2 = \begin{bmatrix} 1.2550 \\ 0.9354 \\ 1.0090 \\ 1.0810 \end{bmatrix}, \qquad \boldsymbol{z}_3 = \begin{bmatrix} 0.0850 \\ -0.2170 \\ 0.0056 \\ -0.0014 \end{bmatrix}, \qquad \boldsymbol{z}_4 = \begin{bmatrix} -0.00005 \\ 0.00060 \\ -0.00700 \\ 0.08510 \end{bmatrix}.$$

El procesador P_3 resuelve los sistemas

$$A_3\boldsymbol{y}_3 = \boldsymbol{d}_3, \qquad A_3\boldsymbol{z}_5 = \boldsymbol{e}_1,$$

cuyas soluciones son

$$\boldsymbol{y}_3 = \begin{bmatrix} 1.2550 \\ 0.9348 \\ 1.0170 \\ 0.9958 \end{bmatrix}, \qquad \boldsymbol{z}_5 = \begin{bmatrix} 0.0850 \\ -0.2170 \\ 0.0056 \\ -0.0014 \end{bmatrix}.$$

Ahora construimos el sistema tridiagonal (5.15) para este caso concreto, que viene dado por

$$\begin{bmatrix} 0.085 & 0.3333 & & & & & \\ 1.000 & 0.0850 & -0.00005 & & & & \\ & -0.0014 & 0.08510 & 0.3333 & & & \\ & & 1.00000 & 0.0850 & -0.00005 & & \\ & & & -0.0014 & 0.08510 & 0.333 & \\ & & & & 1.00000 & 0.085 & \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} -1.085 \\ 1.255 \\ 1.081 \\ 1.255 \\ 1.081 \\ 1.255 \end{bmatrix},$$

cuya solución es

$$\begin{bmatrix} -1.000 \\ -3.000 \\ -1.000 \\ -3.001 \\ -1.000 \\ -3.001 \end{bmatrix}.$$

Una vez calculada la solución $\boldsymbol{\alpha}$*, cada procesador utiliza ciertas componentes de esta solución para calcular las soluciones parciales del sistema inicial. Así, el procesador* P_0 *calcula las cuatro primeras componentes de la solución final de la forma*

$$\boldsymbol{x}_0 = \boldsymbol{y}_0 + \alpha_0 \boldsymbol{z}_0 = \begin{bmatrix} 1.000 \\ 1.001 \\ 0.993 \\ 1.085 \end{bmatrix} - \begin{bmatrix} -0.00005 \\ 0.00060 \\ -0.00720 \\ 0.08510 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

De una forma análoga, el procesador P_1 *calcula las cuatro siguientes componentes de la solución final mediante la expresión*

$$\boldsymbol{x}_1 = \boldsymbol{y}_1 + \alpha_1 \boldsymbol{z}_1 + \alpha_2 \boldsymbol{z}_2 = \begin{bmatrix} 1.2550 \\ 0.9354 \\ 1.0090 \\ 1.0810 \end{bmatrix} - 3 \begin{bmatrix} 0.0850 \\ -0.2170 \\ 0.0056 \\ -0.0014 \end{bmatrix} - \begin{bmatrix} -0.00005 \\ 0.00060 \\ -0.00700 \\ 0.08510 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

El procesador P_2 calcula

$$\boldsymbol{x}_2 = \boldsymbol{y}_2 + \alpha_3 \boldsymbol{z}_3 + \alpha_4 \boldsymbol{z}_4 = \begin{bmatrix} 1.2550 \\ 0.9354 \\ 1.0090 \\ 1.0810 \end{bmatrix} - 3.001 \begin{bmatrix} 0.0850 \\ -0.2170 \\ 0.0056 \\ -0.0014 \end{bmatrix} - 1 \begin{bmatrix} -0.00005 \\ 0.00060 \\ -0.00700 \\ 0.08510 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Finalmente, el procesador P_3 calcula las últimas cuatro componentes de la solución de una forma similar a los casos anteriores

$$\boldsymbol{x}_3 = \boldsymbol{y}_3 + \alpha_5 \boldsymbol{z}_5 = \begin{bmatrix} 1.2550 \\ 0.9348 \\ 1.0170 \\ 0.9958 \end{bmatrix} - 3.001 \begin{bmatrix} 0.0850 \\ -0.2170 \\ 0.0056 \\ -0.0014 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

De esta forma queda calculada la solución del sistema propuesto inicialmente en este ejemplo que, como cabía esperar, tiene todas sus componentes iguales a uno.

5.3 Algoritmos BSP divide y vencerás

5.3.1 Algoritmo BSP basado en el método de Bondeli

De acuerdo con las características de este método expuestas en la sección 5.2 y observando detalladamente el ejemplo 5.1, llegamos a la conclusión de que únicamente son necesarios tres superpasos para construir un algoritmo BSP que resuelva un sistema tridiagonal por el método de Bondeli. Un superpaso será necesario para las comunicaciones iniciales; en un segundo superpaso se resolverán los sistemas

iniciales de forma independiente en cada procesador, mientras que en el tercer superpaso se construirá y resolverá el sistema tridiagonal auxiliar que nos permitirá obtener la solución general. El siguiente algoritmo BSP recoge las características fundamentales del método divide y vencerás desarrolladas en la sección 5.2.

Algoritmo 5.1 Método BSP divide y vencerás de Bondeli para sistemas tridiagonales.

Superpaso 1

Comunicación de datos a los procesadores. Para $i = 1, 2, \ldots, p-1$, el procesador P_i recibe la submatriz A_i y el vector $\boldsymbol{d}_i$ desde el procesador principal.

Superpaso 2

1 Resolución de los subsistemas intermedios y envío de los vectores auxiliares al procesador principal.

  1.1 En el procesador P_0 se resuelven los sistemas (5.19).

  1.2 En el procesador P_i, para $i = 1, 2, \ldots, p-2$, se resuelven los sistemas (5.20).

  1.3 En el procesador P_{p-1} se resuelven los sistemas (5.21).

2 Comunicación. El procesador P_i, para $i = 1, 2, \ldots, p-1$, envía los vectores $\boldsymbol{y}_i, \boldsymbol{z}_{2i-1}, \boldsymbol{z}_{2i}$ al procesador P_0.

Superpaso 3

Obtención de la solución final. En el procesador P_0,

1 se calculan H y el vector $\boldsymbol{\beta}$ mediante las expresiones (5.16), (5.17) y (5.18),

2 se resuelve el sistema (5.15),

3 se calcula la solución $\boldsymbol{x}$ utilizando la expresión (5.12).

Ya se ha comentado en la sección 5.2 que en el segundo superpaso se realizan la mayoría de los cálculos aritméticos para calcular los vectores $\boldsymbol{y}$ y $\boldsymbol{z}$ mediante la resolución de los sistemas (5.19), (5.20) y (5.21).

Si observamos las expresiones (5.19), (5.20) y (5.21), notamos que debemos resolver sistemas con la misma matriz de coeficientes y distintos vectores de términos independientes. Como ya vimos en la sección 1.1, estas características nos llevan a la utilización de la factorización LU de la matriz de coeficientes como método óptimo para el cálculo de los vectores $\boldsymbol{y}_i$, $\boldsymbol{z}_{2i-1}$ y $\boldsymbol{z}_{2i}$, para $i = 0, 1, \ldots, p-1$, teniendo en cuenta que $\boldsymbol{z}_{2p-2} = 0$.

En consecuencia, podemos describir los cálculos que realiza cada uno de los procesadores en el superpaso 2 del algoritmo 5.1 mediante el siguiente esquema:

Procesador P_0:

- Cálculo de $A_0 = L_0 U_0$.
- Resolución de los sistemas $L_0 \boldsymbol{u}_0 = \boldsymbol{d}_0$ y $U_0[\ \boldsymbol{y}_0 \quad \boldsymbol{z}_0\ ] = [\ \boldsymbol{u}_0 \quad \boldsymbol{e}_k\ ]$.

Procesador P_i, para $i = 1, 2, \ldots, p-2$:

- Cálculo de $A_i = L_i U_i$.
- Resolución de los sistemas $L_i[\ \boldsymbol{u}_i \quad \boldsymbol{v}_i\ ] = [\ \boldsymbol{d}_i \quad \boldsymbol{e}_1\ ]$ y $U_i[\ \boldsymbol{y}_i \quad \boldsymbol{z}_{2i-1} \quad \boldsymbol{z}_{2i}\ ] = [\ \boldsymbol{u}_i \quad \boldsymbol{v}_i \quad \boldsymbol{e}_k\ ]$.

Procesador P_{p-1}:

- Cálculo de $A_{p-1} = L_{p-1} U_{p-1}$.
- Resolución de los sistemas $L_{p-1}[\ \boldsymbol{u}_{p-1} \quad \boldsymbol{v}_{p-1}\ ] = [\ \boldsymbol{d}_{p-1} \quad \boldsymbol{e}_1\ ]$ y $U_{p-1}[\ \boldsymbol{y}_{p-1} \quad \boldsymbol{z}_{2p-3}\ ] = [\ \boldsymbol{u}_{p-1} \quad \boldsymbol{v}_{p-1}\ ]$.

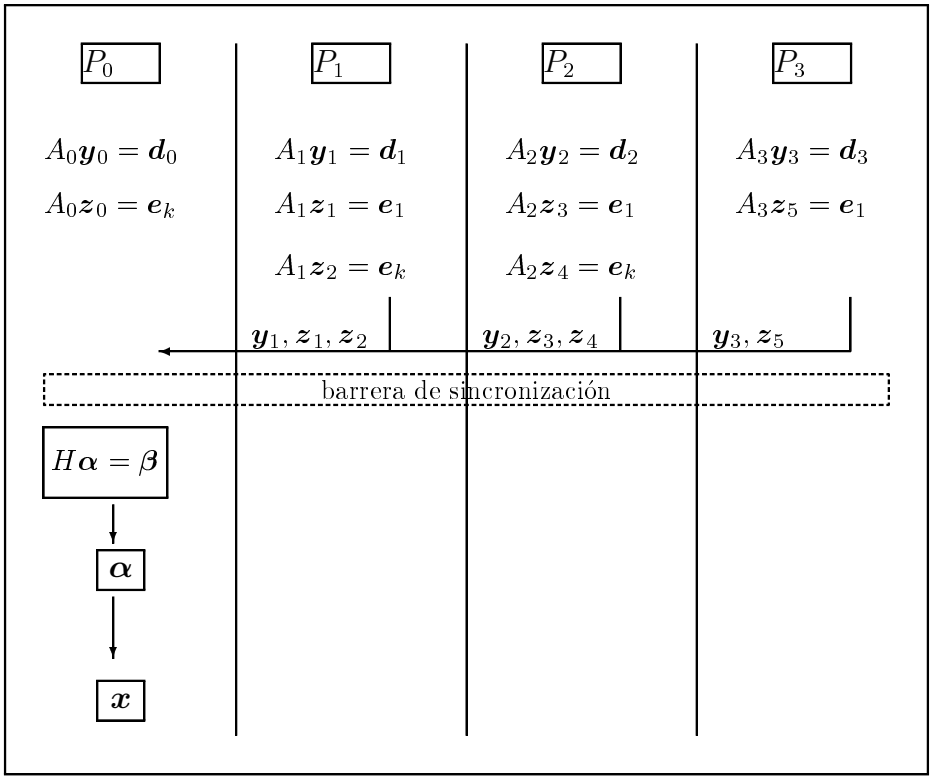


Figura 5.1: *Esquema de ejecución del algoritmo 5.1, para 4 procesadores.*

La figura 5.1 muestra un ejemplo de ejecución del algoritmo 5.1 para el caso particular en que $p = 4$.

Una característica fundamental que presenta el algoritmo 5.1 es que los cálculos que se realizan en el superpaso 3 los realiza únicamente el procesador principal; por eso, al final del superpaso 2 se produce una comunicación de elementos desde todos los procesadores al procesador principal, que es el encargado de formar el sistema (5.15) y resolverlo, de forma secuencial, para obtener la solución final.

5.3.2 Coste computacional

Seguidamente, calculamos los costes de cada uno de los superpasos que integran el algoritmo 5.1 con el fin de obtener su coste computacional.

Coste del superpaso 1. El coste del superpaso viene dado únicamente por el proceso de comunicación habitual desde el procesador principal al resto de procesadores. En total son $4k$ los elementos que recibe cada procesador, por lo que el coste de este superpaso es de

$$[4k(p-1)]g + l = 4(n-k)g + l \quad \text{flops.} \tag{5.22}$$

Coste del superpaso 2. En el superpaso 2 tenemos coste aritmético y coste de comunicación, por lo que los calculamos de forma separada.

Para obtener el coste aritmético, se deben tener en cuenta el número de operaciones aritméticas que realiza cualquiera de los procesadores P_i, para $i = 1, 2, \ldots, p-2$, ya que resuelven tres sistemas. Estas tareas, con su coste computacional, se resumen en los siguientes puntos

- La factorización LU de la matriz A_i requiere $3k - 3$ flops.
- La resolución del sistema $A_i \boldsymbol{y}_i = \boldsymbol{d}_i$, mediante la factorización LU requiere $5k - 4$ flops.

- La resolución del sistema $A_i \boldsymbol{z}_{2i-1} = \boldsymbol{e}_1$ supone un total de $4k-3$ flops ya que, al ser e_1 el vector de términos independientes, no se realiza ninguna suma.
- La resolución del sistema $A_i \boldsymbol{z}_{2i} = \boldsymbol{e}_k$ requiere sólamente $2k-1$ flops.

En consecuencia, necesitamos efectuar un total de

$$14k - 11 \quad \text{flops} \tag{5.23}$$

para completar las tareas aritméticas que se llevan a cabo en este superpaso.

Por otro lado, el coste de comunicación está relacionado con el envío de los vectores $\boldsymbol{y}_i$, para $i = 1, 2, \ldots, p-1$, y los vectores $\boldsymbol{z}_j$, para $j = 1, 2, \ldots, 2p-3$, al procesador principal. El procesador principal recibe dos vectores de tamaño k del procesador P_{p-1} y tres vectores de tamaño k de los procesadores no extremos. Esto significa que el procesador principal recibe un total de $3k(p-2) + 2k = 3n - 4k$ unidades de datos, por lo que el coste de comunicación de este superpaso es de

$$[3k(p-2) + 2k]g \quad \text{flops.} \tag{5.24}$$

En consecuencia, sumando los costes aritmético y computacional dados por las expresiones (5.23) y (5.24), tenemos que el coste total del superpaso 2 es de

$$(14k - 11) + (3n - 4k)g + l \quad \text{flops.} \tag{5.25}$$

Coste del superpaso 3. En este superpaso sólo se realizan operaciones aritméticas en el procesador principal y no se producen comunicaciones entre los procesadores. Las tareas que se realizan en el procesador principal, con su coste computacional son

- La obtención de la matriz H requiere $2p-2$ flops.

- Para resolver el sistema (5.15), utilizando el método de eliminación de Gauss, son necesarios $8(2p-2)-7$ flops.
- El cálculo de la solución $\boldsymbol{x}$ a partir de la expresión (5.12) supone la realización de $4(n-k)$ flops.

En consecuencia, el coste del superpaso 3 es de

$$4n + 18p - 4k - 25 \quad \text{flops.} \tag{5.26}$$

Sumando las expresiones (5.22), (5.25) y (5.26) obtenemos que el coste computacional del algoritmo 5.1 es de

$$4n + 18p + 10k - 36 + (7n - 8m)g + 2l \quad \text{flops.}$$

5.3.3 Algoritmo BSP modificado basado en el método de Bondeli

En el algoritmo 5.1 advertimos la pérdida de paralelismo en el último superpaso, donde los cálculos se realizan en el procesador principal y el resto están inactivos. La ventaja que presenta es que minimiza el número de superpasos y, por tanto, el número de barreras de sincronización necesarias para ejecutar el algoritmo.

Con el fin de evitar esta pérdida de paralelismo en el superpaso 3 podemos considerar una modificación consistente en que cada procesador calcule y resuelva el sistema (5.15) de forma simultánea. Posteriormente, cada procesador calcula en paralelo las componentes de la solución final que le corresponden y las envía al procesador principal. El siguiente algoritmo recoge esta modificación respecto del algoritmo 5.1.

Algoritmo 5.2 Método divide y vencerás modificado según el modelo BSP para sistemas tridiagonales basado en el método divide y vencerás de Bondeli.

Superpaso 1

Comunicación de datos a los procesadores. Para $i = 1, 2, \ldots, p-1$, el procesador P_i recibe la submatriz A_i y el vector $\boldsymbol{d}_i$ desde el procesador principal.

Superpaso 2

1 Resolución de los subsistemas intermedios y envío de los vectores auxiliares al procesador principal.

  1.1 En el procesador P_0 se resuelve el sistema (5.19).

  1.2 En el procesador P_i, para $i = 1, 2, \ldots, p-2$, se resuelve el sistema (5.20).

  1.3 En el procesador P_{p-1} se resuelven los sistemas (5.21).

2 Comunicación. El procesador P_i, para $i = 0, 1, \ldots, p-1$, envía los vectores $\boldsymbol{y}_i$, $\boldsymbol{z}_{2i-1}$, $\boldsymbol{z}_{2i}$ al resto de procesadores, siendo $\boldsymbol{z}_{-1} = \boldsymbol{z}_{2p-2} = \boldsymbol{0}$.

Superpaso 3

Obtención de la solución final.

1 En el procesador P_i, para $i = 0, 1, \ldots, p-1$,

  1.1 Se calculan H y el vector β mediante las expresiones (5.16), (5.17) y (5.18).

  1.2 Se resuelve el sistema (5.15).

  1.3 Se calcula el vector solución parcial $\boldsymbol{x}_i$ utilizando la expresión (5.12).

2 Comunicación. El procesador P_i, para $i = 1, 2, \ldots, p-1$, envía su vector solución parcial $\boldsymbol{x}_i$ al procesador P_0.

La figura 5.2 nos muestra la ejecución del algoritmo 5.2 para el caso particular en que $p = 4$. Notemos las diferencias más importantes entre los algoritmos 5.1 y 5.2. En el algoritmo 5.2 no perdemos el paralelismo a lo largo de todo el proceso, como ya se ha mencionado;

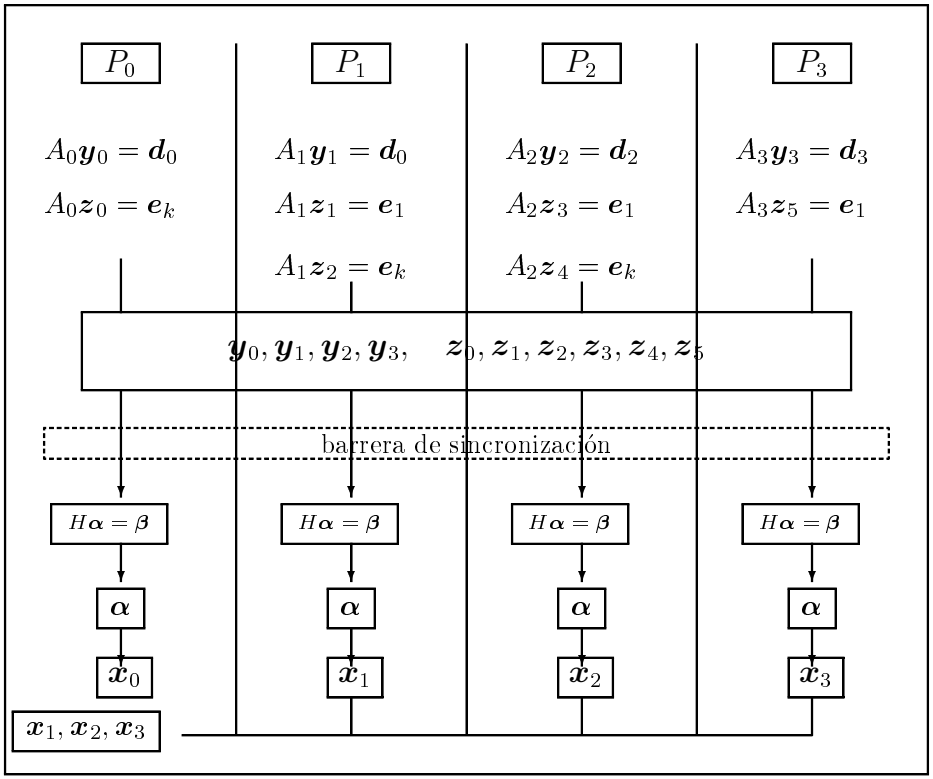


Figura 5.2: *Esquema de ejecución del algoritmo 5.2, para 4 procesadores.*

sin embargo, se requieren tres barreras de sincronización, ya que en el superpaso 3 hay una comunicación final de las soluciones parciales al procesador principal.

Otro aspecto que los diferencia es el proceso de comunicación que se lleva a cabo en el superpaso 2. En el algoritmo 5.2 se produce una comunicación donde cada procesador no extremo envía al resto de procesadores tres vectores de tamaño k. Los procesadores principal y final únicamente envía al resto de procesadores dos vectores de tamaño k. Este modelo de comunicación es un *broadcast* de cada procesador al resto. En el algoritmo 5.1 la comunicación es desde todos los procesadores al principal.

5.3.4 Coste computacional

Calculamos ahora los costes de cada uno de los superpasos que componen el algoritmo 5.2.

Coste del superpaso 1. Este superpaso es análogo al superpaso 1 del algoritmo 5.1, por lo que su coste computacional viene determinado por la expresión (5.22).

Coste del superpaso 2. La diferencia entre el superpaso 2 del algoritmo 5.2 y el superpaso 2 del algoritmo 5.1 se encuentra en la comunicación. Las tareas aritméticas que cada procesador desarrolla son las mismas, por lo que el coste aritmético viene dado por la expresión (5.23).

Para determinar el coste de comunicación de este superpaso hemos de tener en cuenta el máximo de las unidades de datos enviadas o recibidas por el procesador que más datos comunica o recibe. El número máximo de unidades de datos enviadas por los procesadores P_i, para $i = 1, 2, \ldots, p-2$, es de $3k(p-1)$ mientras que el número máximo de datos recibidos por los procesadores que más elementos reciben es $3k(p-2)+2k$. Así, el número máximo de datos enviados o recibidos por cualquiera de estos procesadores es $3k(p-1)$, por lo que el modelo de comunicación es el de una $3k(p-1)$-relación. En consecuencia, el coste de comunicación es de

$$3k(p-1)g = 3(n-k)g \quad \text{flops.} \tag{5.27}$$

Por tanto, sumando las expresiones (5.23) y (5.27)obtenemos que el coste del superpaso 2 es de

$$14k - 11 + 3(n-k)g + l \quad \text{flops.} \tag{5.28}$$

Coste del superpaso 3. Para calcular el coste del superpaso 3 hemos de tener en cuenta el coste aritmético y de comunicación. El coste aritmético de este superpaso viene determinado por la ejecución de las siguientes tareas en cada uno de los procesadores

- La obtención de la matriz H requiere $2p-2$ flops.
- La resolución del sistema (5.15), utilizando el método de eliminación de Gauss, supone $8(2p-2)-7$ flops.
- El cálculo del vector solución parcial $\boldsymbol{x}_i$, a partir de la expresión (5.12), en cada uno de los procesadores centrales representa la realización de $4k$ operaciones.

En consecuencia, el coste aritmético es de

$$4k + 18(p-1) - 7 \quad \text{flops.} \tag{5.29}$$

Por otro lado, el coste de comunicación del superpaso viene determinado por la comunicación desde cada procesador al procesador principal de su vector correspondiente de soluciones parciales, lo cual representa un coste de

$$n - k \quad \text{flops.} \tag{5.30}$$

Así, si sumamos las expresiones (5.29) y (5.30) obtenemos que el coste computacional de este superpaso es de

$$4k + 18(p-1) - 7 + (n-k)g + l \quad \text{flops.} \tag{5.31}$$

Como consecuencia de todo lo dicho, sumando las expresiones (5.22), (5.28) y (5.31) obtenemos que el coste computacional del algoritmo 5.2 es de

$$18k + 18p - 36 + 8(n-k)g + 3l \quad \text{flops.}$$

5.3.5 Algoritmo BSP utilizando el método *recursive doubling*

Al igual que sucedía en el capítulo 4 con el algoritmo 4.2, los algoritmos 5.1 y 5.2 se basan en la resolución de un sistema tridiagonal auxiliar cuya solución nos permite obtener la solución del sistema inicial. Este sistema tridiagonal auxiliar se resuelve de forma secuencial en estos algoritmos. Podemos plantearnos, de la misma forma que en el capítulo 4 con los algoritmos 4.3, 4.4 y 4.5, la resolución de este sistema tridiagonal en paralelo utilizando todos los procesadores disponibles. En el capítulo 4 comprobamos que las diferencias entre los algoritmos eran mínimas y que los tiempos de los algoritmos que utilizaban diversos métodos de resolución en paralelo del sistema tridiagonal auxiliar eran prácticamente idénticos. Por esta razón, únicamente consideramos un nuevo algoritmo basado en el método de Bondeli que resuelva el sistema tridiagonal auxiliar en paralelo utilizando el método del *recursive doubling*. El motivo por el que utilizamos este método en lugar del de reducción cíclica se basa en que se obtienen tiempos ligeramente mejores en el IBM SP2 y en el cluster de Pentiums, como se aprecia en la sección 4.6.

A partir de las características del método *recursive doubling* expuestas en la sección 1.3.2, en esta sección estudiamos un método del tipo divide y vencerás basado en el método divide y vencerás de Bondeli, utilizando el método *recursive doubling* para resolver el sistema 5.15 en paralelo. Así, la diferencia básica entre este nuevo algoritmo y el algoritmo 5.1 es la forma en que se resuelve el sistema intermedio (5.15).

La matriz de coeficientes H del sistema (5.15), para el caso $p = 4$, viene dada por

$$H = \begin{bmatrix} \delta_0 & \mu_0 & & & & \\ \nu_1 & \delta_1 & \mu_1 & & & \\ & \nu_2 & \delta_2 & \mu_2 & & \\ & & \nu_3 & \delta_3 & \mu_3 & \\ & & & \nu_4 & \delta_4 & \mu_4 \\ & & & & \nu_5 & \delta_5 \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_k^T \boldsymbol{z}_0 & \frac{1}{c_{k+1}} & & & & \\ \frac{1}{b_k} & \boldsymbol{e}_1^T \boldsymbol{z}_1 & \boldsymbol{e}_1^T \boldsymbol{z}_2 & & & \\ & \boldsymbol{e}_k^T \boldsymbol{z}_1 & \boldsymbol{e}_k^T \boldsymbol{z}_2 & \frac{1}{c_{2k+1}} & & \\ & & \frac{1}{b_{2k}} & \boldsymbol{e}_1^T \boldsymbol{z}_3 & \boldsymbol{e}_1^T \boldsymbol{z}_4 & \\ & & & \boldsymbol{e}_k^T \boldsymbol{z}_3 & \boldsymbol{e}_k^T \boldsymbol{z}_4 & \frac{1}{c_{3k+1}} \\ & & & & \frac{1}{b_{3k}} & \boldsymbol{e}_1^T \boldsymbol{z}_5 \end{bmatrix} \tag{5.32}$$

Si observamos detenidamente la matriz (5.32) notamos, en primer lugar, que la situación de los elementos en los distintos procesadores sigue un esquema similar al del algoritmo 4.3. Los elementos de la primera fila se encuentran en el procesador P_0, los elementos de la segunda y tercera filas se encuentran en el procesador P_1, los de la tercera y cuarta filas están en P_2, mientras que el procesador P_3 tiene los elementos de la última fila. Una situación análoga sucede con los elementos del vector $\boldsymbol{\beta}$. En segundo lugar, observamos que, a diferencia de lo que ocurría en el algoritmo 4.3, la formación de la matriz H supone que cada uno de los procesadores P_i, para $i = 1, 2, \ldots, p-2$, realiza dos divisiones para calcular cada uno de los elementos ν_i, μ_i y δ_i, para $i = 0, 2, \ldots, 2p-4$.

En consecuencia, el nuevo algoritmo que resume estas características se diferencia básicamente del algoritmo 5.1 en los siguientes puntos:

(i) En el superpaso 2 del algoritmo 5.1, después de resolver los sistemas (5.19), (5.20) y (5.21) en los procesadores correspondientes, tenemos en el procesador P_i, para $i = 0, 1, \ldots, p-1$, los elementos de las filas $(2i-1)$-ésima y $2i$-ésimas de H y $\boldsymbol{\beta}$, respectivamente.

(ii) El paso de comunicación del superpaso 2 se elimina.

(iii) A partir del superpaso 2 se aplica el método *recursive doubling* para resolver el sistema (5.15).

El modelo de comunicación que se emplea para la ejecución del método *recursive doubling* es el mismo que se emplea en el algoritmo 4.4, ya que es el modelo que minimiza el número de elementos que circulan por la red utilizando un esquema de comunicación de tipo *fan-in*. Así mismo, el cálculo de las matrices B_i, para $i = 0, 1, \ldots, 2p-3$, se realiza utilizando las expresiones que aparecen en la sección 1.3.2.

De acuerdo con los comentarios anteriores y teniendo en cuenta la descripción del método *recursive doubling* de la sección 1.3.2, podemos dar el siguiente algoritmo BSP que resume todas estas características.

Algoritmo 5.3 Algoritmo BSP divide y vencerás basado en el método de Bondeli utilizando el método *recursive doubling*.

Superpaso 1

Comunicación de datos a los procesadores. Para $i = 1, 2, \ldots, p-1$, el procesador P_i recibe la submatriz A_i y el vector $\boldsymbol{d}_i$ desde el procesador principal. Inicialmente tomamos $\mu = -1$.

Superpaso 2

1 Resolución de los subsistemas intermedios.

- 1.1 En el procesador P_0 se resuelve el sistema (5.19).
- 1.2 En el procesador P_i, para $i = 1, 2, \ldots, p-2$, se resuelve el sistema (5.20).
- 1.3 En el procesador P_{p-1} se resuelve el sistema (5.21).
- 1.4 El procesador P_i, para $i = 0, 1, \ldots, p-1$,
  - 1.4.1 Calcula las matrices B_{2i-1} y B_{2i}.
  - 1.4.2 Calcula el producto $B[2i \mid 2i-1]$.

2 Comunicación.

- 2.1 El procesador P_i, para $i = 1, 2, \ldots, p-1$, comunica al procesador P_0 las matrices B_{2i-1} y B_{2i}.

2.2 El procesador P_{2i+1}, para $i = 0, 1, \ldots, 2^{m-1} - 1$, comunica al procesador P_{2i} la matriz $B[2i \mid 2i - 1]$.

Superpaso $m - q + 2$, para $q = m - 1, m - 2, \ldots, 2, 1$

1 El procesador $P_{2^{m-q}i}$, para $i = 0, 1, \ldots, 2^q - 1$ calcula la matriz

$$B[2^{m-q} + 2^{m-q+1}i + 2(\mu + 1) \mid 2^{m-q+1}i - 1].$$

2 El procesador $P_{2^{m-q}+2^{m-q+1}i}$, para $i = 0, 1, \ldots, \lfloor \frac{2^q-1}{2} \rfloor$ comunica al procesador $P_{2^{m-q+1}i}$ la matriz

$$B[2^{m-q} + 2^{m-q+1}(2i + 1) + 2(\mu + 1) \mid 2^{m-q+1}(2i + 1) - 1].$$

3 Actualizamos μ, como $\mu = 2(\mu + 1)$

Superpaso $m + 2$

El procesador P_0,

1 Calcula $B[2p - 2 \mid -1]$.

2 Calcula α_0 utilizando la expresión (4.26) y, mediante la expresión (4.25), obtiene el resto de componentes α_i, para $i = 1, 2, \ldots, 2p-3$.

3 Comunica las componentes $\alpha_{2i-1}, \alpha_{2i}$ al procesador P_i, para $i = 1, 2, \ldots, p - 1$.

Superpaso $m + 3$

El procesador P_i, para $i = 0, 1, \ldots, p - 1$,

1 Calcula $\boldsymbol{x}_i$ mediante la ecuación (5.12).

2 Envía el vector de soluciones parciales $\boldsymbol{x}_i$ al procesador principal.

Destacamos las escasas diferencias existentes entre este algoritmo y el algoritmo 4.4 basado en la fórmula de Sherman-Morrison-Woodbury. En ambos, el proceso de cálculo y comunicaciones que seguimos es muy parecido. Las diferencias se encuentran en la forma en la que se obtiene la matriz tridiagonal auxiliar que nos permite obtener la solución del sistema inicial y en algunos elementos de dicha matriz. Estas analogías van a provocar un coste y un comportamiento similar de ambos algoritmos.

5.3.6 Coste computacional

Coste del superpaso 1. Este superpaso es análogo al superpaso 1 del algoritmo 5.1, por lo que su coste viene dado por la expresión (5.22).

Coste del superpaso 2. Los apartados 1.1, 1.2 y 1.3 del superpaso 2 del algoritmo 5.3 coinciden con los apartados 1.1, 1.2 y 1.3 del superpaso 2 del algoritmo 5.1, por lo que el coste de estos apartados viene dado por la expresión (5.23), es decir, $14k - 11$ flops. El cálculo de las matrices B_{2i-1} y B_{2i}, que aparecen en la sección 1.3.2, supone 6 operaciones, mientras que por la forma especial que tienen estas matrices el producto de matrices $B[2i \mid 2i-1]$ requiere 5 operaciones. Por lo tanto, el coste aritmético de este superpaso es de $14k$ flops.

Por otro lado notemos que el modelo de comunicación que seguimos en este superpaso coincide con el modelo de comunicación del superpaso 2 del algoritmo 4.4, que viene dado por la expresión (4.33), es decir, $(6p-3)g$ flops.

Así pues, sumando los costes aritmético y de comunicación se obtiene que el coste computacional del superpaso 2 es de

$$14k + (6p-3)g + l \quad \text{flops.} \tag{5.33}$$

Coste de los superpasos $\boldsymbol{m-q+2}$, para $q = m-1, m-2, \ldots, 2, 1$. Tanto las operaciones aritméticas como las comunicaciones que se realizan en este conjunto de $m-1$ superpasos coincidene exactamente con las que se realizan en el algoritmo 4.4, por lo que su

coste computacional vendrá dado por la expresión (4.35), es decir,

$$20(m-1) + 6(m-1)g + (m-1)l \quad \text{flops.} \tag{5.34}$$

Coste del superpaso $m+2$. De nuevo, este superpaso coincide exactamente con el superpaso $m+2$ del algoritmo 4.4, por lo que su coste viene dado por la expresión (4.36), es decir

$$10p + 6 + (2p-3)g + l \quad \text{flops.} \tag{5.35}$$

Coste del superpaso $m+3$. Finalmente, el cálculo de las soluciones parciales $\boldsymbol{x}_i$ mediante la expresión (5.12) requiere $4k$ flops, mientras que el coste de comunicación es de $(n-k)g$ flops, ya que el procesador principal recibe $n-k$ datos desde todos los procesadores.

Por tanto, el coste de este superpaso es de

$$4k + (n-k)g + l \quad \text{flops.} \tag{5.36}$$

Sumando las expresiones (5.22), (5.33), (5.34), (5.35) y (5.36) y realizando las simplificaciones adecuadas obtenemos el coste total del algoritmo 5.3, que es de

$$18k + 20m + 10p - 14 + (5n - 5k + 6m + 8p - 12)\, g + (m+3)l \quad \text{flops.} \tag{5.37}$$

Como cabía esperar dadas las escasas diferencias existentes entre este algoritmo y el algoritmo 4.4, los costes teóricos obtenidos son casi idénticos, como se desprende de las expresiones (5.37) y (4.32).

5.4 Resultados teóricos y comparaciones

En esta sección presentamos los tiempos teóricos de ejecución de los algoritmos 5.2, 5.3 y 5.4 para las tres máquinas paralelas con las que se ha venido realizando las comparaciones en los capítulos anteriores. Recordemos que los parámetros de estas máquinas vienen dados por la tabla 2.1.

5.4.1 Tiempos en un IBM SP2

Las tablas 5.1 y 5.2 resumen, respectivamente, los tiempos comparados de los algoritmos estudiados en este capítulo para un IBM SP2 dotado con un switch de alto rendimiento y una conexión de tipo ethernet.

Analizamos de forma separada los resultados en esta máquina para los dos tipos de conexión que se utilizan. De la tabla 5.1 se observan varios hechos destacables. El primero de ellos es que el algoritmo 5.3 siempre es el más rápido de los tres para matrices con $n \geq 4096$. Para tamaños inferiores el más rápido es el algoritmo 5.1, aunque no podemos decir que existan diferencias significativas entre ellos dado el tamaño tan pequeño de las matrices. Para cualquier valor de n, el algoritmo 5.2 resulta el más lento.

Otro aspecto destacable es que mientras que en los algoritmos 5.1 y 5.2 los tiempos crecen al aumentar el número de procesadores de 2 a 4 y de 4 a 8, no sucede lo mismo con el algoritmo 5.3. El algoritmo 5.3 reduce los tiempos de ejecución al pasar de 2 a 4 procesadores para matrices con valores de n superiores a 32768, como se aprecia en la tabla 5.1. Cuando se aumenta de 4 a 8 el número de procesadores, vuelven a aumentar los tiempos aunque no excesivamente.

También se observa que las diferencias de tiempo entre el algoritmo más rápido y el más lento crecen notablemente al aumentar el número de procesadores. Por ejemplo, si consideramos el caso $n = 4194304$ vemos que la diferencia en los tiempos entre el algoritmo 5.3 (el más rápido) y el algoritmo 5.2 (el más lento), para $p = 2$, es de 0.7621 segundos, mientras que para $p = 4$, la diferencia en tiempos

IBM SP2 switch									
	p								
	2			4			8		
n	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3
512	0.0005	0.0007	0.0006	0.0007	0.0009	0.0010	0.0010	0.0012	0.0016
1024	0.0009	0.0011	0.0010	0.0012	0.0014	0.0014	0.0015	0.0017	0.0020
2048	0.0016	0.0019	0.0016	0.0021	0.0023	0.0020	0.0025	0.0028	0.0027
4096	0.0031	0.0036	0.0030	0.0038	0.0042	0.0033	0.0045	0.0048	0.0040
8192	0.0060	0.0070	0.0056	0.0074	0.0079	0.0059	0.0086	0.0090	0.0068
16384	0.0118	0.0138	0.0109	0.0145	0.0154	0.0111	0.0168	0.0173	0.0122
32768	0.0234	0.0274	0.0216	0.0287	0.0303	0.0215	0.0332	0.0339	0.0231
65536	0.0467	0.0547	0.0428	0.0570	0.0602	0.0423	0.0659	0.0672	0.0450
131072	0.0932	0.1091	0.0854	0.1137	0.1199	0.0839	0.1313	0.1338	0.0887
262144	0.1862	0.2180	0.1704	0.2271	0.2394	0.1671	0.2621	0.2669	0.1762
524288	0.3722	0.4358	0.3406	0.4540	0.4783	0.3334	0.5237	0.5330	0.3510
1048576	0.7442	0.8714	0.6809	0.9077	0.9563	0.6662	1.0470	1.0654	0.7008
2097152	1.4883	1.7425	1.3614	1.8151	1.9121	1.3316	2.0936	2.1301	1.4002
4194304	2.9765	3.4847	2.7226	3.6300	3.8237	2.6625	4.1867	4.2595	2.7992

Tabla 5.1: *Tiempos teóricos para los algoritmos 5.1, 5.2 y 5.3 medidos en un IBM SP2 dotado con un switch de alto rendimiento.*

IBM SP2 ethernet									
	p								
	2			4			8		
n	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3
512	0.0071	0.0096	0.0076	0.0230	0.0284	0.0227	0.0829	0.0989	0.0776
1024	0.0126	0.0169	0.0123	0.0422	0.0514	0.0372	0.1556	0.1849	0.1314
2048	0.0237	0.0316	0.0216	0.0807	0.0975	0.0660	0.3041	0.3568	0.2388
4096	0.0460	0.0610	0.0403	0.1577	0.1896	0.1237	0.5991	0.7006	0.4538
8192	0.0904	0.1198	0.0775	0.3116	0.3738	0.2391	1.1891	1.3884	0.8838
16384	0.1793	0.2374	0.1521	0.6194	0.7421	0.4698	2.3691	2.7638	1.7437
32768	0.3571	0.4726	0.3012	1.2351	1.4788	0.9313	4.7291	5.5146	3.4635
65536	0.7127	0.9429	0.5994	2.4664	2.9523	1.8543	9.4490	11.0163	6.9031
131072	1.4239	1.8836	1.1959	4.9290	5.8991	3.7004	18.8888	22.0197	13.7823
262144	2.8463	3.7650	2.3887	9.8543	11.7928	7.3924	37.7685	44.0264	27.5408
524288	5.6910	7.5278	4.7745	19.7049	23.5802	14.7766	75.5278	88.0399	55.0577
1048576	11.3806	15.0533	9.5460	39.4060	47.1550	29.5449	151.0464	176.0668	110.0916
2097152	22.7596	30.1044	19.0891	78.8083	94.3046	59.0814	302.0837	352.1207	220.1593
4194304	45.5178	60.2066	38.1752	157.6128	188.6039	118.1546	604.1583	704.2285	440.2947

Tabla 5.2: *Tiempos teóricos para los algoritmos 5.1, 5.2 y 5.3 medidos en un IBM SP2 utilizando conexión ethernet.*

entre ambos algoritmos es de 1.46 segundos. Esta diferencia de tiempos significa que ejecutar el algoritmo 5.3 supone un ahorro de tiempo del 34% aproximadamente frente a la ejecución del algoritmo 5.2, lo que representa un valor significativo.

La tabla 5.2 recoge los valores teóricos cuando se utiliza una conexión ethernet. Las características más notables descritas en el caso de conexión con switch se pueden extrapolar para esta conexión. El algoritmo 5.3 sigue siendo el más rápido y el algoritmo 5.2 es el más lento. Ahora esta característica es independiente del valor de n.

Una característica que no se observa en este caso es la reducción de tiempos en el algoritmo 5.3 al pasar de 2 a 4 procesadores; cuando se utiliza una conexión ethernet los tiempos aumentan de forma considerable para los tres algoritmos. En general podemos decir que al duplicar el número de procesadores los tiempos aumentan en un factor superior a tres. Esto nos da una idea del comportamiento paralelo altamente negativo de estos algoritmos en estas máquinas. Sin duda, el peso de las comunicaciones y el alto coste de las mismas cuando se utiliza este tipo de conexión determina este comportamiento negativo al duplicar el número de procesadores.

En cuanto a las diferencias de tiempo que se miden para los algoritmos 5.2 y 5.3, que son el más lento y más rápido respectivamente, debemos decir que aumentan ligeramente respecto al caso con switch, aunque no de forma significativa; los porcentajes de diferencia que antes se aproximaban al 35% ahora están entre e 38 y el 40%.

5.4.2 Tiempos en un CRAY T3D

Las tablas 5.3 y 5.4 muestran los resultados teóricos esperados en un CRAY T3D para los tres algoritmos estudiados en este capítulo.

La primera conclusión que podemos extraer al observar las tablas 5.3 y 5.4 es que, a diferencia de lo que ocurría en el IBM SP2, en esta máquina el algoritmo más lento es el algoritmo 5.1 a partir de 4 procesadores. El algoritmo más rápido sigue siendo el algoritmo 5.3, para cualquier número de procesadores y para cualquier valor de n. Para dos procesadores el algoritmo 5.2 ofrece peores resultados que el algoritmo 5.1; sin embargo, como ya se ha comentado, al aumentar a 4 el número de procesadores, los resultados que se obtienen al

CRAY T3D												
	p											
	2			4			8			16		
n	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3
2048	0.0018	0.0018	0.0018	0.0015	0.0012	0.0011	0.0014	0.0010	0.0009	0.0016	0.0011	0.0009
4096	0.0035	0.0036	0.0034	0.0029	0.0023	0.0021	0.0027	0.0019	0.0015	0.0029	0.0019	0.0014
8192	0.0069	0.0072	0.0068	0.0057	0.0046	0.0041	0.0054	0.0036	0.0029	0.0054	0.0034	0.0025
16384	0.0138	0.0143	0.0136	0.0113	0.0091	0.0080	0.0106	0.0071	0.0056	0.0106	0.0065	0.0047
32768	0.0275	0.0285	0.0270	0.0226	0.0181	0.0160	0.0210	0.0140	0.0111	0.0209	0.0126	0.0091
65536	0.0549	0.0569	0.0540	0.0451	0.0361	0.0318	0.0419	0.0277	0.0220	0.0415	0.0249	0.0180
131072	0.1098	0.1137	0.1079	0.0902	0.0722	0.0636	0.0837	0.0553	0.0439	0.0828	0.0495	0.0356
262144	0.2196	0.2272	0.2158	0.1803	0.1443	0.1271	0.1672	0.1105	0.0876	0.1652	0.0986	0.0710
524288	0.4391	0.4544	0.4315	0.3605	0.2884	0.2541	0.3344	0.2208	0.1749	0.3302	0.1969	0.1416
1048576	0.8782	0.9088	0.8630	0.7210	0.5768	0.5080	0.6686	0.4414	0.3497	0.6600	0.3936	0.2829
2097152	1.7564	1.8176	1.7258	1.4419	1.1535	1.0159	1.3371	0.8827	0.6992	1.3197	0.7868	0.5655
4194304	3.5128	3.6351	3.4515	2.8837	2.3070	2.0317	2.6740	1.7653	1.3983	2.6392	1.5732	1.1308

Tabla 5.3: *Tiempos teóricos para los algoritmos 5.1, 5.2 y 5.3 medidos en un CRAY T3D utilizando* 2, 4, 8 *y* 16 *procesadores.*

CRAY T3D												
	p											
	32			64			128			256		
n	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3
2048	0.0017	0.0012	0.0009	0.0019	0.0014	0.0010	0.0024	0.0020	0.0015	0.0032	0.0028	0.0024
4096	0.0031	0.0021	0.0015	0.0032	0.0021	0.0015	0.0037	0.0027	0.0020	0.0046	0.0037	0.0028
8192	0.0058	0.0037	0.0026	0.0058	0.0036	0.0024	0.0064	0.0043	0.0030	0.0074	0.0053	0.0038
16384	0.0113	0.0070	0.0048	0.0110	0.0065	0.0043	0.0118	0.0073	0.0049	0.0130	0.0086	0.0058
32768	0.0223	0.0136	0.0092	0.0213	0.0122	0.0080	0.0226	0.0135	0.0088	0.0242	0.0153	0.0100
65536	0.0442	0.0268	0.0180	0.0421	0.0237	0.0155	0.0442	0.0258	0.0167	0.0467	0.0285	0.0184
131072	0.0880	0.0531	0.0356	0.0836	0.0468	0.0305	0.0873	0.0504	0.0323	0.0916	0.0550	0.0351
262144	0.1757	0.1058	0.0709	0.1665	0.0929	0.0605	0.1735	0.0996	0.0637	0.1814	0.1080	0.0685
524288	0.3510	0.2112	0.1413	0.3324	0.1850	0.1204	0.3459	0.1981	0.1264	0.3609	0.2140	0.1353
1048576	0.7016	0.4220	0.2823	0.6642	0.3693	0.2402	0.6908	0.3950	0.2517	0.7201	0.4259	0.2689
2097152	1.4028	0.8436	0.5643	1.3277	0.7380	0.4798	1.3806	0.7887	0.5024	1.4384	0.8499	0.5362
4194304	2.8053	1.6869	1.1281	2.6548	1.4753	0.9590	2.7601	1.5762	1.0038	2.8750	1.6978	1.0707

Tabla 5.4: *Tiempos teóricos para los algoritmos 5.1, 5.2 y 5.3 medidos en un CRAY T3D para* 32, 64, 128 *y* 256 *procesadores.*

ejecutar el algoritmo 5.2 son sensiblemente mejores que los obtenidos con el algoritmo 5.1. Este hecho no resulta extraño si tenemos en cuenta que la diferencia básica entre ambos algoritmos radica en que el algoritmo 5.3 calcula en paralelo las soluciones finales, mientras que el algoritmo 5.1 las calcula de forma secuencial. Una máquina de este tipo con un coste de las comunicaciones muy bajo permite que la resolución del sistema tridiagonal auxiliar se realice en paralelo y resulte más ventajoso que su resolución secuencial.

También se observa que al aumentar inicialmente el número de procesadores, los tiempos disminuyen, como también cabe esperar en una máquina altamente paralela de estas características. Los tiempos se van reduciendo hasta llegar a un cierto número de procesadores, a partir del cuál vuelven a ir aumentando. Esta característica no es novedosa, ya se había producido anteriormente en los capítulos 3 y 4, cuando se expusieron los resultados numéricos para los métodos basados en las fórmulas de Sherman-Morrison y Sherman-Morrison-Woodbury. Esto nos permite hablar de un número de procesadores óptimo para la ejecución de los distintos algoritmos y este número de procesadores óptimo depende del valor de n. Así, la tabla 5.5 recoge el número óptimo de procesadores para los distintos tamaños de matrices cuando se ejecutan los tres algoritmos estudiados en este capítulo.

Nótese la gran similitud de esta tabla con la tabla 4.5 que nos permite afirmar, salvo para algunos casos concretos con valores pequeños de n, que el número óptimo de procesadores al ejecutar estos algoritmos en esta máquina es de 64.

5.4.3 Tiempos en un cluster de Pentiums

En esta sección se analizan los resultados teóricos sobre un cluster de Pentiums, siguiendo un esquema similar al descrito para las máquinas anteriores. La tabla 5.6 recoge los tiempos teóricos de ejecución de los tres algoritmos estudiados en este capítulo para 2, 4 y 8 procesadores.

La tabla 5.6 nos muestra claramente que el algoritmo 5.3 es el más rápido para cualquier número de procesadores siempre que $n > 2048$. Para valores de n muy pequeños el algoritmo más rápido es el algoritmo 5.1, lo que no resulta extraño ya que la matriz tridiagonal auxiliar no es lo suficientemente grande como para que compense, desde el punto de vista computacional, su resolución en

CRAY T3D			
n	Alg. 5.1	Alg. 5.2	Alg. 5.3
2048	8	8	16
4096	8	16	16
8192	16	16	16
16384	16	64	64
32768	64	64	64
65536	64	64	64
131072	64	64	64
262144	64	64	64
524288	64	64	64
1048576	64	64	64
2097152	64	64	64
4194304	64	64	64

Tabla 5.5: *Número óptimo de procesadores en un CRAY T3D para tamaños de matrices comprendidos entre* $n = 2048$ *y* $n = 4194304$.

Cluster de Pentiums									
	p								
	2			4			8		
n	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3	Alg. 5.1	Alg. 5.2	Alg. 5.3
512	0.0005	0.0006	0.0005	0.0008	0.0010	0.0011	0.0012	0.0016	0.0019
1024	0.0008	0.0010	0.0008	0.0013	0.0016	0.0014	0.0018	0.0022	0.0023
2048	0.0015	0.0019	0.0014	0.0023	0.0027	0.0022	0.0029	0.0035	0.0031
4096	0.0028	0.0036	0.0025	0.0042	0.0050	0.0036	0.0052	0.0060	0.0047
8192	0.0054	0.0070	0.0048	0.0082	0.0096	0.0055	0.0097	0.0111	0.0079
16384	0.0107	0.0138	0.0094	0.0161	0.0188	0.0125	0.0187	0.0212	0.0143
32768	0.0213	0.0273	0.0185	0.0319	0.0372	0.0243	0.0367	0.0415	0.0271
65536	0.0425	0.0544	0.0367	0.0636	0.0739	0.0479	0.0727	0.0821	0.0528
131072	0.0848	0.1086	0.0731	0.1269	0.1474	0.0951	0.1448	0.1633	0.1042
262144	0.1695	0.2171	0.1459	0.2535	0.2944	0.1895	0.2889	0.3256	0.2069
524288	0.3388	0.4339	0.2915	0.5067	0.5885	0.3783	0.5771	0.6503	0.4124
1048576	0.6775	0.8677	0.5826	1.0131	1.1765	0.7558	1.1535	1.2997	0.8233
2097152	1.3549	1.7351	1.1650	2.0259	2.3526	1.5110	2.3064	2.5985	1.6451
4194304	2.7097	3.4700	2.3298	4.0516	4.7047	3.0213	4.6120	5.1962	3.2887

Tabla 5.6: *Tiempos teóricos para los algoritmos 5.1, 5.2 y 5.3 medidos en un Cluster de Pentiums para* 2, 4 *y* 8 *procesadores.*

paralelo. Sin embargo, cuando trabajamos con valores grandes de n, los tiempos de ejecución del algoritmo 5.3 demuestran que resulta muy conveniente utilizar algún método paralelo para resolver el sistema auxiliar (5.15). Así, por ejemplo, para $n = 4096$ y $p = 4$, la diferencia de tiempos entre el algoritmo 5.1 y el algoritmo 5.3 es de 0.0006 segundos a favor del algoritmo 5.3, lo que representa un 14% del tiempo total. Sin embargo, si tomamos $n = 4194304$ y seguimos con $p = 4$, ahora la diferencia de tiempos entre ambos algoritmos es de 1.03 segundos, lo que representa un 25% del tiempo total. Esto significa que resolviendo un sistema tridiagonal de este tamaño con $p = 4$, podemos ahorrar un 25% del tiempo total si utilizamos el algoritmo 5.3 frente al algoritmo 5.1, que es bastante considerable. Si duplicamos el número de procesadores, el comportamiento es el mismo, con pequeñas variaciones en los porcentajes. Para $n = 4096$ ahora el ahorro en tiempo es del 9% únicamente, mientras que si aumentamos hasta $n = 4194304$ el ahorro en tiempo es del 28%.

También destacamos que el comportamiento paralelo de estos algoritmos no es demasiado bueno, ya que aumentan los tiempos al aumentar el número de procesadores, independientemente del tamaño de la matriz y del número de procesadores. El incremento de tiempos es mayor cuando pasamos de 2 a 4 procesadores que cuando duplicamos a 8 procesadores. Por ejemplo, si consideramos el algoritmo 5.1 para $n = 4194304$ tenemos un aumento del 33% al pasar de 2 a 4 procesadores, mientras que el porcentaje disminuye hasta el 12% al pasar de 4 a 8 procesadores. Esta característica se repite para los otros algoritmos, si bien con pequeñas variaciones en los porcentajes.

5.4.4 Estudio del speedup

En esta sección realizamos un estudio del *speedup* y de la eficiencia para los algoritmos 5.1, 5.2 y 5.3 que hemos estudiado a lo largo de este capítulo.

En la tabla 5.7 aparecen los valores del *speedup* y la eficiencia en las tres máquinas que venimos utilizando para obtener los resultados teóricos. Los mejores valores se obtienen en el CRAY T3D, donde se alcanza un valor del 70% en el algoritmo 4.5 para $p = 2$. Al aumentar el número de procesadores va disminuyendo la eficiencia de forma notable. Este descenso se hace más notorio en el IBM SP2 y en el cluster. Los resultados más negativos se obtienen en el IBM SP2 con conexión ethernet, al igual que ha venido ocurriendo en los

IBM SP2 switch						
	speedup			eficiencia		
p	Alg. 5.1	5.2	5.3	Alg. 5.1	5.2	Alg. 5.3
2	0.72	0.62	0.79	36.18	30.90	39.55
4	0.59	0.56	0.81	14.83	14.08	20.22
8	0.51	0.51	0.77	6.43	6.32	9.62

(a) *IBM SP2 switch*

IBM SP2 ethernet						
	speedup			eficiencia		
p	Alg. 5.1	5.2	5.3	Alg. 5.1	5.2	Alg. 5.3
2	0.05	0.04	0.06	2.37	1.79	2.82
4	0.01	0.01	0.02	0.34	0.29	0.46
8	0.004	0.003	0.005	0.05	0.04	0.06

(b) *IBM SP2 ethernet*

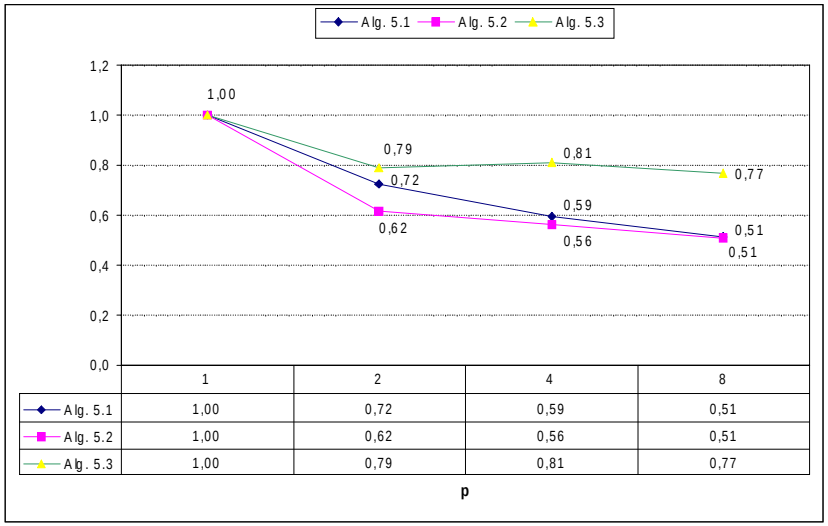
CRAY T3D						
	speedup			eficiencia		
p	Alg. 5.1	5.2	Alg. 5.3	Alg. 5.1	5.2	Alg. 5.3
2	1.38	1.33	1.40	68.90	66.58	70.13
4	1.68	2.10	2.38	48.97	52.46	59.57
8	1.81	2.74	3.46	22.63	34.28	43.27
16	1.83	3.08	4.28	11.46	19.23	26.75
32	1.72	2.87	4.29	5.39	8.97	13.40
64	1.82	3.28	5.05	3.38	6.07	9.34
128	1.75	3.07	4.82	1.37	2.40	3.76
256	1.68	2.85	4.51	0.66	1.11	1.76

(c) *CRAY T3D*

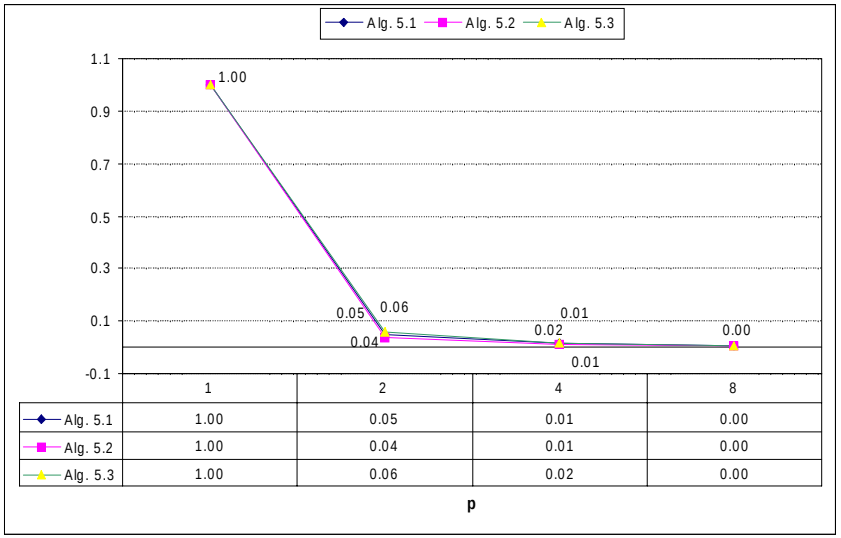
Cluster de Pentiums						
	speedup			eficiencia		
p	Alg. 5.1	5.2	Alg. 5.3	Alg. 5.1	5.2	Alg. 5.3
2	0.24	0.18	0.27	11.74	9.17	13.66
4	0.16	0.14	0.21	3.93	3.38	5.26
8	0.14	0.12	0.19	1.72	1.53	2.42

(d) *Cluster de Pentiums*

Tabla 5.7: *Speedup y eficiencia de los algoritmos 5.1, 5.2 y 5.3 para* $n = 2097152$.



(a) *IBM SP2 switch*



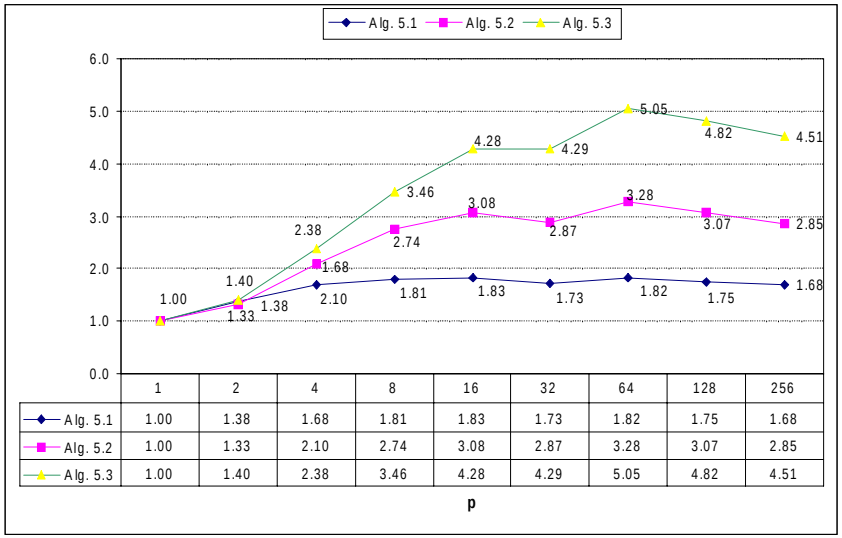
(b) *IBM SP2 ethernet*

Figura 5.3: *Valores del speedup en un IBM SP2*

capítulos anteriores con todos los algoritmos estudiados.

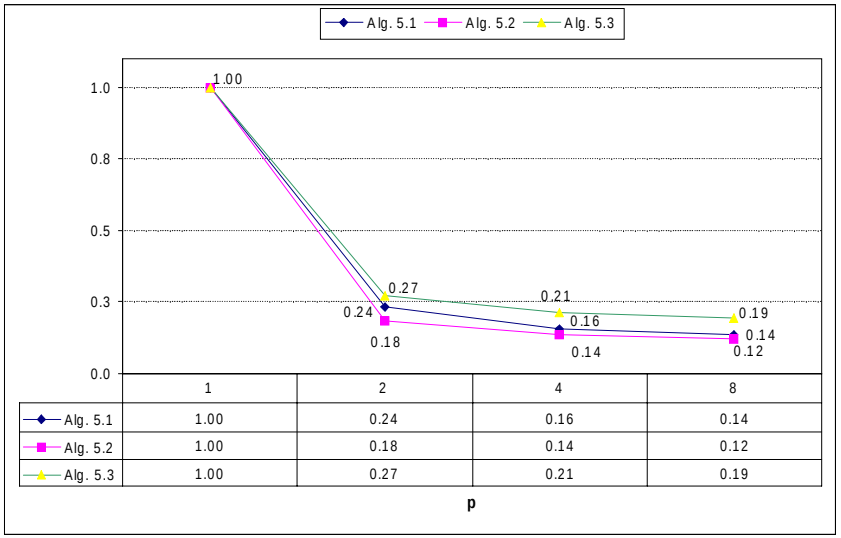
Hay que notar que los valores del *speedup* y la eficiencia que se obtienen para estos algoritmos son sensiblemente peores que los que se obtuvieron para los algoritmos estudiados en los capítulos anteriores. Por ejemplo, en los algoritmos basados en la fórmula de Sherman-Morrison (véase el capítulo 3), se obtuvieron valores del speedup de 9.47. Ahora, sin embargo, el valor máximo del *speedup* es de 5.05, lo que representa casi la mitad del primero. Por otra parte, los valores en el IBM SP2 y en el cluster son muy bajos.

Las figuras 5.3 y 5.4 nos muestran los valores del *speedup* obtenidos en la tabla 5.7.



	1	2	4	8	16	32	64	128	256
Alg. 5.1	1.00	1.38	1.68	1.81	1.83	1.73	1.82	1.75	1.68
Alg. 5.2	1.00	1.33	2.10	2.74	3.08	2.87	3.28	3.07	2.85
Alg. 5.3	1.00	1.40	2.38	3.46	4.28	4.29	5.05	4.82	4.51

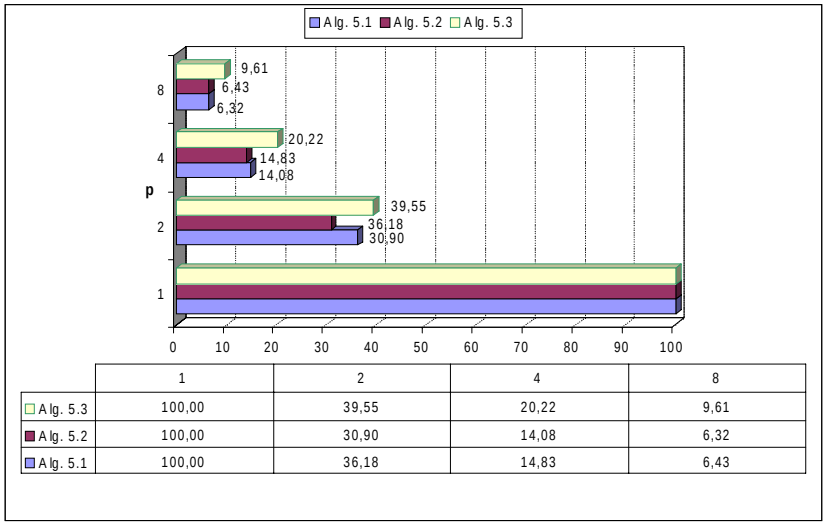
(a) *CRAY T3D*



	1	2	4	8
Alg. 5.1	1.00	0.24	0.16	0.14
Alg. 5.2	1.00	0.18	0.14	0.12
Alg. 5.3	1.00	0.27	0.21	0.19

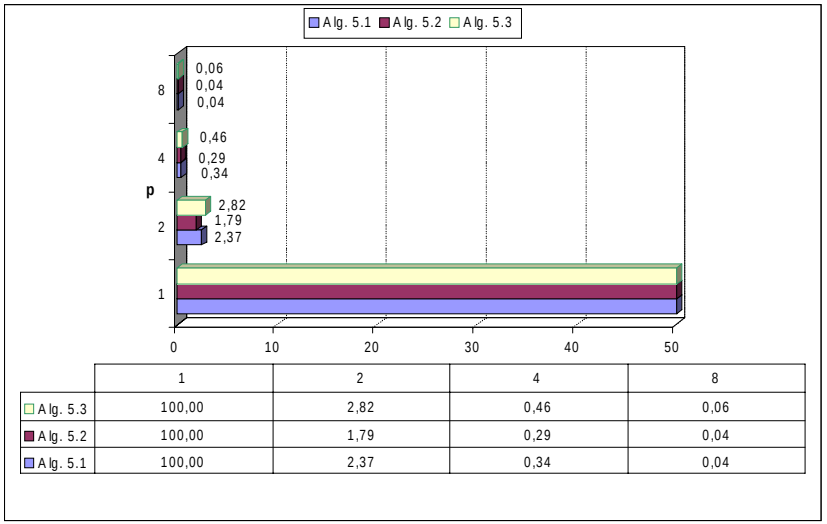
(b) *Cluster de Pentiums*

Figura 5.4: *Valores del speedup en un CRAY T3D y un cluster de Pentiums*



	1	2	4	8
Alg. 5.3	100,00	39,55	20,22	9,61
Alg. 5.2	100,00	30,90	14,08	6,32
Alg. 5.1	100,00	36,18	14,83	6,43

(a) *IBM SP2 switch*

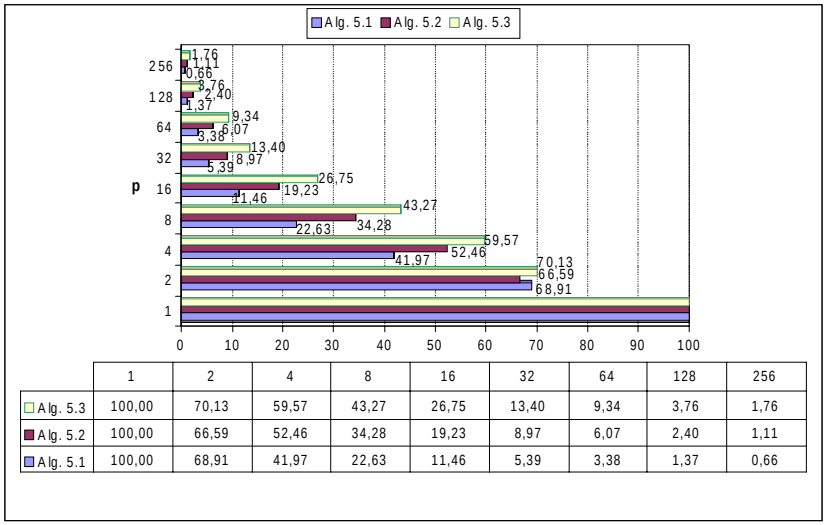


	1	2	4	8
Alg. 5.3	100,00	2,82	0,46	0,06
Alg. 5.2	100,00	1,79	0,29	0,04
Alg. 5.1	100,00	2,37	0,34	0,04

(b) *IBM SP2 ethernet*

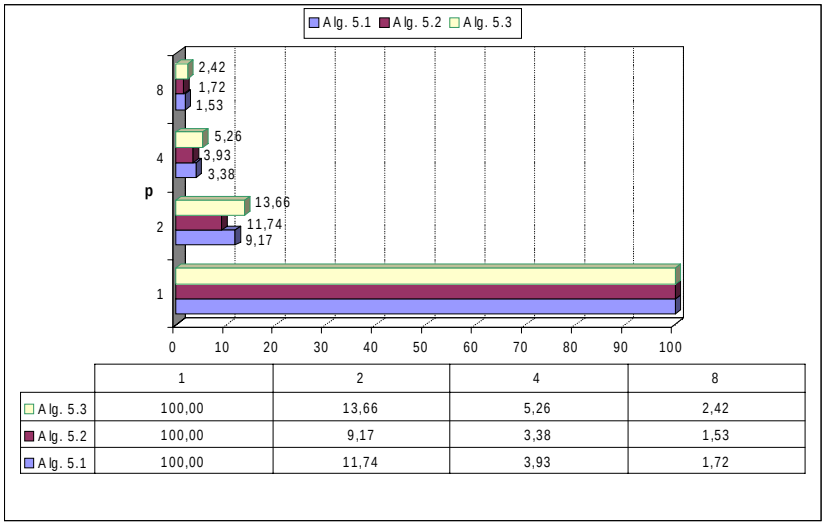
Figura 5.5: *Valores de la eficiencia en un IBM SP2*

Las figuras 5.5 y 5.6 nos muestra los valores obtenidos de la eficiencia para los algoritmos 5.1, 5.2 y 5.3 en las tres máquinas donde se han estudiado los resultados numéricos teóricos de dichos algoritmos.



	1	2	4	8	16	32	64	128	256
Alg. 5.3	100,00	70,13	59,57	43,27	26,75	13,40	9,34	3,76	1,76
Alg. 5.2	100,00	66,59	52,46	34,28	19,23	8,97	6,07	2,40	1,11
Alg. 5.1	100,00	68,91	41,97	22,63	11,46	5,39	3,38	1,37	0,66

(a) *CRAY T3D*



	1	2	4	8
Alg. 5.3	100,00	13,66	5,26	2,42
Alg. 5.2	100,00	9,17	3,38	1,53
Alg. 5.1	100,00	11,74	3,93	1,72

(b) *Cluster de Pentiums*

Figura 5.6: *Valores de la eficiencia en un CRAY T3D y un cluster de Pentiums*