Capítulo 3 Algoritmos *divide y vencerás* basados en la fórmula de Sherman-Morrison

3.1 Introducción

En esta sección presentamos un método general para resolver un sistema lineal

$$A\boldsymbol{x} = \boldsymbol{d} \tag{3.1}$$

siguiendo una estrategia de actualización de rango uno de la matriz A. Empezamos describiendo el método general para, posteriormente en la sección 3.2, estudiar el caso tridiagonal.

Supongamos que podemos obtener una matriz no singular G de manera que

$$A = G + \sum_{i=1}^{m-1} \boldsymbol{u}_i \boldsymbol{v}_i^T \tag{3.2}$$

donde $\boldsymbol{u}_i, \boldsymbol{v}_i \in \mathbb{R}^n$, para $i = 1, 2, \ldots, m-1$.

Sea

$$B_i = B_{i-1} + \boldsymbol{u}_i \boldsymbol{v}_i^T, \qquad i = 1, 2, \ldots, m-1 \tag{3.3}$$

con $B_0 = G$. Claramente $B_{m-1} = A$, por tanto, la única solución $\boldsymbol{x} = A^{-1}\boldsymbol{d}$ del sistema (3.1) puede obtenerse utilizando la fórmula de Sherman-Morrison (véase la sección 1.1 para una descripción más detallada de dicha fórmula), como

$$\boldsymbol{x} = \left(I - \frac{1}{1 + \boldsymbol{v}_{m-1}^T \boxed{B_{m-2}^{-1}\boldsymbol{u}_{m-1}}} \boxed{B_{m-2}^{-1}\boldsymbol{u}_{m-1}} \boldsymbol{v}_{m-1}^T \right) \boxed{B_{m-2}^{-1}\boldsymbol{d}},$$

lo cual implica calcular previamente los vectores $B_{m-2}^{-1}\boldsymbol{d}$ y $B_{m-2}^{-1}\boldsymbol{u}_{m-1}$. Aplicando de nuevo la fórmula de Sherman-Morrison a la ecuación (3.3) para $i = m-2$ obtenemos que

$$B_{m-2}^{-1}\boldsymbol{d} = \left(I - \frac{1}{1 + \boldsymbol{v}_{m-2}^T \boxed{B_{m-3}^{-1}\boldsymbol{u}_{m-2}}} \boxed{B_{m-3}^{-1}\boldsymbol{u}_{m-2}} \boldsymbol{v}_{m-2}^T \right) \boxed{B_{m-3}^{-1}\boldsymbol{d}},$$

$$B_{m-2}^{-1}\boldsymbol{u}_{m-1} = \left(I - \frac{1}{1 + \boldsymbol{v}_{m-2}^T \boxed{B_{m-3}^{-1}\boldsymbol{u}_{m-2}}} \boxed{B_{m-3}^{-1}\boldsymbol{u}_{m-2}} \boldsymbol{v}_{m-2}^T \right) \boxed{B_{m-3}^{-1}\boldsymbol{u}_{m-1}},$$

lo que significa que debemos conocer los vectores $B_{m-3}^{-1}\boldsymbol{d}$, $B_{m-3}^{-1}\boldsymbol{u}_{m-2}$ y $B_{m-3}^{-1}\boldsymbol{u}_{m-1}$.

Siguiendo con este esquema de cálculo, obtenemos que si $i = m-2, m-3, \ldots, 2, 1$, para calcular $B_i^{-1}\boldsymbol{d}$ y $B_i^{-1}\boldsymbol{u}_j$, con $j = i+1, i+2, \ldots, m-1$, necesitamos calcular previamente $B_{i-1}^{-1}\boldsymbol{d}$ y $B_{i-1}^{-1}\boldsymbol{u}_l$, para $l = i, i+1, \ldots, m-1$.

En consecuencia, si por $\boldsymbol{x}_i$ denotamos la solución del sistema $B_i\boldsymbol{x} = \boldsymbol{d}$, es decir $\boldsymbol{x}_i = B_i^{-1}\boldsymbol{d}$ y por $\boldsymbol{w}_{i,k}$ la solución del sistema $B_i\boldsymbol{x} = \boldsymbol{u}_k$, es decir $\boldsymbol{w}_{i,k} = B_i^{-1}\boldsymbol{u}_k$ podemos calcular la solución del sistema (3.1) dada por $\boldsymbol{x} = A^{-1}\boldsymbol{d} = B_{m-1}^{-1}\boldsymbol{d} = \boldsymbol{x}_{m-1}$ utilizando el siguiente algoritmo basado en la fórmula de Sherman-Morrison y los comentarios anteriores.

Algoritmo 3.1 Un algoritmo recursivo general.

1 Inicialización.
Resolver los sistemas $B_0\boldsymbol{x} = \boldsymbol{d}$ y $B_0\boldsymbol{x} = \boldsymbol{u}_i$, para $i = 1, 2, \ldots, m-1$, con el fin de obtener $\boldsymbol{x}_0 = B_0^{-1}\boldsymbol{d}$ y $\boldsymbol{w}_{0,i} = B_0^{-1}\boldsymbol{u}_i$, para $i = 1, 2, \ldots, m-1$.

2 Actualización.
Para $i = 1, 2, \ldots, m-2$ calcular

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} - \frac{\boldsymbol{v}_i^T\boldsymbol{x}_{i-1}}{1+\boldsymbol{v}_i^T\boldsymbol{w}_{i-1,i}}\boldsymbol{w}_{i-1,i} \quad \text{y} \quad \boldsymbol{w}_{i,j} = \boldsymbol{w}_{i-1,j} - \frac{\boldsymbol{v}_i^T\boldsymbol{w}_{i-1,j}}{1+\boldsymbol{v}_i^T\boldsymbol{w}_{i-1,i}}\boldsymbol{w}_{i-1,i} \quad \text{para} \quad j = i+1, i+2, \ldots, m-1.$$

3 Obtención de la solución.
Calcular

$$\boldsymbol{x}_{m-1} = \boldsymbol{x}_{m-2} - \frac{\boldsymbol{v}_{m-1}^T\boldsymbol{x}_{m-2}}{1+\boldsymbol{v}_{m-1}^T\boldsymbol{w}_{m-2,m-1}}\boldsymbol{w}_{m-2,m-1}.$$

3.2 Un algoritmo recursivo para sistemas tridiagonales

A partir de esta sección y en lo que resta del capítulo supondremos que la matriz

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix} \tag{3.4}$$

del sistema (3.1) es tridiagonal, irreducible y diagonal dominante. Con objeto de simplificar la notación, supondremos a partir de ahora que $n = 2^q$ para un cierto entero $q \geq 1$, que

$$G = \begin{bmatrix} G_1 & & & \\ & G_2 & & \\ & & \ddots & \\ & & & G_{2^{q-1}} \end{bmatrix} \quad \text{con} \quad G_i = \begin{bmatrix} \hat{a}_{2i-1} & b_{2i-1} \\ c_{2i} & \hat{a}_{2i} \end{bmatrix}, \quad i = 1, 2, \ldots, 2^{q-1},$$

donde

$$\begin{aligned} \hat{a}_1 &= a_1 \\ \hat{a}_{2i} &= a_{2i} - c_{2i+1}, \quad i = 1, 2, \ldots, 2^{q-1} - 1, \qquad (3.5) \\ \hat{a}_{2i-1} &= a_{2i-1} - b_{2i-2}, \quad i = 2, 3, \ldots, 2^{q-1}, \qquad (3.6) \\ \hat{a}_n &= a_n \end{aligned}$$

y que $\boldsymbol{u}_i = \boldsymbol{e}_{2i} + \boldsymbol{e}_{2i+1}$ y $\boldsymbol{v}_i = c_{2i+1}\boldsymbol{e}_{2i} + b_{2i}\boldsymbol{e}_{2i+1}$, donde $\boldsymbol{e}_{2i}$ y $\boldsymbol{e}_{2i+1}$ son las columnas $2i$-ésima y $(2i+1)$-ésima de la matriz I_n.

Debido a la estructura de las matrices A y G y de los vectores $\boldsymbol{u}_i$ y $\boldsymbol{v}_i$, en esta sección proponemos una modificación del método introducido en la sección 3.1, que puede implementarse utilizando un algoritmo recursivo en el que únicamente se actualizan los bloques de vectores no nulos. Se utiliza la notación $\boldsymbol{x}(k : l)$ con $k \leq l$ para denotar el vector formado por las componentes $k, k+1, \ldots, l$ del vector $\boldsymbol{x}$.

Algoritmo 3.2 Un algoritmo basado en el método de desacoplamiento recursivo para sistemas tridiagonales.

1 Inicialización.

1.1 Para $i = 1, 2, \ldots, 2^{q-1}$, resolver los sistemas

$$G_i \boldsymbol{x}(2i-1:2i) = \boldsymbol{d}(2i-1:2i). \tag{3.7}$$

1.2 Para $i = 1, 2, \ldots, 2^{q-1}-1$, resolver los sistemas

$$G_i \boldsymbol{w}_i(2i-1:2i) = \boldsymbol{u}_i(2i-1:2i) \qquad \text{y} \qquad G_{i+1} \boldsymbol{w}_i(2i+1:2i+2) = \boldsymbol{u}_i(2i+1:2i+2). \tag{3.8}$$

2 Actualización. Para $k = q-2, q-3, \ldots, 2, 1$, y para $i = 1, 2, \ldots, 2^k$, calcular

$$\begin{aligned}\boldsymbol{x}_{(2^{q-k}(i-1)+1:2^{q-k}i)} = & \begin{bmatrix} \boldsymbol{x}_{(2^{q-k}(i-1)+1:2^{q-1-k}(2i-1))} \\ \boldsymbol{x}_{(2^{q-1-k}(2i-1)+1:2^{q-k}i)} \end{bmatrix} \\ & - \frac{\boldsymbol{v}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)^T} \begin{bmatrix} \boldsymbol{x}_{(2^{q-k}(i-1)+1:2^{q-1-k}(2i-1))} \\ \boldsymbol{x}_{(2^{q-1-k}(2i-1)+1:2^{q-k}i)} \end{bmatrix}}{1 + \boldsymbol{v}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)^T} \boldsymbol{w}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)}} \boldsymbol{w}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)},\end{aligned} \tag{3.9}$$

$$\begin{aligned}\boldsymbol{w}_{2^{q-1-k}i(2^{q-k}(i-1)+1:2^{q-k}i)} = & \begin{bmatrix} \boldsymbol{0}_{(1:2^{q-1-k})} \\ \boldsymbol{w}_{2^{q-1-k}i(2^{q-1-k}(2i-1)+1:2^{q-k}i)} \end{bmatrix} \\ & - \frac{\boldsymbol{v}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)^T} \begin{bmatrix} \boldsymbol{0}_{(1:2^{q-1-k})} \\ \boldsymbol{w}_{2^{q-1-k}i(2^{q-1-k}(2i-1)+1:2^{q-k}i)} \end{bmatrix}}{1 + \boldsymbol{v}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)^T} \boldsymbol{w}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)}} \boldsymbol{w}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)},\end{aligned} \tag{3.10}$$

$$\boldsymbol{w}_{2^{q-1-k}(i-1)(2^{q-k}(i-1)+1:2^{q-k}i)} = \begin{bmatrix} \boldsymbol{w}_{2^{q-1-k}(i-1)(2^{q-k}(i-1)+1:2^{q-1-k}(2i-1))} \\ \mathbf{0}_{(1:2^{q-1-k})} \end{bmatrix}$$

$$- \frac{\boldsymbol{v}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)^T} \begin{bmatrix} \boldsymbol{w}_{2^{q-1-k}(i-1)(2^{q-k}(i-1)+1:2^{q-1-k}(2i-1))} \\ \mathbf{0}_{(1:2^{q-1-k})} \end{bmatrix}}{1+\boldsymbol{v}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)^T}\boldsymbol{w}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)}} \boldsymbol{w}_{2^{q-2-k}(2i-1)(2^{q-k}(i-1)+1:2^{q-k}i)}. \quad (3.11)$$

En este paso, suponemos que $\boldsymbol{w}_0$ y $\boldsymbol{w}_{2^{q-1}}$ son vectores nulos.

3 Obtención de la solución. Calcular

$$\boldsymbol{x}_{(1:2^q)} = \begin{bmatrix} \boldsymbol{x}_{(1:2^{q-1})} \\ \boldsymbol{x}_{(2^{q-1}+1:2^q)} \end{bmatrix} - \frac{\boldsymbol{v}_{2^{q-2}(1:2^q)^T} \begin{bmatrix} \boldsymbol{x}_{(1:2^{q-1})} \\ \boldsymbol{x}_{(2^{q-1}+1:2^q)} \end{bmatrix}}{1+\boldsymbol{v}_{2^{q-2}(1:2^q)^T}\boldsymbol{w}_{2^{q-2}(1:2^q)}} \boldsymbol{w}_{2^{q-2}(1:2^q)}. \quad (3.12)$$

El algoritmo 3.2 constituye una modificación del método propuesto por Evans [44], Spaletta y Evans [101] y Mehrmann [90]. Spaletta y Evans [101] aplican la fórmula de Sherman-Morrison para calcular A^{-1} mediante la expresión

$$\left(G + \sum_{i=1}^{m-1} \boldsymbol{u}_i \boldsymbol{v}_i^T\right)^{-1} = G^{-1} - \sum_{i=1}^{m-1} \alpha_i G^{-1} \boldsymbol{u}_i \boldsymbol{v}_i^T G^{-1} \quad (3.13)$$

siendo

$$\alpha_i = \frac{1}{1+\boldsymbol{v}_i^T G^{-1} \boldsymbol{u}_i}, \qquad i = 1, 2, \ldots, m-1.$$

Sin embargo, esta fórmula no es correcta para $m > 2$ como se puede ver en el ejemplo siguiente.

Ejemplo 3.1 *Consideremos el problema de resolver el sistema (3.1) con*

$$A = \begin{bmatrix} 2 & -1 & & & & & & \\ -1 & 2 & -1 & & & & & \\ & -1 & 2 & -1 & & & & \\ & & -1 & 2 & -1 & & & \\ & & & -1 & 2 & -1 & & \\ & & & & -1 & 2 & -1 & \\ & & & & & -1 & 2 & -1 \\ & & & & & & -1 & 2 \end{bmatrix} \qquad y \qquad \boldsymbol{d} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

De acuerdo con la expresión (3.3) y teniendo en cuenta las definiciones de los vectores $\boldsymbol{u}_i$ y $\boldsymbol{v}_i$, podemos escribir la matriz A como

$$A = \begin{bmatrix} G_1 & & & \\ & G_2 & & \\ & & G_3 & \\ & & & G_4 \end{bmatrix} + \boldsymbol{u}_1\boldsymbol{v}_1^T + \boldsymbol{u}_2\boldsymbol{v}_2^T + \boldsymbol{u}_3\boldsymbol{v}_3^T,$$

siendo

$$\boldsymbol{u}_1 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{u}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{u}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad y \qquad \boldsymbol{v}_1 = \begin{bmatrix} 0 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{v}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{v}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ 0 \end{bmatrix}.$$

Entonces

$$G = \begin{bmatrix} 2 & -1 & & & & & & \\ -1 & 3 & & & & & & \\ & & 3 & -1 & & & & \\ & & -1 & 3 & & & & \\ & & & & 3 & -1 & & \\ & & & & -1 & 3 & & \\ & & & & & & 3 & -1 \\ & & & & & & -1 & 2 \end{bmatrix} \qquad y \qquad G^{-1} - \sum_{i=1}^{3} \alpha_i G^{-1} \boldsymbol{u}_i \boldsymbol{v}_i G^{-1} = \begin{bmatrix} \frac{7}{9} & \frac{5}{9} & \frac{1}{3} & \frac{1}{9} & 0 & 0 & 0 & 0 \\ \frac{5}{9} & \frac{10}{9} & \frac{2}{3} & \frac{2}{9} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{2}{3} & \frac{17}{16} & \frac{25}{48} & \frac{3}{16} & \frac{1}{16} & 0 & 0 \\ \frac{1}{9} & \frac{2}{9} & \frac{25}{48} & \frac{145}{144} & \frac{9}{16} & \frac{3}{16} & 0 & 0 \\ 0 & 0 & \frac{3}{16} & \frac{9}{16} & \frac{145}{144} & \frac{25}{48} & \frac{2}{9} & \frac{1}{9} \\ 0 & 0 & \frac{1}{16} & \frac{3}{16} & \frac{25}{48} & \frac{17}{16} & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & \frac{2}{9} & \frac{2}{3} & \frac{10}{9} & \frac{5}{9} \\ 0 & 0 & 0 & 0 & \frac{1}{9} & \frac{1}{3} & \frac{5}{9} & \frac{7}{9} \end{bmatrix}.$$

Sin embargo,

$$A^{-1} = \begin{bmatrix} \frac{8}{9} & \frac{7}{9} & \frac{2}{3} & \frac{5}{9} & \frac{4}{9} & \frac{1}{3} & \frac{2}{9} & \frac{1}{9} \\ \frac{7}{9} & \frac{14}{9} & \frac{4}{3} & \frac{10}{9} & \frac{8}{9} & \frac{2}{3} & \frac{4}{9} & \frac{2}{9} \\ \frac{2}{3} & \frac{4}{3} & 2 & \frac{5}{3} & \frac{4}{3} & 1 & \frac{2}{3} & \frac{1}{3} \\ \frac{5}{9} & \frac{10}{9} & \frac{5}{3} & \frac{20}{9} & \frac{16}{9} & \frac{4}{3} & \frac{8}{9} & \frac{4}{9} \\ \frac{4}{9} & \frac{8}{9} & \frac{4}{3} & \frac{16}{9} & \frac{20}{9} & \frac{5}{3} & \frac{10}{9} & \frac{5}{9} \\ \frac{1}{3} & \frac{2}{3} & 1 & \frac{4}{3} & \frac{5}{3} & 2 & \frac{4}{3} & \frac{2}{3} \\ \frac{2}{9} & \frac{4}{9} & \frac{2}{3} & \frac{8}{9} & \frac{10}{9} & \frac{4}{3} & \frac{14}{9} & \frac{7}{9} \\ \frac{1}{9} & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & \frac{5}{9} & \frac{2}{3} & \frac{7}{9} & \frac{8}{9} \end{bmatrix}.$$

La modificación que introduce el algoritmo 3.2 respecto del método que proponen Evans [44], Spaletta y Evans [101] y Mehrmann [90], se basa en una reducción del número de actualizaciones que se realizan, ya que únicamente se consideran las componentes no nulas de los vectores que son actualizados. A continuación resolvemos detalladamente un ejemplo donde ponemos de manifiesto el funcionamiento de dicho algoritmo.

Ejemplo 3.2 *Consideremos el problema de resolver el sistema (3.1), donde A y* $\boldsymbol{d}$ *tienen la misma estructura que en el ejemplo 3.1 pero ahora* $n = 16$.

Utilizando las expresiones (3.5) y (3.6) podemos determinar los bloques diagonales G_i*, para* $i = 1, 2, \ldots, 8$*, de la matriz G. Así, tendremos que*

$$G_1 = \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}, \qquad G_i = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}, \quad \textit{para} \quad i = 2, 3 \ldots, 7 \qquad \textit{y} \qquad G_8 = \begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}.$$

Además, para $i = 1, 2, \ldots, 7$*,*

$$\boldsymbol{u}_i = \boldsymbol{e}_{2i} + \boldsymbol{e}_{2i+1} \qquad \textit{y} \qquad \boldsymbol{v}_i = -\boldsymbol{e}_{2i} - \boldsymbol{e}_{2i+1},$$

donde $\boldsymbol{e}_{2i}$ *y* $\boldsymbol{e}_{2i+1}$ *son las columnas* $2i$*-ésima y* $(2i+1)$*-ésima de la matriz* I_{16}.

Con estos datos estamos ya en condiciones de aplicar el algoritmo 3.2.

1 ***Inicialización****.*

1.1 Resolvemos los sistemas

$$\begin{array}{llll} G_1\boldsymbol{x}_{(1:2)} = \boldsymbol{d}_{(1:2)}, & G_2\boldsymbol{x}_{(3:4)} = \boldsymbol{d}_{(3:4)}, & G_3\boldsymbol{x}_{(5:6)} = \boldsymbol{d}_{(5:6)}, & G_4\boldsymbol{x}_{(7:8)} = \boldsymbol{d}_{(7:8)}, \\ G_5\boldsymbol{x}_{(9:10)} = \boldsymbol{d}_{(9:10)}, & G_6\boldsymbol{x}_{(11:12)} = \boldsymbol{d}_{(11:12)}, & G_7\boldsymbol{x}_{(13:14)} = \boldsymbol{d}_{(13:14)}, & G_8\boldsymbol{x}_{(15:16)} = \boldsymbol{d}_{(15:16)}, \end{array}$$

cuyas soluciones son

$$\boldsymbol{x}_{(1:2)} = \begin{bmatrix} 4/5 \\ 3/5 \end{bmatrix}, \qquad \boldsymbol{x}_{(3:4)} = \boldsymbol{x}_{(5:6)} = \boldsymbol{x}_{(7:8)} = \boldsymbol{x}_{(9:10)} = \boldsymbol{x}_{(11:12)} = \boldsymbol{x}_{(13:14)} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \qquad y \qquad \boldsymbol{x}_{(15:16)} = \begin{bmatrix} 3/5 \\ 4/5 \end{bmatrix}.$$

Nótese cómo la estructura particular de la matriz de coeficientes y del vector de términos independiente hace que muchos cálculos a lo largo del algoritmo sean repetitivos y coincidentes.

1.2 Debemos resolver ahora los sistemas

$$\begin{array}{llll} & G_2\boldsymbol{w}_{1(3:4)} = \boldsymbol{u}_{1(3:4)}, & G_3\boldsymbol{w}_{2(5:6)} = \boldsymbol{u}_{2(5:6)}, & G_4\boldsymbol{w}_{3(7:8)} = \boldsymbol{u}_{3(7:8)}, \\ G_1\boldsymbol{w}_{1(1:2)} = \boldsymbol{u}_{1(1:2)}, & G_2\boldsymbol{w}_{2(3:4)} = \boldsymbol{u}_{2(3:4)}, & G_3\boldsymbol{w}_{3(5:6)} = \boldsymbol{u}_{3(5:6)}, & G_4\boldsymbol{w}_{4(7:8)} = \boldsymbol{u}_{4(7:8)}, \end{array}$$

$$\begin{array}{llll} G_5\boldsymbol{w}_{4(9:10)} = \boldsymbol{u}_{4(9:10)}, & G_6\boldsymbol{w}_{5(11:12)} = \boldsymbol{u}_{5(11:12)}, & G_7\boldsymbol{w}_{6(13:14)} = \boldsymbol{u}_{6(13:14)}, & G_8\boldsymbol{w}_{7(15:16)} = \boldsymbol{u}_{7(15:16)}, \\ G_5\boldsymbol{w}_{5(9:10)} = \boldsymbol{u}_{5(9:10)}, & G_6\boldsymbol{w}_{6(11:12)} = \boldsymbol{u}_{6(11:12)}, & G_7\boldsymbol{w}_{7(13:14)} = \boldsymbol{u}_{7(13:14)}. & \end{array}$$

Como ya se comentó anteriormente, la estructura del sistema que estamos resolviendo provoca que muchos vectores solución de los

sistemas anteriores sean iguales. Así, es fácil comprobar que

$$\boldsymbol{w}_1(1:2) = \begin{bmatrix} 1/5 \\ 2/5 \end{bmatrix},$$

$$\boldsymbol{w}_1(3:4) = \boldsymbol{w}_2(5:6) = \boldsymbol{w}_3(7:8) = \boldsymbol{w}_4(9:10) = \boldsymbol{w}_5(11:12) = \boldsymbol{w}_6(13:14) = \begin{bmatrix} 3/8 \\ 1/8 \end{bmatrix},$$

$$\boldsymbol{w}_2(3:4) = \boldsymbol{w}_3(5:6) = \boldsymbol{w}_4(7:8) = \boldsymbol{w}_5(9:10) = \boldsymbol{w}_6(11:12) = \boldsymbol{w}_7(13:14) = \begin{bmatrix} 1/8 \\ 3/8 \end{bmatrix},$$

$$\boldsymbol{w}_7(15:16) = \begin{bmatrix} 2/5 \\ 1/5 \end{bmatrix}.$$

2 ***Actualización****.*

En nuestro ejemplo, $n = 2^4$*, por lo que* $q = 4$*. Así, existen dos pasos de actualización para* $k = q - 2 = 2$ *y* $k = q - 3 = 1$*.*

Para $k = 2$*, tendremos que* $i = 1, 2, 3, 4$ *por lo que para* $i = 1$*, se calculan los vectores* $\boldsymbol{x}_{(1:4)}$ *y* $\boldsymbol{w}_{2(1:4)}$ *únicamente, ya que el vector* $\boldsymbol{w}_0$ *es nulo. Se utilizan las expresiones (3.9), (3.10) y (3.11) que aparecen en el algoritmo 3.2. Así,*

$$\boldsymbol{x}(1:4) = \begin{bmatrix} \boldsymbol{x}(1:2) \\ \boldsymbol{x}(3:4) \end{bmatrix} - \frac{\boldsymbol{v}_1(1:4)^T \begin{bmatrix} \boldsymbol{x}(1:2) \\ \boldsymbol{x}(3:4) \end{bmatrix}}{1 + \boldsymbol{v}_1(1:4)^T \boldsymbol{w}_1(1:4)} \boldsymbol{w}_1(1:4) = \begin{bmatrix} 4/5 \\ 3/5 \\ 1/2 \\ 1/2 \end{bmatrix} + \frac{44}{9} \begin{bmatrix} 4/5 \\ 3/5 \\ 1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 16/9 \\ 23/9 \\ 7/3 \\ 10/9 \end{bmatrix},$$

$$\boldsymbol{w}_2(1:4) = \begin{bmatrix} \mathbf{0}(1:2) \\ \boldsymbol{w}_2(3:4) \end{bmatrix} - \frac{\boldsymbol{v}_1(1:4)^T \begin{bmatrix} \mathbf{0}(1:2) \\ \boldsymbol{w}_2(3:4) \end{bmatrix}}{1 + \boldsymbol{v}_1(1:4)^T \boldsymbol{w}_1(1:4)} \boldsymbol{w}_1(1:4) = \begin{bmatrix} 0 \\ 0 \\ 1/8 \\ 3/8 \end{bmatrix} + \frac{40}{72} \begin{bmatrix} 1/5 \\ 2/5 \\ 3/8 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 1/9 \\ 2/9 \\ 1/3 \\ 4/9 \end{bmatrix},$$

Para $i = 2$*, se calculan los vectores* $\boldsymbol{x}(5:8)$*,* $\boldsymbol{w}_4(5:8)$ *y* $\boldsymbol{w}_2(5:8)$ *aplicando las mismas expresiones que en el caso* $i = 1$*.*

$$\boldsymbol{x}(5:8) = \begin{bmatrix} \boldsymbol{x}(5:6) \\ \boldsymbol{x}(7:8) \end{bmatrix} - \frac{\boldsymbol{v}_3(5:8)^T \begin{bmatrix} \boldsymbol{x}(5:6) \\ \boldsymbol{x}(7:8) \end{bmatrix}}{1 + \boldsymbol{v}_3(5:8)^T \boldsymbol{w}_3(5:8)} \boldsymbol{w}_3(5:8) = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} + 4 \begin{bmatrix} 1/8 \\ 3/8 \\ 3/8 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \end{bmatrix},$$

$$\boldsymbol{w}_4(5:8) = \begin{bmatrix} \mathbf{0}(1:2) \\ \boldsymbol{w}_4(7:8) \end{bmatrix} - \frac{\boldsymbol{v}_3(5:8)^T \begin{bmatrix} \mathbf{0}(1:2) \\ \boldsymbol{w}_4(7:8) \end{bmatrix}}{1 + \boldsymbol{v}_3(5:8)^T \boldsymbol{w}_3(5:8)} \boldsymbol{w}_3(5:8) = \begin{bmatrix} 0 \\ 0 \\ 1/8 \\ 3/8 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1/8 \\ 3/8 \\ 3/8 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 1/16 \\ 3/16 \\ 5/16 \\ 7/16 \end{bmatrix},$$

$$\boldsymbol{w}_2(5:8) = \begin{bmatrix} \boldsymbol{w}_2(5:6) \\ \mathbf{0}(1:2) \end{bmatrix} - \frac{\boldsymbol{v}_3(5:8)^T \begin{bmatrix} \boldsymbol{w}_2(5:6) \\ \mathbf{0}(1:2) \end{bmatrix}}{1 + \boldsymbol{v}_3(5:8)^T \boldsymbol{w}_3(5:8)} \boldsymbol{w}_3(5:8) = \begin{bmatrix} 3/8 \\ 1/8 \\ 0 \\ 0 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1/8 \\ 3/8 \\ 3/8 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 7/16 \\ 5/16 \\ 3/16 \\ 1/16 \end{bmatrix}.$$

Para $i = 3$, se calculan los vectores $\boldsymbol{x}_{(9:12)}$, $\boldsymbol{w}_{6(9:12)}$ y $\boldsymbol{w}_{4(9:12)}$ realizando cálculos similares a los casos anteriores.

$$\boldsymbol{x}_{(9:12)} = \begin{bmatrix} \boldsymbol{x}_{(9:10)} \\ \boldsymbol{x}_{(11:12)} \end{bmatrix} - \frac{\boldsymbol{v}_{5(9:12)}{}^T \begin{bmatrix} \boldsymbol{x}_{(9:10)} \\ \boldsymbol{x}_{(11:12)} \end{bmatrix}}{1 + \boldsymbol{v}_{5(9:12)}{}^T \boldsymbol{w}_{5(9:12)}} \boldsymbol{w}_{5(9:12)} = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} + 4 \begin{bmatrix} 1/8 \\ 3/8 \\ 3/8 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \end{bmatrix},$$

$$\boldsymbol{w}_{6(9:12)} = \begin{bmatrix} \boldsymbol{0}_{(1:2)} \\ \boldsymbol{w}_{6(11:12)} \end{bmatrix} - \frac{\boldsymbol{v}_{5(9:12)}{}^T \begin{bmatrix} \boldsymbol{0}_{(1:2)} \\ \boldsymbol{w}_{6(11:12)} \end{bmatrix}}{1 + \boldsymbol{v}_{5(9:12)}{}^T \boldsymbol{w}_{5(9:12)}} \boldsymbol{w}_{5(9:12)} = \begin{bmatrix} 0 \\ 0 \\ 1/8 \\ 3/8 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1/8 \\ 3/8 \\ 3/8 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 1/16 \\ 3/16 \\ 5/16 \\ 7/16 \end{bmatrix},$$

$$\boldsymbol{w}_{4(9:12)} = \begin{bmatrix} \boldsymbol{w}_{4(9:10)} \\ \boldsymbol{0}_{(1:2)} \end{bmatrix} - \frac{\boldsymbol{v}_{5(9:12)}{}^T \begin{bmatrix} \boldsymbol{w}_{4(9:10)} \\ \boldsymbol{0}_{(1:2)} \end{bmatrix}}{1 + \boldsymbol{v}_{5(9:12)}{}^T \boldsymbol{w}_{5(9:12)}} \boldsymbol{w}_{5(9:12)} = \begin{bmatrix} 3/8 \\ 1/8 \\ 0 \\ 0 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1/8 \\ 3/8 \\ 3/8 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 7/16 \\ 5/16 \\ 3/16 \\ 1/16 \end{bmatrix}.$$

Finalmente, para $i = 4$, se calculan los vectores $\boldsymbol{x}_{(13:16)}$ y $\boldsymbol{w}_{6(13:16)}$ de forma análoga a las anteriores.

$$\boldsymbol{x}_{(13:16)} = \begin{bmatrix} \boldsymbol{x}_{(13:14)} \\ \boldsymbol{x}_{(15:16)} \end{bmatrix} - \frac{\boldsymbol{v}_{7(13:16)}{}^T \begin{bmatrix} \boldsymbol{x}_{(13:14)} \\ \boldsymbol{x}_{(15:16)} \end{bmatrix}}{1 + \boldsymbol{v}_{7(13:16)}{}^T \boldsymbol{w}_{7(13:16)}} \boldsymbol{w}_{7(13:16)} = \begin{bmatrix} 1/2 \\ 1/2 \\ 3/5 \\ 4/5 \end{bmatrix} + \frac{44}{9} \begin{bmatrix} 1/8 \\ 3/8 \\ 2/5 \\ 1/5 \end{bmatrix} = \begin{bmatrix} 10/9 \\ 7/3 \\ 23/9 \\ 16/9 \end{bmatrix},$$

$$\boldsymbol{w}_6(13:16) = \begin{bmatrix} \boldsymbol{w}_6(13:14) \\ \mathbf{0}_{(1:2)} \end{bmatrix} - \frac{\boldsymbol{v}_7(13:16)^T \begin{bmatrix} \boldsymbol{w}_6(13:14) \\ \mathbf{0}_{(1:2)} \end{bmatrix}}{1 + \boldsymbol{v}_7(13:16)^T \boldsymbol{w}_7(13:16)} \boldsymbol{w}_7(13:16) = \begin{bmatrix} 3/8 \\ 1/8 \\ 0 \\ 0 \end{bmatrix} + \frac{40}{72} \begin{bmatrix} 1/8 \\ 3/8 \\ 2/5 \\ 1/5 \end{bmatrix} = \begin{bmatrix} 4/9 \\ 1/3 \\ 2/9 \\ 1/9 \end{bmatrix}.$$

Una vez cumplida la actualización para $k = 2$, procedemos a la actualización para $k = 1$, por lo que $i = 1, 2$. Para $i = 1$ se calculan los vectores $\boldsymbol{x}_{(1:8)}$ y $\boldsymbol{w}_{4(1:8)}$ utilizando de nuevo las expresiones (3.9), (3.10) y (3.11).

$$\boldsymbol{x}(1:8) = \begin{bmatrix} \boldsymbol{x}(1:4) \\ \boldsymbol{x}(5:8) \end{bmatrix} - \frac{\boldsymbol{v}_2(1:8)^T \begin{bmatrix} \boldsymbol{x}(1:4) \\ \boldsymbol{x}(5:8) \end{bmatrix}}{1 + \boldsymbol{v}_2(1:8)^T \boldsymbol{w}_2(1:8)} \boldsymbol{w}_2(1:8) = \begin{bmatrix} 64/17 & 111/17 & 141/17 & 154/17 & 150/17 & 129/17 & 91/17 & 36/17 \end{bmatrix}^T,$$

$$\boldsymbol{w}_4(1:8) = \begin{bmatrix} \mathbf{0}_{(1:4)} \\ \boldsymbol{w}_4(5:8) \end{bmatrix} - \frac{\boldsymbol{v}_2(1:8)^T \begin{bmatrix} \mathbf{0}_{(1:4)} \\ \boldsymbol{w}_4(5:8) \end{bmatrix}}{1 + \boldsymbol{v}_2(1:8)^T \boldsymbol{w}_2(1:8)} \boldsymbol{w}_2(1:8) = \begin{bmatrix} 1/17 & 2/17 & 3/17 & 4/17 & 5/17 & 6/17 & 7/17 & 8/17 \end{bmatrix}^T.$$

Para $i = 2$ se calculan, como en el caso anterior, los vectores $\boldsymbol{x}_{(9:16)}$ y $\boldsymbol{w}_{4(9:16)}$.

$$\boldsymbol{x}(9:16) = \begin{bmatrix} \boldsymbol{x}(9:12) \\ \boldsymbol{x}(13:16) \end{bmatrix} - \frac{\boldsymbol{v}_6(9:16)^T \begin{bmatrix} \boldsymbol{x}(9:12) \\ \boldsymbol{x}(13:16) \end{bmatrix}}{1 + \boldsymbol{v}_6(9:16)^T \boldsymbol{w}_6(9:16)} \boldsymbol{w}_6(9:16) = \begin{bmatrix} 36/17 & 91/17 & 129/17 & 150/17 & 154/17 & 141/17 & 111/17 & 64/17 \end{bmatrix}^T,$$

$$\boldsymbol{w}_4(9:16) = \begin{bmatrix} \boldsymbol{w}_4(9:12) \\ \mathbf{0}_{(1:4)} \end{bmatrix} - \frac{\boldsymbol{v}_6(9:16)^T \begin{bmatrix} \boldsymbol{w}_4(9:12) \\ \mathbf{0}_{(1:4)} \end{bmatrix}}{1 + \boldsymbol{v}_6(9:16)^T \boldsymbol{w}_6(9:16)} \boldsymbol{w}_6(9:16) = \begin{bmatrix} 8/17 & 7/17 & 6/17 & 5/17 & 4/17 & 3/17 & 2/17 & 1/17 \end{bmatrix}^T.$$

3 ***Obtención de la solución.***

Una vez terminada la fase de actualización, ya podemos calcular el vector $\boldsymbol{x}(1:16)$ *mediante la expresión (3.12), que ya es la solución del sistema inicial.*

$$\boldsymbol{x}(1:16) = \begin{bmatrix} \boldsymbol{x}(1:8) \\ \boldsymbol{x}(9:16) \end{bmatrix} - \frac{\boldsymbol{v}_4(1:16)^T \begin{bmatrix} \boldsymbol{x}(1:8) \\ \boldsymbol{x}(9:16) \end{bmatrix}}{1 + \boldsymbol{v}_4(1:16)^T \boldsymbol{w}_4(1:16)} \boldsymbol{w}_4(1:16) = \begin{bmatrix} 7/2 & 6 & 15/2 & 8 & 15/2 & 6 & 7/2 & 0 & 0 & 7/2 & 6 & 15/2 & 8 & 15/2 & 6 & 7/2 \end{bmatrix}^T.$$

Como se desprende del ejemplo 3.2, en la fase de inicialización, se resuelven una serie de sistemas de tamaño 2×2. Estos sistemas se utilizan en la fase de actualización para actualizar los bloques de los vetores $\boldsymbol{x}_i$ y $\boldsymbol{w}_i$ que van duplicando el número de sus componentes y que nos permiten en la última fase, calcular la solución general.

En la figura 3.1 se visualizan las componentes de los vectores $\boldsymbol{w}_i$, para $i = 1, 2, \ldots, 7$, involucradas en la inicialización y actualización del algoritmo 3.2 para el ejemplo 3.2. En dicha figura no aparecen las componentes que se actualizan del vector solución $\boldsymbol{x}$ debido a que su actualización no ofrece ninguna dificultad de tipo conceptual. En la fase de inicialización, se calculan todas las componentes del vector $\boldsymbol{x}$ por bloques de tamaño dos, resolviendo sistemas de tamaño 2×2. En los pasos siguientes de actualización se duplica cada vez el tamaño de los bloques que se actualizan hasta llegar a la obtención de la solución final de acuerdo con la expresión (3.12).

En la fase de inicialización observamos que todos los sistemas que se resuelven son de tamaño 2×2. Como se observa en la figura 3.1, el resultado de esta fase es que hemos calculado cuatro componentes de cada uno de los vectores $\boldsymbol{w}_i$, para $i = 1, 2, \ldots, 7$. Esto significa que estos vectores tienen nulas todas sus componentes salvo a lo sumo estas cuatro, que son las que se han actualizado.

En el primer paso de la fase de actualización, es decir, para $k = 2$, combinamos los resultados de la etapa de inicialización para actualizar bloques de cuatro componentes de los vectores $\boldsymbol{w}_2$, $\boldsymbol{w}_4$ y $\boldsymbol{w}_6$, de manera que después de dicha fase de actualización cada uno de estos vectores tiene nulas todas sus componentes salvo a lo sumo estas ocho componentes que se han actualizado. La figura 3.1 nos muestra las componentes de la fase 1 que son necesarias en cada una de las actualizaciones que se llevan a cabo en cada fase. Se utilizan

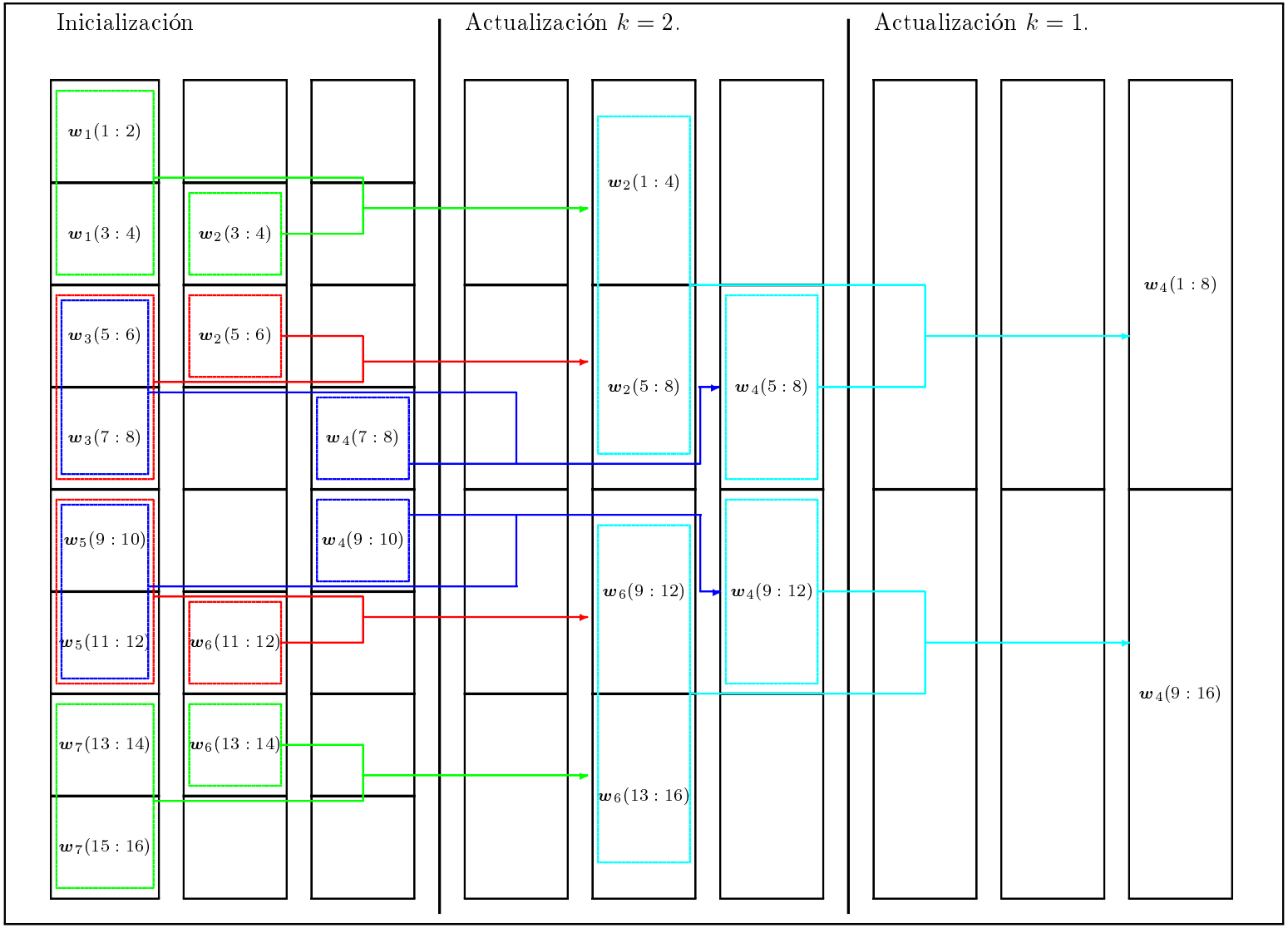


Figura 3.1: *Inicialización y actualización del algoritmo 3.2 para el ejemplo 3.2.*

colores distintos para visualizar de una forma más clara qué componentes son necesarias en cada caso. Así, por ejemplo se observa que para calcular la actualización del vector $\boldsymbol{w}_2(1:4)$ utilizamos las componentes actualizadas en la fase anterior de los vectores $\boldsymbol{w}_2$ y $\boldsymbol{w}_1$, calculadas previamente, es decir, $\boldsymbol{w}_2(3:4)$ y $\boldsymbol{w}_1(1:4)$.

Para $k = 1$ únicamente se actualizan las componentes del vector $\boldsymbol{w}_4$ en dos bloques de ocho componentes. Notemos que ahora para actualizar el bloque $\boldsymbol{w}_4(1:8)$ se utilizan los bloques calculados previamente $\boldsymbol{w}_2(1:8)$ y $\boldsymbol{w}_4(5:8)$, como se aprecia en la figura 3.1. El resultado de esta actualización nos permite obtener la solución final. A lo largo de este proceso queda de manifiesto una característica esencial de esta implementación: únicamente actualizamos bloques de vectores a partir de bloques de vectores no nulos calculados previamente a través de un proceso recursivo.

En la figura 3.1 se pone de manifiesto también que, debido a las características del método, los cálculos que se realizan en las distintas fases conducentes a la obtención del vector $\boldsymbol{w}_4$ pueden desarrollarse de forma independiente unos de otros. Si observamos detenidamente las relaciones gráficas que se establecen en la figura 3.1 entre las distintas componentes, podemos llegar a la conclusión de que el cálculo de las componentes $\boldsymbol{w}_4(1:8)$ que se realiza en el paso de actualización para $k = 1$ es independiente del cálculo de las componentes $\boldsymbol{w}_2(9:16)$. Esto nos lleva a que si consideramos que disponemos de dos procesadores, P_0 y P_1, cada uno de ellos puede desarrollar perfectamente en paralelo los cálculos conducentes a la obtención de $\boldsymbol{w}_4(1:8)$ y $\boldsymbol{w}_4(1:16)$, respectivamente. Para la obtención de la solución final ya es necesaria una comunicación entre procesadores ya que estos dos vectores deben encontrarse en el mismo procesador para calcular $\boldsymbol{x}(1:16)$ mediante la expresión (3.12). Notemos que si en lugar de disponer de dos procesadores dispusiéramos de cuatro procesadores, la comunicación de elementos sería necesaria antes de calcular $\boldsymbol{w}_4(1:8)$, ya que el bloque $\boldsymbol{w}_2(1:4)$ estaría en el procesador P_0 y los bloques $\boldsymbol{w}_2(5:8)$ y $\boldsymbol{w}_4(5:8)$ se encontrarían en el procesador P_1, por lo que su cálculo requiere una comunicación previa.

Dedicamos el resto de la sección al estudio del coste computacional del algoritmo 3.2. Para ello calculamos los costes de cada una de las tres fases que componen dicho algoritmo.

Comenzamos analizando el coste de la fase 1. Sabemos que, para $i = 1, 2, \ldots, 2^{q-1}$, las matrices G_i son de tamaño 2×2, por lo que para calcular los elementos de su diagonal mediante las expresiones (3.5) y (3.6) son necesarios $2(2^{q-1} - 1)$ flops. Ahora, la resolución de

los sistemas involucrados supone un total de $7 \cdot 2^{q-1} + 6$ flops. Por lo tanto, el coste total de esta fase es de

$$2(2^{q-1} - 1) + 7 \cdot 2^{q-1} + 6 = 9 \cdot 2^{q-1} + 4 \quad \text{flops.} \tag{3.14}$$

En cuanto al coste computacional de la fase de actualización, éste puede dividirse en varias partes. Calculamos primeramente el coste de cada una de las operaciones que se realizan en cada paso de la actualización.

- El cálculo del denominador de las expresiones (3.9), (3.10) y (3.11) requiere 4 flops. El cálculo del numerador de la expresión (3.9) únicamente requiere 3 flops, mientras que el cálculo de los numeradores de las expresiones (3.10) y (3.11) requieren 1 flop respectivamente.
- El cálculo del vector $\boldsymbol{x}_{(2^{q-k}(i-1)+1\,:\,2^{q-k}i)}$ supone $1 + 2^{q-k+1}$ flops.
- Para calcular los vectores $\boldsymbol{w}_{2^{q-1-k}i\,(2^{q-k}(i-1)+1\,:\,2^{q-k}i)}$ y $\boldsymbol{w}_{2^{q-1-k}(i-1)\,(2^{q-k}(i-1)+1\,:\,2^{q-k}i)}$ necesitamos $2(1 + 3 \cdot 2^{q-k-1})$ flops.

Ahora, para calcular el coste total, sumamos los costes de cada una de las etapas que se realizan en esta fase de actualización, obteniendo

$$\sum_{k=1}^{q-2} \sum_{i=1}^{2^k} \left[10 + 2^{q-k+1} + 2(1 + 3 \cdot 2^{q-k-1})\right] = 12(2^{q-1} - 2) + 5 \cdot 2^q(q-2) \quad \text{flops.} \tag{3.15}$$

Finalmente, el coste computacional de la obtención del vector solución utilizando la expresión (3.12) es de

$$8 + 2^{q+1} \quad \text{flops.} \tag{3.16}$$

Consecuentemente, si sumamos las expresiones (3.14), (3.15) y (3.16) obtenemos el coste computacional del algoritmo 3.2, que es de

$$(5 + 10q) \cdot 2^{q-1} - 20 \quad \text{flops.} \tag{3.17}$$

3.3 Un algoritmo BSP de tipo *fan-in*

3.3.1 Descripción del algoritmo

Supondremos en adelante que el número de procesadores disponibles es $p = 2^m$, para un cierto entero $m \geq 1$, ya que si $m = 0$ tendremos el algoritmo secuencial. Las características del método expuestas en la sección 3.2 sugieren la implementación del mismo utilizando un algoritmo paralelo del tipo *fan-in*. Se describen a continuación las características esenciales del proceso de cálculo y de comunicación que requiere un algoritmo de este tipo.

Inicialmente, la responsabilidad de la realización de las operaciones de la fase 1 del algoritmo 3.2 se divide entre todos los procesadores. Después de estos cálculos, cada procesador P_j, para $j = 0, 1, \ldots, 2^m - 1$, es responsable de la actualización de los bloques de vectores que se especifican en la fase 2 del algoritmo 3.2. Al final de esta fase, se obtienen los vectores

$$\boldsymbol{x}_{(2^{q-m}j+1:2^{q-m}j+2^{q-m}i)}, \quad \boldsymbol{w}_{2^{q-1-m}(j+1)(2^{q-m}j+1:2^{q-m}j+2^{q-m}i)} \quad \text{y} \quad \boldsymbol{w}_{2^{q-1-m}(j)(2^{q-m}j+1:2^{q-m}j+2^{q-m}i)}. \tag{3.18}$$

Nótese que en todo el proceso de cálculo conducente a la obtención de los vectores (3.18) no es necesario realizar ninguna comunicación de elementos entre los procesadores, debido a que cada procesador tiene los elementos que necesita para efectuar los cálculos. En total, se llevan a cabo $q - m$ pasos de actualización, en los que cada procesador calcula sucesivamente bloques de vectores utilizando las expresiones (3.9), (3.10) y (3.11) de forma recursiva.

Consideramos inicialmente el caso en que $m = 1$, es decir, cuando $p = 2$. En este caso, cada procesador desarrolla los primeros $q - 1$ pasos de la fase 2 de forma independiente. Posteriormente, el procesador P_1 comunica al procesador P_0 los bloques de vectores que permiten al procesador P_0 obtener la solución final. En consecuencia, únicamente se produce una comunicación de datos entre los dos procesadores, por lo que el modelo de comunicación que utilicemos es indiferente ya que disponemos únicamente de dos procesadores. Así, el siguiente algoritmo resume las características expuestas anteriormente para el caso en que $m = 1$.

Algoritmo 3.3 Un algoritmo BSP de tipo *fan-in* para sistemas tridiagonales cuando $p = 2$.

Superpaso 1

El procesador P_0 envía al procesador P_1 los elementos $a_{2^{q-1}j+i}$, $b_{2^{q-1}j+i-1}$ y $c_{2^{q-1}j+i+1}$, para $i = 1, 2, \ldots, 2^{q-1}$, suponiendo que $c_{n+1} = 0$.

Superpaso 2

1 En el procesador P_j, para $j = 0, 1$,

1.1 calcular $\hat{a}_{2^{q-1}j+i}$, para $i = 1, 2, \ldots, 2^{q-1}$, utilizando las expresiones (3.5) y (3.6),

1.2 calcular $\boldsymbol{x}(2^{q-1}j + 2i - 1 : 2^{q-1}j + 2i)$, para $i = 1, 2, \ldots, 2^{q-2}$ utilizando la expresión (3.7),

1.3 calcular $\boldsymbol{w}_{2^{q-2}j+i}(2^{q-1}j + 2i - 1 : 2^{q-1}j + 2i)$ y $\boldsymbol{w}_{2^{q-2}j+i-1}(2^{q-1}j + 2i - 1 : 2^{q-1}j + 2i)$, para $i = 1, 2, \ldots, 2^{q-2}$, utilizando las expresiones (3.8) y suponiendo que $\boldsymbol{w}_0$ y $\boldsymbol{w}_{2^q}$ son vectores nulos,

1.4 para $k = q-2, q-3, \ldots, 1$, y para $i = 1, 2, \ldots, 2^{k-1}$ calcular

1.4.1 $\boldsymbol{x}(2^{q-1}j + 2^{q-k}(i-1) + 1 : 2^{q-1}j + 2^{q-k}i)$ utilizando la expresión (3.9),

1.4.2 $\boldsymbol{w}_{2^{q-2}j+2^{q-k-1}i}(2^{q-1}j + 2^{q-1}(i-1) + 1 : 2^{q-1}j + 2^{q-k}i)$ utilizando la expresión (3.10),

1.4.3 $\boldsymbol{w}_{2^{q-2}j+2^{q-k-1}(i-1)}(2^{q-1}j + 2^{q-1}(i-1) + 1 : 2^{q-1}j + 2^{q-k}i)$ utilizando la expresión (3.11).

2 El procesador P_1 envía al procesador P_0 los bloques de vectores $\boldsymbol{x}(2^{q-1}j + 1 : 2^{q-1}j + 2^{q-1})$, $\boldsymbol{w}_{2^{q-2}j+2^{q-2}}(2^{q-1}j + 1 : 2^{q-1}j + 2^{q-1})$ y $\boldsymbol{w}_{2^{q-2}j}(2^{q-1}j + 1 : 2^{q-1}j + 2^{q-1})$.

Superpaso 3

El procesador P_0 calcula $\boldsymbol{x}(1 : 2^q)$ utilizando la expresión (3.12).

Consideramos ahora el caso general en que $m \geq 2$. Cada procesador inicialmente desarrolla los primeros $q - m$ pasos de la fase 2 de forma independiente. En los siguientes m pasos, los procesadores deben comunicar sus bloques de vectores actualizados a otros

procesadores, que utilizan estos resultados para actualizar bloques de vectores de tamaño cada vez mayor, siguiendo un esquema de comunicación de tipo *fan-in*. En dicho esquema de comunicación, los procesadores pueden dividirse en dos categorías: activo, cuando el procesador realiza cálculos e inactivo, cuando no realiza cálculos.

Al principio del proceso, la matriz A del sistema se distribuye entre todos los procesadores. En los pasos $k = q-1, q-2, \ldots, m$, todos los procesadores trabajan al mismo tiempo de forma independiente. Al final del paso m de actualización, cada procesador ha calculado bloques de vectores de 2^{q-m} componentes. Para poder seguir el proceso de actualización, se hace necesario mezclar los resultados obtenidos por procesadores distintos. Así, cada procesador impar, por ejemplo P_d, envía sus bloques de vectores a su procesador vecino P_{d-1}. El procesador P_{d-1} utiliza estos bloques, mezclándolos y calculando nuevos bloques de vectores con doble número de componentes en el paso $m+1$. En este paso, los procesadores pares se han convertido en procesadores activos, ya que han recibido bloques de vectores de otros procesadores mientras que los procesadores impares se han convertido en procesadores inactivos. Al final del paso $m+1$, se hace necesario mezclar de nuevo los resultados obtenidos por dos procesadores activos. En el paso $m-2$ cada procesador cuyo número es múltiplo de 4 se convierte en procesador activo, mientras que el resto de procesadores permanecen inactivos. Este proceso continúa de forma recursiva hasta que en el último paso el procesador P_0 es el único que permanece activo, estando inactivos los restantes.

El siguiente algoritmo BSP resume las características de este método siguiendo un esquema de comunicaciones *fan-in* cuando el número de procesadores es mayor que dos.

Algoritmo 3.4 Un algoritmo BSP de tipo *fan-in* para sistemas tridiagonales cuando $p > 2$.

Superpaso 1

El procesador P_0 envía al procesador P_j, para $j = 1, 2, \ldots, 2^m - 1$, los elementos $a_{2^{q-m}j+i}$, $b_{2^{q-m}j+i-1}$ y $c_{2^{q-m}j+i+1}$, para $i = 1, 2, \ldots, 2^{q-m}$, suponiendo que $c_{n+1} = 0$.

Superpaso 2

1 En el procesador P_j, para $j = 0, 1, \ldots, 2^m - 1$,

1.1 calcular $\hat{a}_{2^{q-m}j+i}$, para $i = 1, 2, \ldots, 2^{q-m}$, utilizando las expresiones (3.5) y (3.6).

1.2 calcular $\boldsymbol{x}_{(2^{q-m}j+2i-1:2^{q-m}j+2i)}$, para $i = 1, 2, \ldots, 2^{q-m-1}$ utilizando la expresión (3.7).

1.3 calcular $\boldsymbol{w}_{2^{q-m-1}j+i(2^{q-m}j+2i-1:2^{q-m}j+2i)}$ y $\boldsymbol{w}_{2^{q-m-1}j+i-1(2^{q-m}j+2i-1:2^{q-m}j+2i)}$, para $i = 1, 2, \ldots, 2^{q-m-1}$, utilizando las expresiones (3.8) y suponiendo que $\boldsymbol{w}_0$ y $\boldsymbol{w}_{2^q}$ son vectores nulos.

1.4 para $k = q-2, q-3, \ldots, m$, y para $i = 1, 2, \ldots, 2^{k-m}$ calcular

1.4.1 $\boldsymbol{x}_{(2^{q-m}j+2^{q-k}(i-1)+1:2^{q-m}j+2^{q-k}i)}$ utilizando la expresión (3.9),

1.4.2 $\boldsymbol{w}_{2^{q-m-1}j+2^{q-k-1}i(2^{q-m}j+2^{q-m}(i-1)+1:2^{q-m}j+2^{q-k}i)}$ utilizando la expresión (3.10),

1.4.3 $\boldsymbol{w}_{2^{q-m-1}j+2^{q-k-1}(i-1)(2^{q-m}j+2^{q-m}(i-1)+1:2^{q-m}j+2^{q-k}i)}$ utilizando la expresión (3.11).

2 El procesador P_{2j+1}, con $j = 0, 1, \ldots, 2^{m-1}-1$, envía al procesador P_{2j} los bloques de vectores

2.1 $\boldsymbol{x}_{(2^{q-m}(2j+1)+1:2^{q-m}(2j+1)+2^{q-m})}$,

2.2 $\boldsymbol{w}_{2^{q-m-1}(2j+1)+2^{q-m-1}(2^{q-m}(2j+1)+1:2^{q-m}(2j+1)+2^{q-m})}$,

2.3 $\boldsymbol{w}_{2^{q-m-1}(2j+1)(2^{q-m}(2j+1)+1:2^{q-m}(2j+1)+2^{q-m})}$.

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

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

1.1 $\boldsymbol{x}_{(2^{m-k+2}i+1:2^{m-k+2}(i+1))}$, utilizando la expresión (3.9),

1.2 $\boldsymbol{w}_{2^{m-k+1}(i+1)(2^{m-k+2}i+1:2^{m-k+2}(i+1))}$, utilizando la expresión (3.10),

1.3 $\boldsymbol{w}_{2^{m-k+1}i(2^{m-k+2}i+1:2^{m-k+2}(i+1))}$, utilizando la expresión (3.11).

2 El procesador $P_{2^{m-k}+2^{m-k+1}i}$, para $i = 0, 1, \ldots, \lfloor\frac{2^k-1}{2}\rfloor$, comunica al procesador $P_{2^{m-k+1}i}$ los bloques de vectores $\boldsymbol{x}_{(2^{m-k+2}(2i+1)+1:2^{m-k+3}(i+1))}$, $\boldsymbol{w}_{2^{m-k+2}(i+1)(2^{m-k+2}(2i+1)+1:2^{m-k+3}(i+1))}$ y $\boldsymbol{w}_{2^{m-k+1}(2i+1)(2^{m-k+2}(2i+1)+1:2^{m-k+3}(i+1))}$.

Superpaso $m+2$

El procesador P_0 calcula $\boldsymbol{x}(1:2^q)$ utilizando la expresión (3.12).

La diferencia entre este algoritmo y el algoritmo 3.3 está en que ahora es necesario introducir el superpaso $m-k+2$, para $k = m-1, m-2, \ldots, 2, 1$, en el que se produce la comunicación de bloques de vectores entre los procesadores activos y la actualización y cálculo de las nuevas componentes de estos vectores.

Con el fin de analizar detalladamente la estructura que presenta el algoritmo 3.4, analizamos los cálculos aritméticos y los esquemas de comunicación que se requieren para resolver un ejemplo concreto, donde A es una matriz tridiagonal de tamaño 32×32. En este caso, $n = 2^5$, por lo que $q = 5$. Además, supondremos que el número de procesadores es $p = 2^2$, en consecuencia, $m = 2$. En estas condiciones,

$$A = \begin{bmatrix} G_1 & & & & \\ & G_2 & & & \\ & & \ddots & & \\ & & & G_{15} & \\ & & & & G_{16} \end{bmatrix} + \boldsymbol{u}_1\boldsymbol{v}_1^T + \boldsymbol{u}_2\boldsymbol{v}_2^T + \cdots + \boldsymbol{u}_{15}\boldsymbol{v}_{15}^T.$$

A lo largo de todo el proceso tendremos que calcular y actualizar los bloques de vectores que denotamos por $\boldsymbol{x}$ y $\boldsymbol{w}_i$, para $i = 1, 2, \ldots, 15$, utilizando las expresiones (3.9), (3.10) y (3.11) tal como se indica en el algoritmo 3.2.

La figura 3.2 recoge las actualizaciones y las comunicaciones que se realizan en cada procesador hasta obtener la solución final cuando se ejecuta el algoritmo 3.4 para este caso concreto.

Como se desprende de la figura 3.2, el algoritmo 3.4 presenta dos partes claramente diferenciadas: en la primera de ellas se producen todos los cálculos aritméticos de forma independiente en cada procesador sin realizar ningún tipo de comunicación entre los mismos. En la segunda parte se desarrolla un esquema de comunicaciones de tipo *fan-in*, conducente a la obtención de la solución final en el procesador principal. En nuestro ejemplo, la primera parte de cálculo aritmético del algoritmo constaría de la fase 1 y los primeros dos pasos de la fase 2 de actualización de los vectores. Después, es necesaria una comunicación de elementos desde los procesadores impares a los procesadores pares, que son los únicos activos en el siguiente paso. Los procesadores pares calculan cada uno un bloque de 16

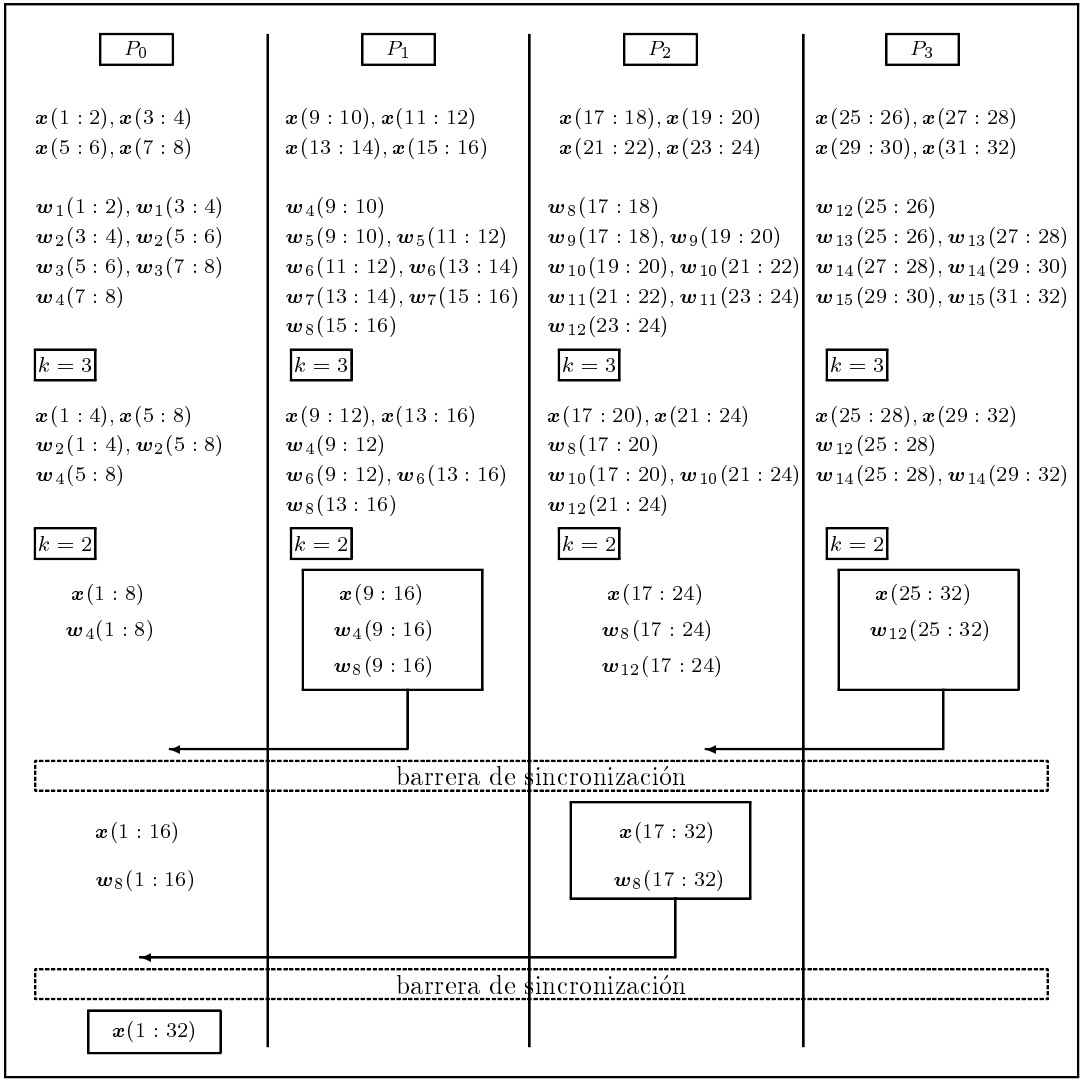


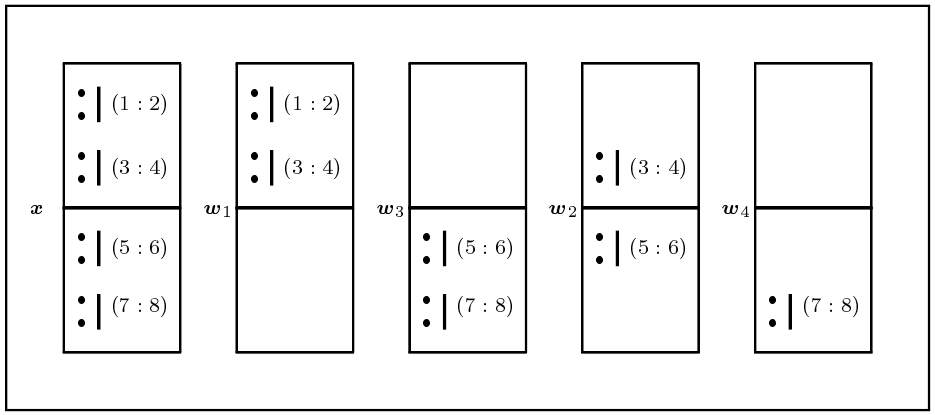
Figura 3.2: *Esquema del algoritmo 3.4 para* $n = 32$ *y* $p = 4$.

componentes del vector $\boldsymbol{x}$ y $\boldsymbol{w_8}$ y el procesador P_2 comunica al procesador P_0 sus bloques actualizados para que el procesador principal calcule la solución utilizando la expresión (3.12). Para $p = 4$, basta con realizar dos barreras de sincronización, ya que después de la segunda comunicación ya se puede obtener la solución final en el procesador principal.

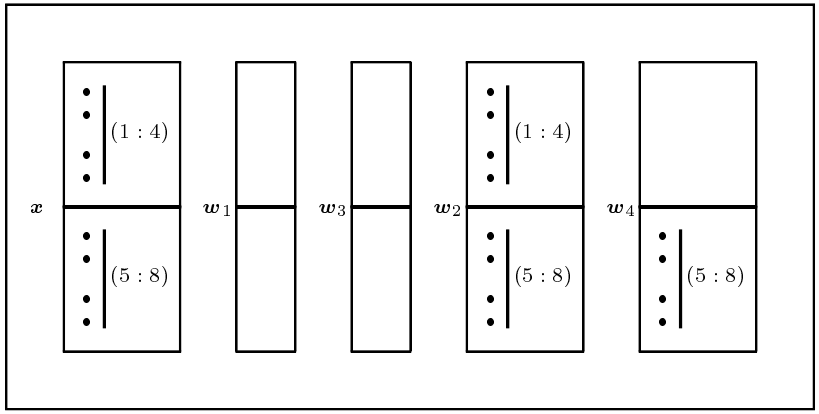
Se observa también que el proceso no es totalmente equilibrado en el sentido que los procesadores centrales tienen una carga computacional mayor. Es interesante destacar que, cuando el número de procesadores es bajo, la mayor parte de los cálculos aritméticos se realizan antes de que sea necesaria una comunicación entre procesadores; el tamaño de los bloques de vectores que se envían en la primera comunicación es $\frac{n}{p}$. En la siguiente comunicación el número de componentes del bloque se va duplicando hasta llegar a la última comunicación, en la que se envían dos bloques de $\frac{n}{2}$ componentes desde el procesador $P_{\frac{n}{2}}$ al procesador P_0.

La figura 3.3 nos muestra los cálculos y actualizaciones que lleva a cabo el procesador P_0 para el caso $n = 32$ que estamos considerando. En dicha figura se ponen de manifiesto todas las actualizaciones que realiza P_0 hasta que es necesaria una comunicación de elementos con otros procesadores. Por tanto, se detalla únicamente el trabajo computacional que realiza dicho procesador en el superpaso 2 del algoritmo 3.4 para este ejemplo. Notemos que el tamaño máximo del bloque que puede actualizar es de $2^{q-m} = 8$. Como se aprecia en la figura 3.3(a), los primeros cálculos se realizan con bloques de dos componentes; posteriormente, la figura 3.3(b) nos muestra cómo las actualizaciones son de bloques de cuatro componentes, para acabar en la figura 3.3(c) actualizando las ocho componentes de los vectores $\boldsymbol{x}$ y $\boldsymbol{w}_8$.

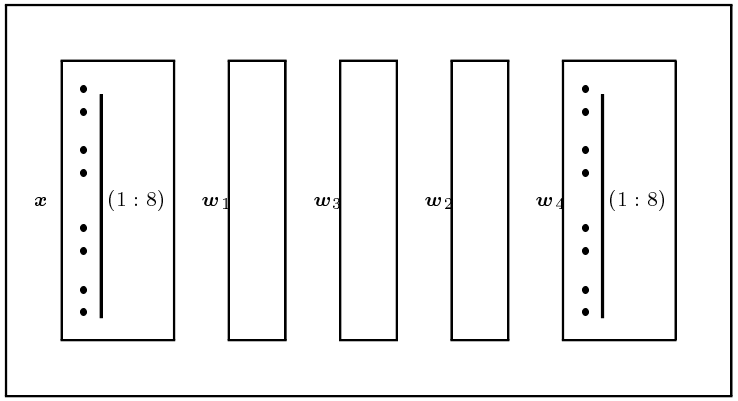
La figura 3.3 pone de manifiesto, de nuevo, la característica fundamental de esta implementación en la que no se tienen en cuenta los bloques nulos de los vectores que se actualizan en cada paso. Así, por ejemplo, para el cálculo del bloque $\boldsymbol{w}_4(1:8)$ que aparece en la figura 3.3(c) utilizamos los bloques $\boldsymbol{w}_4(5:8)$, $\boldsymbol{w}_2(1:4)$ y $\boldsymbol{w}_2(5:8)$ calculados previamente, no siendo necesarias el resto de sus componentes para los cálculos que realiza este procesador.



(a) *Inicialización*



(b) *Actualización,* $k = 2$



(c) *Actualización,* $k = 1$

Figura 3.3: *Inicialización y actualizaciones que realiza el procesador* P_0 *cuando ejecuta el algoritmo 3.4 para el ejemplo 3.2.*

3.3.2 Coste computacional

En esta sección calculamos el coste computacional de los algoritmos 3.3 y 3.4 estudiando el coste de cada uno de los superpasos que los componen. Comenzamos calculando el coste del algoritmo 3.3.

Coste del superpaso 1. En este superpaso no se realiza ninguna operación aritmética, por lo que el coste computacional viene dado por el coste de comunicación. El procesador principal envía $4n - 4\frac{n}{p}$ elementos de la matriz A y del vector $\boldsymbol{d}$ al procesador P_1, quien también recibe un elemento situado fuera de los bloques tridiagonales de A. El último procesador únicamente recibe un elemento situado fuera de los bloques tridiagonales. Por lo tanto, el coste de comunicación es de $(2n+1)g$ flops, por lo que el coste total de este superpaso es de

$$(2n+1)g + l \quad \text{flops.} \tag{3.19}$$

Coste del superpaso 2. Comenzamos calculando el coste aritmético. Para calcular los elementos diagonales de las matrices G_i, utilizando las expresiones (3.5) y (3.6) se necesitan 2^{q-1} flops; es decir, $\dfrac{n}{2}$ flops. Para resolver los sistemas de los apartados 1.2 y 1.3 del superpaso 2 se necesitan $7 \cdot 2^{q-2}$ flops en cada procesador. En consecuencia, el coste aritmético de realizar los apartados 1.1, 1.2 y 1.3 del superpaso 2 es de

$$9 \cdot 2^{q-2} = \frac{9}{4}n \quad \text{flops.} \tag{3.20}$$

Debido a la estructura particular de los vectores que aparecen en el apartado 1.4.3 del superpaso 2, son necesarios

$$10 + 2^{q-k+1} + 2(1 + 3 \cdot 2^{q-k+1}) = 12 + 5 \cdot 2^{q-k} \quad \text{flops}$$

para calcular los bloques de vectores $\boldsymbol{x}$, $\boldsymbol{w}_{2^{q-1-k}i}$ y $\boldsymbol{w}_{2^{q-1-k}(i-1)}$.

Como el número de bloques de vectores que se actualizan para cada procesador en esta etapa es 2^{k-1}, tenemos que el coste aritmético del apartado 1.4.3 del algoritmo es

$$2^{k-1}\left(12 + 5 \cdot 2^{q-k}\right) \quad \text{flops.} \tag{3.21}$$

Sumando las expresiones (3.20) y (3.21), se obtiene el coste aritmético del superpaso 2, que es de

$$\frac{9}{4}n + \sum_{k=1}^{q-2} 2^{k-1}\left[12 + 5 \cdot 2^{q-k}\right] = \left[5(q-1) + \frac{11}{2}\right]\frac{n}{2} - 12 \quad \text{flops.} \tag{3.22}$$

Para obtener el coste de comunicación de este superpaso, hay que tener en cuenta que el último procesador envía sólamente dos vectores de tamaño 2^{q-1} a su procesador par contiguo. Así, el coste de comunicación corresponde al de una $2 \cdot 2^{q-1}$-relación, es decir, ng flops.

Sumando el coste aritmético y de comunicación en este superpaso y efectuando las oportunas simplificaciones, se obtiene un coste computacional para el superpaso 2 de

$$\left[5(q-1) + \frac{11}{2}\right]\frac{n}{2} - 12 + ng + l \quad \text{flops.} \tag{3.23}$$

Coste del superpaso 3. En este superpaso, únicamente el procesador principal realiza operaciones aritméticas, ya que es el encargado de calcular el vector solución $\boldsymbol{x}(1 : 2^q)$ y no realiza ninguna comunicación, por lo que el coste computacional es de

$$8 + 2n + l \quad \text{flops.} \tag{3.24}$$

Finalmente, sumando las expresiones (3.19), (3.23) y (3.24) y realizando las simplificaciones oportunas, se obtiene que el coste computacional total del algoritmo 3.3 es de

$$\left[\frac{5}{2}q + \frac{9}{4}\right]n - 4 + (3n+1)g + 3l \quad \text{flops.} \tag{3.25}$$

Una vez calculado el coste del algoritmo 3.3 procedemos a calcular el coste del algoritmo 3.4 de forma análoga.

Coste del superpaso 1. En este superpaso no se realiza ninguna operación aritmética, por lo que el coste computacional viene dado por el coste de comunicación. El procesador principal envía $4n - 4\frac{n}{p}$ elementos de la matriz A y del vector $\boldsymbol{d}$ al resto de procesadores. Además, todos los procesadores excepto el último, reciben dos elementos situados fuera de los bloques tridiagonales de A. El último procesador únicamente recibe un elemento situado fuera de los bloques tridiagonales. En consecuencia, el coste de comunicación es de $(4n - 4\frac{n}{p} + 2p - 3)g$ flops, por lo que el coste total de este superpaso es de

$$\left(4n - 4\frac{n}{p} + 2p - 3\right) g + l \quad \text{flops.} \tag{3.26}$$

Coste del superpaso 2. Comenzamos calculando el coste aritmético. Para calcular los elementos diagonales de las matrices G_i, utilizando las expresiones (3.5) y (3.6) se necesitan 2^{q-m} flops; es decir, $\dfrac{n}{p}$ flops. Para resolver los sistemas de los apartados 1.2 y 1.3 del superpaso 2 se necesitan $7 \cdot 2^{q-m-1}$ flops en cada uno de los procesadores no extremos. En consecuencia, el coste aritmético de realizar los apartados 1.1, 1.2 y 1.3 del superpaso 2 es de

$$9 \cdot 2^{q-m-1} = \frac{9}{2}\frac{n}{p} \quad \text{flops.} \tag{3.27}$$

Debido a la estructura particular de los vectores que aparecen en el apartado 1.4.3 del superpaso 2, son necesarios

$$10 + 2^{q-k+1} + 2(1 + 3 \cdot 2^{q-k+1}) = 12 + 5 \cdot 2^{q-k} \quad \text{flops}$$

para calcular los bloques de vectores $\boldsymbol{x}$, $\boldsymbol{w}_{2^{q-1-k}i}$ y $\boldsymbol{w}_{2^{q-1-k}(i-1)}$.

Como el número de bloques de vectores que se actualizan para cada procesador en esta etapa es 2^{k-m}, tenemos que el coste aritmético del apartado 1.4.3 del algoritmo es

$$2^{k-m}\left(12 + 5 \cdot 2^{q-k}\right) \quad \text{flops.} \tag{3.28}$$

Sumando las expresiones (3.27) y (3.28), se obtiene el coste aritmético del superpaso 2, que es de

$$\frac{9}{2}\frac{n}{p} + \sum_{k=m}^{q-2} 2^{k-m}\left[12 + 5\cdot 2^{q-k}\right] = \left(5q - 5m + \frac{11}{2}\right)\frac{n}{p} - 12 \quad \text{flops.} \tag{3.29}$$

Para obtener el coste de comunicación de este superpaso, hay que tener en cuenta que únicamente la mitad de los procesadores envían datos a otros procesadores. Cada procesador impar, salvo el último, envía tres vectores de tamaño 2^{q-m} al procesador contiguo, mientras que el último procesador envía sólamente dos vectores de tamaño 2^{q-m} a su procesador par contiguo. Así, como cada procesador, salvo el último, envía o recibe $3\cdot 2^{q-m}$ unidades de datos, el coste de comunicación es el de una $3\cdot 2^{q-m}$-relación, es decir, $3\dfrac{n}{p}g$ flops.

Sumando el coste aritmético y de comunicación en este superpaso y efectuando las oportunas simplificaciones, se obtiene un coste computacional para el superpaso 2 de

$$\left(5q - 5m + \frac{11}{2}\right)\frac{n}{p} - 12 + 3\frac{n}{p}g + l \quad \text{flops.} \tag{3.30}$$

Coste del superpaso $\boldsymbol{m-k+2}$, para $k = m-1, m-2, \ldots, 2, 1$. Para calcular el coste de estos $m-1$ superpasos, fijamos un valor de k y calculamos el coste computacional de este superpaso concreto. Después, sumamos los costes de todos los superpasos en función de k.

Por lo tanto, dado un cierto k, calculamos en primer lugar el coste del superpaso $m-k+2$. En este superpaso cada procesador activo sólo efectúa una actualización de los bloques de los vectores $\boldsymbol{x}$, $\boldsymbol{w}_{2^{q-1-k}i}$ y $\boldsymbol{w}_{2^{q-1-k}(i-1)}$, utilizando las expresiones (3.9), (3.10) y (3.11). Por la tanto, el coste aritmético de este superpaso es de

$$12 + 5\cdot 2^{q-k} \quad \text{flops.} \tag{3.31}$$

En cuanto al coste de comunicación, notemos que en todos los superpasos se repite el mismo patrón de comunicación: unos cuantos procesadores envían datos a sus procesadores vecinos. La diferencia en cada superpaso radica en el tamaño de la h-relación. Fijado k, la

comunicación se corresponde con una $3 \cdot 2^{q-k}$-relación, por lo que el coste de comunicación es de $3 \cdot 2^{q-k}g$ flops. Por lo tanto, sumando los costes aritmético y de comunicación obtenemos que el coste computacional del superpaso $m-k+2$ es de

$$12 + 5 \cdot 2^{q-k} + 3 \cdot 2^{q-k}g + l \quad \text{flops.}$$

En consecuencia, calculamos ahora el coste computacional de los superpasos $m-k+2$, para $k = m-1, m-2, \ldots, 2, 1$, que viene dado por

$$\sum_{k=1}^{m-1} \left(12 + 5 \cdot 2^{q-k} + 3 \cdot 2^{q-k}g + l\right)$$

o equivalentemente después de hacer las simplificaciones oportunas

$$12(m-1) + 5\left(1 + \frac{2}{p}\right)n + 3\left(1 - \frac{2}{p}\right)ng + (m-1)l \quad \text{flops.} \tag{3.32}$$

Coste del superpaso $\mathbf{m+2}$. Este superpaso es análogo al superpaso 3 del algoritmo 3.3, por lo que su coste computacional viene dado por la expresión (3.24). Así pues, para obtener el coste computacional del algoritmo 3.4 sumamos las expresiones (3.26), (3.30), (3.24) y (3.32) y realizamos las simplificaciones oportunas, lo que produce un coste total de

$$\left[\frac{1}{p}\left(5q - 5m - \frac{9}{2}\right) + 7\right]n + 12m - 16 + \left(7n - 3\frac{n}{p} + 2p - 3\right)g + (m+2)l \quad \text{flops.} \tag{3.33}$$

3.4 Un algoritmo BSP basado en el método *recursive doubling*

3.4.1 Descripción del algoritmo

Podemos considerar una modificación en el método descrito por medio del algoritmo 3.4. Esta modificación se basa en el modo en que se llevan a cabo las comunicaciones que nos conducen al cálculo y actualización de los bloques de vectores a partir de los que se obtiene $\boldsymbol{x}(1:2^q)$. Consideramos un nuevo algoritmo dividido también en dos fases claramente diferenciadas; una primera fase totalmente análoga a la primera parte del algoritmo 3.4 en la que se realizan únicamente cálculos de tipo aritmético, sin efectuar comunicaciones. En la segunda parte del algoritmo se realizan las comunicaciones de forma distinta al modelo *fan-in* que seguimos en el algoritmo 3.4. Ahora se sigue un modelo en el que todos los procesadores comunican sus bloques de vectores al procesador principal, que es el encargado de calcular el resto de actualizaciones hasta llegar a la obtención de la solución final. Nótese que, para el caso particular en que $m = 1$, el algoritmo que produce este nuevo modelo de comunicación coincide exactamente con el algoritmo 3.3, ya que las comunicaciones se producen desde el procesador P_1 al P_0, al igual que ocurría en el caso del modelo de comunicación *fan-in*.

El algoritmo 3.5 resume las características esenciales de este método siguiendo el esquema de comunicación anteriormente expuesto y teniendo en cuenta que $m \geq 2$.

Algoritmo 3.5 Un algoritmo BSP para sistemas tridiagonales para $m \geq 2$.

Superpaso 1

El procesador P_0 envía al procesador P_j, para $j = 1, 2, \ldots, 2^m - 1$, los elementos $a_{2^{q-m}j+i}$, $b_{2^{q-m}j+i-1}$ y $c_{2^{q-m}j+i+1}$, para $i = 1, 2, \ldots, 2^{q-m}$, suponiendo que $c_{n+1} = 0$.

Superpaso 2

1 En el procesador P_j, para $j = 1, 2, \ldots, 2^m - 1$,

1.1 calcular $\hat{a}_{2^{q-m}j+i}$, para $i = 1, 2, \ldots, 2^{q-m}$, utilizando las expresiones (3.5) y (3.6),

1.2 calcular $\boldsymbol{x}_{(2^{q-m}j+2i-1:2^{q-m}j+2i)}$, para $i = 1, 2, \ldots, 2^{q-m-1}$ utilizando la expresión (3.7),

1.3 calcular $\boldsymbol{w}_{2^{q-m-1}j+i(2^{q-m}j+2i-1:2^{q-m}j+2i)}$ y $\boldsymbol{w}_{2^{q-m-1}j+i-1(2^{q-m}j+2i-1:2^{q-m}j+2i)}$, para $i = 1, 2, \ldots, 2^{q-m-1}$, utilizando (3.8) y suponiendo que $\boldsymbol{w}_0$ y $\boldsymbol{w}_{2^q}$ son vectores nulos,

1.4 para $k = q-2, q-3, \ldots, m$, y para $i = 1, 2, \ldots, 2^{k-m}$ calcular

1.4.1 $\boldsymbol{x}_{(2^{q-m}j+2^{q-k}(i-1)+1:2^{q-m}j+2^{q-k}i)}$ utilizando la expresión (3.9),

1.4.2 $\boldsymbol{w}_{2^{q-m-1}j+2^{q-k-1}i(2^{q-m}j+2^{q-m}(i-1)+1:2^{q-m}j+2^{q-k}i)}$ utilizando la expresión (3.10),

1.4.3 $\boldsymbol{w}_{2^{q-m-1}j+2^{q-k-1}(i-1)(2^{q-m}j+2^{q-m}(i-1)+1:2^{q-m}j+2^{q-k}i)}$ utilizando la expresión (3.11).

2 El procesador P_j, para $j = 1, 2, \ldots, p-1$, envía al procesador P_0 los bloques de vectores $\boldsymbol{x}_{(2^{q-m}j+1:2^{q-m}j+2^{q-m})}$, $\boldsymbol{w}_{2^{q-m-1}j+2^{q-m-1}(2^{q-m}j+1:2^{q-m}j+2^{q-m})}$, y $\boldsymbol{w}_{2^{q-m-1}j(2^{q-m}j+1:2^{q-m}j+2^{q-m})}$.

Superpaso 3

1 Para $k = m-1, m-2, \ldots, 2, 1$ y para $i = 0, 1, \ldots, 2^k - 1$, el procesador P_0 calcula

1.1 $\boldsymbol{x}_{(2^{m-k+3}i+1:2^{m-k+3}(i+1))}$, utilizando la expresión (3.9),

1.2 $\boldsymbol{w}_{2^{m-k+2}(i+1)(2^{m-k+3}i+1:2^{m-k+3}(i+1))}$, utilizando la expresión (3.10),

1.3 $\boldsymbol{w}_{2^{m-k+2}i(2^{m-k+3}i+1:2^{m-k+3}(i+1))}$, utilizando la expresión (3.11).

2 El procesador P_0 calcula $\boldsymbol{x}(1:2^q)$ utilizando la expresión (3.12).

La figura 3.4 muestra de forma detallada un ejemplo para una matriz de tamaño 32×32, siguiendo el nuevo esquema de comunicaciones expuesto al inicio de esta sección.

Notemos que el algoritmo 3.5 únicamente requiere tres superpasos, ya que la comunicación se produce de forma masiva al final del superpaso 2. En este superpaso cada procesador envía al procesador principal sus bloques de vectores actualizados de manera que es el

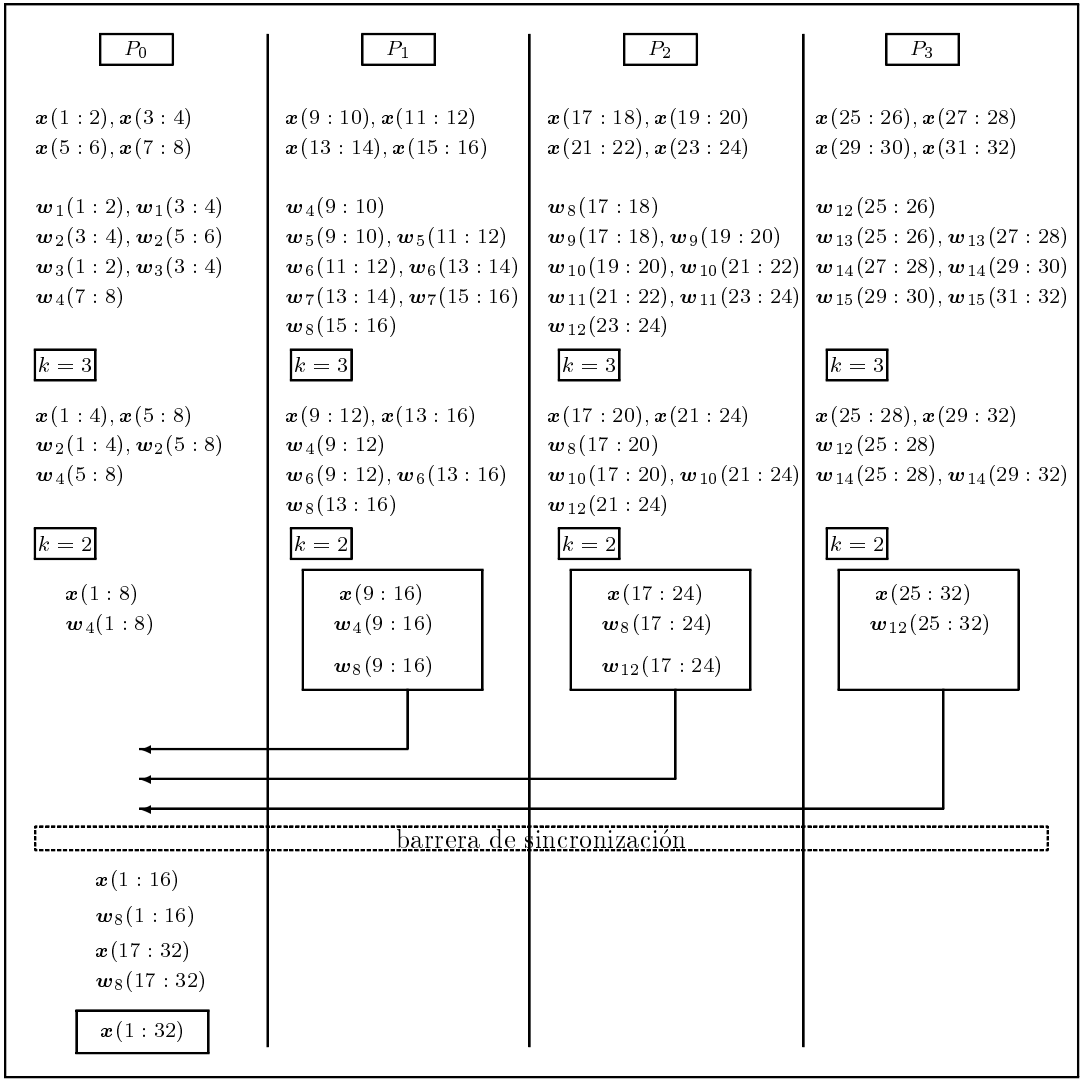


Figura 3.4: *Esquema del algoritmo 3.4 para el caso* $n = 32$ *y* $p = 4$.

procesador principal el encargado de finalizar el proceso de una forma secuencial. Esta es una de las diferencias más importantes con el algoritmo 3.4, en el que el número de superpasos necesarios estaba en función del parámetro m, por lo que depende del número de procesadores, ya que $p = 2^m$. Cuando el número de procesadores que se utilizan es bajo no hay diferencias significativas entre ambas implementaciones en lo que respecta al número total de superpasos requeridos. Así, si $p = 4$ tendremos que $m = 2$, por lo que en el caso del algoritmo 3.4 se necesitan cuatro superpasos, mientras que en el caso del algoritmo 3.5 únicamente se requieren tres superpasos. Si aumentamos el número de procesadores hasta por ejemplo $p = 128$, la situación cambia ya que se necesitarán 9 y 3 superpasos para ejecutar los algoritmos 3.4 y 3.5, respectivamente. Esta diferencia sí puede resultar significativa, especialmente si la máquina en la que se trabaja presenta un coste alto de sincronización.

Otro aspecto digno de análisis en esta nueva implementación es el modelo de comunicación que se establece en el superpaso 2 del algoritmo 3.5. De acuerdo con este modelo cada procesador, excepto el último, envía al procesador P_0 tres bloques de vectores de tamaño 2^{q-m}, mientras que el último procesador únicamente envía dos bloques de vectores de tamaño 2^{q-m}. En consecuencia, el coste de comunicación es de

$$\left(3n - 4\frac{n}{p}\right) g = \left(3 - \frac{4}{p}\right) ng \quad \text{flops}, \tag{3.34}$$

lo que supone un coste bastante mayor que el que se obtiene siguiendo el proceso de comunicación descrito en el superpaso 2 del algoritmo 3.4, que viene dado por la expresión (3.30). Si comparamos las expresiones (3.34) y (3.30), llegamos a la conclusión que

$$\frac{3}{p} ng \leq \left(3 - \frac{4}{p}\right) ng$$

siempre que $p \geq \frac{7}{3}$, lo que se cumple para $p \geq 3$. Hay que tener en cuenta que en el algoritmo 3.5 no se realizan más comunicaciones a lo largo de todo el proceso, mientras que en el algoritmo 3.4 necesitamos llevar a cabo otros tres pasos de comunicación.

3.4.2 Coste computacional

En esta sección se calcula el coste computacional del método descrito en la sección 3.3, de acuerdo con la implementación en paralelo que proporciona el algoritmo 3.5. Para ello, se calcula el coste de cada uno de los superpasos que integran el algoritmo.

Coste del superpaso 1. Este superpaso es análogo al superpaso 1 del algoritmo 3.4, por lo que su coste viene dado por la expresión (3.26).

Coste del superpaso 2. La única diferencia existente entre este superpaso y el superpaso 2 del algoritmo 3.4 se encuentra en el proceso de comunicación, por lo que ambos superpasos tienen el mismo coste aritmético dado por la expresión (3.29).

En cuanto al coste de comunicación, éste viene dado por la expresión (3.34), como ya se analizó anteriormente. En consecuencia, sumando las expresiones (3.29) y (3.34) tendremos el coste computacional del superpaso 2 del algoritmo 3.5, que es de

$$\left(5q - 5m + \frac{11}{2}\right)\frac{n}{p} - 12 + \left(3n - 4\frac{n}{p}\right)g + l \quad \text{flops.} \tag{3.35}$$

Coste del superpaso 3. En este superpaso no se producen comunicaciones de elementos entre los procesadores, por lo que únicamente contamos las operaciones aritméticos que se realizan.

Recordemos que son necesarios $12 + 5 \cdot 2^{q-k}$ flops (véase la expresión (3.31)), para calcular los bloques de vectores $\boldsymbol{x}$, $\boldsymbol{w}_{2^{q-1-k}i}$ y $\boldsymbol{w}_{2^{q-1-k}(i-1)}$. Además, como el número de bloques de vectores que se actualizan en el procesador principal es 2^k, tenemos que el coste aritmético de cualquiera de los k pasos que se llevan a cabo en este superpaso es de $2^{k-m}\left(12 + 5 \cdot 2^{q-k}\right)$ flops.

De esta forma, el coste aritmético de este superpaso es de

$$\sum_{k=1}^{m-1} 2^k \left[12 + 5 \cdot 2^{q-k}\right] = 5(m-1)n + 12(p-2) \quad \text{flops.}$$

Finalmente, el procesador principal calcula la solución al final del superpaso 3, lo que representa un coste de $8 + 2n$ flops.

Por lo tanto, el coste total del superpaso 3 es de

$$5(m-3)n + 12(p-2) + 8 + l \quad \text{flops.} \tag{3.36}$$

Sumando las expresiones (3.26), (3.35) y (3.36) y realizando las simplificaciones oportunas, se obtiene que el coste computacional del algoritmo 3.5 es

$$\frac{1}{p}\left[m(5p-3) + 5q - 3p - \frac{11}{2}\right] n + 12(p-2) - 4 + \left(7n - \frac{8}{p}n + 2p - 3\right) g + 3l \quad \text{flops.} \tag{3.37}$$

3.5 Resultados teóricos y comparaciones

En esta sección se presentan los tiempos teóricos de ejecución de los algoritmos 3.4 y 3.5 sobre las tres máquinas paralelas que se describieron brevemente en la sección 2.5. Los valores que aparecen en las columnas de 2 procesadores se han calculado utilizando la expresión (3.25) que proporciona el coste computacional del algoritmo 3.3. Cuando el número de procesadores aumenta ya se comparan los tiempos obtenidos para los algoritmos 3.4 y 3.5 utilizando las expresiones (3.33) y (3.37), respectivamente.

3.5.1 Tiempos en un IBM SP2

Analizamos de forma separada los resultados que se obtienen en esta máquina cuando se utilizan los dos tipos de conexión: switch y ethernet. La tabla 3.4 recoge los tiempos comparados de los algoritmos estudiados en este capítulo para un IBM SP2 dotado con un switch de alto rendimiento y con una conexión de tipo ethernet.

IBM SP2 switch					
	p				
	2	4		8	
n	Alg. 3.3	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5
512	0.0009	0.0013	0.0010	0.0017	0.0014
1024	0.0017	0.0020	0.0017	0.0024	0.0021
2048	0.0033	0.0035	0.0031	0.0038	0.0036
4096	0.0068	0.0066	0.0060	0.0066	0.0067
8192	0.0142	0.0131	0.0119	0.0123	0.0130
16384	0.0297	0.0263	0.0241	0.0239	0.0256
32768	0.0622	0.0537	0.0494	0.0475	0.0514
65536	0.1305	0.1099	0.1016	0.0954	0.1037
131072	0.2735	0.2255	0.2090	0.1929	0.2099
262144	0.5719	0.4631	0.4302	0.3910	0.4255
524288	1.1940	0.9508	0.8852	0.7934	0.8630
1048576	2.4886	1.9514	1.8203	1.6110	1.7505
2097152	5.1786	4.0033	3.7410	3.2713	3.5507
4194304	10.7602	8.2077	7.6833	6.6424	7.2016

(a) *Switch*

IBM SP2 ethernet					
	p				
	2	4		8	
n	Alg. 3.3	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5
512	0.0081	0.0312	0.0247	0.1015	0.0868
1024	0.0141	0.0554	0.0442	0.1830	0.1608
2048	0.0261	0.1039	0.0831	0.3461	0.3088
4096	0.0503	0.2010	0.1612	0.6724	0.6048
8192	0.0992	0.3954	0.3175	1.3250	1.197
16384	0.1978	0.7846	0.6305	2.6305	2.3814
32768	0.3966	1.5638	1.2572	5.2418	4.7508
65536	0.7972	3.1237	2.5124	10.4652	9.4903
131072	1.6048	6.2468	5.0257	20.9136	18.971
262144	3.2327	12.4991	10.0588	41.8135	37.9354
524288	6.5135	25.0165	20.1374	83.6197	75.8705
1048576	13.1256	50.0765	40.3200	167.2446	151.7534
2097152	26.4505	100.2468	80.7356	334.5197	303.5444
4194304	53.3022	200.6883	161.6675	669.1203	607.1767

(b) *Ethernet*

Tabla 3.1: *Tiempos teóricos para los algoritmos 3.3, 3.4 y 3.5 medidos en un IBM SP2 dotado con un switch de alto rendimiento y conexión ethernet.*

La tabla 3.1(a) nos muestra los tiempos teóricos esperados con switch. Analizando de forma general esta tabla se observa que para 4 procesadores el algoritmo 3.4, que recordemos se basaba en una comunicación de tipo *fan-in*, siempre es más lento que el algoritmo 3.5. Sin embargo, cuando el número de procesadores aumenta a 8 ya no podemos afirmar esta conclusión para cualquier valor de n. A partir de $n = 4096$ ya se observa que el algoritmo 3.4 es más rápido que el algoritmo 3.5. Estas diferencias se van acrecentando a medida que aumenta n. También se observa que al ir aumentando p, los tiempos de ejecución van disminuyendo, especialmente para valores grandes de n. Este descenso en los tiempos es más notable para el algoritmo 3.4 que para el algoritmo 3.5, en el que para valores pequeños de n se observa un ligero incremento en los tiempos al aumentar p.

La tabla 3.1(b) nos muestra los tiempos teóricos esperados utilizando una conexión ethernet. En dicha tabla se observa que para 4 y 8 procesadores el algoritmo 3.4 siempre es más lento que el algoritmo 3.5. Las diferencias se van reduciendo a medida que aumenta el número de procesadores. De la misma forma, se observa que al aumentar el número de procesadores, los tiempos de ejecución de los dos algoritmos aumentan, lo que nos da una idea de que no resultan eficientes en una máquina con estas características.

Con el objetivo de realizar un estudio comparativo de los algoritmos presentados en este capítulo, no sólamente nos interesa establecer qué algoritmo es más rápido sino también las diferencias que existen entre ambos en los tiempos previstos de ejecución. Para determinar estos valores formamos una nueva tabla en la que calculamos el pocentaje de tiempo que nos ahorramos al ejecutar el algoritmo más rápido para esos valores concretos. Así, si para un determinado valor de n y p, $T_{alg.\ 3.4}$ representa el tiempo de ejecución del algoritmo 3.4, $T_{alg.\ 3.5}$ representa el tiempo de ejecución del algoritmo 3.5 y suponemos que $T_{alg.\ 3.4} > T_{alg.\ 3.5}$, entonces el valor que aparece en la tabla 3.2 viene dado por la expresión

$$\frac{T_{alg.\ 3.4} - T_{alg.\ 3.5}}{T_{alg.\ 3.4}} 100. \tag{3.38}$$

Como en la mayoría de los casos el más rápido es el algoritmo 3.5, únicamente señalaremos en la tabla cuando el porcentaje de ahorro de tiempos sea a favor del algoritmo 3.4. Señalizaremos en la tabla este caso con una F.

Se observa que, por ejemplo utilizando ethernet, al resolver el sistema mediante el algoritmo 3.5 nos podemos ahorrar aproximadamente

IBM SP2				
	p			
	4		8	
n	sw	et	sw	et
512	23.08	20.83	17.65	14.48
1024	15.00	20.22	12.50	12.13
2048	11.43	20.02	5.26	10.78
4096	9.09	19.80	1.49(F)	10.05
8192	9.16	19.70	5.38(F)	9.66
16384	8.37	19.64	6.64(F)	9.47
32768	8.01	19.61	7.59(F)	9.37
65536	7.55	19.57	8.00(F)	9.32
131072	7.32	19.55	8.10(F)	9.29
262144	7.10	19.52	8.11(F)	9.27
524288	6.90	19.50	8.06(F)	9.27
1048576	6.72	19.48	7.97(F)	9.26
2097152	6.55	19.46	7.87(F)	9.26
4194304	6.39	19.44	7.76(F)	9.26

Tabla 3.2: *Diferencias de tiempos entre los algoritmos 3.4 y 3.5, medidos en porcentajes.*

el 20% del tiempo de ejecución del algoritmo 3.4 si se utilizan 4 procesadores y de alrededor del 10% cuando empleamos los 8 procesadores. Estas cifras resultan muy significativas.

De la lectura de la tabla 3.2 se pueden destacar varios aspectos notables.

- Al aumentar el número de procesadores de 4 a 8, las diferencias de tiempo a favor del algoritmo 3.5 se van reduciendo para cualquier valor de n y utilizando cualquier hardware para las comunicaciones. Para el caso de 8 procesadores y comunicación switch, cuando $n > 2048$, ya resulta más conveniente utilizar el algoritmo 3.4.

- Para $p = 4$, las diferencias de tiempos entre los dos algoritmos son mayores para la máquina con conexión ethernet que para la misma máquina con switch. Por ejemplo, cuando $n = 1048576$, se observa en la tabla 3.2 que utilizando el switch el algoritmo 3.4 es un 6.7% aproximadamente más lento que el algoritmo 3.5. Sin embargo, cuando se utiliza la conexión ethernet, el algoritmo 3.5 es un 20% aproximadamente más rápido que el algoritmo 3.4. Esto representa una variación significativa a tener en cuenta. Además, las diferencias se mantienen constantes alrededor del 20% para $p = 4$ y alrededor del 10% para $p = 8$, independientemente del tamaño de la matriz.

- Para 4 procesadores el máximo ahorro de tiempo se produce para ethernet y $n = 512$, siendo el valor del porcentaje del 20.83%. Para 8 procesadores, el máximo ahorro de tiempo se produce para switch y $n = 512$, siendo del 17.65%. Nótese el descenso en los valores máximos al ir aumentando el número de procesadores.

En consecuencia, como conclusión general para una máquina de este tipo se puede decir que el algoritmo 3.4, basado en un modelo de comunicación *fan-in*, siempre es más lento que el algoritmo 3.5, salvo para 8 procesadores y conexión con switch. Cuando se utiliza el switch de alto rendimiento, las diferencias en tiempos disminuyen sensiblemente al aumentar el número de procesadores. Este comportamiento nos lleva a pensar que el algoritmo 3.4 presenta un mejor comportamiento en máquinas donde las comunicaciones no son muy costosas.

CRAY T3D															
	p														
	2	4		8		16		32		64		128		256	
n	Alg. 3.3	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5
2048	0.0053	0.0034	0.0034	0.0026	0.0035	0.0024	0.0041	0.0024	0.0049	0.0025	0.0059	0.0029	0.0072	0.0037	0.0088
4096	0.0114	0.0071	0.0072	0.0052	0.0070	0.0045	0.0080	0.0044	0.0096	0.0043	0.0112	0.0048	0.0133	0.0056	0.0158
8192	0.0245	0.0150	0.0152	0.0106	0.0143	0.0089	0.0159	0.0085	0.0189	0.0081	0.0218	0.0086	0.0257	0.0094	0.0299
16384	0.0523	0.0316	0.0320	0.0220	0.0293	0.0180	0.0320	0.0169	0.0376	0.0157	0.0431	0.0162	0.0504	0.0171	0.0582
32768	0.1115	0.0665	0.0673	0.0455	0.0601	0.0364	0.0645	0.0338	0.0752	0.0309	0.0859	0.0314	0.0999	0.0326	0.1147
65536	0.2365	0.1397	0.1414	0.0942	0.1235	0.0743	0.1304	0.0680	0.1509	0.0617	0.1716	0.0620	0.1989	0.0636	0.2277
131072	0.5003	0.2930	0.2964	0.1950	0.2537	0.1516	0.2640	0.1373	0.3032	0.1237	0.3436	0.1233	0.3972	0.1257	0.4539
262144	1.0552	0.6131	0.6199	0.4035	0.5209	0.3097	0.5346	0.2776	0.6095	0.2484	0.6882	0.2464	0.7943	0.2501	0.9065
524288	2.2195	1.2808	1.2944	0.8341	1.0689	0.6326	1.0825	0.5616	1.2255	0.4996	1.3792	0.4935	1.5892	0.4993	1.8121
1048576	4.6575	2.6707	2.6980	1.7227	2.1923	1.2922	2.1919	1.1365	2.4642	1.0055	2.7647	0.9893	3.1808	0.9987	3.6243
2097152	9.7518	5.5597	5.6143	3.5544	4.4937	2.6387	4.4382	2.2999	4.9554	2.0240	5.5425	1.9844	6.3674	1.9991	7.2502
4194304	20.3774	11.5563	11.6655	7.3271	9.2058	5.3863	8.9853	4.6540	9.9651	4.0748	11.1117	3.9814	12.7475	4.0034	14.5055

Tabla 3.3: *Tiempos teóricos para los algoritmos 3.3, 3.4 y 3.5 medidos en un CRAY T3D para* 2, 4, 8, 16, 32, 64, 128 *y* 256 *procesadores.*

3.5.2 Tiempos en un CRAY T3D

La tabla 3.3 muestra los resultados teóricos esperados en un CRAY T3D, una máquina altamente paralela que dispone de hasta 256 procesadores y en la que las comunicaciones son muy rápidas. En las tablas que resumen los resultados teóricos los tamaños de la matriz de coeficientes comienzan a partir de 2048 debido al elevado número de procesadores disponibles en dicha máquina.

La primera conclusión que extraemos observando la tabla 3.3 es que, salvo para el caso en que $p = 2$, el algoritmo 3.4 es más rápido

que el algoritmo 3.5. Para $p = 4$ los resultados que se obtienen en ambos algoritmos son muy parecidos, especialmente en matrices de tamaño pequeño y medio. Por tanto, el comportamiento general de ambos algoritmos difiere radicalmente de lo que ocurría en el IBM SP2 en el que el algoritmo 3.4 era siempre más lento que el algoritmo 3.5, salvo cuando la conexión era a través del switch y $p = 8$.

También se observa de forma generalizada que al ir aumentando el número de procesadores, los tiempos van disminuyendo, como es de esperar en una máquina de estas características. Esta tendencia a la reducción de tiempos se cumple en el algoritmo 3.4 hasta $p = 128$. Para $p = 128$ ya únicamente se obtienen mejores tiempos para matrices con $n > 65536$. Para valores de $n < 65536$ los tiempos son muy parecidos, aunque ya se advierte que aumentan muy ligeramente al pasar de 64 a 128 procesadores. Al aumentar el múmero de procesadores de 128 a 256 se produce un aumento generalizado en los tiempos de ejecución para todos los tamaños, aunque dicho aumento de tiempo no es significativo. Para el algoritmo 3.5, se produce un aumento de tiempos para cualquier tamaño a partir de $p = 32$, aunque para matrices pequeñas ya se constatan aumentos a partir de $p = 8$. En general, el comportamiento paralelo de este algoritmo es peor que el del algoritmo 3.4.

En consecuencia, en una máquina de estas características, podemos hablar de un número de procesadores óptimo para la ejecución de cada uno de los algoritmos. Así, por ejemplo, observamos que resolver un sistema con $n = 1048576$ mediante el algoritmo 3.4 utilizando 128 procesadores cuesta un total de 0.9893 segundos, mientras que si utilizamos los 256 procesadores, costará 0.9987 segundos. Así, en general, podemos decir que, dependiendo del tamaño del sistema que queramos resolver, nos interesa utilizar un número alto de procesadores para ejecutar el algoritmo 3.4, por ejemplo, 64 o 128 procesadores; sin embargo, si ejecutamos el algoritmo 3.5 para resolver un sistema tridiagonal nos interesa utilizar un número bajo de procesadores para obtener los mejores resultados, por ejemplo, 8 o 16 procesadores. La tabla 3.4(a) resume el número óptimo de procesadores para los que se obtienen los mejores tiempos, dependiendo del tamaño de la matriz de coeficientes.

Volvemos a construir una tabla similar a la que construimos para la máquina IBM SP2 donde se reflejan las diferencias en procentajes de los tiempos de ejecución de ambos algoritmos. La tabla 3.4(b) muestra estos resultados, teniendo en cuenta que la expresión (3.38) es la utilizada para la obtención de estos porcentajes.

CRAY T3D		
n	Alg. 3.4	Alg. 3.5
2048	32	4
4096	64	8
8192	64	8
16384	64	8
32768	64	8
65536	64	8
131072	128	8
262144	128	8
524288	128	8
1048576	128	16
2097152	128	16
4194304	128	16

(a) *Número óptimo de procesadores*

CRAY T3D							
	p						
n	4	8	16	32	64	128	256
2048	0	25.71	41.46	51.02	57.63	59.72	57.95
4096	1.39	25.71	43.75	54.17	61.61	63.91	64.56
8192	1.32	25.87	44.02	55.03	62.84	66.54	68.56
16384	1.25	24.91	43.75	55.05	63.57	67.86	70.62
32768	1.19	24.29	43.57	55.05	64.03	68.57	71.58
65536	1.20	23.72	43.02	54.94	64.04	68.83	72.07
131072	1.15	23.13	42.58	54.72	64.00	68.96	72.31
262144	1.10	22.54	42.07	54.45	63.91	68.98	72.41
524288	1.05	21.97	41.56	54.17	63.78	68.95	72.45
1048576	1.01	21.42	41.05	53.88	63.63	68.90	72.44
2097152	0.97	20.90	40.56	53.59	63.48	68.83	72.43
4194304	0.94	20.41	40.05	53.30	63.33	68.77	72.40

(b) *Diferencias de tiempos entre los algoritmos 3.4 y 3.5*

Tabla 3.4: *Número óptimo de procesadores y diferencias de tiempos para la ejecución de los algoritmos 3.4 y 3.5 en un CRAY T3D.*

A partir de 2 procesadores, el algoritmo 3.4 siempre es más rápido. Se observa que al ir aumentando el número de procesadores, el porcentaje de ahorro de tiempos al ejecutar dicho algoritmo va aumentando progresivamente, especialmente al pasar de 8 a 16 y de 16 a 32 procesadores. Para $p = 32$ se observa que el tiempo de ejecución del algoritmo 3.5 es más del doble del tiempo de ejecución del algoritmo 3.4. Los valores máximos se obtienen para 256 procesadores, donde el tiempo de ejecución del algoritmo 3.4 es casi tres cuartas partes el tiempo de ejecución del algoritmo 3.5.

Así pues, la conclusión general que podemos extraer es que en esta máquina, salvo que trabajemos con dos procesadores, siempre es más rápido el algoritmo 3.4, aumentando las diferencias de tiempos de ejecución a medida que aumenta el número de procesadores. Los valores óptimos del algoritmo 3.4 se obtienen para 64 y 128 procesadores, dependiendo del tamaño de la matriz, mientras que los valores óptimos del algoritmo 3.5 se obtienen para 8 y 16 procesadores, dependiendo también del tamaño de la matriz.

3.5.3 Tiempos en un cluster de Pentiums

Ahora se analizan los resultados esperados sobre un cluster de Pentiums, siguiendo un esquema similar al descrito para las máquinas anteriores. Los resultados numéricos esperados se muestran en la tabla 3.5(a).

De esta tabla se desprende que, al igual que sucedía en el caso del IBM SP2, los tiempos de ejecución del algoritmo 3.4 son siempre mayores que el del algoritmo 3.5, para cualquier valor de n y de p. En este caso, a diferencia de lo que ocurría en las máquinas anteriores, no encontramos ningún caso en el que el algoritmo 3.4 sea más rápido que el algoritmo 3.5. A la vista de esta tabla sí podemos afirmar que, a medida que aumenta el número de procesadores, las diferencias entre los tiempos de ejecución se van acercando cada vez más, especialmente para matrices de tamaño medio y grande. Esto ya nos ocurría en las máquinas anteriores. En cuanto a los tiempos óptimos que se obtienen para los diferentes tamaños de la matriz, podemos decir que hasta $n = 524288$ los mejores tiempos se obtienen para $p = 2$ mediante el algoritmo 3.3. Sin embargo, cuando el valor de $n \geq 524288$ los mejores tiempos nos los proporciona el algoritmo 3.5 para $p = 4$, ya que al aumentar a $p = 8$, los tiempos también experimentan un incremento para este algoritmo. El comportamiento del algoritmo 3.4 es distinto; aunque no produce valores óptimos del tiempo, dichos tiempos se reducen al pasar de 4 a 8 procesadores para

Cluster de Pentiums					
	p				
	2	4		8	
n	Alg. 3.3	Alg. 3.4	Alg. 3.5	Alg. 3.4	Alg. 3.5
512	0.0006	0.0012	0.0010	0.0020	0.0015
1024	0.0011	0.0019	0.0015	0.0026	0.0022
2048	0.0020	0.0032	0.0027	0.0040	0.0034
4096	0.0039	0.0060	0.0050	0.0066	0.0060
8192	0.0079	0.0115	0.0096	0.0119	0.0111
16384	0.0160	0.0227	0.0191	0.0226	0.0215
32768	0.0328	0.0453	0.0382	0.0441	0.0423
65536	0.0673	0.0910	0.0768	0.0874	0.0841
131072	0.1381	0.1834	0.1551	0.1744	0.1682
262144	0.2835	0.3699	0.3136	0.3494	0.3373
524288	0.5817	0.7468	0.6342	0.7012	0.6775
1048576	1.1930	1.5079	1.2828	1.4085	1.3615
2097152	2.4453	3.0450	2.5950	2.8306	2.7370
4194304	5.0095	6.1490	5.2493	5.6896	5.5029

(a) *Tiempos*

Cluster de Pentiums		
	p	
n	4	8
512	16.67	25.00
1024	21.05	15.38
2048	15.63	15.00
4096	16.67	9.09
8192	16.52	6.72
16384	15.86	4.87
32768	15.67	4.08
65536	15.60	3.78
131072	15.43	3.56
262144	15.22	3.46
524288	15.08	3.38
1048576	14.93	3.34
2097152	14.78	3.31
4194304	14.63	3.28

(b) *Diferencias de tiempos*

Tabla 3.5: *Tiempos teóricos y diferencias de tiempos para los algoritmos 3.3, 3.4 y 3.5 en un cluster de Pentiums.*

$n \geq 16384$.

De la misma forma que hemos realizado en las máquinas anteriores un estudio de las diferencias entre los tiempos de ejecución de los algoritmos, ahora formamos la tabla 3.5(b), que muestra las diferencias de tiempo, en porcentaje, al ejecutar los algoritmos 3.4 y 3.5, siempre a favor del segundo. Como se observa en dicha tabla, las diferencias en los tiempos varían aproximadamente alrededor del 15% para 4 procesadores y disminuyen hasta el 3% para 8 procesadores. Las diferencias más importantes se producen para 4 procesadores y tamaños pequeños de la matriz de coeficientes. En ningún caso las diferencias de tiempo llegan al 50% como ocurría en las máquinas anteriores.

Como conclusión podemos decir que en este tipo de máquinas nuevamente el algoritmo 3.4 es más lento que el algoritmo 3.5, para cualquier valor de n y de p, aunque presenta un mejor comportamiento paralelo. También hay que resaltar que los valores óptimos de los tiempos hasta $n = 524288$ se obtienen para $p = 2$ utilizando el algoritmo 3.3 y, para tamaños de $n > 524288$, los valores óptimos se obtienen para $p = 4$ procesadores utilizando el algoritmo 3.4.

3.5.4 Estudio del *speedup*

En esta sección realizamos un estudio del speedup para los algoritmos 3.4 y 3.5 que hemos estudiado en este capítulo. Recordemos brevemente que el *speedup* es un parámetro que representa una medida de comparación de un algoritmo cuando se ejecuta utilizando 1 y p procesadores. Para una descripción más detallada de este parámetro, véase la sección 2.1.

La tabla 3.6 nos muestra tres tablas en las que aparecen los valores del *speedup* y de la eficiencia para una matriz de tamaño $n = 2097152$ en las tres máquinas en las que se han calculado tiempos teóricos.

La tabla 3.6(a) nos muestra el speedup y la eficiencia de los algoritmos 3.4 y 3.5 en un IBM SP2 para una matriz de gran tamaño, $n = 2097152$. La eficiencia la medimos en porcentaje.

IBM SP2 switch						
	speedup			eficiencia		
p	Alg. 3.3	Alg. 3.4	Alg. 3.5	Alg. 3.3	Alg. 3.4	Alg. 3.5
2	1.59			79.30		
4		2.17	2.32		54.15	57.95
8		2.65	2.44		33.13	30.53

(a) *IBM SP2 switch*

IBM SP2 ethernet						
	speedup			eficiencia		
p	Alg. 3.3	Alg. 3.4	Alg. 3.5	Alg. 3.3	Alg. 3.4	Alg. 3.5
2	0.29			14.32		
4		0.09	0.11		2.16	2.68
8		0.03	0.03		0.32	0.36

(b) *IBM SP2 ethernet*

CRAY T3D						
	speedup			eficiencia		
p	Alg. 3.3	Alg. 3.4	Alg. 3.5	Alg. 3.3	Alg. 3.4	Alg. 3.5
2	1.85			92.71		
4		3.38	3.35		84.48	83.66
8		5.29	4.18		66.07	52.26
16		7.12	4.23		44.50	26.46
32		8.17	3.79		25.53	11.85
64		9.28	3.39		17.19	6.28
128		9.47	2.95		7.40	2.31
256		9.40	2.59		3.67	1.01

(c) *CRAY T3D*

Cluster de Pentiums						
	speedup			eficiencia		
p	Alg. 3.3	Alg. 3.4	Alg. 3.5	Alg. 3.3	Alg. 3.4	Alg. 3.5
2	0.95			47.74		
4		0.84	0.99		21.03	24.68
8		0.91	0.94		11.31	11.70

(d) *Cluster de Pentiums*

Tabla 3.6: *Speedup y eficiencia de los algoritmos 3.3, 3.4 y 3.5 para* $n = 2097152$.

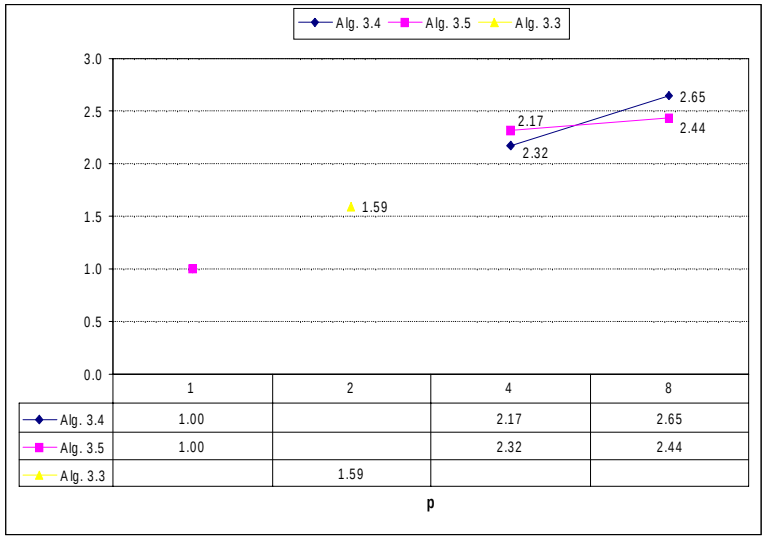
Notemos la gran diferencia en los valores del speedup y de la eficiencia en los dos algoritmos cuando se utiliza el switch de alto rendimiento frente a una conexión de tipo ethernet. Con el switch, la eficiencia alcanza un valor máximo del 79.30% para el algoritmo 3.3 cuando se utilizan 2 procesadores; sin embargo, con la conexión ethernet para efectuar comunicaciones, únicamente se alcanza un 14.32% de eficiencia como valor máximo para el mismo algoritmo con 2 procesadores. En general, cuando se utiliza un hardware lento para las comunicaciones como la conexión ethernet, la pérdida de eficiencia en ambos algoritmos es notable, obteniéndose unos resultados muy discretos para el speedup. La situación cambia cuando las comunicaciones son rápidas, obteniéndose valores muy aceptables del speedup.

Observamos que, aunque los porcentajes de eficiencia son mayores para el algoritmo 3.5 que para el algoritmo 3.4, este último algoritmo presenta un mejor comportamiento paralelo, como se desprende del aumento del speedup para conexión ethernet. Dicho aumento es mayor en proporción que el que obtenemos para el algoritmo 3.5.

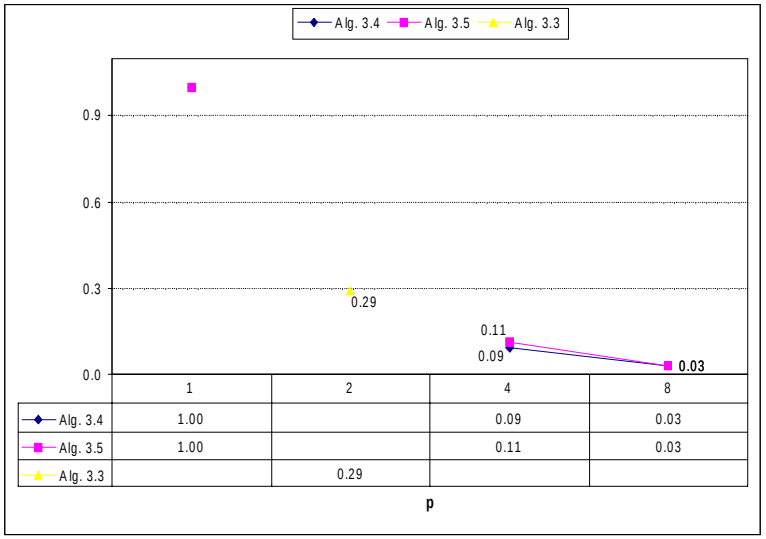
La tabla 3.6(c) es similar a las tablas 3.6(a) y 3.6(b) pero utilizando ahora una máquina del tipo CRAY T3D. En la misma se observa que se obtienen unos valores muy buenos del speedup y de la eficiencia, especialmente en el algoritmo 3.4, llegando a unos valores máximos de 9.47 para 128 procesadores, que es cuando se obtienen los mejores tiempos. También vemos claramente que los valores de la eficiencia y speedup del algoritmo 3.4 mejoran los del algoritmo 3.5, especialmente cuando el número de procesadores es alto, en el que las diferencias aumentan significativamente. En el caso $p = 2$, la eficiencia del algoritmo 3.5 es de 92.71%, lo que representa un máximo en este conjunto de valores. Sin embargo, al aumentar el número de procesadores, la eficiencia para este algoritmo decrece más rápidamente de lo que lo hace cuando se ejecuta el algoritmo 3.4.

A la vista de estos datos concluimos que, contrariamente a lo que ocurría con el IBM SP2, para una máquina como el CRAY T3D, el algoritmo 3.4 resulta no sólamente más rápido que el algoritmo 3.5, sino que presenta unos valores de *speedup* y eficiencia muy buenos, que lo convierten en un algoritmo altamente paralelo.

La tabla 3.6(d) nos muestra los valores del speedup y la eficiencia para un cluster de Pentiums, utilizando hasta un máximo de 8 procesadores. En este caso, siempre los valores de speedup y eficiencia son mejores para el algoritmo 3.5 frente a los obtenidos para el



(a) *IBM SP2 switch*



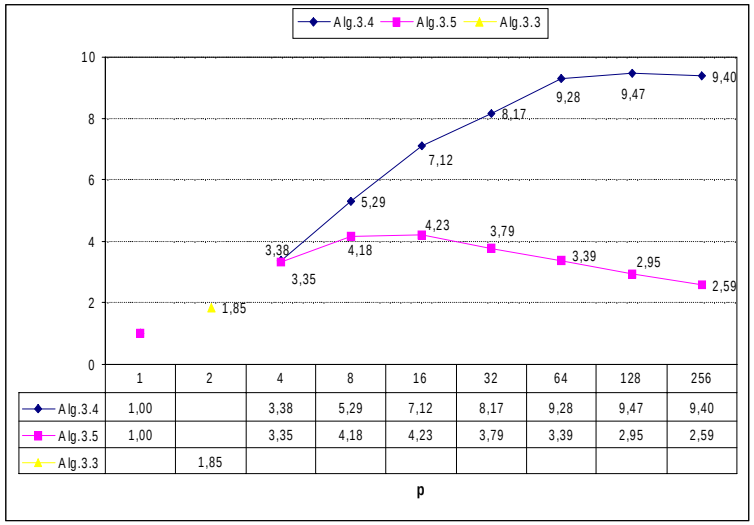
(b) *IBM SP2 ethernet*

Figura 3.5: *Valores del speedup en un IBM SP2*

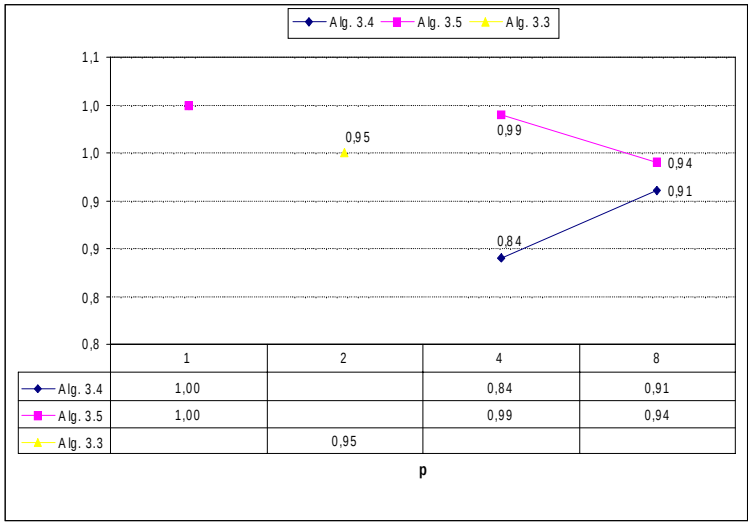
algoritmo 3.4. Los valores de la eficiencia no son tan buenos como los que se obtienen en el CRAY T3D.

Las figuras 3.5 y 3.6 nos muestran los valores del *speedup* obtenidos en la tabla 3.6.

Las figuras 3.7 y 3.8 nos muestra los valores obtenidos de la eficiencia para los algoritmos 3.4 y 3.5 en las tres máquinas donde se han estudiado los resultados numéricos teóricos de dichos algoritmos.

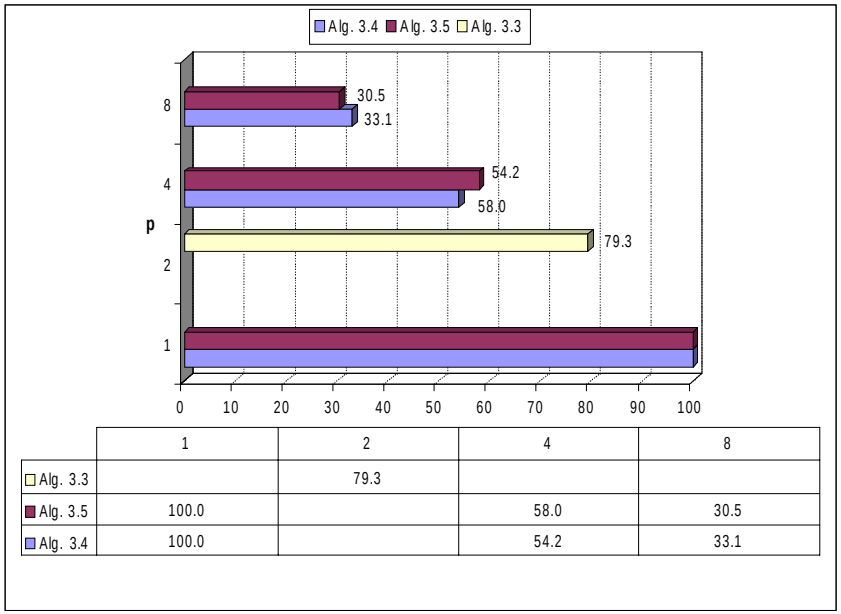


(a) *CRAY T3D*



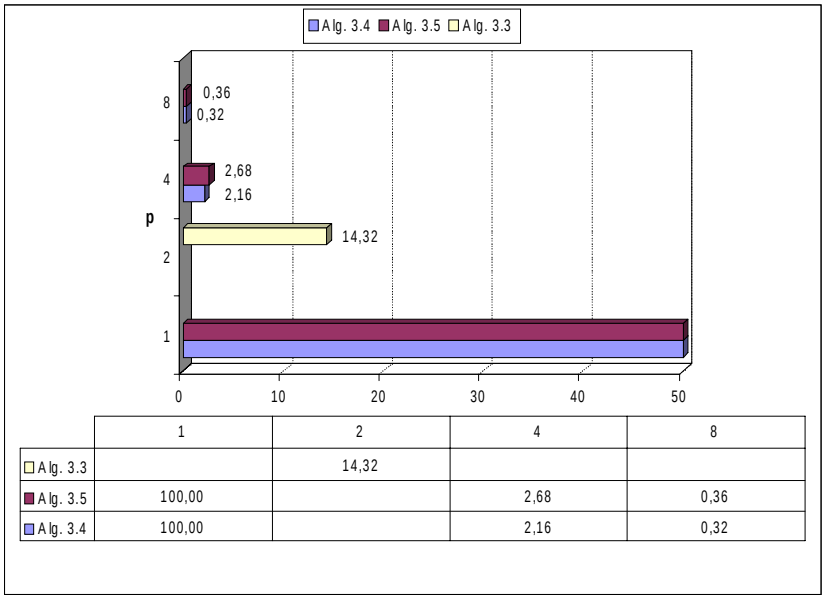
(b) *Cluster de Pentiums*

Figura 3.6: *Valores del speedup en un CRAY y un cluster de Pentiums*



	1	2	4	8
Alg. 3.3		79.3		
Alg. 3.5	100.0		58.0	30.5
Alg. 3.4	100.0		54.2	33.1

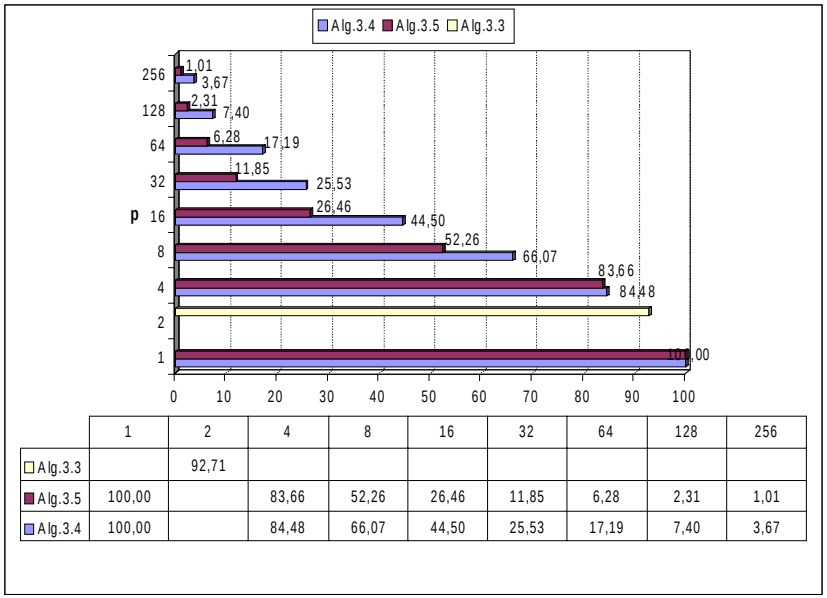
(a) *IBM SP2 switch*



	1	2	4	8
Alg. 3.3		14,32		
Alg. 3.5	100,00		2,68	0,36
Alg. 3.4	100,00		2,16	0,32

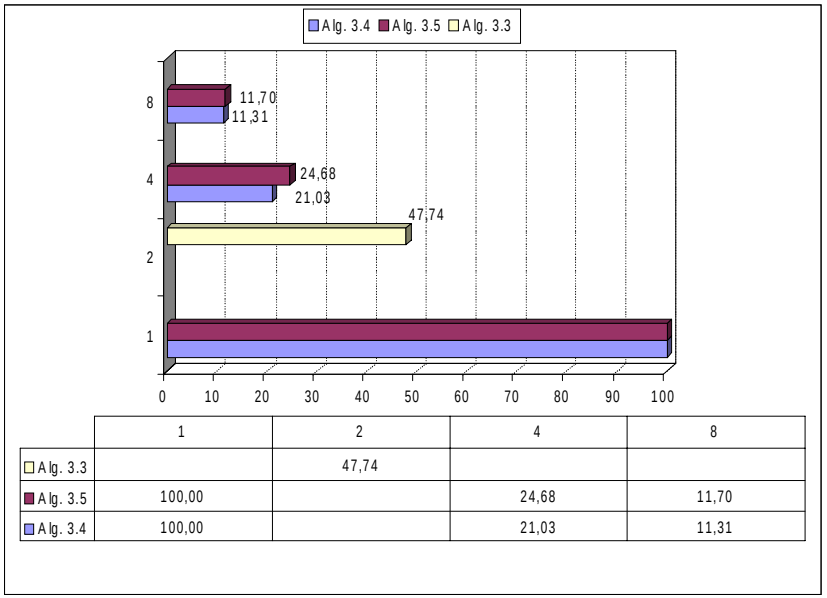
(b) *IBM SP2 ethernet*

Figura 3.7: *Valores de la eficiencia en un IBM SP2*



	1	2	4	8	16	32	64	128	256
Alg.3.3		92,71							
Alg.3.5	100,00		83,66	52,26	26,46	11,85	6,28	2,31	1,01
Alg.3.4	100,00		84,48	66,07	44,50	25,53	17,19	7,40	3,67

(a) *CRAY T3D*



	1	2	4	8
Alg. 3.3		47,74		
Alg. 3.5	100,00		24,68	11,70
Alg. 3.4	100,00		21,03	11,31

(b) *Cluster de Pentiums*

Figura 3.8: *Valores del speedup en un CRAY y un cluster de Pentiums*