UNIVERSIDAD DE OVIEDO

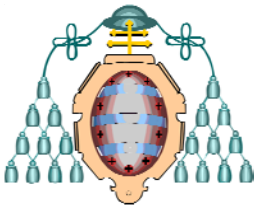
DEPARTAMENTO DE INFORMÁTICA

EL MÉTODO DE NEVILLE: UN ENFOQUE BASADO EN COMPUTACIÓN DE ALTAS PRESTACIONES

Tesis Doctoral

Septiembre 2008

Autora: Raquel Cortina Parajón
Directores: Dr. Pedro Alonso Velázquez
Dr. Vicente Hernández García



Edita: Universidad de Oviedo
Biblioteca Universitaria, 2009
Colección Tesis Doctoral-TDR nº 40
ISBN: 978-84-692-1365-0
D.L.: AS.00737-2009

Índice General

Índice de Figuras

Índice de Gráficas

Índice de Tablas

Índice de Algoritmos

1

Planteamiento y Objetivos

A continuación mostraremos la motivación del trabajo desarrollado, dando una visión global de la tesis doctoral, planteando los objetivos que pretendemos cubrir, así como la estructura del resto de la memoria.

1.1 Motivación

Las aplicaciones de la computación de altas prestaciones y en concreto del paralelismo, se extienden prácticamente a todos los ámbitos donde la programación es necesaria. En la actualidad, la computación paralela está siendo utilizada en multitud de campos para el desarrollo de aplicaciones y el estudio de problemas que requieren gran capacidad de cómputo, bien por el gran tamaño de los problemas que abordan o por la necesidad de trabajar con problemas en tiempo real.

De esta forma, el paralelismo en la actualidad, además de formar parte de diversas líneas abiertas de intensa labor investigadora, puede encontrarse en infinidad de aplicaciones de campos muy variados (aplicaciones en ingeniería y diseño, aplicaciones médicas, aplicaciones comerciales, etc.).

En muchas aplicaciones científicas y técnicas, la formulación del problema mediante un modelo matemático y su tratamiento numérico, nos lleva a la resolución de un sistema de ecuaciones, normalmente de gran tamaño. Para sistemas de orden elevado, la implementación de métodos iterativos en máquinas secuenciales presenta problemas (falta de memoria y tiempo de ejecución elevado) que incluso, en determinados casos, pueden hacerla inviable. Una forma de subsanar las limitaciones de las máquinas secuenciales es el procesamiento paralelo. Por esos motivos, la aparición de computadoras paralelas ha provocado una adaptación de los algoritmos clásicos de la computación matricial para su implementación en dichas máquinas y, además, ha dado lugar a nuevas líneas de investigación dirigidas a diseñar nuevos métodos que obtengan un buen rendimiento en las máquinas paralelas.

La eliminación de Neville es un procedimiento alternativo a la eliminación de Gauss para transformar una matriz cuadrada *A* en una matriz triangular superior. Estrictamente hablando, la eliminación de Neville hace ceros en una columna de *A* añadiendo a cada fila un múltiplo de la fila previa. Esta estrategia se ha probado especialmente útil cuando se trabaja con cierto tipo de matrices, como por ejemplo, las totalmente positivas o las signo-regulares (ver [Ando 87] y [Gasca *et al* 96]). Una matriz se dice totalmente positiva si todos sus menores son no negativos. Este tipo de matrices aparecen en muchas ramas de la ciencia, como por ejemplo en, Matemáticas, Estadística, Economía (ver [Gasca *et al* 96]), o en Diseño Geométrico Asistido por Ordenador (ver [Peña 99]). En esta línea, en los trabajos de [Gasca *et al* 92], [Gassó *et al* 04], [Demmel *et al* 05], [Cortes *et al* 07] y [Gemignani 08] se muestra que la eliminación de Neville es una alternativa interesante a la de Gauss para cierto tipo de estudios.

En noviembre de 1995, Pedro Alonso Velázquez defiende su tesis doctoral titulada Eliminación de Neville y Análisis del Error. Dicha tesis tenía como objetivo principal el estudio del error del método de Neville. Sin embargo, en uno de los capítulos y con la colaboración de los profesores Vicente Hernández García y Jose Ranilla Pastor, se inició el estudio del comportamiento del método de Neville sobre arquitecturas paralelas. Para ello se estimó el tiempo de ejecución de la estrategia que supone dividir la matriz en bloques de filas consecutivas y distribuir éstos entre los procesadores. Los resultados obtenidos fueron que el tiempo de cálculo empleado era igual que en Gauss, mientras que el tiempo de comunicación de Neville era menor. Esto último viene motivado por la propia estructura algorítmica de la eliminación de Neville que sólo precisa comunicaciones entre vecinos, a diferencia de Gauss que requiere operaciones colectivas de comunicación. Estos resultados permitían avanzar que un análisis más profundo de este método sobre arquitecturas paralelas proporcionaría resultados muy satisfactorios comparables a los ya conocidos para el método de Gauss.

En el diseño de algoritmos paralelos que tratan con matrices, la organización por bloques se ha demostrado como la más eficiente para obtener el máximo rendimiento de las máquinas actuales. De todas formas, la distribución de los datos de la matriz afecta al rendimiento del sistema paralelo como podremos comprobar a lo largo de la presente memoria. Por ello, es importante determinar qué esquema de distribución es el más apropiado para cada algoritmo. En nuestro trabajo analizaremos, de manera

detallada, diversas organizaciones por bloques del algoritmo de eliminación de Neville para computadores que sigan el modelo de paso de mensajes.

1.2 Objetivos

La presente tesis doctoral aborda por tanto un estudio en profundidad del método de Neville desde el punto de vista de la Computación de Altas Prestaciones y en particular de la Computación en Paralelo, siendo los objetivos que nos plantearemos en la elaboración del presente trabajo los siguientes:

- Análisis exhaustivo del comportamiento teórico del método de Neville en paralelo basándonos en tres métricas: tiempo de ejecución, eficiencia y escalabilidad.
- Modelizar el comportamiento de los dos entornos de experimentación a utilizar, siendo éstos dos tipos de máquinas representativas del modelo de paso de mensajes: una red de estaciones de trabajo y un multicomputador.
- Implementación del algoritmo secuencial y de los diferentes algoritmos paralelos con objeto de obtener resultados empíricos para de ese modo validar el estudio teórico.

1.3 Estructura de la Memoria de la Tesis

Esta memoria está organizada del siguiente modo

- En el capítulo 2 se realiza una introducción a la eliminación de Neville, así como una recapitulación de algunos resultados sobre su aplicación a matrices totalmente positivas.

- El capítulo 3 presenta el estado del arte de las arquitecturas paralelas, de los paradigmas de programación y de las técnicas sobre evaluación de las prestaciones de algoritmos paralelos. Además se describen y modelizan las plataformas sobre las que se han realizado los experimentos.

- En el capítulo 4 se analiza el comportamiento del método de eliminación de Neville cuando la matriz de datos se distribuye de forma unidimensional. Mostraremos tanto la orientación por filas como por columnas, analizando las prestaciones de cada una de ellas, así como sus implementaciones y conclusiones.

- En el capítulo 5 se analizan las prestaciones del método cuando se considera un particionado bidimensional de los datos, esto es, dividiremos la matriz en

submatrices y las distribuiremos entre los procesadores. Abordaremos un estudio teórico acerca de la conducta de las diferentes particiones para posteriormente cotejarlo con los resultados experimentales obtenidos en los dos entornos de experimentación utilizados.

- Dedicamos un último apartado a las conclusiones que podemos obtener del trabajo expuesto, incluyendo una mención de las publicaciones a las que ha dado lugar dicho estudio. Comentaremos algunas de las líneas abiertas que proyecta la presente memoria de tesis.

- Finalmente, figuran dos anexos. Uno de ellos reúne los resultados de los experimentos que hemos realizado para comparar las diferentes estrategias de distribución. Y el otro, recoge una serie de estimaciones que contribuyen a evaluar la conducta de las diversas formas de abordar la eliminación de Neville en paralelo.

2

La Eliminación de Neville

La modelización de problemas en Física, Mecánica, etc., conducen, en muchas ocasiones, directamente o después de un proceso de discretización, a la resolución de un sistema de ecuaciones de orden finito, dicho sistema será lineal o no lineal de acuerdo con el carácter del problema. Los sistemas de ecuaciones lineales son tan habituales, que es importante ser capaces de resolverlos de forma eficiente y precisa. Es fundamental, por ello, desarrollar métodos eficientes que obtengan la solución de sistemas de ecuaciones lineales.

En [Gasca *et al* 87] se estudiaron estrategias generales de eliminación donde una de ellas, llamada eliminación de Neville, se ha mostrado especialmente adecuada para trabajar con algunas clases especiales de matrices, en particular, las matrices totalmente positivas. Estas matrices se caracterizan por tener todos sus menores no negativos. Esto fue el origen de una serie de trabajos donde las propiedades de la eliminación de Neville han sido estudiadas, destacando su aplicación al caso de matrices totalmente positivas y sus subclases.

En este capítulo, expondremos una serie de resultados básicos acerca de la eliminación de Neville y la total positividad, que serán fundamentales para comprender y motivar los estudios que se llevarán a cabo en los capítulos posteriores.

En primer lugar, describiremos el proceso de eliminación de Neville para matrices no singulares, centrándonos en aquellas en las que no son necesarios cambios de filas. En el apartado 2.2, presentaremos el algoritmo del proceso de eliminación de Neville analizando su coste computacional.

En la sección 2.3 definiremos las matrices totalmente positivas y sus subclases, mostrando la utilidad de la eliminación de Neville para caracterizar este tipo de matrices. Finalizaremos este capítulo haciendo referencia a algunos de los campos de aplicación de la eliminación de Neville.

2.1 Eliminación de Neville

En primer lugar, introduciremos la siguiente notación. Para $k, n \in \mathbb{N}$, $1 \leq k \leq n$, denotaremos por $Q_{k,n}$ al conjunto de todas las sucesiones de k números naturales no mayores que n ordenados crecientemente. Para $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k)$, $\beta = (\beta_1, \beta_2, \ldots, \beta_k) \in Q_{k,n}$ y A una matriz real n x n, denotaremos por $A[\alpha|\beta]$ la submatriz de A, de tamaño k x k, formada por las filas $\alpha_1, \ldots, \alpha_k$ y las columnas $\beta_1, \ldots, \beta_k$ de A. $Q^0_{k,n}$ denotará el conjunto de sucesiones de k números naturales consecutivos no mayores que n.

Los problemas habituales de interpolación dan lugar a un sistema lineal que hay que resolver. Recíprocamente todo sistema lineal se puede interpretar como un problema de interpolación. Así pues, todo método de resolución de uno de ambos problemas puede interpretarse en términos del otro. Partiendo de la fórmula de interpolación de Aitken-Neville, Gasca y Mühlbach introducen en [Gasca *et al* 87] la idea general de lo que llamamos eliminación de Neville, debido a que dicho proceso surge de manera natural cuando la estrategia de interpolación de Neville es usada para resolver sistemas lineales, de manera similar a como ocurre con el método de Gauss y la estrategia de Aitken.

La eliminación de Neville fue descrita con detalle para matrices finitas en [Gasca *et al* 92a]. Aquí nos ceñiremos al caso de matrices no singulares. Para una matriz no singular A de orden n este proceso de eliminación consta de n-1 pasos sucesivos, resultando una serie de matrices de la forma siguiente

$$A = \tilde{A}^{(1)} \rightarrow A^{(1)} \rightarrow \tilde{A}^{(2)} \rightarrow A^{(2)} \rightarrow \ldots \rightarrow \tilde{A}^{(n)} = A^{(n)} = U,$$

donde U es una matriz triangular superior.

Para cada t, $1 \le t \le n$, la matriz $A^{(t)}=(a_{ij}^{(t)})_{1\le i,j\le n}$ tiene ceros por debajo de la diagonal principal en sus t-1 primeras columnas

$$A^{(t)}=\begin{pmatrix} a_{11}^{(t)} & a_{12}^{(t)} & \cdots & \cdots & \cdots & \cdots & a_{1n}^{(t)} \\ 0 & a_{22}^{(t)} & \cdots & \cdots & \cdots & \cdots & a_{2n}^{(t)} \\ \vdots & 0 & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{t-1,t-1}^{(t)} & a_{t-1,t}^{(t)} & \cdots & a_{t-1,n}^{(t)} \\ \vdots & \vdots & \vdots & 0 & a_{tt}^{(t)} & \cdots & a_{tn}^{(t)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & a_{nt}^{(t)} & \cdots & a_{nn}^{(t)} \end{pmatrix}, \tag{2.1}$$

teniéndose además que

$$a_{it}^{(t)}=0,\ i\ge t \Rightarrow a_{ht}^{(t)}=0,\ \forall h\ge i. \tag{2.2}$$

La matriz $A^{(t)}$ se obtiene a partir de la matriz $\tilde{A}^{(t)}$ situando en las últimas filas, si esto fuera necesario, aquellas que poseen un cero en la columna t, según (2.2). Las filas se mueven y se posicionan en el mismo orden relativo en el que aparecían en $\tilde{A}^{(t)}$. Para obtener $\tilde{A}^{(t+1)}$ a partir de $A^{(t)}$ produciremos ceros en la columna t por debajo de la diagonal mediante la resta de un múltiplo de la fila i-ésima a la fila $(i+1)$-ésima para $i=n-1, n-2, \ldots, t$, según la siguiente expresión

$$\tilde{a}_{ij}^{(t+1)}=\begin{cases} a_{ij}^{(t)} & si\ 1\le i\le t \\ a_{ij}^{(t)}-\dfrac{a_{it}^{(t)}}{a_{i-1,t}^{(t)}}\,a_{i-1,j}^{(t)} & si\ t+1\le i,j\le n\wedge a_{i-1,t}^{(t)}\neq 0 \\ a_{ij}^{(t)} & si\ t+1\le i,j\le n\wedge a_{i-1,t}^{(t)}=0. \end{cases} \tag{2.3}$$

Podemos observar que en el tercer caso, $a_{i-1,t}^{(t)}=0$, lo cual por (2.2) implica que $a_{it}^{(t)}=0$. Cuando los cambios de filas no son necesarios, entonces $\tilde{A}^{(t)}=A^{(t)}$ para todo t, esto ocurre, por ejemplo, cuando

$$\det A\left[\alpha\,|\,1,2,\ldots,t\right]\neq 0,\ 1\le t\le n,\ \forall\,\alpha\in Q_{t,n}^{0}, \tag{2.4}$$

(véase Lema *2.6* de [Gasca *et al* 92a]) o cuando A es una matriz no singular y totalmente positiva (Corolario *5.5* de [Gasca *et al* 92a]).

El elemento

$$p_{ij} = a_{ij}^{(j)}, 1 \le i, j \le n, \tag{2.5}$$

se denomina elemento pivote (i,j) de la eliminación de Neville de A, y el número

$$m_{ij} = \begin{cases} \dfrac{a_{ij}^{(j)}}{a_{i-1,j}^{(j)}} \left(= \dfrac{p_{ij}}{p_{i-1,j}} \right) & si\ a_{i-1,j}^{(j)} \neq 0 \\ 0 & si\ a_{i-1,j}^{(j)} = 0, \end{cases} \tag{2.6}$$

el multiplicador (i,j). Obsérvese que $m_{ij} = 0$ si y sólo si $p_{ij} = 0$ y que por (2.2)

$$m_{ij} = 0 \Rightarrow m_{hj} = 0, \forall h > i. \tag{2.7}$$

Resulta interesante observar en qué orden se van obteniendo los ceros por debajo de la diagonal principal de A a través del proceso de eliminación de Neville. De este modo, para cada t, al conseguir $\tilde{A}^{(t+1)}$ a partir de $A^{(t)}$ produciremos un primer cero en la posición (n,t), a continuación en la entrada $(n\text{-}1,t)$, y así sucesivamente hasta la posición $(t\text{+}1,t)$. De forma general, los ceros se van logrando de abajo hacia arriba y recorriendo las columnas de la *1* hasta la n-1. Podríamos representar este proceso del siguiente modo

$$\begin{pmatrix} \ddots & & & & & \\ \uparrow & \ddots & & & & \\ \uparrow & \uparrow & \ddots & & & \\ \vdots & \vdots & \uparrow & \ddots & & \\ \uparrow & \uparrow & \vdots & \uparrow & \ddots & \\ \uparrow & \uparrow & \uparrow & \cdots & \uparrow & \ddots \end{pmatrix}.$$

La eliminación de Neville completa de una matriz A consiste en realizar la eliminación de Neville de A para obtener U, y a continuación proceder con la eliminación de Neville de U^T, la traspuesta de U. El elemento pivote (i,j) (multiplicador respectivamente) de la eliminación de Neville completa de A es el de la eliminación de Neville de A si $i \geq j$ y el elemento pivote (j,i) (multiplicador respectivamente) de la eliminación de U^T si $j \geq i$.

Vamos a considerar con más detalle el caso de una matriz no singular A cuya eliminación de Neville puede llevarse a cabo sin cambios de filas. Como en [Gasca *et al* 94a], cuando este hecho suceda, diremos que dichas matrices verifican la condición *WR*. En este caso, el proceso de eliminación puede ser descrito matricialmente mediante matrices elementales sin utilizar matrices de permutación.

Con este fin, denotaremos por $E_i(\alpha)$ ($2 \le i \le n$) la matriz triangular inferior bidiagonal cuyos elementos (r,s), $1 \le r, s \le n$ vienen dados por

$$\begin{cases} 1 & si \;\; r = s \\ \alpha & si \;\; (r,s) = (i, i-1) \\ 0 & en \; otro \; caso \end{cases} \tag{2.8}$$

esto es

$$E_i(\alpha) = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & \alpha & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}. \tag{2.9}$$

Podemos observar que

$$\begin{cases} E_i^{-1}(\alpha) = E_i(-\alpha) \\ E_i(\alpha)E_i(\beta) = E_i(\alpha + \beta) \\ E_i(\alpha)E_j(\beta) = E_j(\beta)E_i(\alpha) \; excepto \, para \; | \, i - j \, | = 1 \;\; con \;\; \alpha\beta \neq 0. \end{cases} \tag{2.10}$$

Si una matriz A satisface la condición *WR*, el proceso de eliminación de Neville puede expresarse a través de matrices elementales $E_i(\alpha)$ como

$$E_n(-m_{n,n-1}) \ldots (E_3(-m_{32}) \ldots \; E_n(-m_{n2}))\,(E_2(-m_{21}) \ldots E_{n-1}(-m_{n-1,1}) E_n(-m_{n1}))A = U, \tag{2.11}$$

donde U es una matriz triangular superior, no singular, y los m_{ij} son los multiplicadores de (2.6) satisfaciendo (2.7).

Teniendo en cuenta (2.9), se puede comprobar fácilmente que

$$E_i(m_{ij})E_{i+1}(m_{i+1,j}) \ldots E_n(m_{nj}) = \begin{pmatrix} 1 & & & & & & \\ 0 & 1 & & & & & \\ & \ddots & \ddots & & & & \\ & & 0 & 1 & & & \\ & & & m_{ij} & 1 & & \\ & & & & \ddots & \ddots & \\ & & & & & m_{nj} & 1 \end{pmatrix}, \tag{2.12}$$

es una matriz bidiagonal, triangular inferior, cuyas entradas (2,1), ..., (i-1,i-2) son cero. Por el contrario, el producto $E_n(m_{nj}) \ldots E_{i+1}(m_{i+1,j})E_i(m_{ij})$ es una matriz triangular inferior, por lo general, no bidiagonal. Así, (2.11) puede ser escrita en la forma

$$L_{n-1} L_{n-2} \ldots L_1 \, A = U, \tag{2.13}$$

con

$$L_i = \begin{pmatrix} 1 & & & & & & \\ 0 & 1 & & & & & \\ & \ddots & \ddots & & & & \\ & & 0 & 1 & & & \\ & & & -m_{i+1,i} & 1 & & \\ & & & & \ddots & \ddots & \\ & & & & & -m_{ni} & 1 \end{pmatrix}. \tag{2.14}$$

De (2.13) se deduce la sucesión de matrices intermedias, $A^{(t)}$, que se producen en el proceso de eliminación. Partiendo de $A^{(1)}=A$ obtendremos la matriz $A^{(2)}$ a través de L_1, esto es $A^{(2)}=L_1A^{(1)}$, de la misma forma $A^{(3)}=L_2A^{(2)}$ y, reiterando el proceso, se tiene que para cada t, $2 \le t \le n$,

$$A^{(t)} = L_{t-1} \, A^{(t-1)}. \tag{2.15}$$

Teniendo en cuenta que, por (2.10), $E_i^{-1}(\alpha)=E_i(-\alpha)$, podemos deducir de manera inmediata la siguiente factorización para A

$$A = (E_n(m_{n1}) \, E_{n-1}(m_{n-1,1}) \ldots \, E_2(m_{21})) \, (E_n(m_{n2}) \ldots E_3(m_{32})) \ldots E_n(m_{n,n-1})U. \tag{2.16}$$

El siguiente resultado (Teorema 2.2 de [Gasca *et al* 94a]), nos permite relacionar la factorización anterior y la condición *WR*:

Teorema 2.1. Una matriz no singular A satisface la condición *WR* si y sólo si puede ser factorizada en la forma (2.16) con los m_{ij} satisfaciendo (2.7). Si A satisface esta condición, la factorización es única y m_{ij} es el multiplicador (i,j) de la eliminación de Neville de A.

Cuando no son necesarios los cambios de filas en el proceso de eliminación de Neville, A puede expresarse como

$$A = LU, \tag{2.17}$$

donde

$$L = L_1^{-1} L_2^{-1} \ldots L_{n-1}^{-1} = (E_n(m_{n1})\, E_{n-1}(m_{n-1,1}) \ldots\ E_2(m_{21}))\, (E_n(m_{n2}) \ldots E_3(m_{32})) \ldots E_n(m_{n,n-1}), \tag{2.18}$$

con U una matriz triangular superior y m_{ij} satisfaciendo (2.7).

En la factorización expresada en (2.17), U es una matriz triangular superior no singular, y en (2.14) se observa que la matriz L es triangular inferior con unos en la diagonal. Por la bien conocida unicidad de dichas factorizaciones se tiene que las matrices L y U, obtenidas mediante el proceso de eliminación de Neville con aritmética exacta, coinciden con las calculadas por eliminación de Gauss sobre la matriz A.

2.2 Algoritmo del Proceso de Eliminación de Neville

Sea $Ax=b$ un sistema de ecuaciones lineales no singular, con $A=(a_{ij})_{1\le i,j\le n}$ y $b=(b_i)_{1\le i\le n}$, supondremos que sobre la matriz A del sistema puede llevarse a cabo un proceso de eliminación de Neville por filas sin necesidad de realizar cambios de filas, esto es, $\tilde{A}^{(t)} = A^{(t)}$ para todo t.

Comenzando con $A^{(1)}=A$ y aplicando el proceso de eliminación, vamos obteniendo una sucesión de matrices intermedias

$$A^{(1)} \to A^{(2)} \to \ldots \to A^{(n)},$$

cuyos elementos denotaremos por $A^{(t+1)}=(a_{ij}^{(t+1)})_{1\le i,j\le n}$, con t = 1, 2, ..., n-1 y donde

$$a_{ij}^{(t+1)} = \begin{cases} a_{ij}^{(t)} & si\ 1 \le i \le t\ y\ 1 \le j \le n \\ a_{ij}^{(t)} - m_{it}\, a_{t-1,j}^{(t)} & si\ t+1 \le i,j \le n \end{cases} \tag{2.19}$$

siendo m_{it} el multiplicador (i,t) del proceso de eliminación.

De manera similar, el vector de términos independientes se va modificando; así, partiendo de $b^{(1)}=b$, se obtiene $b^{(1)} \to b^{(2)} \to \ldots \to b^{(n)}$, con $b^{(t+1)}=(b_i^{(t+1)})_{1\le i\le n}$, con t = 1, 2, ..., n-1 y donde

$$b_i^{(t+1)} = \begin{cases} b_i^{(t)} & si\ 1 \le i \le t \\ b_i^{(t)} - m_{it}\, b_{t-1}^{(t)} & si\ t+1 \le i \le n. \end{cases} \tag{2.20}$$

A continuación, mostramos el algoritmo de la eliminación de Neville obtenido a partir de las expresiones (2.19) y (2.20). Este algoritmo transforma un sistema de ecuaciones lineales $Ax=b$ en un sistema triangular superior $Ux=y$, pero no recoge el proceso de sustitución regresiva utilizado para resolver dicho sistema. Esto es debido a que este proceso ha sido estudiado habitualmente al abordar otros problemas, como por ejemplo, el proceso de eliminación de Gauss, por lo que si se desea puede verse en el capítulo 3 de Golub [Golub *et al* 96]. La matriz U ocupa las posiciones del triangulo superior de A y el vector y las posiciones del vector b de entrada. Los elementos que ocupan la posición (i,j) de la matriz A, serán denotados como $A[i,j]$, actuando de manera análoga para el vector b. Además, este algoritmo asume que $A[i,j] \neq 0$ cuando es usado como dividendo en la línea 4.

```
1.   procedimiento ELIMINACIÓN_NEVILLE (A, b)
2.      para j := 1 hasta n - 1 hacer    /* bucle externo */
3.         para i :=  n descendiendo hasta j + 1 hacer
4.            A[i, j] := A[i, j] / A[i – 1, j];          /* cálculo de multiplicadores */
5.            para r :=  j + 1 hasta n hacer
6.               A[i, r] := A[i, r] – A[i, j] * A[i – 1, r];    /* paso de eliminación */
7.            b[i] := b[i] – A[i, j] * b[i – 1];
8.         fin para;      /* línea 3 */
9.      fin para       /* línea 2 */
10.  fin ELIMINACIÓN_NEVILLE
```

Algoritmo 2.1 Eliminación de Neville secuencial

Hemos supuesto al comienzo de esta sección que sobre la matriz A del sistema se lleva a cabo un proceso de eliminación de Neville sin necesidad de realizar cambios de filas, si esto fuera necesario, al comienzo de cada etapa del proceso tendría lugar una reordenación de las filas de A y b de modo que si $a_{ik}=0$, $i \geq k$ entonces $a_{hk}=0$, $\forall\ h \geq i$.

A continuación analizaremos el número de pasos de computación básicos del Algoritmo 2.1, para lo cual calcularemos el número de operaciones que se realizan.

El número total de restas que se realizan en el proceso anteriormente descrito se recogen en la siguiente expresión:

$$\sum_{j=1}^{n-1}\sum_{i=j+1}^{n}\left(\sum_{r=j+1}^{n}1+1\right)=\frac{1}{3}n^3-\frac{1}{3}n\ . \tag{2.21}$$

El número de multiplicaciones coincide con el número de restas, por tanto

$$\sum_{j=1}^{n-1}\sum_{i=j+1}^{n}\left(\sum_{r=j+1}^{n}1+1\right)=\frac{1}{3}n^3-\frac{1}{3}n\ . \tag{2.22}$$

Y en cuanto al número de cocientes, se llevan a cabo

$$\sum_{j=1}^{n-1}\sum_{i=j+1}^{n} 1 = \frac{1}{2}n^2 - \frac{1}{2}n \ . \tag{2.23}$$

Consideraremos que cada operación aritmética emplea un tiempo t_c, así, el coste total de realizar un proceso de eliminación de Neville sobre la matriz A sin cambios de filas, será

$$T_{SECUENCIAL} = \left(\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n\right)t_c \ . \tag{2.24}$$

Por simplicidad, si nos centramos exclusivamente en las operaciones sobre la matriz A, despreciando las efectuadas sobre el vector b de términos independientes, se tiene que

$$T_{SECUENCIAL} = \left(\frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n\right)t_c \ . \tag{2.25}$$

2.3 Matrices Totalmente Positivas y Eliminación de Neville

Diremos que una matriz A es totalmente positiva (abreviadamente *TP*) si todos sus menores son no negativos. Un excelente informe sobre matrices totalmente positivas fue realizado por T. Ando en [Ando 87]. En la sección 2.4 veremos abundantes situaciones en las que aparecen matrices totalmente positivas.

En [Gasca *et al* 87], Gasca y Mühlbach introdujeron la idea general de la eliminación de Neville como estrategia de eliminación, aplicando este procedimiento para obtener un test que permitiera averiguar si una matriz era totalmente positiva. En los últimos años una serie de estudios han mostrado la potencia de este método de eliminación para trabajar con matrices totalmente positivas y sus subclases. Presentaremos a continuación algunos de los resultados más destacados.

El siguiente teorema nos proporciona un test para comprobar si una cierta matriz es totalmente positiva a través de la eliminación de Neville (ver Teorema 5.4 de [Gasca *et al* 92a]).

Teorema 2.2. Una matriz A es totalmente positiva si y sólo si la eliminación de Neville completa se puede llevar a cabo con pivotes no negativos y sin intercambiar filas o columnas salvo las nulas.

El Corolario *5.5* de [Gasca *et al* 92a] proporciona una caracterización para matrices no singulares totalmente positivas que simplifica la proporcionada en el teorema anterior:

Corolario 2.1. Sea A una matriz no singular, entonces A es totalmente positiva si y sólo si no son necesarios cambios de filas o columnas en la eliminación de Neville completa de A y todos los pivotes son no negativos.

Una matriz se dice estrictamente totalmente positiva si todos sus menores son positivos. Algunos ejemplos de este tipo de matrices son las matrices de Hilbert y las matrices de Vandermonde de la forma $\left(t_j^i\right)_{0\le i\le n-1,1\le j\le n}$ con $0 < t_1 < t_2 < \ldots < t_n$ (ver sección 7 de [Ando 87]).

En el Teorema 4.1 de [Gasca *et al* 94a], se prueba que una matriz cuadrada es estrictamente totalmente positiva si y sólo si se puede llevar a cabo la eliminación de Neville completa sin necesidad de realizar cambios de filas (llamaremos a esta condición *WRC*) y, los multiplicadores y los pivotes diagonales del proceso de eliminación de Neville completa son todos positivos.

Una clase importante de matrices totalmente positivas son las matrices casi estrictamente totalmente positivas. Una matriz A totalmente positiva de orden n se dice casi estrictamente totalmente positiva si todos sus menores son no negativos y éstos son estrictamente positivos si y sólo si los elementos diagonales son estrictamente positivos. Tales matrices aparecen a menudo en la práctica. Así, las matrices de colocación asociadas a los B-splines y las matrices de Hurwitz son casi estrictamente totalmente positivas. El siguiente teorema nos sirve para comprobar si una matriz no singular es casi estrictamente totalmente positiva, reduciendo el problema al estudio de submatrices con filas y columnas consecutivas:

Teorema 2.3. Sea A una matriz no singular. Entonces A es casi estrictamente totalmente positiva si y sólo si se satisfacen las dos siguientes condiciones:

i) Todos los menores de A son no negativos .

ii) Los menores formados por filas y columnas consecutivas son estrictamente positivos si y sólo si sus elementos diagonales son estrictamente positivos.

También podemos caracterizar una matriz casi estrictamente totalmente positiva no singular a través de su eliminación de Neville:

Teorema 2.4. Una matriz no singular A es casi estrictamente totalmente positiva si y sólo si la eliminación de Neville completa de A puede ser realizada sin cambios de filas o columnas y los pivotes son no nulos si y sólo si los correspondientes elementos de A lo son.

De manera análoga a lo hecho para matrices totalmente positivas y estrictamente totalmente positivas, podemos caracterizar las matrices casi estrictamente totalmente positivas en términos de sus factorizaciones.

Por otro lado, mencionemos que la eliminación de Neville también ha servido para obtener un test que permite comprobar si una matriz es estrictamente signo regular, es decir, si todos los menores del mismo orden tienen el mismo signo para todos los órdenes, así como para analizar si una matriz rectangular tiene los menores de orden máximo del mismo signo (ver [Peña 95]).

La eliminación de Neville también ha sido útil para mejorar las caracterizaciones por menores de las matrices totalmente positivas y sus subclases. Como se puede ver en los trabajos de Gasca y Peña [Gasca *et al* 92a] [Gasca *et al* 93], tales caracterizaciones reducen considerablemente el número de menores usados en otras caracterizaciones previas. Así, en el Teorema 3.1 de [Gasca *et al* 93] se mejora la caracterización de Cryer [Cryer 73] para matrices totalmente positivas; para el caso de matrices estrictamente totalmente positivas, el teorema 4.1 de [Gasca *et al* 92a] reduce el número de menores necesarios para caracterizar tales matrices, respecto al estudio realizado por Fekete y Pólya en [Fekete *et al* 12], ya que basta con ver el signo de los menores que usan columnas consecutivas iniciales y filas consecutivas y los que usan filas consecutivas iniciales y columnas consecutivas en vez de tener que comprobar el signo de todos los menores con filas y columnas consecutivas.

Recientemente, Gasca y Peña (ver [Gasca *et al* 06]) han completado las caracterizaciones realizadas para matrices casi estrictamente total positivas.

En lo que respecta al coste computacional de la eliminación de Neville, es, por lo general, del mismo orden que el de la eliminación de Gauss, pero para algunas familias de matrices el coste de Neville es menor que el de Gauss. Esto ocurre, por ejemplo, para ciertas matrices A, donde en la descomposición de A en la forma LU, la matriz L resulta ser la inversa de una matriz tipo banda.

Conviene recordar dos resultados interesantes obtenidos en [Gasca *et al* 94a]. Por un lado, si A es no singular, admite eliminación de Neville sin cambios de filas si y sólo si es descomponible LU y L la admite. En ese caso los multiplicadores de A y L son los mismos. Que lo mismo sucede en la eliminación gaussiana es ya conocido.

Pero por otro lado, en el mismo trabajo (ver Teorema 2.7) se demuestra que una matriz A no singular, descomponible LU, admite eliminación de Neville sin cambios de filas si y sólo si lo mismo sucede con otra $B=L^{-1}V$ con V triangular superior. En este caso el número de multiplicadores no nulos de la eliminación de Neville de A y B es el mismo, y si L es de diagonal unidad unos multiplicadores y otros son opuestos, aunque usados en general en distinto orden.

Así pues, tomando $V=I$ en lo anterior, se deduce que las transformaciones elementales (es decir, los multiplicadores) que llevan la matriz $A=LU$ a U son las mismas que las que llevan L a I, tanto en la eliminación de Neville como de Gauss. En el caso de la eliminación de Neville son inversas (esto es, los multiplicadores son los opuestos), pero usados en distinto orden, a las que llevan por esa eliminación L^{-1} a I, lo cual no sucede en la de Gauss.

Si L^{-1} es una matriz bandeada, y como caso particular es bidiagonal, en su eliminación de Neville y en la de Gauss se ahorran muchos cálculos. Pero como en general L es triangular inferior llena, si realizamos en ella o en $A=LU$ eliminación gaussiana no ahorraremos ningún cálculo. En cambio el coste de realizar eliminación de Neville a L o A es el mismo que el de L^{-1}. En general se puede estimar que el número de multiplicadores no nulos de la eliminación de Neville de A es el mínimo de multiplicadores no nulos que resulten en las eliminaciones gaussianas de L y L^{-1}.

Conviene observar que los papeles de L y L^{-1} en todo esto pueden ser intercambiados.

Para ilustrar lo anterior, tomemos como ejemplo la siguiente matriz triangular superior y bidiagonal

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{pmatrix}.$$

Si realizamos un proceso de eliminación de Neville sobre la matriz L para llevarla a forma triangular superior, en este caso diagonal, el número de operaciones necesarias para llevar a cabo el proceso coincide con el que sería necesario si deseamos utilizar eliminación gaussiana.

Construyamos ahora la matriz inversa de L

$$L^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 1 & 0 & 0 \\ 9 & -3 & 1 & 0 \\ -27 & 9 & -3 & 1 \end{pmatrix}.$$

Al llevar a cabo eliminación gaussiana sobre L^{-1} podemos observar que necesitamos seis pasos en la eliminación. Con este proceso haremos ceros en las posiciones (2,1), (3,1), (4,1), (3,2), (4,2) y (4,3). Sin embargo, si aplicamos eliminación de Neville bastará utilizar tres pasos para conseguir nuestro propósito. Si hacemos cero en la posición (4,1), con $m_{41}=3$, vemos que la última fila se transforma en (0,0,0,1); utilizando $m_{31}=3$ la tercera fila pasa a ser (0,0,1,0) y, por último, con m_{21}=3 la segunda fila resulta ahora (0,1,0,0), consiguiendo así con tres pasos (los mismos que son necesarios en L) una matriz diagonal.

Además de lo ya comentado, también se ha abordado el análisis de error asociado al proceso de eliminación de Neville con aritmética de precisión finita, en sus dos vertientes, *backward* y f*orward* (ver [Alonso 95]). Tanto los resultados teóricos, como los correspondientes experimentos numéricos, muestran que el comportamiento de la eliminación de Neville respecto al error, es similar al de Gauss (ver [Wilkinson 63] y [Wilkinson 65]).

En el caso del error *backward* se ha probado en [Alonso *et al* 97] que para matrices totalmente positivas, las cotas alcanzadas (funciones de la unidad de redondeo y de la matriz) son mejores que las calculadas por de Boor y Pinkus [Boor *et al* 77] para el caso gaussiano. Para el error *forward* se han obtenido aproximaciones lineales del error y del residuo (ver [Alonso *et al* 98]) utilizando técnicas similares a las desarrolladas por Stummel en [Stummel 85].

Ideas similares a las desarrolladas en la eliminación de Neville han sido abordadas por otros autores con el fin de obtener estrategias de pivotaje eficientes para resolver sistemas lineales en

computación paralela. Estas estrategias han sido llamadas *pairwise pivoting* (ver [Davis *et al* 88], [Gallivan *et al* 90], [Sorensen 85] y [Tiskin 07]) y con ellas se intenta reducir el tiempo dedicado a las comunicaciones en el proceso de eliminación.

La búsqueda de estrategias de pivotaje eficientes, también ha llevado en los últimos años al profesor Peña a utilizar la eliminación de Neville sobre cierto tipo de matrices estructuradas (en particular matrices signo-regulares no singulares) para obtener factores de crecimiento óptimos con un coste computacional reducido (ver [Cortes *et al* 07]). También podemos observar el uso de Neville sobre matrices estructuras en [Gemignani 08].

Por último, conviene recordar que en [Alonso 95] se inicia el estudio del comportamiento de la eliminación de Neville sobre arquitecturas multiprocesador, comprobando como el coste de las comunicaciones asociado al método de Neville es algo menor que en Gauss, lo cual es debido a que éstas se producen entre procesadores vecinos.

2.4 Aplicaciones

Ya hemos comentado anteriormente que el proceso de eliminación de Neville surge de manera natural a partir de la estrategia de Neville para resolver el problema de interpolación polinomial clásico. La idea de la consecutividad que aparece en la eliminación de Neville, la podemos ver también en procesos de extrapolación, como por ejemplo en la extrapolación de Richardson (ver [Gasca *et al* 95]). Por otro lado, la eliminación de Neville se ha utilizado para caracterizar matrices totalmente positivas, estrictamente totalmente positivas y casi estrictamente totalmente positivas, pudiendo ver algunos de estos resultados en la sección anterior.

Conceptos íntimamente ligados al de total positividad son los de sistemas de Tchebyshev de funciones (un sistema de funciones se llama de Tchebyshev si todas sus matrices de colocación cuadradas tienen determinante mayor estrictamente que cero), las funciones de Green asociadas a un gran número de problemas de contorno de Sturm-Liouville, etc.

Concretamente, del estudio de las vibraciones en sistemas mecánicos acoplados llevado a cabo por Gantmacher y Krein (ver [Gantmacher *et al* 35] y [Gantmacher *et al* 37]), se deriva la primera monografía sobre matrices totalmente positivas. En él establecían notables propiedades espectrales para este tipo de matrices. Anteriormente, Schoenberg [Schoenberg 30] descubrió la conexión entre la total no negatividad y la propiedad de variación de los signos.

El importante libro de Karlin [Karlin 68], que sigue siendo después de muchos años referencia básica y obligada en este tema, recogió y amplió gran parte de las investigaciones de Schoenberg sobre positividad total. Además, Karlin aplicó este concepto a la Economía, y en particular, a la teoría de inventarios.

En el caso de Schoenberg, aquellas investigaciones estaban relacionadas con la teoría de *splines*. Estudió la interpolación mediante funciones *splines*, introdujo los *B-splines* y conjeturó la total positividad de la matriz de colocación de los *B-splines*, lo cual fue probado por Karlin en su libro. De hecho, dicha matriz pertenece a la clase de matrices casi estrictamente totalmente positivas. A esta clase también pertenecen las matrices de Hurwitz, que proveen un criterio para que un polinomio tenga sus ceros con parte real estrictamente negativa, propiedad que tiene importantes aplicaciones en el estudio de la estabilidad de sistemas dinámicos.

La total positividad también aparece asociada a otras ramas de la ciencia, como la estadística, destacando los trabajos de Karlin, problemas de combinatoria, muchas técnicas asociadas a la total positividad pueden ser usadas para resolver problemas unimodales (ver [Brenti 95], [Fomin *et al* 00] y [Fomin 01]), teoría de grafos [Peña 98], teoría de politopos (ver [Sturmfels 88] y [Chapoton *et al* 02]), economía [Bapat *et al* 97], programación lineal [Berman *et al* 94], tomografía por impedancia eléctrica, etc.

Algunas de las investigaciones en éstos y otros campos pueden verse en *Total Positivity and Its Applications* (ver [Gasca et al 96]).

En cualquier caso, desde el momento en que Schoenberg y Karlin detectan la relación entre las matrices totalmente positivas y los problemas de interpolación con *splines*, el estudio de este tipo de matrices se desarrolla dentro de la Teoría de Aproximación, más concretamente en sus aplicaciones al diseño geométrico asistido por computador *(CAGD)* (ver [Carnicer *et al* 94], [Peña 99] y [Delgado *et al* 07]).

Cuando nos planteamos un problema de *CAGD* deseamos diseñar una curva (o una superficie) que responda a un cierto perfil, imitando las propiedades geométricas de un cierto modelo. Lo que se pretende es que el diseñador controle la curva y que modificando ciertos parámetros de control ésta vaya cambiando hasta alcanzar una forma satisfactoria para aquel. Centrándonos en el caso de curvas planas, la forma de enfocar el problema es diseñar una línea poligonal (polígono de control), cuya forma debe ser imitada por la curva.

Tras decidir a que espacio funcional pertenecerán las componentes paramétricas de la curva, se elige una base de ese espacio, sea la formada por las funciones $(u_0,u_1,\ldots,u_n)$ definidas en un intervalo $[a,b]$, y se construye el polígono de control.

Este sistema de funciones se puede caracterizar por el hecho de que sus matrices de colocación, correspondientes a la base usada, pertenecen a uno de los tipos de matrices considerados en la sección anterior, es decir, matrices totalmente positivas, matrices estrictamente totalmente positivas y matrices casi estrictamente totalmente positivas. Por otro lado, en esa sección 2.3 vimos como estas clases de matrices pueden ser caracterizadas a través de su eliminación de Neville. Así, las propiedades de los sistemas de funciones se pueden determinar frecuentemente por las propiedades que obtengamos al aplicar la eliminación de Neville a sus matrices de colocación.

Un sistema de funciones se dice que es totalmente positivo si todas sus matrices de colocación son totalmente positivas. El sistema es totalmente positivo y normalizado *(NTP)* si es totalmente positivo y $\sum_{i=0}^{n} u_i = 1$.

Pues bien, los sistemas de funciones *NTP*, al ser usados para la construcción de curvas, satisfacen interesantes propiedades de preservación de formas. La curva construida pertenece a la envolvente convexa del polígono de control, generando una imitación suave del mismo, produciéndose una disminución de la variación respecto a éste. Además la curva comienza y termina respectivamente, en el primer y último vértice del polígono de control, siendo tangente al primer y último lado del polígono (propiedad de interpolación en los extremos).

Ejemplos típicos de sistemas de funciones *NTP* son los polinomios de Bernstein en el intervalo [0,1], la base de los *B-splines* en el caso de usar funciones *spline* con una sucesión dada de nudos, o la de los *B-splines* en el caso de funciones *spline* generalizadas con condiciones de continuidad prescritas.

Uno de los problemas más interesantes que ha surgido en los últimos años es la búsqueda de bases óptimas en el sentido de que para una curva dada, el polígono de control que mejor imite a la curva sea el asociado a esa base.

En [Carnicer *et al* 94], Carnicer y Peña demostraron la optimalidad en sus respectivos espacios de algunas bases de funciones bien conocidas como polinomios de Bernstein y *B*-splines en el contexto de representaciones que preserven formas, presentando un método de construcción de bases óptimas inspirado en el proceso de eliminación de Neville.

Dado un cierto vector *x* de *n* componentes reales, se denota por $V^-(x)$ al número de cambios de signo que presenta la sucesión de sus componentes excluyendo las nulas. Entre otras muchas propiedades las matrices totalmente positivas *A* de dimensión *m x n*, tienen la de que su producto por *x* causa una disminución de la variación de éste, $V^-(Ax) \leq V^-(x)$, para cualquier *x* de R^n.

El hecho fundamental que relaciona la eliminación de Neville con la disminución de la variación es que aunque dicha eliminación produce, obviamente, la misma factorización de la matriz como producto de una triangular inferior por una superior, las matrices elementales del proceso, cuyo producto da lugar a la triangular inferior, son bidiagonales en la eliminación de Neville mientras que no lo son en la gaussiana. Las matrices bidiagonales si usan sólo elementos positivos son *TP*, y por consiguiente su producto también. Y si además son estocásticas (matrices con elementos no negativos y suma 1 en cada una de sus filas) dan lugar a lo que en diseño gráfico se conoce como algoritmos de recorte de esquinas *(corner cutting algorithms)*, que es el algoritmo típico que aplicado al polígono de control disminuye la variación de éste.

La factorización de matrices *TP* utilizando matrices bidiagonales, también ha sido utilizada por Fallat en [Fallat 01].

Los algoritmos del álgebra lineal numérica constituyen un cuello de botella para la eficiencia y precisión de muchos de los algoritmos que aparecen en el resto de las disciplinas numéricas, es por ello especialmente importante que su comportamiento sea eficaz desde distintos puntos de vista. A este respecto, en los últimos años hay una gran actividad investigadora en aquellos métodos que nos permitan dar una aproximación de los valores y vectores propios de una matriz con alta precisión relativa. Este tipo de algoritmos permiten asegurar un elevado número de dígitos correctos, incluso para los autovalores más pequeños, con un reducido coste computacional, utilizando para ello distintas características de la eliminación de Neville y de algunos tipos de matrices para las que este procedimiento es adecuado (ver [Demmel *et al* 99], [Demmel *et al* 04], [Dopico *et al* 06] y [Koev 07]).

Otro ejemplo del uso del método de Neville para obtener algoritmos numéricos eficientes, se puede observar en [Demmel *et al* 05], donde Demmel y Koev utilizan ciertas propiedades de este método para conseguir algoritmos de alta precisión relativa para matrices de Vandermonde totalmente positivas.

Otras propiedades y aplicaciones de la eliminación de Neville, así como de las matrices totalmente positivas, pueden encontrarse en la bibliografía.

3

Rendimiento y Escalabilidad de Sistemas Paralelos

Un algoritmo secuencial es evaluado en términos de su tiempo de ejecución, expresado éste como función del tamaño de la entrada. El tiempo de ejecución de un algoritmo paralelo depende no sólo del tamaño del problema sino también del número de elementos de proceso utilizados y sus relativas velocidades de cálculo y de comunicación, etc. Por ello, un algoritmo paralelo no puede ser evaluado aisladamente de la arquitectura paralela. La combinación de un algoritmo y de la arquitectura paralela en la que está implementado es lo que se denomina *sistema paralelo*.

En este capítulo, en primer lugar describiremos y clasificaremos las arquitecturas paralelas actuales, detallando las plataformas que utilizaremos en la experimentación. En la sección 3.2 expondremos el modelo de algoritmo paralelo desarrollado. Posteriormente, en el apartado 3.3 mostraremos el paradigma de programación usado así como el software utilizado para realizar la validación y evaluación de los algoritmos. Ya en la sección 3.4 expondremos las operaciones básicas de comunicación indicando el coste de aquellas que serán utilizadas por nuestros algoritmos y que nos ayudarán a estimar los tiempos de comunicación de los mismos. En la sección 3.5, recogeremos las técnicas habituales de evaluación de prestaciones de algoritmos paralelos que usaremos en posteriores capítulos en los que se realizará un estudio exhaustivo de distintas formulaciones paralelas de la eliminación de Neville. Por último, en la sección 3.6 presentamos los valores estimados en nuestros entornos de experimentación para aquellas variables que surgen en las expresiones de las secciones previas.

3.1 Modelos de Computadores Paralelos

Los computadores secuenciales tradicionales están basados en el modelo introducido en la década de los cuarenta por John von Neumann, el cual consiste en una unidad central de proceso *(CPU)* y una memoria. Este modelo computacional emplea una secuencia de instrucciones y opera en una secuencia de datos. Los computadores de este tipo se denominan computadores *SISD* (*Single Instruction Single Data*).

La velocidad de un computador *SISD* está limitada por dos factores: la rapidez en la ejecución de instrucciones y la velocidad en la que la información es intercambiada entre memoria y *CPU*. Esta última puede ser incrementada a base de aumentar el número de canales a través de los cuales se puede acceder a los datos de modo independiente. Esto se realiza dividiendo la memoria en un número de bancos, accediéndose a cada uno de ellos de forma independiente (*memory interleaving*).

Otra forma de mejorar esa velocidad es mediante el uso de memorias rápidas de enlace (memorias *cache*), que actúan de intermediario entre la memoria principal, mucho más lenta, y la *CPU*, incrementando sustancialmente el rendimiento del sistema.

Por otro lado, el ritmo de ejecución de instrucciones puede ser incrementado mediante la ejecución solapada de distintas etapas de varias instrucciones (*pipelining*).

Estas tres técnicas de mejora de la velocidad de un computador *SISD* son usadas comúnmente en los computadores *SISD* de alto rendimiento; no obstante tienen sus limitaciones.

Una forma alternativa para incrementar la tasa de ejecución de instrucciones es usar múltiples unidades de proceso y múltiples unidades de memoria interconectadas de algún modo (*paralelismo*). La velocidad de procesamiento de dicho sistema crece cuando el número de *CPUs* y unidades de memoria aumenta.

Si pensamos en el número de procesadores utilizados en la construcción del computador, nos estaremos refiriendo a la *granularidad*. Así, un computador paralelo puede estar compuesto por un número pequeño de procesadores con una gran potencia, tanto de cálculo como de comunicación, o por un gran número de procesadores cuya capacidad es menor. Los computadores del primer tipo se denominan *computadores de grano grueso*, mientras que los segundos, son *computadores de grano fino*.

Además de lo ya comentado, los computadores difieren en otras características tales como la multiplicidad del flujo de instrucciones y de datos, la organización del espacio de direcciones o la red de interconexión.

3.1.1 Multiplicidad del Flujo de Instrucciones y de Datos

En 1966 Michael J. Flynn propuso una forma de clasificación de las computadoras. La taxonomía de Flynn [Flynn 66] es la manera clásica de organizar las computadoras, aunque no cubre todas las posibles arquitecturas actuales. Esta organización se basa en la pluralidad de instrucciones y de datos que la computadora utiliza para procesar información. Puede haber secuencias de instrucciones sencillas o múltiples y secuencias de datos sencillas o múltiples. Es precisamente en esta división cuando aparecen por primera vez las máquinas paralelas divididas en tres clases:

- *SIMD (Single Instruction Multiple Data)*
- *MISD (Multiple Instruction Single Data) y*
- *MIMD (Multiple Instruction Multiple Data).*

De estas tres clases, la más general y que ofrece mayor paralelismo es la que sigue el esquema *MIMD*. Aunque este tipo de modelo de paralelismo es el más complejo, es el que se ha impuesto últimamente.

La clasificación de Flynn pone de manifiesto dos tipos de paralelismo, de datos y funcional. El primero de ellos se da cuando la misma función o instrucción se ejecuta repetidas veces con datos distintos y se da en los modelos *SIMD* y puede darse en los *MIMD* si la aplicación se implementa como un programa paralelo *SPMD* (*Single Program Multiple Data*). El paralelismo funcional ocurre cuando diferentes funciones o instrucciones se ejecutan en paralelo con distintos datos (modelos *MIMD*). Este paralelismo funcional puede darse a nivel de instrucciones, a nivel de bucle, a nivel de funciones o a nivel de programas.

3.1.2 Organización del Espacio de Direcciones

A la hora de solucionar cooperativamente un problema en un conjunto de procesadores se requiere la interacción de éstos. Las arquitecturas de espacio de direcciones compartido y de espacio de direcciones distribuido proporcionan dos formas diferentes de esa comunicación.

La *arquitectura de espacio de direcciones compartido* (figura 3.1) proporciona un soporte *hardware* para los accesos de lectura y escritura de todos los procesadores a un espacio de direcciones compartido. Los procesadores se relacionan mediante la modificación de los datos almacenados en el espacio de direcciones compartido. El cuello de botella en este tipo de sistemas se encuentra precisamente en el acceso intensivo a la memoria por parte de los procesadores. Estos computadores se clasifican en dos categorías atendiendo a la uniformidad, o no, en el tiempo de acceso a todas las posiciones del espacio de memoria: *UMA* (*Uniform Memory Access*) y *NUMA* (*NonUniform Memory Access*).

La mayoría de los computadores de memoria compartida tienen una memoria *cache* local en cada procesador para incrementar así el ancho de banda efectivo entre memoria y procesador. El uso de las memorias *cache* introduce *el problema de la coherencia*. Este problema ocurre cuando un procesador modifica una variable compartida en su memoria *cache*; tras esta modificación, los diferentes procesadores tienen diferentes valores de la variable, salvo que las copias de la variable en sus memorias *cache* se invaliden o se modifiquen de manera simultánea. Los computadores de memoria compartida se denominan habitualmente *multiprocesadores*. Ejemplos de este tipo de computadores paralelos son *C.nmp*, *NYU Ultracomputer* y *Finis Terrae* del Centro de Supercomputación de Galicia (*CESGA*).

En una *arquitectura de memoria distribuida* (figura 3.2), los procesadores están conectados usando una red de interconexión y no comparten recursos. Cada procesador tiene su propia memoria local, la cual es accesible sólo desde ese procesador, y ejecuta su propia copia del sistema paralelo. Obviamente la relación entre los distintos procesadores se realiza mediante el intercambio de información, siendo el *paradigma de paso de mensaje* la técnica más empleada. Los computadores *MIMD* de memoria distribuida se denominan comúnmente *multicomputadores*. La programación es más compleja que en el caso de memoria compartida, ya que cuando los datos a usar por un procesador están en el espacio de direcciones de otro hace falta solicitarlas y transferirlas a través de mensajes. Así, es preciso potenciar la localidad de los datos para minimizar la comunicación entre procesadores y obtener un buen rendimiento. La ventaja que proporcionan es su escalabilidad, es decir, el sistema puede crecer a un número más grande de procesadores que los sistemas de memoria compartida y, por tanto, es más idóneo para las máquinas paralelas. Ejemplos de computadores paralelos de memoria distribuida son *Cosmic Cube*, *Paragon XP/S*, *iPSC*, *CM-E*, *nCUBE2*, *Mare Nostrum* del Centro Nacional de Supercomputación (*CNS*), grupos de estaciones de trabajo (*PCs*), etc. Este último es un exponente de *MIMD* de memoria distribuida débilmente acoplado, que está teniendo un gran auge por sus prestaciones y bajo precio.

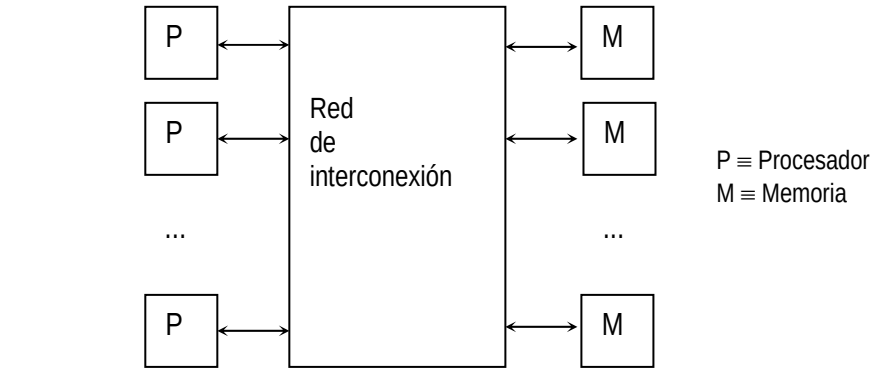


Figura 3.1 Típica arquitectura de espacio de direcciones compartido

Hay un tercer tipo de organización, la *memoria distribuida-compartida*, que combina las ventajas de las dos organizaciones: la memoria está físicamente distribuida y, por tanto, el sistema es escalable, pero se accede con un espacio único de direcciones y, consecuentemente, es fácilmente programable.

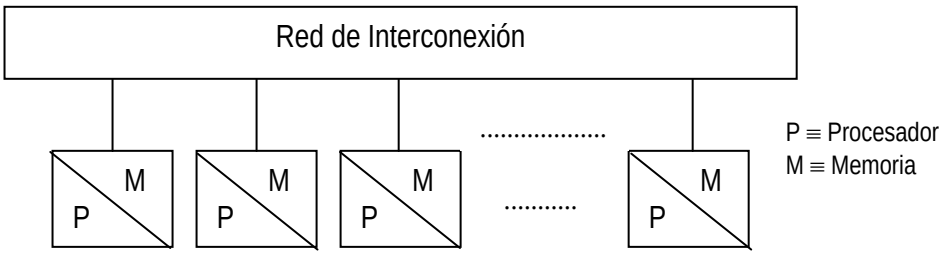


Figura 3.2 Típica arquitectura de paso de mensajes

3.1.3 Redes de Interconexión

Los computadores de espacio de direcciones compartido y distribuido pueden ser construidos mediante la conexión de procesadores y de unidades de memoria de acuerdo a una variedad de redes de interconexión.

Atendiendo a la posibilidad de cambiar las conexiones entre los nodos, podemos encontrar redes estáticas o dinámicas [Almasi *et al* 94].

Una *red estática* es una red cuya topología queda definida de manera definitiva y estable durante la construcción de la máquina paralela. La red simplemente une los diversos elementos de acuerdo a una configuración dada. Se utiliza sobre todo en el caso de los multicomputadores para conectar los diversos procesadores que posee la máquina. Por la red sólo circulan los mensajes entre procesadores, por lo que se dice que la red presenta un *acoplamiento débil*.

Una *red dinámica* es una red cuya topología puede variar durante el curso de la ejecución de un programa paralelo o entre dos ejecuciones de programas. Las redes dinámicas se utilizan sobre todo en los multiprocesadores. En este caso, la red une los procesadores a los bancos de memoria central. Cualquier acceso de un procesador a la memoria, bien sea para acceder a los datos o a las instrucciones, debe pasar a través de la red, por lo que se dice que la red tiene un *acoplamiento fuerte*. La red debe poseer un rendimiento muy elevado para no demorar demasiado a los procesadores que acceden a memoria.

Topologías de Red

En las redes de interconexión se han utilizado una amplia variedad de topologías de red como las siguientes:

- *Red basada en bus*, es quizás la red más simple y en ella, todos los elementos están conectados a un bus rápido.
- *Red crossbar*, es una forma simple de conectar *p* procesadores a *b* bancos de memoria y consiste en una malla de *p* x *b* elementos de conmutación. Es una red sin bloqueo en el sentido de que la conexión de un nodo de proceso a un banco de memoria no obstaculiza la conexión de los demás nodos de proceso a otros bancos de memoria.
- *Red multietapa*, tiene como objetivo mejorar, tanto como sea posible, el rendimiento de la red crossbar, haciendo uso de un número menor de conmutadores, pero, eso sí, con más tiempo para su recorrido. Una red de conexión multietapa comúnmente utilizada es la red *Omega*.
- *Red conectada completamente*, en este tipo de red cada procesador tiene un enlace directo con cada uno de los restantes procesadores de la red.

- *Red de conexión en estrella*, donde un procesador actúa como procesador central. Cada uno de los restantes procesadores tiene un enlace directo que lo conecta a ese procesador central.
- *Array lineal*, es una forma simple de conectar los procesadores. Cada procesador en esta red tiene conexión directa con dos procesadores (figura 3.3), excepto los extremos que sólo la tienen con uno. Si los procesadores de los extremos están conectados entre sí, entonces ese tipo de red se denomina *anillo*.

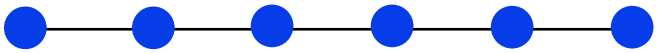


Figura 3.3 *Array* lineal de *6* procesadores

- *Malla*, una *malla abierta* de dos dimensiones es la extensión del *array* lineal al plano. En una malla abierta de dos dimensiones, cada procesador tiene conexión directa con cuatro procesadores, excepto los extremos que sólo la tienen con dos. La siguiente figura ilustra una malla de k^2 procesadores con $k = 6$. Si los procesadores extremos tuvieran también conexión con cuatro procesadores, se denominaría *malla cerrada*.

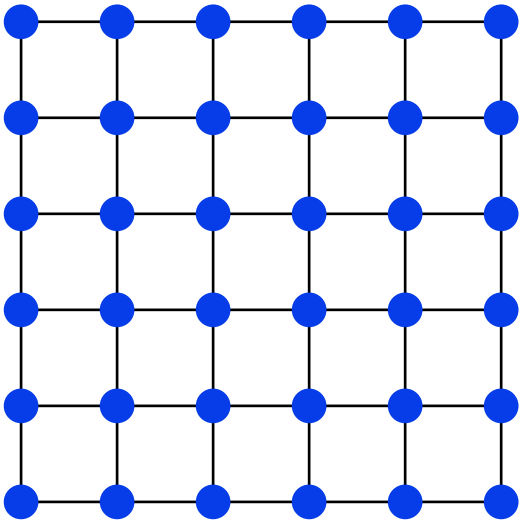


Figura 3.4 Malla de dos dimensiones con *36* procesadores

- *Árbol*, es una red en la que sólo existe un camino entre cualquier par de procesadores. Los tipos de red *array* lineal y estrella son clases especiales del tipo de red árbol.

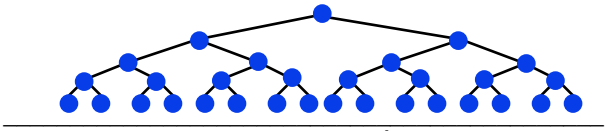


Figura 3.5 Árbol

- *Hipercubo*, este tipo de red es una malla multidimensional de procesadores con exactamente dos procesadores en cada dimensión. Un hipercubo de dimensión *d* consiste en $p = 2^d$ procesadores. Un hipercubo puede construirse recursivamente del siguiente modo: un hipercubo de dimensión 0 es un único procesador, un hipercubo de dimensión 1 se construye conectando dos hipercubos de dimensión 0; en general, un hipercubo de dimensión $d+1$ se forma conectando los procesadores correspondientes de dos hipercubos de dimensión *d*. Este tipo de red es una buena arquitectura por su bajo diámetro. En las siguientes figuras aparecen hipercubos de dimensiones 0, 1, 2, 3 y 4.

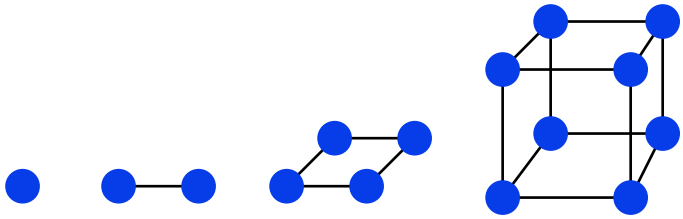


Figura 3.6 Hipercubos de dimensiones 0, 1, 2 y 3

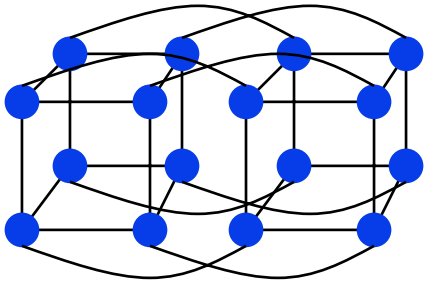


Figura 3.7 Hipercubo de dimensión 4

El hipercubo admite la codificación *Gray*, donde cada nodo se numera de forma binaria con tantos dígitos como dimensión tenga el hipercubo. Partiendo de una etiqueta, se obtiene la etiqueta de los nodos conectados directamente cambiando uno de sus bits. Esto ayuda, por ejemplo, al diseño de algoritmos de encaminamiento.

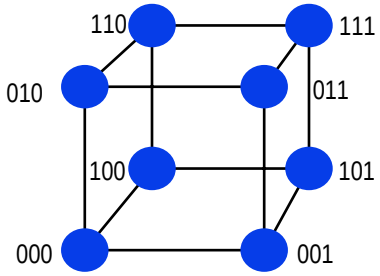


Figura 3.8 Etiquetas en los procesadores de un hipercubo de dimensión 3

Una de las mayores sobrecargas en la ejecución de programas paralelos proviene del intercambio de información entre los elementos de proceso. El coste de la comunicación es dependiente de una variedad de características como la semántica del modelo de programación, la topología de la red, el manejo de los datos, etc.

El tiempo empleado en comunicar un mensaje entre dos procesadores en sistemas distribuidos es la suma del tiempo que se dedica a preparar un mensaje para su transmisión y el tiempo empleado por el mensaje en atravesar la red hasta su destino. Los principales parámetros que determinan el coste de la comunicación son los siguientes:

- *Tiempo de arranque* (t_s), es el tiempo requerido para manejar un mensaje en los nodos emisor y receptor. Esto incluye el tiempo para preparar el mensaje, el tiempo para ejecutar el algoritmo de encaminamiento y el tiempo para establecer un interfaz entre el procesador local y el *router*. Este tiempo sólo se produce una vez para la transferencia de un mensaje simple.
- *Tiempo de transferencia por palabra* (t_w), si el ancho de banda de un canal es r palabras por segundo, entonces cada palabra emplea un tiempo $t_w = 1/r$ para atravesar esa unión. Este tiempo se denomina tiempo de transferencia por palabra.
- *Tiempo por nodo* (t_h), una vez que un mensaje abandona un procesador, emplea una cantidad finita de tiempo en alcanzar el siguiente procesador en su camino. El tiempo empleado por la cabecera del mensaje para viajar entre dos procesadores conectados directamente en la red es lo que se denomina tiempo por nodo.

Uno de los factores que influye en los costes de comunicación en una red es la técnica de encaminamiento utilizada. Las dos técnicas de conmutación típicas para hacer llegar la información a su destino en los computadores paralelos se denominan almacenar-reenviar y segmentación.

En el caso de *almacenar-reenviar*, cuando un mensaje atraviesa un camino con múltiples enlaces, cada procesador intermedio envía el mensaje al siguiente procesador después de recibir y almacenar el mensaje completo. Supongamos que un mensaje de tamaño m se va a transmitir por una red, la cual tiene e enlaces, en cada enlace, el mensaje incurre en un coste t_h debido a la cabecera y mt_w por el resto del mensaje que atraviesa el enlace. Dado que hay e enlaces, el coste total es $(t_h+mt_w)e$. Por tanto, para la técnica almacenar-reenviar, el coste total de enviar un mensaje de m palabras a través de e enlaces de comunicación es

$$T_{ENVÍO\_ALMACENAR\_REENVIAR} = t_s + (t_h + m\, t_w)\, e. \tag{3.1}$$

En los actuales computadores paralelos [Kumar *et al* 03], el tiempo t_h es bastante pequeño, por lo que para la mayoría de los algoritmos paralelos es mucho menor que mt_w, incluso para valores pequeños de m, y por tanto puede ser ignorado, simplificando así la ecuación anterior del siguiente modo

$$T_{ENVÍO\_ALMACENAR\_REENVIAR} = t_s + e\, m\, t_w. \tag{3.2}$$

La técnica de *segmentación* consiste en dividir el mensaje en unidades de tamaño fijo denominadas unidades de control de flujo o *flits*. La primera unidad es enviada desde el nodo origen al nodo destino para establecer de ese modo la comunicación. Una vez que ésta se haya establecido, el resto de unidades serán enviadas una tras otra. Un procesador intermedio no esperará por el mensaje completo; en cuanto le llegue un trozo del mensaje original, lo pasará al siguiente nodo. Por lo tanto, esta técnica emplea menos memoria y menos ancho de banda en los procesadores intermedios y es más rápida.

Si consideramos un mensaje que está atravesando la red de ese modo y el mensaje tiene que recorrer e enlaces, entonces la cabecera del mensaje emplea un tiempo et_h para alcanzar su destino. Además, si el mensaje tiene m palabras, entonces el mensaje completo tardará un tiempo mt_w tras la llegada de la cabecera del mensaje. Así, el tiempo total de comunicación viene dado por

$$T_{ENVÍO\_SEGMENTACIÓN} = t_s + e\, t_h + m\, t_w. \tag{3.3}$$

Este tiempo es una mejora sobre el tiempo de almacenar-reenviar ya que, dejando aparte el tiempo de arranque (t_s), vemos que el coste de enviar un mensaje en (3.3) es $O(e+m)$ mientras que el coste que se muestra en (3.2) es $O(em)$. Nótese que en el caso de que la comunicación fuera entre procesadores vecinos o el tamaño del mensaje fuese pequeño, entonces los tiempos de ambos esquemas serían similares.

La mayoría de los computadores paralelos actuales y muchas redes de área local soportan la técnica de segmentación, siendo determinado el tamaño de un *flit* por una variedad de parámetros de la red, al igual que sucede con las constantes t_s, t_w y t_h.

La ecuación 3.3 nos indica que para optimizar el coste de transferencia de mensajes deberíamos:

- Priorizar los mensajes largos frente a los cortos, de ese modo ahorraríamos el tiempo de arranque. Esto es debido a que en plataformas como *clusters* y máquinas de paso de mensajes el valor de t_s es mucho mayor que t_w y t_h.
- Disminuir el volumen de datos, para minimizar la sobrecarga debida a t_w.
- Minimizar la distancia de la transferencia de datos, reduciendo de ese modo el término asociado a t_h.

Se puede apreciar fácilmente que los dos primeros objetivos pueden ser considerados a la hora de diseñar el algoritmo paralelo, no sucede lo mismo con el tercero de ellos. Esto es debido a una serie de características de las plataformas paralelas. Una de ellas es el hecho de que en la mayoría de las librerías de paso de mensajes, el programador tiene muy poco control en la relación entre los procesos a realizar y los procesadores físicos. En estos paradigmas, ocurre que mientras en el diseño del sistema paralelo las tareas están bien definidas y las comunicaciones se realizan entre procesadores vecinos en la topología de red considerada, a la hora de ejecutarse ese planteamiento puede no mantener la estructura previamente diseñada.

Por otro lado, el tiempo por nodo está dominado habitualmente o bien por t_s, en el caso de mensajes cortos; o bien por mt_w en mensajes largos. Dado que el número máximo de nodos en la mayoría de la redes es relativamente pequeño, el tiempo por nodo puede ser ignorado con poca pérdida de precisión.

Por todo ello, en [Kumar *et al* 03] se propone un modelo de coste más simple que (3.3) en el que el coste de transferir un mensaje entre dos nodos viene dado por:

$$T_{ENVÍO} = t_s + m\,t_w. \tag{3.4}$$

Este mismo modelo simplificado (3.4) se puede asumir para cuantificar el coste de compartir un bloque de m palabras entre dos procesadores en el modelo de memoria compartida. Si bien es cierto que el valor de la constante t_s en relación a t_w es probablemente mucho más pequeña en una máquina de memoria compartida que en una máquina de memoria distribuida.

3.1.4 Entornos de Experimentación

Siguiendo la clasificación de Flynn [Flynn 66] dada en la subsección 3.1.1 los entornos utilizados para la experimentación efectuada en este trabajo corresponden a la clase *MIMD*, y

considerando la organización del espacio de direcciones (subsección 3.1.2) deben ser catalogados como sistemas de memoria distribuida.

Consecuentemente, el paradigma de paso de mensaje es la técnica más apropiada para su programación, y la técnica de ruta que tendremos en cuenta será la de segmentación, debido a que realiza un uso más eficiente de los recursos de comunicación y reduce el tiempo de comunicación. Estas consideraciones serán las hipótesis de partida para el análisis que llevaremos a cabo en los capítulos 4 y 5.

Finalmente, en las siguientes subsecciones se enumeran los entornos usados en la experimentación, así como sus características principales.

CLÚSTER

Conjunto formado por 20 ordenadores tipo *PC* con un nivel de acoplamiento muy bajo, únicamente comparten su conexión de red vía un *switch*.

Cada nodo consta de 160 *MB* de memoria *RAM*, un procesador *Intel Pentium III* a 733 *Mhz* con 256 *KB* de *L2* (*cache* de nivel *2*) y una tarjeta de red de 100 *Mbps*.

El *switch* es un *Cisco* con 24 puertos 10/100 *Mbps* y un puerto adicional 10/100/1000 *Mbps*. Este último puerto se usa para conectar un servidor de disco por *NFS*, dado que los nodos de proceso carecen de él. Además, el servidor es usado para el arranque por red de los nodos, usando *LTSP* (*Linux Terminal Server Project*). Por tanto, todos los nodos son equivalentes tanto a nivel *software* como *hardware*. La topología de la red se puede asimilar al tipo *full-connection*.

IBM SP2

El *IBM SP2 (IBM SCALABLE POWERPARALLEL SYSTEM 2*) del *CESCA* (*CE*ntro de *S*upercomputación de *CA*taluña) es un multiprocesador de memoria distribuida compuesto por procesadores *RS6000* (*RISC -REDUCED INSTRUCTION SET COMPUTER- SYSTEM/6000*) con arquitectura *POWER2* (*PERFORMANCE OPTIMIZED WITH ENHANCED RISC*). Consta de 32 nodos, cada uno de ellos ejecuta su propia imagen del sistema operativo, y un único procesador *SP Thin Node* 2 (*TN2*) por nodo. La figura 3.9 muestra el diagrama de bloques de los distintos procesadores que puede montar el *IBM SP2* del *CESCA*. Los *TN2* del *CESCA*, tienen las siguientes características:

- Frecuencia de reloj *160 MHz* con *POWER2 SUPER CHIP (PSC2)*.
- 128 *KB* de caché asociativa para datos (*DCU*), 32 *KB* de *cache* para instrucciones (*ICU*) y 256 *MB* de *RAM*.
- Dos unidades de punto flotante (*FPU*) de doble precisión (128 *bits*) capaces, cada una de ellas, de realizar en paralelo una operación aditiva y otra multiplicativa, en total, 4 operaciones en punto flotante simultáneas si el tipo base es *double* (64 *bits*)
- Dos unidades para la aritmética entera y otro tipo de operaciones (*FXU*).

Según la información facilitada por el *CESCA*, sus *TN2* alcanzan los siguientes valores:

- *R_{punta}:* 640 *MFLOPS*
- *Linpack 100:* 315 *MFLOPS* *Linpack TPP:* 532 *MFLOPS*
- *SPECint95:* 8.61 *SPECfp95:* 25.8

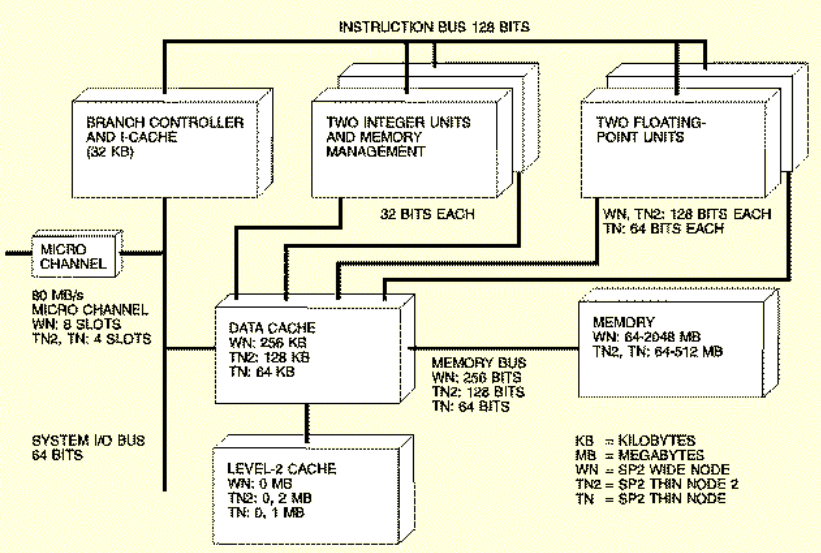


Figura 3.9 Diagrama de bloques de los procesadores del *IBM SP2*

Respecto a las comunicaciones, los nodos del *IBM SP2* del *CESCA* están conectados a un *switch* de alto rendimiento; concretamente a un *SCALABLE POWERPARALLEL SWITCH* (*SP*) *TRAILBLAZER3* (*TB3*). Es un conmutador bidireccional de dos niveles (*two-level crossbar switch*) con un rendimiento punta por dirección de 160 *MB/sec*. La figura 3.10 muestra su diagrama de conexiones para el caso concreto de 64 procesadores.

El *SP-TB3* pertenece a la categoría de redes de interconexión dinámicas multietapa (con todas las etapas iguales), con una topología equivalente a una red *Omega*, donde todos los procesadores están conectados entre sí a través de caminos que se establecen. Su diámetro es *O(log p)*, siendo *p* el número de procesadores.

Finalmente, *IBM* define al *SP2* como un multiprocesador, atendiendo seguramente a su nivel de acoplamiento y a las potenciales posibilidades, tanto *hardware* como *software*, que ofrece. Sin embargo, la categoría más apropiada, desde el punto de vista lógico, para el *SP2* del *CESCA* sería la de multicomputador.

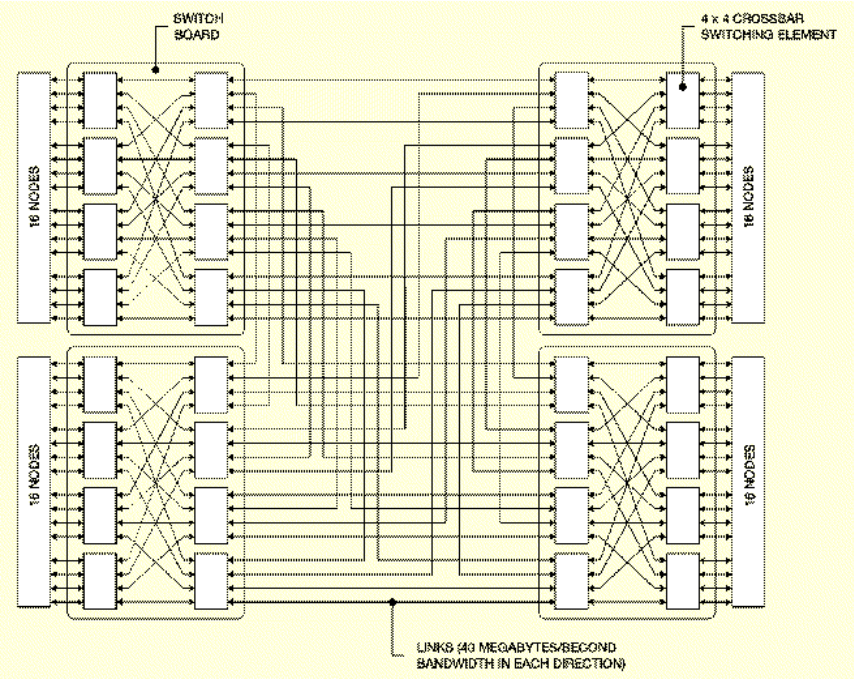


Figura 3.10 Diagrama de conexiones del *switch SP-TB3* para 64 nodos

3.2 Diseño de Algoritmos Paralelos

El desarrollo de un algoritmo es un componente crítico en la solución de problemas usando computadoras. Mientras que un algoritmo secuencial es esencialmente una secuencia de pasos básicos para solucionar un problema dado, un algoritmo paralelo debe indicar como solucionar un problema usando múltiples procesadores. Por ello, especificar un algoritmo paralelo envuelve algo más que mostrar los pasos que lo integran. Un algoritmo paralelo tiene una dimensión añadida de concurrencia y el diseñador del algoritmo debe especificar conjuntos de pasos que pueden ser ejecutados simultáneamente. El desarrollo de un algoritmo paralelo puede incluir algunos de los siguientes aspectos:

- Identificar porciones del trabajo que pueden ser ejecutadas concurrentemente.
- Mapear las porciones de trabajo en los múltiples procesos.
- Distribuir los datos de entrada, salida e intermedios asociados con el algoritmo.
- Manejar accesos a datos compartidos por múltiples procesadores.
- Sincronizar los procesadores en las distintas etapas del algoritmo.

El proceso de dividir una computación en partes más pequeñas, donde algunas o todas se puedan ejecutar en paralelo, se denomina *descomposición*. Estas unidades de computación en las que se divide la computación principal se denominan *tareas*. Algunas de las técnicas de descomposición más utilizadas son las siguientes:

- *Descomposición recursiva*, es una técnica que induce concurrencia en problemas que pueden ser resueltos utilizando el paradigma *Divide y Vencerás*.
- *Descomposición de los datos*, es un método potente para derivar concurrencia en algoritmos que operan en grandes estructuras de datos. La descomposición se realiza en dos pasos, en primer lugar se dividen los datos y posteriormente, esta distribución de los datos es usada para inducir una partición de la computación en tareas. Esta división de los datos puede ser realizada de muchas formas (dividir datos de salida, dividir datos de entrada, dividir ambos, dividir datos intermedios, …).
- *Descomposición exploradora*, está técnica se utiliza para descomponer problemas cuyos cálculos consisten en la búsqueda de la solución en un espacio de soluciones posibles. La técnica consiste en dividir el espacio de búsqueda en partes más pequeñas y buscar en cada una de las partes de manera concurrente hasta encontrar la solución buscada.
- *Descomposición especulativa*, este procedimiento se utiliza cuando un programa debe elegir una opción de entre muchas posibles dependiendo de la salida de los cómputos que la preceden. En esta situación, mientras una tarea está realizando la computación cuya salida será usada para decidir la siguiente computación, otras tareas pueden concurrentemente comenzar los cálculos de la siguiente etapa. Así, cuando la primera tarea haya finalizado, el cálculo de la opción correcta será utilizada mientras que el resto de las opciones serán desestimadas.

Las dos primeras técnicas son de propósito general y se pueden utilizar para una amplia variedad de problemas, mientras que las dos últimas se aplican a unos tipos de problemas muy específicos. Teniendo en cuenta que nuestro objetivo es operar sobre una matriz de datos densa, la técnica que hemos utilizado es la técnica de descomposición de los datos de entrada.

Las diferentes técnicas de descomposición descritas previamente nos permiten identificar la concurrencia que está disponible en el problema y descomponerlo en tareas que pueden ser ejecutadas en paralelo. El siguiente paso en el proceso de diseño de un algoritmo paralelo es tomar esas tareas y asignarlas a los procesos disponibles. Las técnicas de mapeo usadas en los algoritmos paralelos pueden ser clasificadas en dos categorías: *estáticas* y *dinámicas*. La asignación estática distribuye las tareas entre los procesos antes de la ejecución del algoritmo mientras que la técnica dinámica las distribuye durante la ejecución del algoritmo. El paradigma de programación paralelo, las características de las tareas y las interacciones entre ellas determinan cuál de las dos técnicas es más adecuada. La técnica de mapeo estática es a menudo utilizada en conjunción con una descomposición basada en los datos. En nuestros estudios hemos optado por realizar una asignación estática basada en la división de la matriz en la que se representan los datos. De ese modo, hemos analizado las técnicas más comunes de distribuir matrices entre los procesos:

- *Distribución por bloques*, consiste en dividir la matriz en bloques de filas o de columnas consecutivas, en el caso de una dimensión, y en fraccionar la matriz en bloques cuadrados consecutivos, en el caso de dos dimensiones. En ambos casos, el número de bloques en los que se divide la matriz será igual al número de procesadores de los que disponemos y el tamaño de los bloques será igual. Posteriormente, se asignará cada uno de estos bloques a un procesador.
- *Distribución cíclica*, esta modalidad distribuye las filas, las columnas o los elementos de la matriz de manera cíclica entre los procesadores.
- *Distribución por bloques cíclicos,* la división de los datos es del mismo tipo que la que se realiza en Distribución por bloques, tanto en unidimensional como en bidimensional. La diferencia radica en que el número de bloques en los que se divide la matriz puede variar desde el número de procesadores hasta el número de filas (o columnas) en el caso unidimensional o hasta el número de elementos de la matriz en el caso bidimensional. Posteriormente, se distribuirán estos bloques de manera cíclica entre los procesadores. Nótese que las dos primeras distribuciones son casos particulares de ésta.

En conclusión, en nuestro modelo las tareas son mapeadas estáticamente en los procesos y cada tarea realiza operaciones similares en datos diferentes. Este trabajo puede ser realizado en fases, en las cuales habrá interacciones entre los procesos para sincronizar las tareas. Este tipo de paralelismo que es resultado de operaciones idénticas aplicadas concurrentemente en diferentes datos se denomina *paralelismo de datos*.

3.3 Programación Usando el Paradigma Paso de Mensajes

Numerosas librerías y lenguajes de programación han sido desarrollados para la programación paralela explícita. Éstas difieren en la visión del espacio de direcciones que tiene el programador, así como en el grado de sincronización impuesto en las actividades concurrentes y en la multiplicidad de programas. El *paradigma de programación paso de mensajes* es uno de los más ampliamente utilizados. Las dos características principales de este paradigma son: asumir un espacio de direcciones dividido y soportar sólo paralelismo explícito.

La visión lógica de una máquina que soporta el paradigma de paso de mensajes consiste en *k* procesos, cada uno de los cuales tiene su propio espacio de direcciones. Esto implica por un lado que cada dato debe pertenecer a una de las divisiones del espacio de datos; por tanto, los datos deben partirse y localizarse explícitamente. Por otro lado, supone que todas las interacciones requieren la cooperación de dos procesos, el proceso que tiene los datos y el proceso que quiere acceder a los mismos; por tanto, se necesita el envío y la recepción de mensajes. Así, las operaciones básicas en el paradigma de programación paso de mensajes son: enviar y recibir.

Estas operaciones de comunicación pueden ser *con bloqueo* o *sin bloqueo*, permitiendo estas últimas el solapamiento entre el cálculo y las comunicaciones.

Muchas computadoras paralelas de las primeras generaciones estaban basadas en la arquitectura de paso de mensajes debido a su menor coste en comparación con las arquitecturas de memoria compartida. Puesto que el paso de mensajes era el paradigma de programación natural en este tipo de máquinas, esto dió lugar al desarrollo de muy diversas librerías: *Express* [Parasoft], *Parmacs* (ver [Bomans *et al* 90] y [Calkin *et al* 94]), *PICL* [Geist *et al* 91], *Zipcode* (ver [Skjellum *et al* 90] , [Skjellum *et al* 92] y [Skjellum *et al* 94]), *PVM* (ver [Geist *et al* 94] y [Dongarra *et al* 94]), etc. Así, todo fabricante de *hardware* proporcionaba su propia librería, que funcionaba muy bien en su propio *hardware* pero no era compatible con computadoras paralelas ofertadas por otros comerciales. Muchas de las diferencias eran sólo sintácticas, pero a menudo había serias diferencias semánticas que requerían un esfuerzo notable para trasladar un programa de paso de mensaje de una librería a otra.

MPI

Para resolver el citado problema, surgió el *interfaz de paso de mensajes* (ver [Doss *et al* 93], [Dongarra *et al* 93] y [Pacheco 96]) o *MPI* como es comúnmente conocido. *MPI* define una librería estándar para paso de mensaje que puede ser utilizada para desarrollar programas de paso de mensajes portables usando *C*, *Fortran*, *Perl*, etc. El *MPI* estándar define tanto la semántica como la sintaxis de un conjunto base de rutinas de librerías que son muy útiles para escribir programas de paso de mensajes.

En la reunión de principios de marzo de 1995 del *MPI Forum* se aprobaron un conjunto de extensiones y correcciones del primer estándar del *MPI*, el *MPI 1.0*, dando lugar a la primera especificación completa y competitiva de *MPI*, la *MPI 1.1*, publicada en junio de 1995 [MPI Forum 95]. Desde 1995 hasta 2005 dicha estandarización ha sufrido correcciones y clasificaciones, siendo la versión actual, la *1.2.7*, de noviembre de 2005. Paralelamente, los expertos del *MPI Forum* han venido trabajando en una nueva especificación, o extensión de la existente, más actual y que aborda nuevos retos como las comunicaciones *one-side* o la gestión dinámica de procesos. En 1997 se publica la primera versión de este nuevo estándar, llamado *MPI 2.0*.

Existen varias implementaciones de *MPI* que cumplen con el estándar como *MPICH* (ver [Gropp *et al* 96a] y [Gropp *et al* 96b]) de *Argonne Nacional Laboratory* y de *Mississippi State University* y como *LAM-MPI* de *Ohio Supercomputing Center* y *Open System Laboratory* de *Indiana University*.

MPI permite comunicaciones punto a punto mediante operaciones de envío y recepción. También permite operaciones colectivas como por ejemplo barreras de sincronización, difusiones, recolecciones, distribuciones, etc.

Un concepto clave usado a través de *MPI* es el *dominio de la comunicación*. Éste surge como la necesidad de restringir operaciones de comunicación a ciertos grupos de procesos. *MPI* proporciona el mecanismo *comunicador* para definir el conjunto de procesos que puede comunicarse. Este conjunto de procesos forma el dominio de la comunicación.

Nuestros algoritmos se apoyan en la biblioteca *MPI* debido a su actualidad y a la disponibilidad de distribuciones en prácticamente cualquier arquitectura.

La distribución de *MPI* instalada en nuestros entornos de experimentación es *MPICH 1.2.7*, basada en la especificación *1.2* de *MPI*. Dicha distribución, así como nuestros programas, ha sido compilada con el compilador *C* de *GNU* (*gcc*).

3.4 Operaciones Básicas de Comunicación

En la mayoría de los algoritmos paralelos, los procesadores necesitan intercambiar datos. Este intercambio de datos afecta significativamente a la eficiencia del algoritmo paralelo introduciendo retardos de comunicación durante la ejecución. Por ello, una apropiada implementación de estas operaciones de comunicación en las diversas arquitecturas paralelas es la clave de la ejecución eficiente de los algoritmos paralelos que usemos.

Las operaciones básicas de comunicación que pueden tener lugar y sus descripciones son las siguientes:

- *Transferencia de un mensaje entre dos procesadores*, es la operación más básica de comunicación y consiste en enviar un mensaje de un procesador a otro.
- *Envío de uno a todos*, un procesador envía idénticos datos a todos los procesadores o a un subconjunto de ellos.
- *Envío de todos a todos*, es una generalización del envío de uno a todos, en el que todos los procesadores inician simultáneamente la transmisión.
- *Envío de uno a todos personalizado*, en esta operación el procesador origen del envío transmite p mensajes, uno a cada uno de los p procesadores.
- *Desplazamiento circular*, consiste en que cada procesador envía un mensaje a un único procesador.

En los estudios que abordaremos en posteriores capítulos tendremos la necesidad de conocer el coste de dos de las operaciones mencionadas anteriormente. Esas operaciones son el envío entre dos procesadores y el envío de uno a todos.

El coste de la primera de ellas ya se ha mostrado en (3.4). El segundo tipo de operación básica que precisaremos es el envío de un procesador al resto de procesadores. Asumiremos que hay k procesadores implicados, siendo k potencia de 2, que el tamaño del mensaje es m, que el mensaje se envía entero y que el algoritmo está implementado en el contexto de un hipercubo. El cálculo del coste de este tipo de comunicación es muy sencillo. El procedimiento del envío de uno a todos implica $\log_2 k$ transferencias de mensaje punto a punto, el coste de cada una de ellas es $t_s + m\, t_w$. Por lo tanto, el tiempo total de la operación que estamos analizando es

$$T_{ENVÍO\_UNO\_TODOS} = (t_s + m\, t_w)\, \log_2 k \qquad (3.5)$$

Si bien es cierto que es el coste de la operación en un hipercubo, éste es aplicable a otra topología de red y puede ser fácilmente extendido para trabajar con cualquier número de procesadores.

El coste (3.5) puede ser mejorado en el caso de que el tamaño del mensaje, *m*, sea suficientemente grande. Si esto ocurre, el mensaje de tamaño *m* puede ser dividido en *k* partes de tamaño *m/k*. El envío de uno a todos se implementa en dos pasos:

- enviar el trozo *i*-ésimo del mensaje al procesador *i*,
- enviar de todos a todos el mensaje de tamaño *m/k* residente en cada procesador.

Este proceso puede ser realizado con un coste

$$T_{ENVÍO\_UNO\_TODOS} = 2 \log_2 k\, t_s + 2\, m\, ((k-1)/k)\, t_w, \tag{3.6}$$

el cual puede considerarse de manera aproximada como

$$2\,(\log_2 k\, t_s + m\, t_w).$$

Comparando (3.5) con esta aproximación se observa que el término t_s es el doble en esta última expresión mientras que el término t_w se reduce por un factor $(\log_2 k)/2$. En los estudios que realizaremos en los capítulos 4 y 5, asumiremos (3.5) como el coste de la operación colectiva envío de uno a todos que tiene lugar en algunas de las formulaciones paralelas de Neville. Esto es debido a que cuando el tamaño del mensaje o el número de procesadores es pequeño, siendo nuestro caso, *MPICH* 1.2.7 utiliza un algoritmo de árbol binomial para este tipo de envío, cuyo coste es (3.5). Para mayor información se puede consultar, por ejemplo, la especificación de la rutina *intra_fns_new.c* de *MPICH* y también [Grama et al 03].

Dado que todos los logaritmos que aparecen en la memoria son en base 2, a partir de ahora y por simplicidad los denotaremos sin subíndice (*log*).

3.5 Métricas de Rendimiento para Sistemas Paralelos

La evaluación del rendimiento de un algoritmo paralelo no puede estar desligada de la arquitectura paralela sobre la cual se está ejecutando e incluso del sistema real que implementa dicha arquitectura. Definimos un *sistema paralelo* como la combinación de un algoritmo paralelo y una arquitectura paralela. El tiempo de ejecución de un algoritmo paralelo depende no sólo del tamaño del problema sino también del número de elementos de proceso utilizados, de sus relativas velocidades de cálculo y de comunicación, de la topología de la red de interconexión, de la técnica de encaminamiento, etc. Un algoritmo que se ejecute de manera eficaz en un sistema, puede que no lo haga si una o varias de las características del sistema se ven modificadas.

Es importante entonces estudiar el rendimiento de los programas paralelos con la visión de determinar el mejor algoritmo, evaluando las plataformas *hardware* y examinando los beneficios del paralelismo. Por ello, en esta sección, introduciremos algunas métricas que son comúnmente usadas para analizar el rendimiento.

El tamaño del problema se define como el valor o conjunto de valores asociados a la entrada del problema que representa una medida de su tamaño respecto de otras entradas posibles. Por otro lado, con W denotaremos la cantidad de operaciones básicas necesarias que hay que realizar sobre los datos de entrada para resolver un problema, siendo tal cantidad función del tamaño del problema. Dado que pueden existir diferentes algoritmos que solucionen el mismo problema y con objeto de tener un único valor de W para un problema dado, en este trabajo W expresará más concretamente: *el número de operaciones básicas requeridas por el algoritmo secuencial conocido más rápido para solucionar dicho problema en un único procesador*.

El tiempo de ejecución de un algoritmo secuencial, $T_{SECUENCIAL}$, es el tiempo transcurrido entre el comienzo y el fin de su ejecución en un único procesador. Si el tiempo empleado por el algoritmo conocido más rápido para solucionar un problema en un único procesador es $T_{SECUENCIAL}$, entonces $T_{SECUENCIAL} \propto W$ ó $T_{SECUENCIAL} = W\, t_c$, donde t_c es una constante dependiente de la máquina y representa el tiempo empleado en realizar una operación en punto flotante.

El tiempo de ejecución de un algoritmo paralelo en k procesadores, $T_{PARALELO}\,(W, k)$, es el tiempo empleado desde que el primer procesador inicia la ejecución de algoritmo hasta que el último procesador completa su ejecución. El tiempo paralelo se compone de una suma de, al menos, dos tiempos, que son:

- Tiempo de cálculo, $T_{CÁLCULO}$ (W, k), lo definiremos como el tiempo de cómputo del procesador que más tarda en realizar sus cálculos, esto es

$$T_{CÁLCULO}\,(W, k) = max\ \{\, T_{CÁLCULO,\, i}\,(W)\,\}\ con\ 1 \le i \le k \tag{3.7}$$

  donde $T_{CÁLCULO,\, i}\,(W)$ es el tiempo de cálculo empleado por el procesador i-ésimo.

- Tiempo de comunicación, $T_{COMUNICACIÓN}$ (W, k), que es el tiempo destinado a las transferencias de datos entre los procesadores. La suma de todos los tiempos dedicados a las comunicaciones en aquel procesador que emplee más tiempo en ese hecho dará lugar a este tiempo total. Por ello, de forma similar a lo que mostramos en (3.7), tenemos

$$T_{COMUNICACIÓN}\,(W, k) = max\ \{\, T_{COMUNICACIÓN,\, i}\,(W)\,\}\ con\ 1 \le i \le k \tag{3.8}$$

  donde $T_{COMUNICACIÓN,\, i}\,(W)$ es el tiempo de comunicación empleado por el procesador i-ésimo.

Si existiera solapamiento entre el cómputo y las comunicaciones, habría que añadir un término negativo para reflejar tal situación, y si hubiera inactividades habría que añadir otro término positivo.

Incremento de Velocidad y Eficiencia

Cuando se evalúa un sistema paralelo, es interesante determinar el aumento de rendimiento obtenido al realizar una implementación paralela de un algoritmo dado. Esta medida, denominada *incremento de velocidad* o *speedup*, *S*, recoge el beneficio relativo de solucionar un problema en una plataforma paralela. Formalmente, *S* está definida como la relación entre el tiempo empleado para solucionar un problema en un único procesador y el tiempo requerido para solucionar el mismo problema en un multicomputador con *k* procesadores idénticos

$$S(W,k) = \frac{T_{SECUENCIAL}(W)}{T_{PARALELO}(W,k)}. \tag{3.9}$$

Debido a las distintas sobrecargas del procesamiento paralelo, los procesadores no pueden dedicar todo su tiempo a computación útil, computación que es también realizada por su homólogo secuencial. La fracción del tiempo empleada por un procesador en tareas útiles determina la *eficiencia*, *E,* de un sistema paralelo. La eficiencia se define como la proporción entre el incremento de la velocidad y el número de procesadores, así

$$E(W,k) = \frac{S(W,k)}{k} = \frac{T_{SECUENCIAL}(W)}{kT_{PARALELO}(W,k)}. \tag{3.10}$$

En un sistema paralelo ideal, el incremento de velocidad es igual al número de procesadores y la eficiencia es igual a *1*. En la práctica, *S* es menor que el número de procesadores y la eficiencia está entre *0* y *1*, dependiendo del grado de efectividad con el que los procesadores han sido utilizados. En ocasiones, se puede observar un valor de *S* superior a *k* (eficiencia mayor que *1*), efecto conocido como condición de *superlinealidad*. Esto es normalmente debido a un algoritmo secuencial no óptimo o a características *hardware* que sitúan al algoritmo secuencial en desventaja. Un ejemplo típico es el efecto de las memorias *cache*, que al dividir las estructuras de datos iniciales *(caso secuencial)* en muchas y pequeñas sub-estructuras *(caso paralelo)* inciden fuertemente en los tiempos de acceso a los datos (memoria).

La Ley de Amdahl [Amdahl 67] pone un límite superior al incremento de velocidad, y por tanto también a la eficiencia, de un sistema paralelo atendiendo al hecho de que los procesos suelen tener partes que no pueden ser ejecutadas en paralelo, sino de forma secuencial pura. Ese límite superior en el caso del *speedup*, denominado *cuello de botella secuencial*, es la inversa de la parte serie del algoritmo, por tanto, la eficiencia máxima tenderá a cero. En conclusión, una pequeña parte serie que no se puede realizar en paralelo en un proceso puede provocar una fuerte disminución del rendimiento del sistema. La Ley de Amdahl es bastante pesimista, sin embargo, es necesario señalar que esta ley no tiene en cuenta algunos beneficios de los sistemas paralelos. Una realidad que sí transmite la Ley de Amdahl es que el rendimiento no aumenta por el hecho de incrementarse indefinidamente el número de procesadores.

La Ley de Gustafson [Gustafson 88] viene a compensar el pesimismo dejado por la Ley de Amdahl. Esta última se refería a problemas con volumen de cálculo fijo en los que se aumenta el número de procesadores. Sin embargo, en la práctica, el volumen del problema no es independiente del número de procesadores, ya que con mayor número de procesadores se pueden abordar problemas de mayores dimensiones. Por ello, la Ley de Gustafson se refiere al crecimiento del volumen de cálculo necesario para resolver un problema. En la mayoría de los casos, cuando el volumen del problema crece lo hace sólo en su parte paralela, no en su parte secuencial. Ello hace que el cuello de botella secuencial tienda a cero cuando el volumen del problema aumenta.

Podría pensarse que hay una aparente contradicción entre las leyes de Amdahl y Gustafson, pero esto no es así debido a que las premisas de ambas leyes son distintas. La Ley de Amdahl se refiere a procesos con un volumen de cálculo fijo mientras que la Ley de Gustafson se refiere a problemas cuyo volumen de cálculo puede aumentar según el número de procesadores. Esto se denomina *escalado del problema* y refleja la situación más cercana a la realidad.

El *coste de un sistema paralelo* está definido como el producto del tiempo de ejecución paralelo y el número de procesadores utilizados. Un sistema paralelo se dice que es de coste óptimo, si y sólo si, el coste es asintóticamente del mismo orden de magnitud que el tiempo de ejecución del programa secuencial, es decir, $k\,T_{PARALELO} = \Theta(W)$. Dado que la eficiencia es la proporción entre el coste del algoritmo secuencial y el coste del algoritmo paralelo, un sistema paralelo de coste óptimo tiene una eficiencia de $\Theta(1)$. El coste también se denomina *producto procesador-tiempo*.

Normalmente, los programas paralelos incurren en sobrecargas tales como tiempo de inactividad por parte de los procesadores, comunicación entre procesadores y exceso de computación. Éstas, y otras sobrecargas, pueden ser recogidas en una función simple denominada *función de sobrecarga* u *overhead*, que denotaremos por T_O, la cual expresamos de la forma

$$T_O(W,k) = kT_{PARALELO}(W,k) - W \ . \tag{3.11}$$

El coste de solucionar un problema de tamaño W en k procesadores es $k\,T_{PARALELO}$; de éste, W unidades son empleadas en realizar trabajo útil, por tanto el resto es sobrecarga.

Escalabilidad

Se define *escalabilidad* de un algoritmo paralelo como su capacidad para hacer un uso efectivo de los recursos del sistema con un número mayor de procesadores dentro de una arquitectura dada. Puede existir interés en diferentes recursos: número de procesadores, tiempo de ejecución, velocidad de ejecución, eficiencia, etc.

Suelen plantearse dos modelos genéricos de escalado del sistema [Martín *et al* 96]:

- Modelo de *escalado por tiempo crítico*, donde el objetivo es disminuir el tiempo de ejecución del algoritmo al aumentar el número de procesadores, manteniendo el tamaño del problema.
- Modelo de *escalado por precisión crítica*, el objetivo en este caso es poder aumentar el tamaño del problema. Este aumento puede realizarse manteniendo constante la eficiencia, el tiempo de ejecución, la velocidad o la memoria. Lo que da lugar a los siguientes submodelos: isoeficiencia [Grama *et al* 03] [Gupta 95] [Gupta *et al* 95], isotiempo [Gustafson 88] [Martín *et al* 97], isovelocidad [Sun et al 94] o isomemoria.

En [Grama *et al* 03] la eficiencia se expresa del siguiente modo

$$E(W,k) = \frac{S(W,k)}{k} = \frac{W}{kT_{PARALELO}(W,k)}, \tag{3.12}$$

utilizando (3.11) se tiene que,

$$E(W,k) = \frac{1}{1+\frac{T_O(W,k)}{W}}. \tag{3.13}$$

En (3.13) se observa que si W se mantiene constante y k se incrementa, la eficiencia disminuye porque T_O se incrementa con el número de procesadores. Si W se incrementa manteniendo fijo el número de procesadores, entonces para sistemas paralelos escalables, la eficiencia se incrementa porque T_O crece más lentamente que W para un k fijo. Para estos sistemas paralelos, la eficiencia puede ser mantenida en un valor fijo (entre 0 y 1) si la relación T_O / W en (3.13) se mantiene como un valor constante. Para un valor deseado E de la eficiencia tenemos

$$\begin{aligned} E &= \frac{1}{1+\frac{T_O(W,k)}{W}}, \\ \frac{T_O(W,k)}{W} &= \frac{1-E}{E}, \\ W &= \frac{E}{1-E}T_O(W,k), \\ W &= KT_O(W,k) \end{aligned} \tag{3.14}$$

donde $K = E / (1 - E)$ es una constante dependiente de la eficiencia que va a ser mantenida constante.

En (3.14) W puede habitualmente ser obtenido como una función de k mediante manipulaciones algebraicas. Esta función marca el crecimiento de W requerido para mantener fija la eficiencia a medida que el número de procesadores k aumenta. Esta función es la *función de isoeficiencia* del sistema paralelo (combinación de arquitectura paralela y algoritmo paralelo), la cual es una métrica útil de la escalabilidad y fue propuesta por primera vez por Kumar y Rao [Kumar *et al* 87] en el contexto de la búsqueda de primero en profundidad.

En el trabajo de Kumar y Rao, un sistema paralelo es considerado escalable si su función de isoeficiencia existe; de otra manera, el sistema paralelo no es escalable. La función de isoeficiencia de un sistema escalable podría, no obstante, ser arbitrariamente grande; es decir, esto podría imponer una tasa de crecimiento de *W* muy grande en relación al número de procesadores. En la práctica, el tamaño del problema puede ser incrementado asintóticamente sólo en una proporción permitida por la cantidad de memoria disponible en cada procesador. Si la restricción de memoria no permite que el tamaño del problema sea incrementado en la proporción necesaria para mantener la eficiencia fija, entonces el sistema paralelo sería considerable no escalable desde un punto de vista práctico.

El análisis de la isoeficiencia ha sido considerado muy útil al caracterizar la escalabilidad de una amplia variedad de sistemas paralelos (ver [Ranka *et al* 90], [Singh *et al* 91], [Hwang 93], [Gupta *et al* 94] y [Gupta *et al* 95]). Una importante característica del análisis de la isoeficiencia es que en una expresión simple se captura los efectos de las características de un algoritmo paralelo al igual que de la arquitectura en la que está implementado. Al efectuar el análisis de la isoeficiencia, uno puede estudiar el rendimiento de un programa paralelo y entonces, predecir su rendimiento en un amplio número de procesadores.

Otra posible forma de evaluar la facilidad de un sistema paralelo para mantener la eficiencia constante es a través de la eficiencia escalada [Prieto *et al* 03]. El tamaño del problema y el número de procesadores se multiplican por el mismo factor y se observa si el algoritmo escala, esto es

$$E_{ESCALADA}(W,k) = \frac{T_{PARALELO}(W,1)}{T_{PARALELO}(Wk,k)}. \tag{3.15}$$

3.6 Entornos de Experimentación

En las secciones previas se han presentado los modelos de computadores paralelos así como las métricas más habituales para estimar la eficacia y eficiencia de los algoritmos paralelos. Una primera conclusión es que, en ciertas situaciones, resulta complejo extraer conclusiones generales de las expresiones obtenidas para las métricas mencionadas, puesto que en ellas aparecen términos ligados a muchas variables.

También se ha observado que la bondad de un algoritmo paralelo depende de la calidad de su diseño y de los parámetros propios del entorno *(hardware/software)* donde se ejecuta. En otras palabras, el mismo algoritmo puede alcanzar eficiencias reales superiores a *0.9* para tamaños razonables en un entorno y, en otro, ni tan siquiera para tamaños desproporcionados; todo depende de las constantes de cálculo, de comunicaciones, de la relación de proporcionalidad existente entre ellas, etc.

Existen gran variedad de *benchmarks* para caracterizar el comportamiento de los distintos subsistemas de los computadores paralelos de memoria distribuida, al cual pertenecen los computadores usados en nuestra experimentación, así como para estimar sus valores de rendimiento máximo. De todos ellos, las velocidades de cálculo y de comunicación son, a nuestro juicio, los de mayor impacto en el comportamiento de los algoritmos paralelos en este tipo de computadores, por lo que en este trabajo nos hemos centramos exclusivamente en ellos.

Conocer los valores de rendimiento máximos (o de *pico*) y/o la cantidad alcanzada en la magnitud que define cierto *benchmark* (*Linpack* o *SPEC*, por ejemplo) es, evidentemente, útil para comparar computadores, pero no lo es, por ejemplo, para predecir desde el plano teórico cómo se comportará la ejecución de los algoritmos paralelos cuando el número de procesadores, o el tamaño del problema, crece en ésta o aquella proporción. Nuestro interés radica precisamente en este segundo aspecto, en predecir desde el plano teórico cuál será el comportamiento de las distintas variantes paralelas de la eliminación de Neville, lo que se traduce en la búsqueda de estimaciones para las constantes de los modelos teóricos que reflejen nuestra forma de programar, el impacto del compilador y los *flags* de compilación utilizados, las dimensiones de las estructuras de datos, etc.

Resulta útil, por tanto, disponer de estimaciones ajustadas de las constantes más influyentes del entorno de desarrollo. En las siguientes subsecciones se detallan las metodologías y *benchmarks* utilizados para la obtención de las estimaciones de los modelos teóricos, así como los valores de dichas constantes obtenidos para nuestros entornos de experimentación.

Constante de Cálculo

Representa la capacidad de la unidad de coma flotante para realizar operaciones. Se expresa en *Flops* (operaciones en punto flotante por segundo) o *MFlops* (millones de operaciones en punto flotante por segundo), entendiendo por operación en punto flotante al conjunto formado por una operación simple (suma, resta, etc.) seguida de una asignación $A = A \Phi b$, siendo A una posición de un vector/matriz y b una constante, o $A = A \psi b \omega C$ donde ψ es una adicción/sustracción, ω una multiplicación/división y C otra posición del vector/matriz. La tendencia actual es considerar la primera expresión como más representativa, si bien a la hora de implementar los *benchmarks* se utiliza la segunda por ser de mayor granularidad. La aproximación de la primera se obtiene, normalmente, desde la segunda dividiendo el tiempo por 2

Para la estimación de la constante de cálculo se ha optado por la implementación de un *benchmark* propio que considera tanto el caso en el que la matriz está almacenada usando una estructura bidimensional como el supuesto donde el almacenamiento es unidimensional. El *benchmark* se muestra en el Algoritmo 3.1, donde se observa que en cada repetición se calcula una nueva posición de inicio buscando una lo suficientemente alejada para que no esté almacenada en la memoria *cache*. Es decir, se provoca un fallo de *cache*, usando operaciones de reubicación de orden constante ($O(1)$). Dentro de cada repetición se aprovechan las ventajas de la *cache*. Este es el escenario típico de la eliminación de Neville.

```
1.  ElementosCache = SizeCacheL2 / sizeof (double);
2.  NumeroElementos = ElementosCache * Cte       /* Cte entera > 1 */
3.  V = Reservar Espacio (NumeroElementos);
4.  para n = 16 hasta NumeroElementos paso * 2 hacer
5.     Rellenar Vector ( V, NumeroElementos);
6.     Inicio := 0;
7.     Fin := n - 1;
8.     Tiempo := Medir Tiempo;
9.     para k := 0 hasta Repeticiones - 1 hacer
10.       para i := Inicio hasta Fin - 1 hacer
11.          V[i] := V[i] + 1.1 * V[i + 1];
12.       Inicio := Nuevo Inicio (Fin, ElementosCache);
13.       Fin := Inicio + n - 1;
14.    fin para      /* línea 9 */
15.    Tiempo := (Medir Tiempo - Tiempo) / 2.0;
16.    Imprimir(n, Tiempo / (Repeticiones * (n + 1)));
17. fin para   /* línea 4 */
18. Liberar Espacio(V);
19. Raiz := Raiz(NumeroElementos);
20. M := Reservar Espacio(Raiz, Raiz);
21. para n := 4 hasta Raiz paso * 2 hacer
22.    Rellenar Matriz(M, Raiz, Raiz);
23.    Inicio := 0;
24.    Fin := n - 1;
25.    Tiempo := Medir Tiempo;
26.    para k := 0 hasta Repeticiones - 1 hacer
27.       para i:= 0 hasta n - 1 hacer
28.          para j := Inicio hasta Fin - 1 hacer
29.             M[i, j] := M[i , j] + 1.1 * M[i, j + 1];
30.       Inicio := Nuevo Inicio (Fin, ElementosCache);
31.       Fin := Inicio + n - 1;
32.    fin para   /* línea 26 */
33.    Tiempo := (Medir Tiempo - Tiempo) / 2.0;
34.    Imprimir ( n, Tiempo / (Repeticiones * (n + 1)));
35. fin para   /* línea 21 */
36. Liberar Espacio (M);
```

Algoritmo 3.1 *Benchmark* para estimar la constante de cálculo

Finalmente, la siguiente expresión

$$Error(x) = 100\frac{\left| f(x) - \hat{f}(x) \right|}{\left| f(x) \right|} \tag{3.16}$$

será usada para cuantificar el error relativo entre la estimación teórica ($f(x)$) y la realidad empírica ($\hat{f}(x)$).

Constantes de Comunicaciones

Como se ha comentado en las secciones previas, el paradigma de paso de mensaje considera dos tipos de operaciones: a) entre vecinos y b) colectivas. Las primeras involucran a dos procesadores y son del tipo punto a punto, conexión directa, y en las segundas interviene un número elevado de procesadores, que normalmente no están conectados entre sí directamente.

Las operaciones entre vecinos están caracterizadas por (3.4) y dependen únicamente de las constantes t_s y t_w, así como del tamaño del mensaje (m).

Por su parte las colectivas dependen del número de procesadores, del tamaño del mensaje, de la topología y del algoritmo de encaminamiento. Pero como se ha comentado en la subsección 3.1.3, seleccionada una técnica de ruta y una topología, se pueden expresar en términos del tamaño del mensaje, del número de procesadores y de las constantes de las operaciones entre vecinos.

Por tanto, el modelado de las operaciones de comunicación se efectuará en base a las constantes t_s y t_w. Para estimarlas se han analizado las metodologías propuestas en *PARKBENCH (PARallel Kernels and BENCHmarks*) [Hockney *et al* 94], *Intel MPI Benchmarks* (*IMB*), formalmente conocido como *Pallas Benchmarks* (*PMB*), incluido en el paquete *Intel Clusters Toolkit*, *HPL* (*High Performance Linpack*), una implementación portable del *benchmark High-Performance Linpack* para sistemas paralelos de memoria distribuida, y *NetPIPE* (*Network Protocol Independent Performance Evaluator*) [Snell *et al* 96], llegando a la conclusión de que el *test PingPong* es el más adecuado, y el *benchmark NetPIPE* el más completo, estable y preciso.

Para un tamaño concreto, el *test PingPong* consiste en el intercambio de dos mensajes consecutivos. En el primero es el primer procesador el emisor, y el receptor el segundo procesador. Para el segundo mensaje los procesadores intercambian sus papeles. El tiempo obtenido se divide por 2 para así caracterizar el coste de una operación de envío/recepción entre vecinos.

PingPong en *NetPIPE*, al igual que en el resto de *benchmarks*, realiza el intercambio de información para un rango de tamaños establecido, usando incrementos no lineales en el tamaño del mensaje. Así, para mensajes pequeños el incremento en el tamaño de dos mensajes consecutivos es una constante, normalmente 1. Para tamaños medios/grandes el incremento es, normalmente, en potencias de 2.

Esta forma de proceder se ampara en el principio de que para tamaños grandes no hay cambios significativos en el comportamiento de las comunicaciones y, así, evitar que el tiempo de ejecución del *benchmark* sea excesivamente grande.

Sin embargo, el comportamiento real del *hardware*/*software* puede ser otro y, entonces, no se detectan cambios de comportamiento potencialmente significativos. Por este motivo, el código fuente ha sido modificado para que los incrementos sean menores, lineales en una constante adaptada al tamaño del mensaje.

NetPIPE permite caracterizar tanto el comportamiento de la librería de paso de mensaje como el del protocolo *TCPI/IP*, lo que puede aportar información sobre el coste adicional del uso de las librerías de paso de mensaje.

Ejecutado el test y *benchmark* elegido, *PingPong* y *NetPIPE* en nuestro caso, se está en condiciones de estimar las constantes t_s y t_w del modelo. Puesto que la modificación realizada en *NetPIPE* arroja una información exhaustiva en el rango de tamaños analizado, se puede proceder de la siguiente manera:

- Si no se observan cambios de comportamiento/pendiente, saltos, transitorios, etc., la constante t_s será el tiempo observado al tratar el mensaje de tamaño 0, y t_w el tiempo obtenido para el mensaje de mayor tamaño.
- Si no se cumplen las condiciones anteriores se realizará un ajuste por mínimos cuadrados.

Tipos de Datos Usados y Tamaños Analizados

Para el *benchmark* que estima la constante de cálculo, coste por *Flop*, el tipo de dato usado es el *double*, y las dimensiones de la matriz potencias de 2 en el rango [4x4,…, 2048x2048]. La memoria *RAM* de nuestros entornos de experimentación oscila entre 160 y 256 *MB*, y la memoria *cache* de datos entre 128 y 256 *KB*, por lo que las dimensiones usadas exceden claramente el tamaño de la *cache* y cubren aproximadamente el 50% de las dimensiones factibles para la eliminación de Neville en nuestros entornos de experimentación. Nótese que, una matriz de dimensiones 4320x4320 de tipo *double* ocupa aproximadamente 140 MB, máximo admisible en el cluster de *PCs* si se quiere garantizar que su ejecución no entre en *swapping*.

Respecto a las comunicaciones el tipo de datos base es, obviamente, el *byte*. Al ser la dimensión máxima admisible para la matriz 4320x4320 consideramos que un intervalo de observación entre 0 *bytes* y 1 *Mbyte* es suficiente para cubrir el abanico de los potenciales tamaños de mensajes de todas las variantes paralelas de la eliminación de Neville estudiadas.

3.6.1 *CLÚSTER*

Constante de Cálculo

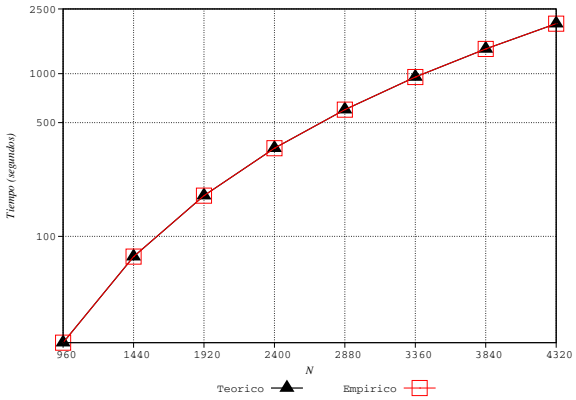
De la ejecución del *benchmark* mostrado en el Algoritmo 3.1 se obtienen dos conclusiones fundamentales:

- La forma de almacenar la matriz, como una estructura bidimensional o como una unidimensional, no influye en el tiempo empleado en la ejecución de un *Flop*.
- Desde tamaños relativamente pequeños, matrices de 64x64 elementos, el tiempo por *Flop* es muy estable, habiendo ligeras fluctuaciones que a medida que el tamaño de la matriz aumenta desaparecen o son de menor magnitud.

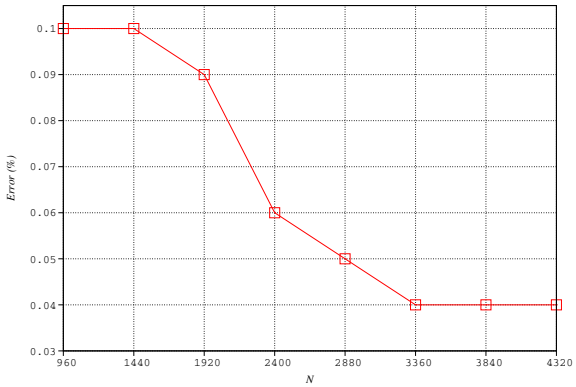
La estimación obtenida para la constante de cálculo es: $t_c = 3.8E^{-08}$ segundos por *Flop*.

Para validar la estimación se ha comparado el tiempo resultante de la ejecución de la variante secuencial de Neville con el procedente del modelo teórico usando dicha estimación. El resultado de la comparación se muestra en la gráfica 3.1, siendo la escala del eje *y* logarítmica. Por su parte, la gráfica 3.2 representa el error relativo entre la estimación teórica y la realidad empírica usando (3.16). En ambas figuras se observa un grado de coincidencia muy elevado, siendo siempre el error relativo inferior o igual al 0.1%.

Por todo lo dicho, la estimación obtenida para la constante de cálculo en este entorno se considera apropiada.



Gráfica 3.1 Neville secuencial vs modelo teórico en *CLÚSTER*



Gráfica 3.2 Error relativo Neville secuencial vs modelo teórico en *CLÚSTER*

Constantes de Comunicación

Para el análisis inicial de los datos se usará la metodología propuesta en *NetPIPE*, esto es, representar la *Saturación*, *Signature* y *Productividad* obtenidas con *MPICH* y con el protocolo *TCP/IP*. Dicha información se muestra en las gráficas 3.3 y 3.4, donde se aprecia que:

- Tanto para *MPICH* como *TCP/IP* hay un punto de saturación entorno a 1500 *bytes*. La gráfica 3.5a refleja mejor esta situación. La explicación más plausible del cambio de comportamiento observado es la propia naturaleza del protocolo *TCP/IP*, que divide los mensajes en tramas de aproximadamente 1500 *bytes*, encauzando, cuando es posible, la transmisión de las diferentes tramas. Nótese que al citado tamaño hay que restarle los *bytes* de control del protocolo, y los propios de *MPICH*.
- El coste o sobrecarga de *MPICH* no es elevado hasta mensajes de aproximadamente 32000 *bytes*. A partir de ese tamaño y hasta el final *MPICH* sufre una pérdida de rendimiento respecto a *TCPI/IP* que se acentúa (ver gráfica 3.5b) en torno a 128000 *bytes*.

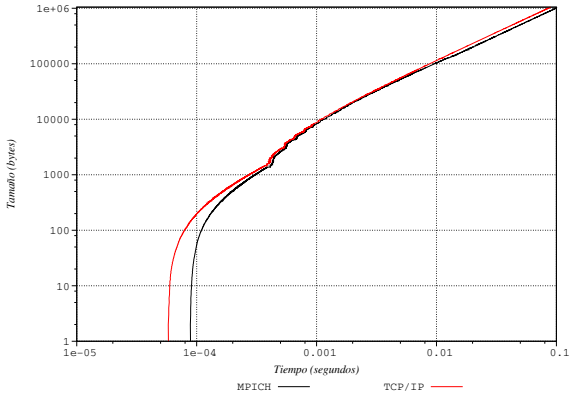
Por todo lo dicho, lo apropiado para estimar las constantes t_s y t_w sería considerar 3 rangos: a) hasta aproximadamente 1500 *bytes*, b) entre 1500 y 128000 *bytes* y c) desde 128000 *bytes* hasta el final. Ahora bien, esto complicaría el modelo teórico haciendo su uso inviable por lo que los dos últimos rangos serán considerados conjuntamente. Los resultados obtenidos para *MPICH* se muestran en la tabla 3.1.

Directamente de los Datos	
t_S = 8.77E^{-05}s	t_W = 1.20E^{-06} s por *double*

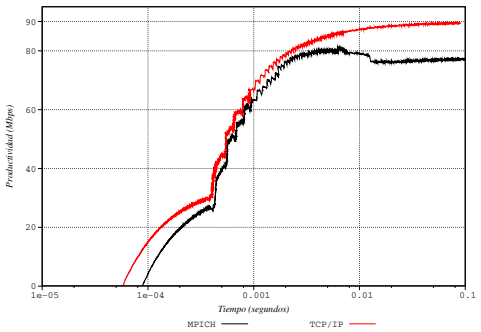
Por Mínimos Cuadrados		
1 Ajuste	2 Ajustes	
[1,..., 1024x1024] *bytes*	[1,..., 1342] *bytes*	[1350,..., 1024x1024] *bytes*
t_S =8.87E^{-05} s t_W = 1.25E^{-06} s por *double*	t_S =8.80E^{-05}s t_W = 1.71E^{-06}s por *double*	t_S =2.51E^{-04} s t_W = 7.32E^{-07}s por *double*

Tabla 3.1 Distintas estimaciones de t_s y t_w para *CLÚSTER*

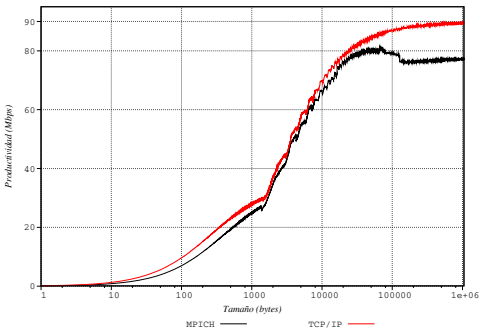
A la vista de los datos de la tabla 3.1 y por simplificar el uso del modelo teórico las estimaciones propuestas son: t_s = 8.8E^{-05} s y t_w = 1.3E^{-06} s.



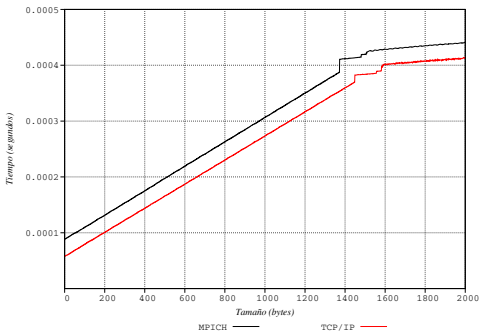
Gráfica 3.3 Saturación en *CLÚSTER*

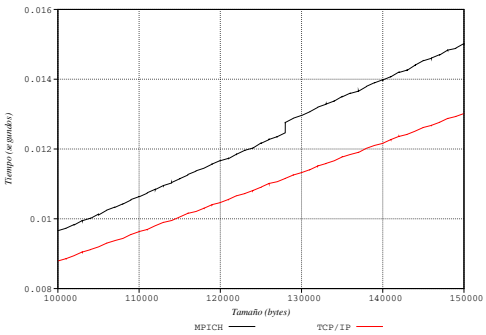


Gráfica 3.4a *Signature* en *CLÚSTER*



Gráfica 3.4b Productividad en *CLÚSTER*





Gráfica 3.5 Evolución del test *PingPong* para ciertos tamaños de mensaje en *CLÚSTER*

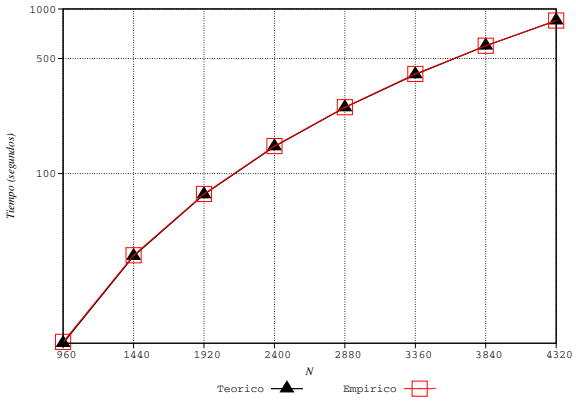
3.6.2 *IBM SP2*

Constante de Cálculo

A diferencia del otro entorno de experimentación, en éste sí influye la forma de almacenar los datos, obteniéndose 2 estimaciones distintas según qué caso. Además, la convergencia o estabilidad de la estimación se obtiene para dimensiones de la matriz más elevadas.

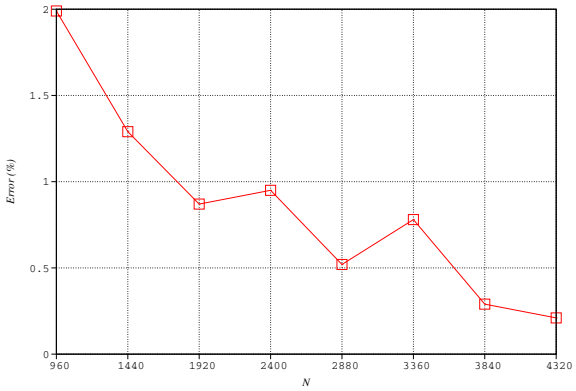
Dado que el almacenamiento bidimensional predomina sustancialmente en las implementaciones de Neville la estimación seleccionada para la constante de cálculo es: t_c = 1.6E^{-08} segundos por *Flop*.

Al igual que en el otro entorno, se ha comparado el tiempo de la ejecución de Neville secuencial con el procedente del modelo teórico usando la estimación. El resultado se muestra en la gráfica 3.6.



Gráfica 3.6 Neville secuencial vs modelo teórico en *IBM SP2*

Como era de esperar el error relativo entre la estimación teórica y la realidad empírica usando (3.16) es mayor que en el otro entorno (gráfica 3.7), pero es inferior al 1% para tamaños del sistema medios/grandes. Por tanto, $1.6E^{-08}$ s es una estimación correcta.



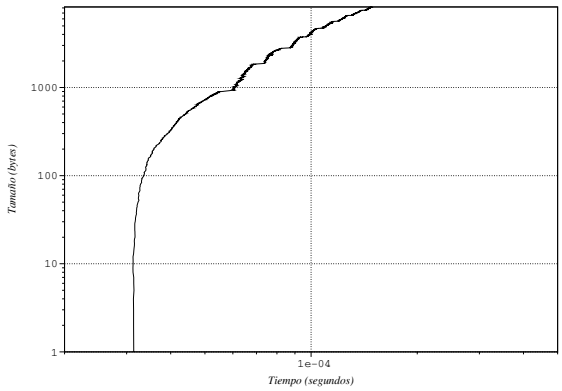
Gráfica 3.7 Error relativo Neville secuencial vs modelo teórico en *IBM SP2*

Constantes de Comunicación

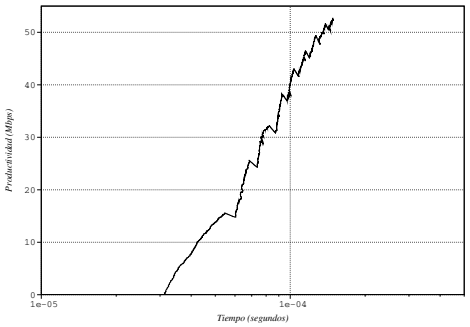
A diferencia del entorno anterior, en éste no es posible modelizar el comportamiento del protocolo subyacente, puesto que para su estudio es necesario usar el *modo de ejecución interactivo* no soportado por el gestor de colas del *IBM SP2*.

El comportamiento de las comunicaciones punto a punto usando *MPICH* es ligeramente distinto al del *CLÚSTER*. Comparten la existencia de un punto de saturación, que en el *IBM SP2* está

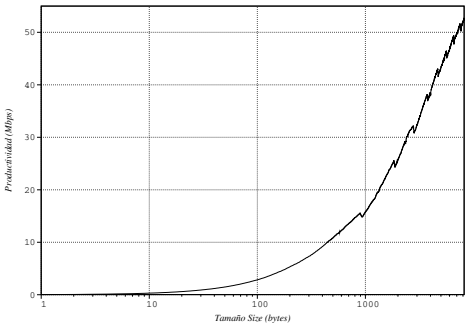
ligeramente por debajo de los 900 *bytes* (ver gráficas 3.8 y 3.10). Desde ese valor, y a intervalos regulares de aproximadamente 900 *bytes*, *IBM SP2* se comporta como una función escalón, como muestra la gráfica 3.10. Nótese que al final del primer intervalo se observa un cambio de pendiente, mientras que en los otros no.



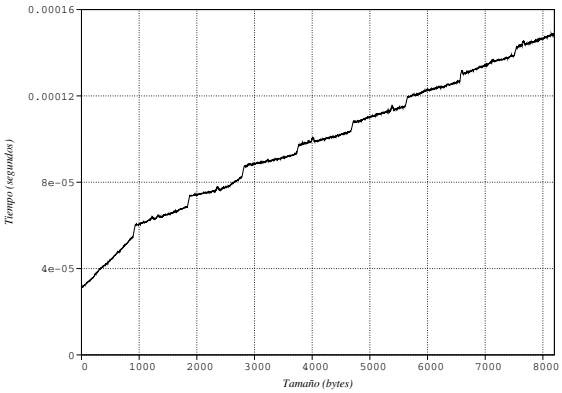
Gráfica 3.8 Saturación en *IBM SP2*



Gráfica 3.9a *Signature* en *IBM SP2*



Gráfica 3.9b Productividad en *IBM SP2*



Gráfica 3.10 Evolución del test *PingPong* para ciertos tamaños de mensaje en *IBM SP2*

Directamente de los Datos	
t_S = 3.05E^{-05}s	t_W = 1.44E^{-07}s por *double*

Por Mínimos Cuadrados		
1 Ajuste	2 Ajustes	
[1,..., 1024x1024] *bytes*	[1,..., 892] *bytes*	[940,..., 1024x1024] *bytes*
t_S =3.17E^{-05}s t_W = 1.09E^{-07}s por *double*	t_S =3.17E^{-05}s t_W = 1.15E^{-07}s por *double*	t_S =5.46E^{-05}s t_W = 9.25E^{-08}s por *double*

Tabla 3.2 Distintas estimaciones de t_s y t_w para *IBM SP2*

Considerando los aspectos mencionados la tabla 3.2 recoge las distintas estimaciones para *MPICH* de t_s y t_w.

A la vista de los datos de la tabla 3.2 y por simplificar el uso del modelo teórico las estimaciones propuestas son: t_s = 3.1E^{-05} s y t_w = 1.2E^{-07} s.

4

Particiones Unidimensionales

Para tratar una matriz en paralelo, debemos dividirla de tal forma que las particiones se puedan asignar a los distintos procesadores. La distribución de los datos de la matriz afecta al rendimiento del sistema paralelo como podremos comprobar a lo largo de los dos próximos capítulos. Por ello, es importante determinar qué esquema de distribución es el más apropiado para cada algoritmo. En las siguientes secciones, analizaremos un tipo de partición en el que la matriz de datos es dividida en grupos de filas o columnas completas consecutivas, asignando a cada procesador uno o varios de esos grupos. Este tipo de esquema de distribución se denomina *Partición Unidimensional,* existiendo dos tipos dependiendo si se agrupan las filas o las columnas y se denominan *Particiones Unidimensionales orientadas a Filas* o *Particiones Unidimensionales orientadas a Columnas* respectivamente. En este capítulo realizaremos un estudio exhaustivo de ambas.

4.1 Particiones Unidimensionales Orientadas a Filas

En esta sección, analizaremos el rendimiento y la escalabilidad de las *Particiones Unidimensionales orientadas a Filas* del algoritmo de Neville y estimaremos las condiciones bajo las cuales una formulación proporciona mejores prestaciones que otra. En primer lugar, estudiaremos el caso general denominado *Bloques de Filas Cíclicos (BFC)*, el cual consiste en dividir la matriz en grupos de m filas completas consecutivas, con $1 \leq m \leq n/k$, donde n es el orden de la matriz y k es el número de procesadores siendo éstos, P_1, P_2, ..., P_k. Estos bloques se asignarán a los procesadores de manera cíclica hasta distribuir todos los bloques. Consideraremos una distribución uniforme, esto es, una distribución en la que todos los bloques tendrán igual número de filas y todos los procesadores tendrán el mismo número de bloques. En esta distribución, determinaremos el tamaño de bloque óptimo que será aquel tamaño de bloque que proporcione el menor tiempo de ejecución. Concluiremos esta sección examinando dos casos particulares de la distribución que acabamos de describir. El primero de ellos, *Bloques de Filas (BF)*, aparece cuando m toma el valor n/k; y el segundo, *Cíclico por Filas (CF)*, cuando m es igual 1. Nótese que las situaciones mencionadas corresponden con los límites, superior e inferior, para el parámetro m (tamaño de bloque).

A lo largo de todo el capítulo consideraremos que estamos llevando a cabo un proceso de eliminación de Neville sobre una matriz A no singular, en el que no es necesario realizar cambios de filas.

4.1.1 Bloques de Filas Cíclicos

Dada la matriz $A=(a_{ij})_{1\leq i,j\leq n}$ dividiremos sus n filas en bloques de m filas completas consecutivas. Estos bloques se asignarán de manera cíclica entre los procesadores, que denotaremos por P_1, P_2, …, P_k, para lo cual consideraremos n divisible por k. Cada procesador tendrá asignados h bloques de tamaño m, donde $m=n/(kh)$, con $h \in [1,n/k]$. Al igual que la matriz A, los elementos del vector $b=(b_i)_{1\leq i\leq n}$ de términos independientes se agruparán en bloques de m elementos consecutivos y dichos bloques se distribuirán entre los procesadores de modo cíclico.

En la figura 4.1 se puede observar cómo se realiza el reparto de los diferentes bloques entre los k procesadores. La matriz se muestra fraccionada en bloques de m filas y cada bloque, empezando por el que contiene las filas de la 1 a la m, se va distribuyendo entre los k procesadores. Una vez asignados los k primeros bloques, se distribuyen los k siguientes y así sucesivamente hasta concluir el total de bloques. En la figura antes mencionada también se puede ver al vector b dividido en bloques de m elementos consecutivos, realizándose la distribución de igual forma que para la matriz de coeficientes.

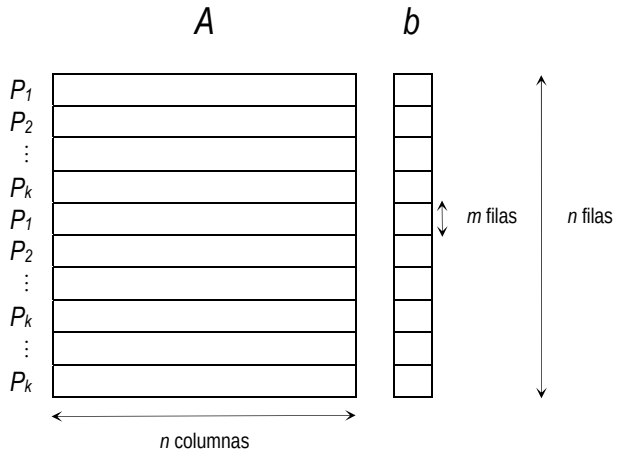
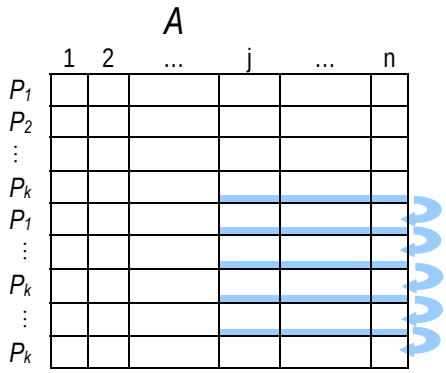


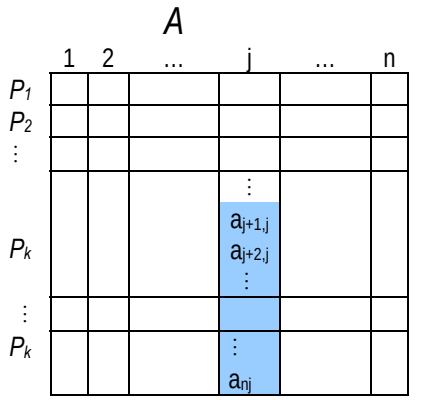
Figura 4.1 Bloques de Filas Cíclicos (*BFC*)

Antes de presentar el algoritmo de la distribución que nos ocupa, vamos a describir brevemente los pasos que tienen lugar a lo largo de la *j*-ésima iteración del proceso de eliminación de Neville:

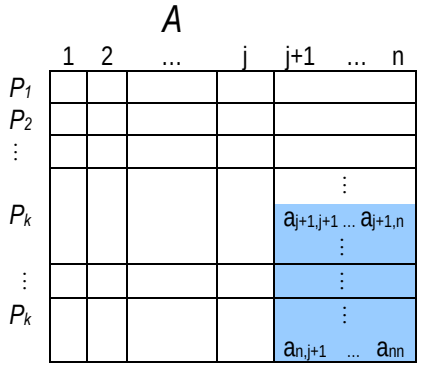
1. Cada procesador activo enviará los elementos a_{ip} de la última fila de cada uno de sus bloques, con $i \geq j$ y $j \leq p \leq n$, al procesador inmediatamente siguiente (ver figura 4.2a). Entenderemos por procesador activo aquel que contiene algún elemento que todavía puede ser modificado a lo largo del proceso de eliminación.
2. Los procesadores tendrán que calcular los multiplicadores que les competan (ver figura 4.2b).
3. Cada procesador realizará las actualizaciones de todos aquellos elementos que corresponden a filas y columnas con un índice superior a *j* (ver figura 4.2c).



(a)
Enviar a_{ip}, bien de P_l a P_{l+1} si $1 \leq l \leq k\text{-}1$ o bien de P_k a P_1, con $j \leq p \leq n$ e $i \geq j$, donde i es la última fila de cada bloque.



(b)
Calcular multiplicadores
$a_{ij} = a_{ij} / a_{i-1,j}$ para $n \geq i \geq j+1$.



(c)
Actualizar la parte activa de la matriz
$a_{ts} = a_{ts} - (a_{ij} / a_{t-1,j})\, a_{t-1,s}$ para $j+1 \leq s \leq n$ y $j+1 \leq t \leq n$.

Figura 4.2 Pasos de la distribución *BFC* durante la iteración *j*

```
1. para j := 1 hasta n – 1 hacer        /* bucle externo */
2.     en paralelo hacer
3.         para todo P_l , con 1 ≤ l < k, hacer
4.             para r := 0 hasta h – 1 hacer
5.                 t := ( r * k + l ) * m;
6.                 si t ≥ j entonces
7.                     para s := j hasta n hacer
8.                         enviar A[t, s] de P_l a P_{l+1};
9.                     enviar b[t] de P_l a P_{l+1};
10.                fin si        /* línea 6 */
11.            fin para        /* línea 4 */
12.        para P_k hacer
13.            para r := 0 hasta h – 2  hacer
14.                t := ( r + 1) * k * m;
15.                si t ≥ j entonces
16.                    para s := j hasta n hacer
17.                        enviar A[t, s] de P_k a P_1;
18.                    enviar b[t] de P_k a P_1;
19.                fin si      /* línea 15 */
20.            fin para      /* línea  13 */
21.    fin en paralelo      /* línea 2 */
22.    en paralelo hacer
23.        para todo P_l , con 1 ≤ l ≤ k, hacer
24.            para r := h – 1 descendiendo hasta 0 hacer
25.                para q := l * m descendiendo hasta (l – 1) * m + 1 hacer
26.                    s := q + r * k * m ;
27.                    si s > j entonces
28.                        A[s, j] := A[s, j] / A[s – 1, j];        /* cálculo de multiplicadores */
29.                        para t := j + 1 hasta n hacer
30.                            A[s, t] := A[s, t] – A[s, j] * A[s – 1, t];        /* paso de eliminación */
31.                        b[s] := b[s] – A[s, j] * b[s – 1];
32.                    fin si      /* línea 27 */
33.                fin para        /* línea 25 */
34.    fin en paralelo        /* línea 22 */
35.  fin para        /* línea 1 */
```

Algoritmo 4.1 Eliminación de Neville mediante *BFC*

En el Algoritmo 4.1 se recogen con más detalle los pasos a realizar sobre la matriz *A* reseñados anteriormente, junto con las operaciones a llevar a cabo en el vector de términos independientes *b*. Como se puede observar, se diferencian las comunicaciones a realizar por el procesador P_k de las comunicaciones a efectuar por el resto de los procesadores (P_l *con* $1 \leq l < k$). Esto es debido a que en el primer caso la comunicación se realiza desde P_k a P_1, mientras que en el segundo caso tiene lugar de P_l a P_{l+1}.

A continuación, estimaremos el tiempo de ejecución de este algoritmo y su eficiencia, así como el tamaño de bloque óptimo. Este tiempo estará expresado como una función del orden de la matriz (*n*), del número de procesadores *(k)*, del tiempo empleado en realizar una operación en punto flotante (t_c), de los parámetros del tiempo de comunicaciones *(*t_s y t_w*)* y, finalmente, del tamaño del bloque de filas *(m)*. Nos concentraremos únicamente en las operaciones sobre la matriz *A* del Algoritmo 4.1, dado que la magnitud de las operaciones sobre el vector *b* es irrelevante. Comenzaremos analizando el tiempo de cálculo para seguidamente abordar el tiempo de comunicaciones.

Tiempo de Cálculo

El primer paso de computación que se desarrolla en el Algoritmo 4.1, tiene lugar cuando cada procesador debe calcular los multiplicadores correspondientes a una iteración (figura 4.2b y línea 28 del Algoritmo 4.1). Obtener un multiplicador supone efectuar un cociente y para determinar el número de cocientes deberemos, en primer lugar, conocer las filas ubicadas en cada procesador. En la siguiente tabla, aparecen los *h* bloques de cada procesador, junto con las filas que integran dichos bloques.

En segundo lugar, vamos a observar cómo se va desarrollando el proceso de eliminación en lo que respecta al número de cocientes, para así poder precisar el número total de los mismos.

En la primera iteración del proceso se elimina la variable x_1 de las ecuaciones *n*, *n*-1, …, 2 . De esta forma todos los procesadores calculan el multiplicador de todas sus filas, esto es, *hm* multiplicadores, excepto el primer procesador que no requiere el multiplicador correspondiente a la fila 1, por tanto, este procesador realiza *hm*-1 cocientes.

En la segunda iteración, en la que se elimina la variable x_2, todos los procesadores realizan sus *hm* cocientes, salvo el primer procesador que efectúa *hm*-2 cocientes debido a que la fila 2 no requiere multiplicador.

PROCESADORES	BLOQUES	FILAS
Procesador 1	Bloque 1	1, 2, …, *m*
	Bloque 2	*km*+1, *km*+2, …, (*k*+1)*m*
	…	…
	Bloque *h*	(*h*-1)*k m*+1, (*h*-1)*k m*+2, …, ((*h*-1)*k*+1)*m*
Procesador 2	Bloque 1	*m*+1, *m*+2, …, 2*m*
	Bloque 2	(*k*+1)*m*+1, (*k*+1)*m*+2, …, (*k*+2)*m*
	…	…
	Bloque *h*	((*h*-1)*k*+1)*m*+1, ((*h*-1)*k*+1)*m*+2, …, ((*h*-1)k+2)*m*
…		…
Procesador k	Bloque 1	(*k*-1)*m*+1, (*k*-1)*m*+2, …, *km*
	Bloque 2	(2*k*-1)*m*+1, (2*k*-1)*m*+2, …, 2*km*
	…	…
	Bloque *h*	(*hk*-1)*m*+1, (*hk*-1)*m*+2, …, *n*

Tabla 4.1 *BFC*: Distribución de filas entre los *k* procesadores

Hasta la iteración *m*, todos los procesadores realizan *hm* cocientes a excepción de P_1 que va disminuyendo el número de cocientes, uno en cada iteración.

En las siguientes *m* iteraciones, el procesador que disminuye el número de cocientes a realizar es el procesador P_2 que pasa de realizar *hm* cocientes a efectuar *hm-m* cocientes. A lo largo de estas iteraciones, el procesador 1 sigue calculando *hm-m* multiplicadores mientras que el resto de los procesadores realiza *hm*.

Agrupando las iteraciones en bloques de tamaño *m*, el proceso anteriormente descrito se repite hasta que en el *k*-ésimo bloque de *m* iteraciones el procesador que ve disminuido el número de multiplicadores a realizar es el procesador P_k, que pasa de *hm* a *hm-m* cocientes, mientras que el resto sigue realizando *hm-m* cocientes.

Esta situación descrita se va repitiendo a lo largo de las restantes iteraciones hasta concluir el proceso de eliminación.

Recordemos que los cálculos se realizan a la vez en todos los procesadores activos y además, como hemos podido observar, el procesador P_k es el que más cocientes efectúa. Por ello, en la siguiente tabla reflejamos el número de cocientes (o de multiplicadores calculados) que lleva a cabo dicho procesador.

ITERACIONES	NÚMERO DE MULTIPLICADORES CALCULADOS
1,2,..., (*k*-1)*m*	*hm*, *hm*, ..., *hm*
(*k*-1)*m*+1, (*k*-1)*m*+2, ..., *km*	*hm*-1, *hm*-2, ..., *hm*-*m*
km+1, *km*+2, ..., (2*k*-1)*m*	*hm*-*m*, *hm*-*m*, ..., *hm*-*m*
(2*k*-1)*m*+1, (2*k*-1)*m*+2, ..., 2*km*	*hm*-(*m*+1), *hm*-(*m*+2), *hm*-2*m*
2*km*+1, 2*km*+2, ..., (3*k*-1)*m*	*hm*-2*m*, *hm*-2*m*, ..., *hm*-2*m*
(3*k*-1)*m*+1, (3*k*-1)*m*+2, ..., 3*km*	*hm*-(2*m*+1), *hm*-(2*m*+2), ..., *hm*-3*m*
3*km*+1, 3*km*+2, ..., (4*k*-1)*m*	*hm*-3*m*, *hm*-3*m*, ..., *hm*-3*m*
...	...
(*h*-1)*km*+1, (*h*-1)*km*+2, ..., (*hk*-1)*m*	*m*, *m*, ..., *m*
(*hk*-1)*m*+1, (*hk*-1)*m*+2, ..., *hkm*-1	*m*-1, *m*-2, ..., 1

Tabla 4.2 *BFC*: Número de multiplicadores calculados por el procesador P_k

La situación anterior se puede recoger en los siguientes sumandos:

$$\sum_{i=0}^{h-1}\sum_{j=ikm+1}^{((i+1)k-1)m}(hm-im)+\sum_{i=1}^{h}\sum_{j=(ik-1)m+1}^{ikm}(hm-(j-i(k-1)m)),$$

siendo el número total de cocientes

$$\frac{1}{2}h^2m^2k+\frac{1}{2}hm^2k-\frac{1}{2}hm^2-\frac{1}{2}hm.$$

El segundo paso de computación consiste en la actualización por parte de cada procesador de la parte activa de la matriz que le corresponde, es decir, la actualización de aquellas filas y columnas cuyo índice sea superior al número de iteración en el que nos encontremos (figura 4.2c y línea 30 del Algoritmo 4.1). En este caso, se realizan una resta y un producto para actualizar cada elemento. El razonamiento para determinar el número de restas y productos que se llevan a cabo es idéntico al seguido en el caso de los cocientes, sólo queda añadir que en cada fila a actualizar se realizan *n-j* actualizaciones de elementos, o lo que es lo mismo *n-j* restas y *n-j* productos.

Por tanto el número de restas resulta ser

$$\sum_{i=0}^{h-1}\sum_{j=ikm+1}^{((i+1)k-1)m}((hm-im)(n-j))+\sum_{i=1}^{h}\sum_{j=(ik-1)m+1}^{ikm}((hm-(j-i(k-1)m))(n-j)),$$

con lo que desarrollando los sumatorios anteriores se obtiene que el número total de restas es

$$\left(3km^2h - 6nmh - 3k^2m^2h^2 - k^2m^2h + 2h - 2m^2h + 6nh^2mk + 3h^2m^2k - 2h^3m^2k^2 + 6nkmh - 6nh\right)\frac{m}{12}.$$

En cuanto al número total de productos, éste es exactamente igual al de restas. Una vez determinado el número de operaciones de los dos pasos de computación, el tiempo de cálculo de esta distribución es la suma del número de cocientes, del número de restas y del número de productos. Asumimos que el coste de una unidad de computación es t_c. Así, el tiempo de cálculo es el siguiente

$$T_{CÁLCULO\_BFC} = \left(\frac{1}{2}km^3h - nm^2h - \frac{1}{2}k^2m^3h^2 - \frac{1}{6}k^2m^3h - \frac{1}{6}hm - \frac{1}{3}m^3h + nh^2m^2k + \frac{1}{2}h^2m^3k - \frac{1}{3}h^3m^3k^2 + \right.$$
$$\left. + nkm^2h - nhm + \frac{1}{2}m^2h^2k + \frac{1}{2}m^2hk - \frac{1}{2}m^2h\right)t_c,$$

y sustituyendo en la expresión anterior h por $n/(km)$, se tiene

$$T_{CÁLCULO\_BFC} = \left(\frac{2}{3}\frac{n^3}{k} + \frac{1}{2}mn^2 - \frac{1}{2}\frac{n^2m}{k} - \frac{1}{2}\frac{n^2}{k} - \frac{1}{6}nkm^2 + \frac{1}{2}m^2n - \frac{1}{3}\frac{m^2n}{k} + \frac{1}{2}nm - \frac{1}{2}\frac{mn}{k} - \frac{1}{6}\frac{n}{k}\right)t_c. \qquad (4.1)$$

Tiempo de Comunicación

En este tipo de distribución tiene lugar una comunicación entre procesadores vecinos como ya mostramos en la figura 4.2a (líneas 8 y 17 del Algoritmo 4.1). Cada procesador contiene h bloques de m filas cada uno y las filas que corresponden a cada procesador aparecen detalladas en la tabla 4.1.

A medida que se van sucediendo las iteraciones del proceso de eliminación veremos cómo se va realizando la comunicación entre los procesadores:

En la primera iteración, donde se elimina la variable x_1 de las ecuaciones n, n-1, ..., 2, se necesita que todo procesador transmita al procesador inmediatamente inferior la parte activa de la última fila de cada uno de sus h bloques, esto es, la parte activa de h filas. Entendemos por parte activa de una fila, todos aquellos elementos cuyo índice de columna sea superior a j, siendo j la iteración en curso. El procesador P_k no tendría que comunicar la última fila correspondiente a su último bloque, por tanto, este procesador transfiere h-1 filas al procesador P_1.

En la segunda iteración, en la que se elimina la variable x_2 de las ecuaciones n, n-1, ..., 3, todos los procesadores comunican h filas, excepto el procesador k. Esto mismo sucede hasta la iteración m inclusive, concretamente en esta iteración se elimina la variable x_m en las ecuaciones n, n-1, ..., m+1.

La situación cambia en la iteración m+1, en la que se elimina la variable x_{m+1} en las ecuaciones n, n-1, ..., m+2. En este momento del proceso, el procesador P_2 no necesita la fila m, que se encuentra

en el procesador P_1, por tanto los procesadores P_1 y P_k transmiten h-1 filas y el resto de los procesadores comunican h filas.

Durante las m-1 iteraciones siguientes, iteración m+2, iteración m+3, …, iteración 2m, la situación se mantiene, modificándose en la iteración 2m+1 donde el procesador P_2 no necesita comunicar su fila 2m al procesador P_3, así que los procesadores P_1, P_2 y P_k transmiten h-1 filas y el resto de los procesadores comunican h filas.

Como se puede apreciar, por cada bloque de m iteraciones, uno de los procesadores pasa de transferir h filas a transferir h-1 filas, siendo P_{k-1} el último procesador que sufre esta disminución. En las iteraciones $(k\text{-}2)m+1$, $(k\text{-}2)m+2$, …, $(k\text{-}1)m$, todos los procesadores comunican h-1 filas excepto el procesador P_{k-1} que transmite h filas. En la iteración $(k\text{-}1)m+1$ este procesador también pasará a comunicar h-1 filas ya que su fila $(k\text{-}1)m$ no es requerida por el procesador P_k. Esta situación se mantendrá en las m-1 iteraciones siguientes.

Por consiguiente, el procesador que realiza mayor número de comunicaciones es el procesador P_{k-1} y la siguiente tabla recoge como avanzan las iteraciones a lo largo del proceso y cuál es el número de filas a transmitir en cada etapa por P_{k-1}:

ITERACIONES	NÚMERO DE FILAS COMUNICADAS
1,2,…, $(k\text{-}1)m$	h, h, …, h
$(k\text{-}1)m$+1, $(k\text{-}1)m$+2, …, $(2k\text{-}1)m$	h-1, h-1, …, h-1
$(2k\text{-}1)m$+1, $(2k\text{-}1)m$+2, …, $(3k\text{-}1)m$	h-2, h-2, …, h-2
$(3k\text{-}1)m$+1, $(3k\text{-}1)m$+2, …, $(4k\text{-}1)m$	h-3, h-3, …, h-3
…	…
$((h\text{-}1)k\text{-}1)m$+1, $((h\text{-}1)k\text{-}1)m$+2, …, $(hk\text{-}1)m$	1

Tabla 4.3 *BFC*: Número de filas comunicadas en cada etapa por el procesador P_{k-1}

Como se puede observar en la tabla, hay un bloque inicial de $(k\text{-}1)m$ iteraciones en las que se comunican h filas mientras que en el resto de iteraciones del proceso, agrupadas en bloques de km iteraciones, el número de filas a comunicar va disminuyendo una unidad de cada vez.

Una vez calculado el número de filas que comunica el procesador P_{k-1} en función de las iteraciones del proceso, podemos obtener el tiempo de comunicación también en función de éstas. Para determinar el coste de comunicación vamos a utilizar dos sumandos; el primero de ellos, contabiliza el número de filas que transmite el procesador P_{k-1} en las primeras $(k\text{-}1)m$ iteraciones y el segundo, registra el número de filas que el procesador P_{k-1} transfiere en el resto de las iteraciones, agrupadas éstas en bloques de km iteraciones como hemos visto anteriormente. Indicar que la información transferida en cada iteración es el número de filas multiplicado por el número de elementos que pertenecen a la parte activa de cada fila, siendo este número, n-j+1.

Como ya se mencionó al comienzo de este apartado, el tipo de comunicación que se realiza es un envío entre procesadores vecinos, siendo el coste de dicha operación el que se indicó en (3.4). Teniendo en cuenta lo ya comentado, el coste de comunicación puede expresarse como

$$\sum_{j=1}^{(k-1)m}\left(t_s + h(n-j+1)t_w\right) + \sum_{i=0}^{h-2}\sum_{j=(k-1)m+ikm+1}^{(k-1)m+(i+1)km}\left(t_s + (h-(i+1))(n-j+1)t_w\right),$$

siendo éste

$$T_{COMUNICACIÓN\_BFC} = (mkh-m)t_s + \left(\frac{1}{2}hnmk - hnm + \frac{1}{4}hmk - \frac{1}{2}hm - \frac{1}{12}hm^2k^2 + \frac{1}{2}hm^2k - \frac{1}{2}hm^2 - \right.$$
$$\left. - \frac{1}{6}h^3m^2k^2 - \frac{1}{4}h^2m^2k^2 + \frac{1}{2}h^2m^2k + \frac{1}{4}h^2mk + \frac{1}{2}h^2nmk\right)t_w,$$

sustituyendo *h* por *n/(km)* se tiene

$$T_{COMUNICACIÓN\_BFC} = (n-m)t_s + \left(\frac{1}{3}\frac{n^3}{km} + \frac{1}{4}n^2 - \frac{1}{2}\frac{n^2}{k} + \frac{1}{4}n - \frac{1}{2}\frac{n}{k} - \frac{1}{12}nkm + \frac{1}{2}nm - \frac{1}{2}\frac{nm}{k} + \frac{1}{4}\frac{n^2}{km}\right)t_w. \qquad (4.2)$$

Una vez obtenidos los tiempos de cálculo y de comunicación analizamos el valor de la eficiencia asintótica de nuestra distribución. A partir de (4.1), de (4.2) y del tiempo del algoritmo secuencial de Neville, siendo éste (2/3) n^3 - (1/2) n^2 - (1/6) $n\,t_c$ (ver (2.25)), podemos obtener la expresión de la eficiencia teórica de *BFC* para valores suficientemente grandes de *n*, siendo ésta

$$E_{BFC} = \frac{2mt_c}{2mt_c + t_w}. \qquad (4.3)$$

Como podemos observar, es una expresión dependiente de la constante de cálculo y de la constante de comunicación t_w, así como del tamaño de bloque utilizado, que a su vez es función de *n* y *k*.

Tamaño de Bloque Óptimo

Como se explicó al inicio de esta sección, la distribución *BFC* es un caso general, esto quiere decir que dependiendo del valor del tamaño de bloque *(m)* se obtienen diferentes divisiones de la matriz de coeficientes *A* entre los *k* procesadores. El presente apartado tiene como objetivo determinar qué tamaño de bloque es el que nos proporciona un menor tiempo de ejecución, y por tanto una mayor eficiencia, para posteriormente utilizar esta información en los experimentos. Teniendo en cuenta que el tiempo de ejecución total del Algoritmo 4.1 depende de las variables n, k, m, t_c, t_s y t_w, siendo estas tres últimas dependientes del entorno de experimentación, realizaremos el estudio para diferentes valores de n y k en los dos ámbitos de pruebas. Recordar que los valores estimados para estas constantes fueron mostrados en el capítulo 3.

k	n	$m_{óptimo}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
4	960	41.296	2.836	0.831
	1440	50.215	9.239	0.861
	1920	57.763	21.459	0.879
	2400	64.423	41.346	0.891
	2880	70.449	70.739	0.900
	3360	75.993	111.469	0.907
	3840	81.154	165.364	0.913
	4320	86.003	234.247	0.918
8	960	27.997	1.556	0.758
	1440	33.777	4.978	0.799
	1920	38.687	11.446	0.824
	2400	43.026	21.906	0.841
	2880	46.956	37.296	0.854
	3360	50.574	58.547	0.864
	3840	53.944	86.589	0.872
	4320	57.111	122.347	0.878
16	960	20.187	0.876	0.673
	1440	24.003	2.740	0.726
	1920	27.293	6.220	0.758
	2400	30.217	11.804	0.781
	2880	32.875	19.971	0.797
	3360	35.326	31.201	0.810
	3840	37.612	45.966	0.821
	4320	39.763	64.741	0.830

Tabla 4.4 *BFC*: Valores de $m_{óptimo}$ para *IBM SP2*

Para realizar el cálculo del tamaño de bloque óptimo de cada entorno hemos procedido a sustituir los valores correspondientes a t_c, t_s y t_w en la expresión del tiempo de ejecución; quedará por tanto una función dependiente de n, k y m. Fijados los valores de n y k, derivamos respecto a m, igualamos a 0 y obtenemos aquellos valores de m que anulan la derivada. De ese modo, aquellos valores de m que anulen la derivada y que sean positivos, junto con m=1 y m=n/k, nos permitirán determinar el tamaño de bloque óptimo ($m_{óptimo}$) para cada n y k.

En la tabla anterior se recogen los resultados obtenidos para el *IBM SP2* con el tipo de dato *double*, con valores de n comprendidos entre 960 y 4320 y para 4, 8 y 16 procesadores. Como se puede observar, figura el tamaño de bloque óptimo ($m_{óptimo}$) para esos valores concretos de n y k, al igual que las estimaciones del tiempo de ejecución (en segundos) y de la eficiencia. Así, podemos ver como la eficiencia estimada para matrices 4320x4320 rebasa el 0.9 para el caso de 4 procesadores, se aproxima a 0.88 en 8 procesadores y es 0.83 cuando se utilizan 16 procesadores.

Los valores de $m_{óptimo}$ que figuran en esta tabla corresponden con el tamaño de bloque óptimo exacto. Estos valores no permiten una distribución uniforme de los datos, es decir, no permiten que todos los bloques tengan el mismo tamaño y que todos los procesadores tengan el mismo número de bloques. Con objeto de realizar los experimentos que permitan verificar las estimaciones realizadas, hemos procedido a calcular aquellos valores cercanos al tamaño de bloque óptimo que proporcionen una distribución uniforme de la matriz A, es decir

$$m_{inferior} = max \{ m \in IN \mid m \leq m_{óptimo} \wedge n \; MOD \; (k \; m) = 0 \} \tag{4.4}$$

y

$$m_{superior} = min \{ m \in IN \mid m \geq m_{óptimo} \wedge n \; MOD \; (k \; m) = 0 \}. \tag{4.5}$$

En las tablas 4.5, 4.6 y 4.7, se muestran dichos valores para 4, 8 y 16 procesadores respectivamente. El tamaño de bloque óptimo exacto ($m_{óptimo}$) para un determinado n aparece en el centro de la fila correspondiente a dicho orden de la matriz, mientras que en la parte superior figura $m_{inferior}$ y en la inferior, $m_{superior}$.

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	40	2.837	0.831
	41.296	2.836	0.831
	48	2.841	0.830
1440	45	9.246	0.861
	50.215	9.239	0.861
	60	9.257	0.860
1920	48	21.500	0.878
	57.763	21.459	0.879
	60	21.461	0.879
2400	60	41.357	0.891
	64.423	41.346	0.891
	75	41.395	0.890
2880	60	70.825	0.899
	70.449	70.739	0.900
	72	70.740	0.900
3360	70	110.502	0.907
	75.993	111.469	0.907
	84	111.518	0.907
3840	80	165.365	0.913
	81.154	165.364	0.913
	96	165.558	0.912
4320	72	234.541	0.916
	86.003	234.247	0.918
	90	234.266	0.918

Tabla 4.5 *BFC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *IBM SP2 y* 4 procesadores

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	24	1.560	0.756
	27.997	1.556	0.758
	30	1.557	0.757
1440	30	4.984	0.798
	33.777	4.978	0.799
	36	4.980	0.799
1920	30	11.504	0.820
	38.687	11.446	0.820
	40	11.447	0.824
2400	30	22.112	0.833
	43.026	21.906	0.841
	50	21.941	0.840
2880	45	37.300	0.854
	46.956	37.296	0.854
	60	37.442	0.850
3360	42	58.673	0.862
	50.574	58.547	0.864
	60	58.652	0.862
3840	48	86.658	0.871
	53.944	86.589	0.872
	60	86.646	0.871
4320	54	122.369	0.878
	57.111	122.347	0.878
	60	122.364	0.878

Tabla 4.6 *BFC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *IBM SP2 y* 8 procesadores

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	20	0.876	0.673
	20.187	0.876	0.673
	20	0.876	0.673
1440	18	2.764	0.720
	24.003	2.740	0.726
	30	2.754	0.722
1920	24	6.230	0.757
	27.293	6.220	0.758
	30	6.226	0.758
2400	30	11.804	0.781
	30.217	11.804	0.781
	30	11.804	0.781
2880	30	19.986	0.797
	32.875	19.971	0.797
	36	19.985	0.797
3360	35	31.201	0.810
	35.326	31.201	0.810
	35	31.201	0.810
3840	30	46.152	0.818
	37.612	45.966	0.821
	40	45.980	0.821
4320	30	65.131	0.825
	39.763	64.741	0.830
	45	64.814	0.829

Tabla 4.7 *BFC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *IBM SP2 y* 16 procesadores

El mismo estudio se ha llevado a cabo para el *CLÚSTER* y sus resultados se muestran en las siguientes tablas (tablas 4.8-4.10).

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	80	8.023	0.698
	91.567	8.005	0.699
	120	8.075	0.693
1440	90	25.511	0.741
	110.435	25.394	0.744
	120	25.413	0.744
1920	120	58.016	0.772
	126.466	58.000	0.773
	160	58.317	0.768
2400	120	110.687	0.791
	140.635	110.427	0.793
	150	110.470	0.792
2880	144	187.297	0.808
	153.467	187.231	0.808
	180	187.641	0.806
3360	140	293.596	0.818
	165.279	292.930	0.820
	168	292.936	0.820
3840	160	432.338	0.830
	176.281	432.020	0.830
	192	432.265	0.830
4320	180	609.035	0.838
	186.619	608.976	0.838
	216	609.943	0.837

Tabla 4.8 *BFC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *CLÚSTER y* 4 procesadores

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	60	4.691	0.597
	66.991	4.684	0.598
	120	4.844	0.578
1440	60	14.647	0.645
	78.149	14.517	0.651
	90	14.552	0.649
1920	80	32.711	0.685
	88.105	32.674	0.686
	120	33.028	0.678
2400	75	62.041	0.705
	97.063	61.564	0.711
	100	61.570	0.711
2880	90	103.836	0.728
	105.249	103.556	0.730
	120	103.747	0.729
3360	105	161.083	0.746
	112.826	160.996	0.746
	140	161.765	0.742
3840	120	236.207	0.759
	119.909	236.207	0.759
	120	236.207	0.759
4320	108	332.322	0.768
	126.582	331.500	0.770
	135	331.633	0.770

Tabla 4.9 *BFC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *CLÚSTER y* 8 procesadores

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	60	2.775	0.504
	64.845	2.771	0.505
	---	---	---
1440	45	8.584	0.555
	63.533	8.472	0.558
	90	8.545	0.553
1920	60	18.849	0.594
	67.698	18.817	0.595
	120	19.334	0.579
2400	50	35.715	0.613
	72.707	35.088	0.624
	75	35.092	0.624
2880	60	59.017	0.641
	77.665	58.534	0.646
	90	58.679	0.644
3360	70	90.676	0.662
	82.426	90.385	0.664
	105	90.977	0.660
3840	80	131.964	0.680
	86.967	131.856	0.680
	120	133.352	0.673
4320	90	184.154	0.693
	91.301	184.15	0.693
	135	187.17	0.682

Tabla 4.10 *BFC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *CLÚSTER y* 16 procesadores

Teniendo en cuenta los resultados obtenidos en las tablas 4.5-4.10, es inmediato comprobar que tanto el tiempo de ejecución como la eficiencia son muy similares para el tamaño de bloque óptimo y sus aproximaciones.

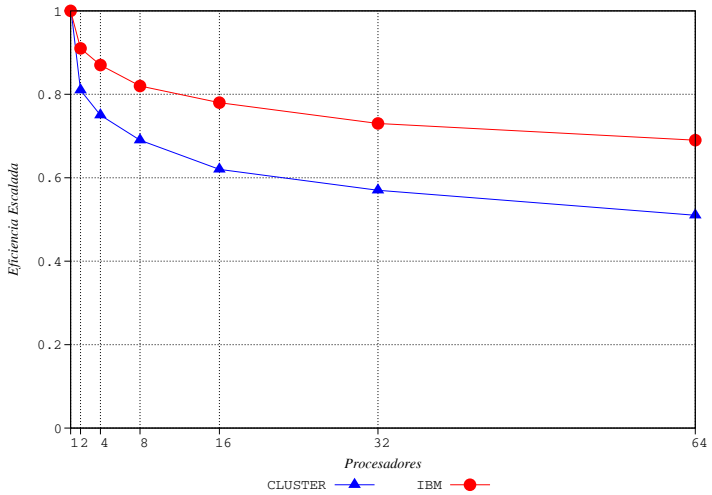
Por otro lado, se puede observar como los valores de las constantes de cada entorno influyen notablemente en los resultados, comprobándose que:

- llega a existir una diferencia a lo sumo de 0.169 en la eficiencia para el caso del menor tamaño de matriz analizado y 16 procesadores,
- a medida que el tamaño de la matriz aumenta, dentro del mismo número de procesadores, la diferencia entre las eficiencias de ambas plataformas se va haciendo algo menor,
- el recorte de la diferencia anteriormente mencionada es más pequeño cuanto mayor sea el número de procesadores que consideremos,
- el tamaño de bloque óptimo del *IBM* viene a ser, en muchos casos, aproximadamente la mitad del obtenido en el *CLÚSTER* para el mismo valor de n y k, y
- en términos generales, el tiempo estimado en el *CLÚSTER* es entorno al triple del tiempo calculado para el *IBM*. Si en lugar de estudiar el tiempo total del algoritmo, analizásemos por separado el tiempo de cálculo y el tiempo de comunicación, comprobaríamos que el primero es alrededor de 2.5 veces mayor en el *CLÚSTER* que en el *IBM*, mientras que el segundo, es 5 veces mayor.

Análisis de la Escalabilidad

En el capítulo 3 se vieron dos alternativas para el cálculo de la escalabilidad, que son: función de isoeficiencia y eficiencia escalada. El uso del enfoque basado en la función de isoeficiencia para el caso general conduce a una expresión compleja donde resulta difícil ver una relación clara y simple de dependencia entre W y k, por lo que se ha optado por el modelo escalado en esta distribución genérica. Cuando su cálculo basado en la función de isoeficiencia nos lleve a una expresión más clara, como es el caso de las distribuciones particulares, se analizarán ambas opciones.

La escalabilidad de esta distribución la examinaremos a través de la eficiencia escalada (ver (3.15)). En la gráfica que figura a continuación se refleja la evolución de la eficiencia a medida que el número de procesadores y W crecen en la misma cuantía. La eficiencia claramente no permanece constante, siendo su degradación más suave en el *IBM* que en el *CLÚSTER*. Sin embargo, como han señalado otros autores, puede considerarse casi escalable si su eficiencia escalada sigue siendo mayor que 0, es decir, $E_{ESCALADA}(W,k) > 0$ (ver [Prieto *et al* 03]). En el Anexo B de esta memoria se recogen los datos que han permitido generar las gráficas de la eficiencia escalada que se muestran en el presente capítulo así como en el siguiente. En la sección B.1.1 de dicho anexo podemos comprobar con más detalle como la eficiencia en el caso del *IBM* decae hasta 0.68 mientras que en el *CLÚSTER* desciende hasta 0.51.



Gráfica 4.1 *BFC*: Eficiencia escalada

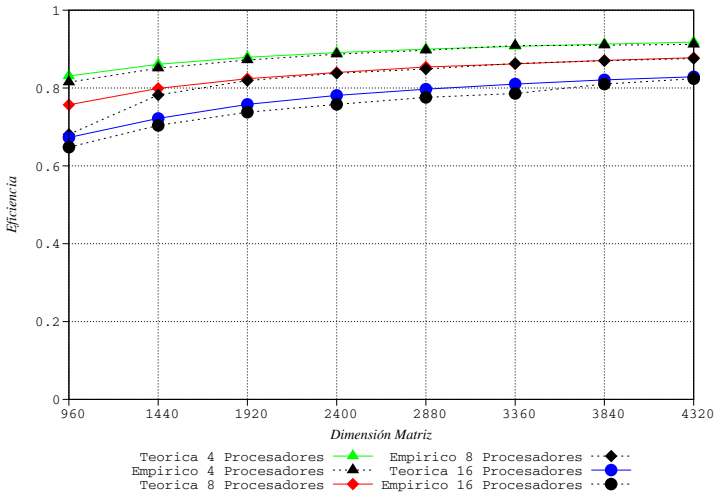
Experimentos

Como hemos visto a lo largo de la sección, esta distribución se caracteriza por tener un modelo de comunicaciones sencillo y un buen balanceado de la carga. Las comunicaciones son siempre entre vecinos, a lo sumo dos vecinos, y con un grado de solapamiento muy elevado. El número de comunicaciones de entrada/salida por etapa y procesador es, al principio, igual al número de bloques que tenga cada procesador y a medida que se suceden las etapas del proceso, ese número de comunicaciones va disminuyendo. Cabe esperar por tanto que esa distribución equitativa de los datos (equilibrio en la intensidad computación) y la simplicidad del modelo de comunicación ofrezca buenos resultados tal y como auguran las estimaciones. En este apartado mostraremos los resultados experimentales obtenidos al implementar el Algoritmo 4.1 en los dos entornos descritos en el capítulo 3. De ese modo podremos verificar los resultados analíticos derivados anteriormente. El tamaño de bloque empleado para los valores de n y k utilizados lo denominaremos $m_{pseudoóptimo}$ y se obtuvo de las tablas 4.5-4.10 del siguiente modo

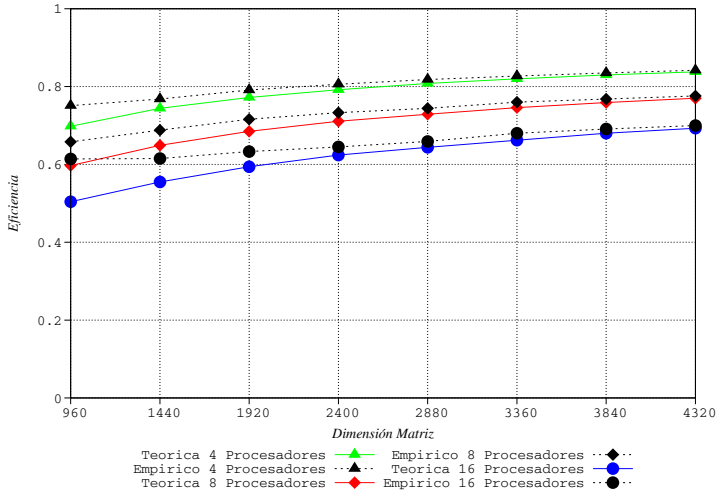
$$m_{pseudoóptimo} = si\ (m_{óptimo} - m_{inferior}) \leq (m_{superior} - m_{óptimo})\ entonces\ m_{inferior}\ si\ no\ m_{superior}\ finsi. \quad (4.6)$$

Las eficiencias obtenidas en las experimentaciones se muestran en las siguientes gráficas donde se confrontan con las eficiencias teóricas. En el Anexo A se recogen los tiempos exactos empleados por cada uno de los experimentos en cada entorno utilizado, así como la eficiencia empírica resultante.

En las gráficas 4.2 y 4.3 se observa que en el *IBM* las expectativas teóricas no son superadas por los valores empíricos, no ocurriendo así en el *CLÚSTER*. Vistas las propiedades de esta distribución, se podría concluir que el valor usado en el modelo teórico para las constantes de cálculo y de comunicaciones es optimista, o pesimista en el otro sistema paralelo utilizado. En un sentido o en el otro, esas diferencias están entre 0.002 y 0.006 para matrices de 4320x4320 y k = 4, 8 y 16, en el caso del *IBM* y entre 0.004 y 0.007, en el caso del *CLÚSTER*. Esto viene a suponer a lo sumo una diferencia en términos porcentuales de 0.65% en el *IBM* y de 1% en el *CLÚSTER*, lo que resulta insignificante. Observar que el grado de coincidencia entre las estimaciones teóricas y los datos derivados de los experimentos va aumentando a medida que el tamaño del problema aumenta, por ejemplo en el caso del *IBM* existe una coincidencia de un 96.3% para 16 procesadores y n=960 y ésta va creciendo paulatinamente hasta ser del 99.4% en el caso de n=4320 y el mismo número de procesadores. La menor similitud en tamaños pequeños puede deberse a efectos colaterales de la red (*cache*, jerarquía de memoria, ...).



Gráfica 4.2 *BFC* con $m_{pseudoóptimo}$: Eficiencia teórica vs. empírica para *IBM SP2*



Gráfica 4.3 *BFC* con $m_{pseudoóptimo}$: Eficiencia teórica vs. empírica para *CLÚSTER*

Por otro lado al comparar las gráficas de *BFC* podemos observar algo previsto con las estimaciones y es el hecho de que las eficiencias empíricas alcanzadas en el *IBM* llegan a tener una diferencia de 0.12 con las obtenidas en el *CLÚSTER*. Esta diferencia es menor que la que indicaban los análisis teóricos ya que las expectativas para el *CLÚSTER* fueron levemente pesimistas.

A través de las gráficas 4.2 y 4.3 hemos podido confirmar como los modelos teórico y empírico tienen un comportamiento similar, alcanzando unas eficiencias muy elevadas tanto en un entorno como en otro. Por ello, podemos deducir la conducta del modelo empírico usando el modelo teórico. El resultado de esta extrapolación se puede observar en la siguiente tabla, donde se puede apreciar las eficiencias alcanzadas para los dos entornos, considerando una matriz de orden 2^{15} y un número de procesadores del tipo 2^i, con i = 2, 3, 4, 5, 6 y 7.

k	*IBM SP2*	*CLÚSTER*
4	0.969	0.935
8	0.953	0.904
16	0.932	0.865
32	0.905	0.816
64	0.870	0.755
128	0.823	0.682

Tabla 4.11 *BFC*: Estimación de la eficiencia para $n=2^{15}$

Vemos como la eficiencia está en su punto álgido en 0.969 y 0.935 para *IBM* y *CLÚSTER* respectivamente, existiendo por tanto una diferencia de 0.034 entre ambas. A medida que el número de

procesadores aumenta, la eficiencia disminuye, llegando a ser 0.823 y 0.682 para *IBM* y *CLÚSTER* respectivamente, lo que hace que la diferencia entre las eficiencias de los dos entornos se acentúe dado que llega a ser de 1.41, siempre a favor del *IBM*.

4.1.2 Bloques de Filas

Seguidamente vamos a analizar uno de los casos particulares de la distribución *BFC*. Nos referimos a la partición *Bloques de Filas (BF)*, en la cual el tamaño de los bloques en los que se divide las filas de la matriz *A* es *n/k*, siendo *n* el orden de la matriz, *k* el número de procesadores y considerando *n* divisible por *k*. De igual forma, los elementos del vector *b* de términos independientes se dividen en grupos de *n/k* elementos consecutivos. A la hora de asignar los bloques entre los *k* procesadores, cada procesador tendrá adjudicado un único bloque de las filas de *A* y un único bloque de los elementos de *b*. En la figura 4.3 se observa como se realiza el reparto de los elementos de *A* y *b*.

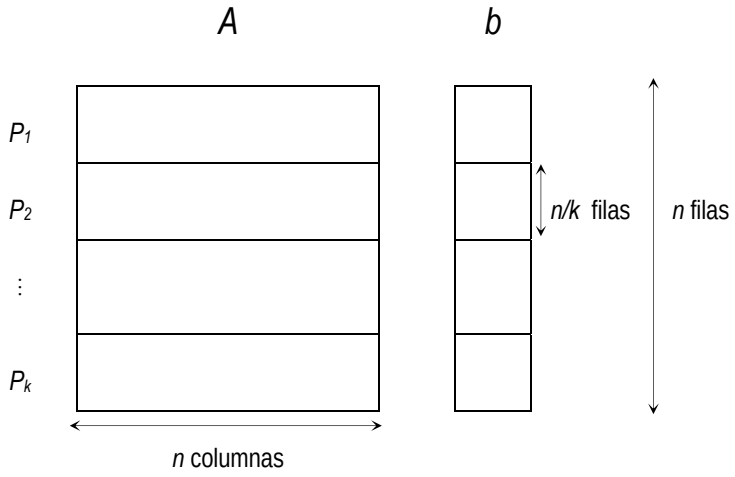


Figura 4.3. Bloques de Filas (*BF*)

Para proceder a realizar un análisis exhaustivo de esta distribución, veamos en primer lugar las expresiones de sus tiempos de cálculo y de comunicación, para así estudiar la eficiencia que nos proporciona dicha formulación paralela.

Las expresiones de los tiempos de cálculo y de comunicación se obtienen al sustituir en (4.1) y (4.2) el tamaño de bloque *(m)* por su valor en esta distribución, es decir, *n/k*, resultando las expresiones (4.7) y (4.8):

$$T_{CÁLCULO\_BF} = \left(\frac{n^3}{k} - \frac{1}{3}\frac{n^3}{k^3} - \frac{1}{2}\frac{n^2}{k^2} - \frac{1}{6}\frac{n}{k} \right) t_c \tag{4.7}$$

y

$$T_{COMUNICACIÓN\_BF} = \left(n - \frac{n}{k} \right) t_s + \left(\frac{1}{2}n^2 - \frac{1}{2}\frac{n^2}{k^2} + \frac{1}{2}n - \frac{1}{2}\frac{n}{k} \right) t_w \,. \tag{4.8}$$

De ese modo, el tiempo de ejecución paralelo total es aproximadamente

$$T_{PARALELO\_BF} \approx \frac{n^3}{k} t_c + n t_s + \frac{1}{2} n^2 t_w \tag{4.9}$$

y la expresión de la eficiencia para valores elevados del tamaño de la matriz resulta ser

$$E_{BF} = \frac{2k^2}{3k^2 - 1}\,, \tag{4.10}$$

dependiente de *k* y que podemos considerar de manera aproximada como *2/3*.

Análisis de la Escalabilidad

El producto tiempo-procesador para esta formulación paralela es $n^3 t_c + n\,k\,t_s + (n^2/2)\,k\,t_w$; si se ignoran los costes de comunicación, la expresión pasa a ser n^3. Así, el coste del algoritmo paralelo es 3/2 el tiempo del algoritmo secuencial (ver (2.25)). Por consiguiente, hay un límite superior de 2/3 en la eficiencia del algoritmo paralelo. Esta ineficiencia en *BF* es debida al tiempo de inactividad de los procesadores, el cual viene motivado por el mal balanceado de la carga. En principio, este problema podría subsanarse en la partición *CF* al conseguirse que haya a lo sumo una diferencia de una fila entre dos procesadores cualesquiera. El análisis de la distribución cíclica se abordará en la subsección 4.1.3.

Analizaremos la escalabilidad de *BF* a través de la función de isoeficiencia, para ello deberemos previamente obtener la sobrecarga relativa a la comunicación de este algoritmo. Recordemos que la función de sobrecarga (T_O) se define como $k\,T_{PARALELO}(W,k) - W$ (ver (3.11)), la cual en la distribución que nos ocupa resulta ser igual a la siguiente expresión:

$$T_{O\_BF} \approx n\,k\,t_s + (n^2/2)\,k\,t_w.$$

Para calcular el término de la isoeficiencia debido a t_s, W tiene que ser proporcional a $K\,n\,k\,t_s$, donde $K = E/(1-E)$, siendo E la eficiencia deseada y que tiene que ser mantenida. La relación de isoeficiencia, $W=KT_O(W,k)$, expresado en (3.14), resulta ser la siguiente:

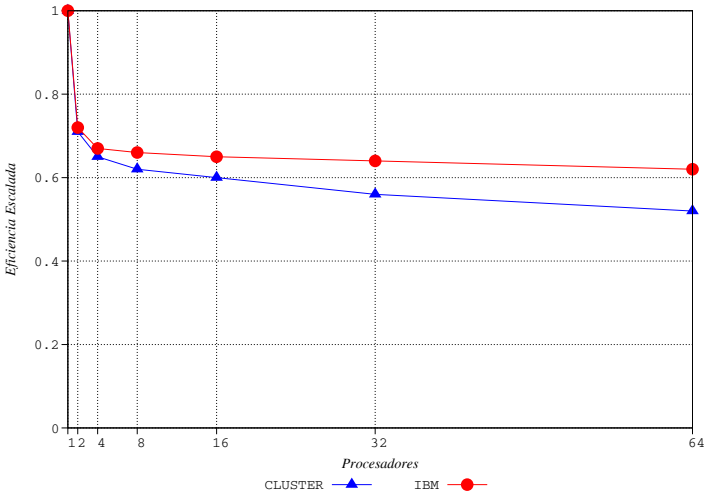
$$\begin{aligned} n^3 &\propto K n k t_s\,, \\ n^3 = W &\propto K^{\frac{3}{2}} k^{\frac{3}{2}} t_s^{\frac{3}{2}}\,. \end{aligned} \tag{4.11}$$

De manera similar, para determinar el término de la isoeficiencia debido a t_w, n^3 tiene que ser proporcional a $(n^2/2)\,k\,t_w$. Por tanto,

$$\begin{aligned} n^3 &\propto K(n^2/2)\,k\,t_w\,, \\ n^3 = W &\propto K^3 (1/2)^3 k^3 t_w{}^3\,. \end{aligned} \tag{4.12}$$

De acuerdo a las ecuaciones (4.11) y (4.12), la función de isoeficiencia asintótica debida a la sobrecarga de comunicación es $\Theta(k^3)$, mejor por tanto que la isoeficiencia de la distribución *BF* de la eliminación gaussiana que es $\Theta(k^3 \log^3 k)$ (ver [Grama *et al* 03]). Además, ésta no es de coste óptimo ya que su coste ($\Theta(n^3 \log n)$ debido al término asociado a t_w) es asintóticamente mayor que el tiempo de ejecución secuencial. El rendimiento de la eliminación gaussiana puede ser mejorado sustancialmente si el número de procesadores trabaja de manera asíncrona; esto es, que ningún procesador espere por los demás para finalizar una iteración antes de comenzar la siguiente. A diferencia del algoritmo síncrono, la versión asíncrona de la eliminación gaussiana es de coste óptimo (ver [Grama *et al* 03]). Por su parte, la eliminación de Neville con una distribución de los datos de acuerdo al esquema *BF* sí es de coste óptimo (ver [Alonso *et al* 03]).

Completaremos el análisis de la escalabilidad mostrando la eficiencia escalada. En este caso, podemos observar como al ir multiplicando por 2 el número de procesadores, y de igual forma *W*, la eficiencia en el *IBM* es casi constante a partir de 4 procesadores (de 0.67 a 0.62), no así en el *CLÚSTER* (de 0.65 a 0.51). La diferencia con la eficiencia escalada mostrada en la gráfica 4.1, correspondiente a *BFC,* es que en *BF* la degradación de la eficiencia se realiza bruscamente en el paso de 1 a 2 procesadores, siendo posteriormente una pérdida de la eficiencia más suave, mientras que en *BFC* la eficiencia va disminuyendo de una forma uniforme. A través de las secciones B.1.1 y B.1.2 del anexo B podemos observar como la eficiencia de *BFC*, en el caso de *IBM*, finaliza en 0.68 mientras que en *BF* desemboca en 0.62. En lo que respeta al *CLÚSTER* en ambos casos termina en 0.51.

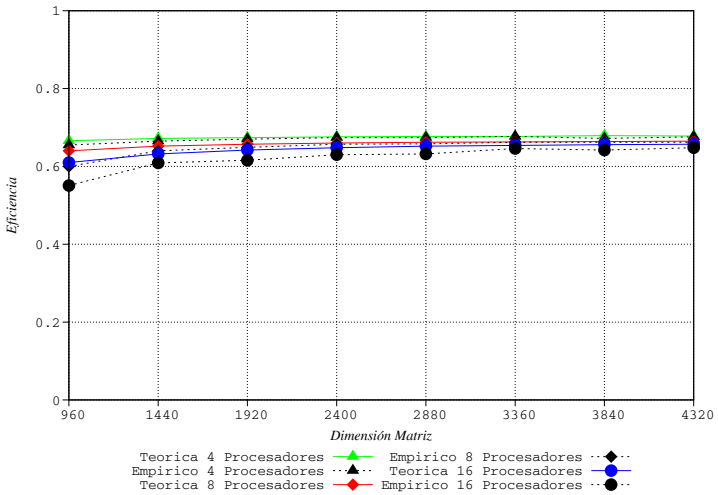


Gráfica 4.4 *BF*: Eficiencia escalada

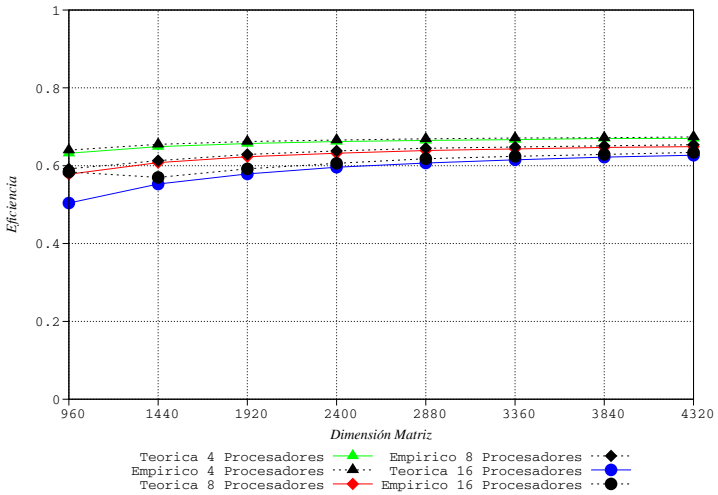
Experimentos

Esta distribución se caracteriza por un modelo de comunicaciones sencillo, al igual que en *BFC*, y por un mal balanceado de la carga. En el caso de las comunicaciones, el número de éstas en cada proceso (procesador) por etapa del algoritmo es muy reducido, siendo el peor caso dos operaciones de envío/recepción: a) el primer procesador únicamente envía a su sucesor una fila o fracción; b) el último simplemente recibe una fila o fracción; c) los intermedios reciben del predecesor y envían al sucesor una fila o fracción. En cuanto a la carga de computación, ésta es diferente en cada procesador. Así, el primer procesador, el que posee el primer bloque del sistema, es el que menos tareas realiza, y el último, el que más.

La conjunción de las dos características anteriores puede tener un efecto beneficioso a la hora de ejecutar los algoritmos. Puesto que el sentido (dependencia) de las comunicaciones es del primero al último y la carga de computación aumenta a medida que el procesador ocupa una posición más alta, cuando un procesador invoca una operación de recepción, el emisor ya ha realizado la operación de envío (*anticipación*); lo que se traduce en una reducción del impacto de las comunicaciones en el tiempo total. Es más, se podría decir que el algoritmo es, en cierto modo, *asíncrono entre etapas*, aunque en la implementación no se ha permitido dicho solapamiento entre etapas. Los procesadores inferiores avanzarán más rápido (menos carga) que los superiores, no habiendo justificación para ralentizar (sincronizar) dicho avance. Nótese que en el modelo teórico se obtiene el tiempo centrándose en el procesador que más carga tiene.



Gráfica 4.5 *BF*: Eficiencia teórica vs. empírica para *IBM SP2*



Gráfica 4.6 *BF*: Eficiencia teórica vs. empírica para *CLÚSTER*

Al igual que sucedía en *BFC*, las expectativas teóricas para el *IBM* no son superadas por los valores empíricos, a diferencia a lo que ocurre en el *CLÚSTER*. Esas diferencias están entre 0.002 y 0.009 para matrices de 4320x4320 y k = 4, 8 y 16, en el caso del *IBM* y entre 0.004 y 0.007, en el caso del *CLÚSTER*. Esto viene a suponer a lo sumo una diferencia en términos porcentuales de 1.37% en el *IBM* y un 1.1% en el *CLÚSTER*.

Se advierte que en ambos entornos el ajuste es mejor cuantos menos procesadores se utilizan y mayor sea el tamaño del problema. Si bien es cierto que en el caso de 16 procesadores existe una diferencia mayor entre estimación y resultados obtenidos en los experimentos para tamaños pequeños, diferencia esperable por los posibles efectos colaterales ya mencionados en tamaños de problema tan pequeños, ésta se va atenuando a medida que el tamaño del problema aumenta. Por ejemplo, en el *IBM* hay una diferencia de 9.67% para matrices de 960x960 y pasa a ser del 1.37% para matrices de 4320x4320, mientras que en el *CLÚSTER* se pasa de un 13.7% a un 1.1%.

Dado que se ha confirmado que en esta distribución existe un grado de similitud entre la eficiencia estimada y la eficiencia empírica de un 98%-99% en el caso de matrices de tamaño grande, para un valor elevado de n como por ejemplo 2^{15} podemos presentar la tabla 4.12. Estos valores que en ella aparecen han sido calculados con las expresiones del tiempo secuencial (2.25), del tiempo de cálculo (4.7) y del tiempo de comunicaciones (4.8) de *BF*.

k	*IBM SP2*	*CLÚSTER*
4	0.681	0.679
8	0.670	0.667
16	0.666	0.662
32	0.664	0.656
64	0.662	0.645
128	0.657	0.625

Tabla 4.12 *BF*: Estimación de la eficiencia para $n=2^{15}$

Los datos mostrados en la tabla nos indican que para un tamaño suficientemente grande del problema la eficiencia está alrededor de 2/3, siendo ligeramente menor en el *CLÚSTER*.

4.1.3 Cíclico por Filas

Esta distribución de los datos corresponde al caso particular de *BFC* en el que el tamaño de bloque *(m)* toma el valor 1. En este caso, son las filas de la matriz *A* las que se distribuyen cíclicamente entre los *k* procesadores, de tal modo que cada procesador tendrá *n/k* bloques de una única fila. La siguiente figura ilustra esta partición de los datos.

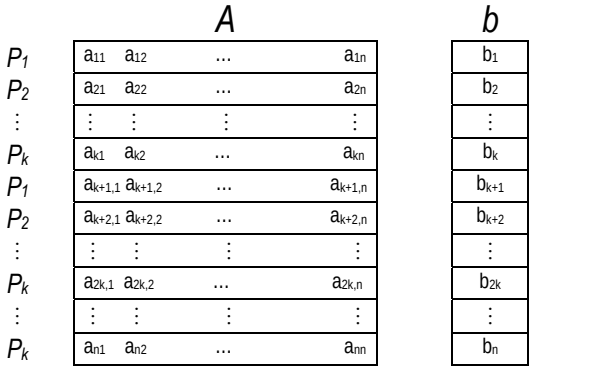


Figura 4.4 Cíclico por Filas (*CF*)

Las expresiones del tiempo de cálculo y comunicación son las que aparecen a continuación, obtenidas al sustituir *m* por 1 en las ecuaciones del tiempo de cálculo (4.1) y comunicación (4.2) de *BFC*:

$$T_{CÁLCULO\_CF} = \left(\frac{2}{3}\frac{n^3}{k} + \frac{1}{2}n^2 - \frac{n^2}{k} - \frac{1}{6}nk + n - \frac{n}{k}\right)t_c \ , \tag{4.13}$$

y

$$T_{COMUNICACIÓN\_CF} = (n-1)t_s + \left(\frac{1}{3}\frac{n^3}{k} + \frac{1}{4}n^2 - \frac{1}{4}\frac{n^2}{k} - \frac{1}{12}nk + \frac{3}{4}n - \frac{n}{k}\right)t_w \ . \tag{4.14}$$

De manera aproximada, el tiempo de ejecución paralelo total es

$$T_{PARALELO\_CF} \approx \frac{2}{3}\frac{n^3}{k}t_c + nt_s + \frac{1}{3}\frac{n^3}{k}t_w \ . \tag{4.15}$$

Como podemos observar, la mejora conseguida en el balanceado de la carga con respecto a *BF* tiene un efecto perjudicial en las comunicaciones, siendo éstas de orden n^3, debido a que en cada etapa del proceso de eliminación los procesadores deben comunicar todas sus filas al procesador vecino.

La expresión de la eficiencia teórica para tallas del problema suficientemente grandes depende exclusivamente de la constantes t_c y t_w como podemos ver a continuación:

$$E_{CF} = \frac{2t_c}{2t_c + t_w} \ . \tag{4.16}$$

Los valores que proporciona (4.16) para el *IBM* y para el *CLÚSTER* son 0.227 y 0.057 respectivamente.

A pesar de lo comentado con anterioridad y que indica claramente un pobre rendimiento de esta formulación paralela, hemos calculado las estimaciones del tiempo de ejecución y eficiencia para el *IBM*, las cuales mostramos en la siguiente tabla. Observar cómo la eficiencia se mantiene constante e independiente por tanto del número de procesadores y de la talla del problema.

k	n	$T_{TEÓRICO}$	$E_{TEÓRICA}$
4	960	11.261	0.209
	1440	37.922	0.210
	1920	89.810	0.210
	2400	175.331	0.210
	2880	302.889	0.210
	3360	480.889	0.210
	3840	717.736	0.210
	4320	1022.000	0.210
8	960	5.663	0.208
	1440	19.023	0.209
	1920	45.005	0.210
	2400	87.812	0.210
	2880	151.647	0.210
	3360	240.711	0.210
	3840	359.207	0.210
	4320	511.339	0.210
16	960	2.864	0.206
	1440	9.573	0.208
	1920	22.602	0.209
	2400	44.052	0.209
	2880	76.025	0.209
	3360	120.622	0.210
	3840	179.943	0.210
	4320	256.091	0.210

Tabla 4.13 *CF*: Estimaciones del tiempo de ejecución y de la eficiencia para *IBM SP2*

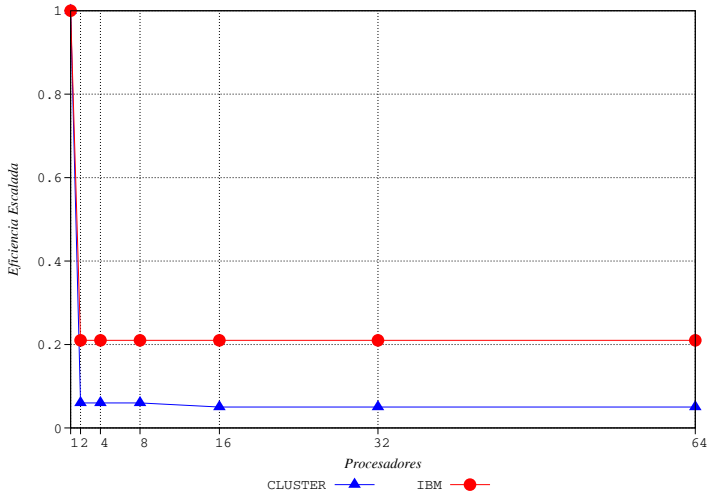
Como hemos mencionado anteriormente, la ineficiencia de *BF* relativa al balanceado de la carga, se ve corregida en esta distribución. En este caso, la diferencia existente entre la carga computacional del procesador más cargado y del menos cargado en cualquier iteración es de a lo sumo una fila, esto es $O(n)$ operaciones aritméticas. Puesto que hay n iteraciones, la sobrecarga acumulada debido a la inactividad de los procesadores es sólo de $O(n^2)$ con una distribución cíclica, comparada con $O(n^3)$ de la distribución por bloques. Precisamente este buen balanceado de la carga en *CF* es el que conlleva como contrapartida tal volumen de comunicaciones, haciendo que éstas sean $\Theta(n^3)$.

Para llevar a cabo el estudio de la función de isoeficiencia de *CF* partimos de la función de sobrecarga relativa a la comunicación de este algoritmo la cual es aproximadamente

$$T_{O\_CF} \approx n\,k\,ts + n^3\,tw.$$

El término de la isoeficiencia debido a t_s es igual al obtenido en *BF* y recogido en (4.11). Puesto que W es n^3, el término $n^3\,t_w$ siempre está balanceado con W. Este término es independiente de k y no contribuye a la función de isoeficiencia. Si calculamos el término de la isoeficiencia relativo al término n^2 de t_w, obtendremos que corresponde con la expresión recogida en (4.12). Teniendo en cuenta el análisis anterior, podemos concluir que la función de isoeficiencia para *CF* es $\Theta(k^3)$.

Concluiremos el presente análisis con la siguiente gráfica donde se muestra la eficiencia escalada correspondiente a esta distribución, pudiendo verse que a partir de un cierto valor de n y para k=2 las eficiencias son 0.21 y 0.06 en el *IBM* y en el *CLÚSTER* respectivamente y que estas eficiencias se mantienen constantes cuando k y W se van multiplicando por el mismo factor (para más detalle ver Anexo B).



Gráfica 4.7 *CF*: Eficiencia escalada

Experimentos

A pesar de las malas expectativas, en esta ocasión también se han realizado experimentaciones y los resultados obtenidos confirman el mal comportamiento pronosticado por los análisis teóricos.

4.1.4 Conclusiones de las Distribuciones Orientadas a Filas

A lo largo de esta sección hemos podido comprobar que las particiones unidimensionales orientadas a filas presentan un modelo de comunicaciones sencillo, el cual consiste en el intercambio de información de cada procesador activo con, a lo sumo, dos procesadores vecinos. La cuantía de las comunicaciones es función del tamaño de los bloques de filas en los que se divide la matriz, siendo de menor magnitud cuanto mayor sea el tamaño de bloque. También hemos podido observar cómo el balanceado de la carga entre los procesadores es mejor cuanto menor es el tamaño del bloque de filas fijado. Estas dos situaciones determinan que es especialmente necesario buscar aquel tamaño de bloque que establezca un compromiso entre carga de trabajo y cantidad de comunicaciones, es decir, aquel tamaño de bloque que permita obtener el mejor equilibrio de trabajo entre los procesadores sin perjudicar el tiempo dedicado a las comunicaciones. Por ello, hemos calculado el tamaño de bloque para cada orden de la matriz (n) y cada número de procesadores (k) que nos proporciona el menor tiempo de ejecución y por consiguiente, la mayor eficiencia.

Además hemos analizado dos situaciones particulares que corresponden; por un lado, a la distribución con el mayor tamaño de bloque posible, lo que supone un mal balanceado de la carga de trabajo entre los procesadores y un coste de comunicaciones mínimo; y por otro lado, la partición con el menor tamaño de bloque (una única fila) lo que implica un buen reparto de la carga computacional y una cuantía de comunicaciones máxima.

Los estudios analíticos realizados nos han permitido comprobar que, el algoritmo orientado a filas que se deriva de aplicar el tamaño de bloque óptimo, no sólo tiene la mayor eficiencia, sino que es además, el más escalable.

Los resultados experimentales obtenidos en dos entornos de experimentación muestran la validez de las estimaciones teóricas y permiten conocer de manera muy fiable el comportamiento de estas tres distribuciones en otras máquinas que sigan el modelo de paso de mensajes.

4.2 Particiones Unidimensionales Orientadas a Columnas

En esta sección vamos a estudiar el comportamiento de las *Particiones Unidimensionales orientadas a Columnas*, de manera similar a como se analizaron las *Particiones Unidimensionales orientadas a Filas* a lo largo de la sección 4.1. Recordemos que las particiones orientadas a columnas se basan en dividir la matriz en grupos de m columnas completas y consecutivas, con $1 \leq m \leq n/k$ donde n es el orden de la matriz y k es el número de procesadores, para posteriormente distribuirlos entre los procesadores de manera cíclica. En primer lugar abordaremos el caso genérico, denominado *Bloques de Columnas Cíclicos* obteniendo el tamaño de bloque óptimo que nos proporciona un menor tiempo de ejecución. A continuación trataremos dos situaciones concretas de esta distribución general: *Bloques de Columnas*, cuando m toma el valor n/k, y *Cíclico por Columnas*, cuando m toma el valor 1.

4.2.1 Bloques de Columnas Cíclicos

Tal y como hemos comentado anteriormente la distribución *Bloques de Columnas Cíclicos* (*BCC*) consiste en dividir las n columnas de una matriz $A=(a_{ij})_{1\leq i,j\leq n}$ en bloques de m columnas completas y consecutivas. Estos bloques se asignarán de manera cíclica entre k procesadores, al igual que sucedía en la partición *BFC*. Una vez hecho el reparto de bloques, cada procesador tendrá asignados h bloques de tamaño m, donde $m = n/(kh)$, con $h \in [1,n/k]$. El vector $b=(b_i)_{1\leq i\leq n}$ de términos independientes se almacenará en el último procesador, P_k.

En la figura 4.5 se muestra como los bloques de columnas de la matriz A se reparten entre los distintos procesadores. De esta forma, el primer bloque, formado por las columnas de la 1 a la m, se asigna al primer procesador (P_1), el siguiente bloque compuesto por las columnas de la m+1 a la 2m se ubica en el segundo procesador (P_2) y así sucesivamente, hasta que las columnas $(k$-1$)m$+1, $(k$-1$)m$+2,

..., *km*, que integran la sección de columnas *k*, se sitúan en el *k*-ésimo procesador (P_k). Este proceso se repite hasta distribuir todos los bloques de *m* columnas de la matriz *A* entre los *k* procesadores, conteniendo cada uno de ellos un total de *h* bloques de columnas de tamaño *m* o lo que es lo mismo *n/k* columnas. El procesador P_k contendrá *(n/k)*+1 columnas, ya que almacenará además el vector *b*.

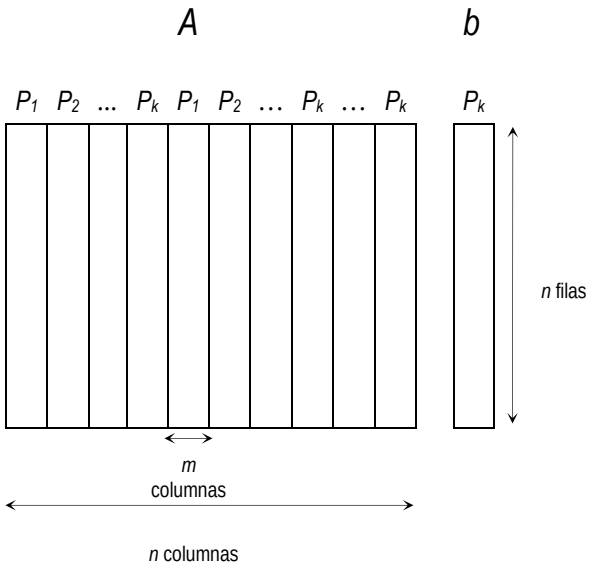
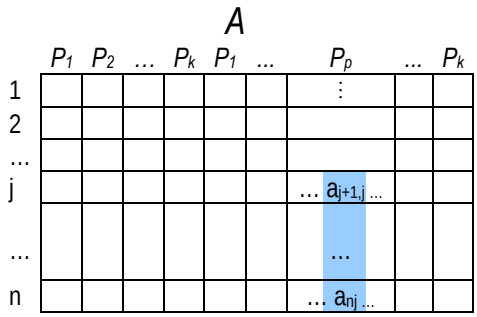


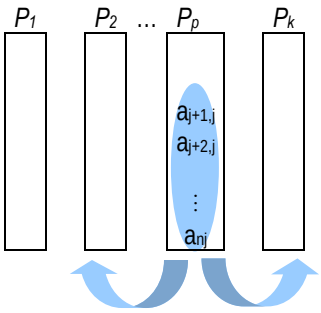
Figura 4.5 Bloques de Columnas Cíclicos (*BCC*)

La formulación paralela de la eliminación de Neville, en la que la distribución de la matriz de datos *A* se realiza de acuerdo a lo explicado anteriormente, presenta unas ideas básicas que se reseñan a continuación:

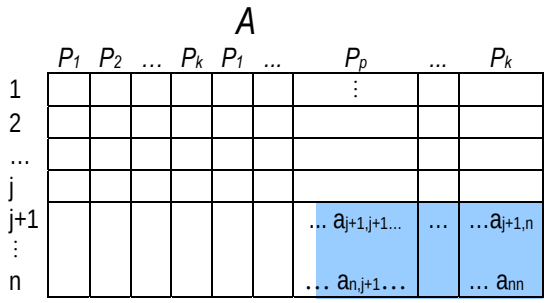
1. El procesador que contenga la columna *j*, siendo *j* el número de iteración en el que nos encontremos, deberá calcular los multiplicadores correspondientes a dicha iteración, es decir, aquellos cuyo índice de fila sea superior a *j* (figura 4.6a).
2. El procesador mencionado en el punto anterior deberá transferir dichos multiplicadores al resto de los procesadores activos (figura 4.6b).
3. En cada procesador se efectuarán las actualizaciones de todos aquellos elementos que correspondan a filas y columnas con un índice superior a *j* (iteración actual) (figura 4.6c).



(a)
Calcular multiplicadores
$a_{ij} = a_{ij} / a_{i-1,j}$ *para* $n \geq i \geq j+1$.



(b)
Enviar multiplicadores al resto de los procesadores activos.



(c)
Actualizar la parte activa de la matriz
$a_{ts} = a_{ts} - (a_{tj} / a_{t-1,j})\, a_{t-1,s}$ para $j+1 \leq s \leq n$ y $n \geq t \geq j+1$.

Figura 4.6 Pasos de la distribución *BCC* durante la iteración *j*

Seguidamente se muestra el algoritmo de la distribución *BCC* de la eliminación de Neville donde figuran todas las operaciones que deben realizar los k procesadores que contienen las n columnas de la matriz A, así como las operaciones que debe efectuar el procesador P_k en el vector de términos independientes. Precisamente este hecho es el que provoca que se diferencien las actualizaciones realizadas por el procesador k de las efectuadas por el resto de los procesadores.

```
1. para j := 1 hasta n – 1 hacer              /* bucle externo */
2.     z := (( j – 1) div m) mod k + 1;       /* P_z contiene la columna j */
3.     para P_z hacer
4.         para i := n descendiendo hasta j + 1 hacer
5.             A[i, j] := A[i, j] / A [i – 1, j];          /* cálculo de multiplicadores */
6.             enviar A[i, j] de P_z al resto de los procesadores;
7.         fin para       /* línea 4 */
8.     en paralelo hacer
9.         para todo P_l , con 1 ≤ l ≤ k, hacer
10.            para r := 0 hasta h – 1 hacer
11.                para q := (l – 1) * m + 1 hasta l * m hacer
12.                    s := q + r * k * m;
13.                    si s > j entonces
14.                        para t := n descendiendo hasta j + 1
15.                            A[t, s] := A[t, s] – A[t, j] * A[t – 1, s];      /* paso de eliminación */
16.                fin para     /* línea 11 */
17.    fin en       /* línea 8 */
18.    para P_k hacer
19.        para t := n descendiendo hasta j+1
20.            b[t] := b[t] – A[t, j] * b[t – 1];
21. fin para       /* línea 1 */
```

Algoritmo 4.2 Eliminación de Neville mediante *BCC*

Al igual que en *BFC*, iniciaremos el estudio de la distribución orientada a columnas estimando el tiempo de ejecución del Algoritmo 4.2 y su eficiencia, así como el tamaño de bloque óptimo. De igual forma que en las distribuciones orientadas a filas, nos centraremos en las operaciones efectuadas sobre los elementos de la matriz de coeficientes.

Tiempo de Cálculo

Para estimar el tiempo de cálculo del Algoritmo 4.2 debemos contabilizar los dos pasos de computación que se realizan en el algoritmo (figuras 4.6a y 4.6c). El primero de ellos consiste en el

cálculo de los multiplicadores asociados a cada etapa, mientras que el segundo corresponde a la actualización de la parte activa de la matriz.

Para proceder al cálculo de los multiplicadores en una etapa *j* del algoritmo (línea 5 del Algoritmo 4.2) se tienen que efectuar *n - j* cocientes. Por ello, el número total de cocientes realizados en el proceso de eliminación de Neville es

$$\sum_{j=1}^{n-1}(n-j) = \frac{1}{2}n^2 - \frac{1}{2}n \ .$$

Para determinar el número de operaciones, restas y productos, que integran el segundo paso de computación (línea 15 del Algoritmo 4.2) deberemos conocer qué columnas corresponden a cada procesador para así determinar las actualizaciones que se llevan a cabo. En la tabla 4.14 se muestran los índices de las columnas que constituyen los *h* bloques de los que disponen los procesadores del 1 al *k*. Como se puede comprobar cada uno de los *h* bloques está compuesto por *m* columnas.

PROCESADORES	BLOQUES	COLUMNAS
Procesador 1	Bloque 1	1, 2, ..., *m*
	Bloque 2	*km*+1, *km*+2, ..., (*k*+1)*m*
	...	...
	Bloque *h*	(*h*-1)*k m*+1, (*h*-1)*k m*+2, ..., ((*h*-1)*k*+1)*m*
Procesador 2	Bloque 1	*m*+1, *m*+2, ..., 2*m*
	Bloque 2	(*k*+1)*m*+1, (*k*+1)*m*+2, ..., (*k*+2)*m*
	...	...
	Bloque *h*	((*h*-1)*k*+1)*m*+1, ((*h*-1)*k*+1)*m*+2, ..., ((*h*-1)*k*+2)*m*
...		...
Procesador k	Bloque 1	(*k*-1)*m*+1, (*k*-1)*m*+2, ..., *km*
	Bloque 2	(2*k*-1)*m*+1, (2*k*-1)*m*+2, ..., 2*km*
	...	...
	Bloque *h*	(*hk*-1)*m*+1, (*hk*-1)*m*+2, ..., *n*

Tabla 4.14 *BCC*: Distribución de columnas entre los *k* procesadores

Con el fin de determinar el número de operaciones necesarias para llevar a cabo el paso de actualización podemos observar lo siguiente:

En la primera iteración se elimina la variable x_1 de las ecuaciones *n, n*-1*, …,* 2; de esta forma, todos los procesadores actualizan todas sus columnas, esto es, *hm* columnas, excepto el primer procesador que actualiza *hm-1* columnas, ya que en la primera columna no se realizan cálculos. En la segunda iteración, en la que se elimina la variable x_2, se actualizan las *filas n, n*-1*, …,* 3; así, todos los procesadores modifican sus *hm* columnas, salvo el primer procesador que actualiza *hm*-2 columnas.

Esta misma situación se repite hasta la iteración m, es decir, a lo largo de estas iteraciones todos los procesadores actualizan sus hm columnas, sin embargo, el primer procesador va disminuyendo el número de columnas a actualizar, esto es, una menos tras cada iteración.

En las siguientes m iteraciones, el procesador que disminuye el número de columnas a actualizar es el procesador P_2 que pasa de actualizar hm columnas a hacerlo en hm-m columnas. El procesador P_1 sigue actualizando hm-m columnas mientras que el resto de los procesadores actualizan hm columnas.

Agrupando las iteraciones en bloques de tamaño m, el proceso anteriormente descrito se repite hasta que en el k-ésimo bloque de m iteraciones, el procesador que ve disminuido el número de columnas a actualizar es el procesador P_k, que pasa de realizar cálculos sobre hm columnas a actualizar hm-m columnas, mientras que el resto sigue actualizando hm-m columnas.

Considerando lo anterior, podemos concluir que el procesador P_k es el procesador que más cálculos realiza en el proceso de actualización. El número de columnas que actualiza a lo largo de las n-1 iteraciones, son las que se muestran en la siguiente tabla:

ITERACIONES	NÚMERO DE COLUMNAS A ACTUALIZAR
1,2,..., (k-1)m	hm, hm, ..., hm
(k-1)m+1, (k-1)m+2, ..., km	hm-1, hm-2, ..., hm-m
km+1, km+2, ..., (2k-1)m	hm-m, hm-m, ..., hm-m
(2k-1)m+1, (2k-1)m+2, ..., 2km	hm-(m+1), hm-(m+2), hm-2m
2km+1, 2km+2, ..., (3k-1)m	hm-2m, hm-2m, ..., hm-2m
(3k-1)m+1, (3k-1)m+2, ..., 3km	hm-(2m+1), hm-(2m+2), ..., hm-3m
3km+1, 3km+2, ..., (4k-1)m	hm-3m, hm-3m, ..., hm-3m
...	...
(h-1)km+1, (h-1)km+2, ..., (hk-1)m	m, m, ..., m
(hk-1)m+1, (hk-1)m+2, ..., hkm-1	m-1, m-2, ..., 1

Tabla 4.15 *BCC*: Número de columnas actualizadas por el procesador P_k

En la tabla 4.15, aparecen las iteraciones agrupadas en bloques, de forma que en ellos se reproducen dos situaciones claramente diferenciadas: en las iteraciones de la 1 a la $(k$-1$)m$ se actualizarán siempre hm columnas de n-j elementos cada una, siendo j la iteración en curso y en el siguiente bloque de m iteraciones, desde la iteración $(k$-1$)m$+1 hasta la km, va disminuyendo unidad a unidad el número de columnas a actualizar.

Debido a que los cálculos se efectúan simultáneamente en los distintos procesadores y como el procesador P_k es el que mayor número de cálculos realiza en las distintas iteraciones, contabilizaremos el número de restas y productos que realiza dicho procesador a lo largo del algoritmo.

Las restas que se realizan, siguiendo el proceso anteriormente descrito, se recogen en la

siguiente expresión:

$$\sum_{i=0}^{h-1}\sum_{j=ikm+1}^{((i+1)k-1)m}((hm-im)(n-j))+\sum_{i=1}^{h}\sum_{j=(ik-1)m+1}^{ikm}((hm-(j-i(k-1)m))(n-j)).$$

La primera parte de la expresión determina las restas que se realizan en los bloques de *(k*-1*)m* iteraciones, en los cuales P_k siempre actualiza el mismo número de columnas, número que varía dependiendo del bloque de iteraciones en el que nos encontremos. La segunda parte establece las restas que tienen lugar en los bloques de *m* iteraciones, en los cuales el número de columnas a actualizar por parte de P_k va disminuyendo unidad en unidad conforme avanzan las iteraciones.

Del resultado de los dos sumatorios podemos concluir que el número total de restas es:

$$\left(3km^2h-6nmh-3k^2m^2h^2-k^2m^2h+2h-2m^2h+6nh^2mk+3h^2m^2k-2h^3m^2k^2+6nkmh-6nh\right)\frac{m}{12},$$

realizándose el mismo número de productos.

De todo lo anterior se tiene que el tiempo de cálculo es

$$T_{CÁLCULO\_BCC}=\left(nkm^2h-nhm-\frac{1}{2}k^2m^3h^2+\frac{1}{3}hm-nm^2h-\frac{1}{6}k^2m^3h-\frac{1}{3}k^2m^3h^3+\frac{1}{2}km^3h-\frac{1}{3}m^3h+\right.$$
$$\left.+nh^2m^2k+\frac{1}{2}h^2m^3k+\frac{1}{2}n^2-\frac{1}{2}n\right)t_c.$$

Sustituyendo en la expresión anterior *h* por su valor, es decir, *n/(km)*, se tiene el tiempo de cálculo como función de *n*, *k*, *m* y t_c, esto es,

$$T_{CÁLCULO\_BCC}=\left(\frac{2}{3}\frac{n^3}{k}+\frac{1}{2}n^2m-\frac{1}{2}\frac{n^2m}{k}+\frac{1}{2}n^2-\frac{n^2}{k}+\frac{1}{3}\frac{n}{k}+\frac{1}{2}m^2n-\frac{1}{3}\frac{nm^2}{k}-\frac{1}{6}nkm^2-\frac{1}{2}n\right)t_c. \quad (4.17)$$

Tiempo de Comunicación

A continuación vamos a determinar el coste de las comunicaciones que tienen lugar en el Algoritmo 4.2 (línea 6). Para ello necesitamos conocer, en primer lugar, el número de multiplicadores que se necesita transferir en cada iteración. Además, se precisará identificar el tipo de comunicación necesaria y el número de procesadores implicados en ella.

Vamos a hacer un recorrido por las distintas etapas del algoritmo, para así deducir el número de multiplicadores que se deben transmitir en cada iteración. En la primera iteración, se elimina la variable x_1 de las ecuaciones n, n-1, ..., 2, y para ello se necesita que P_1 envíe n-1 multiplicadores al resto de los procesadores. En la segunda iteración, en la que se elimina la variable x_2 de las ecuaciones n, n-1, ..., 3, el procesador P_1 transmitirá en este caso n-2 multiplicadores al resto de los procesadores. En la iteración m, en la que se elimina la variable x_m de las ecuaciones n, n-1, ..., m+1, el procesador P_1 envía n-m multiplicadores al resto de los procesadores. Por ello, podemos inferir que en una etapa j del algoritmo el procesador que contiene la variable x_j debe comunicar n-j multiplicadores al resto de los procesadores.

En la siguiente tabla se recoge el número de multiplicadores transmitidos por el procesador que debe realizar la comunicación en cada etapa.

ITERACIÓN	NÚMERO DE MULTIPLICADORES A COMUNICAR
1	n-1
2	n-2
...	...
((h-1)k+1) m	n-((h-1)k+1) m
((h-1)k+1) m+1	n-((h-1) k+1) m+1)
((h-1)k+1) m+2	n-((h-1) k+1) m+2)
...	...
(hk-1) m=n-m	n-(hk-1) m

Tabla 4.16 *BCC*: Número de multiplicadores a comunicar en cada etapa

El tipo de operación de comunicación que se debe realizar en esta distribución de los datos es un envío de uno a todos, dado que en cada etapa, un procesador dispone de los multiplicadores necesarios y debe transmitirlos al resto de los procesadores para su conocimiento. En las iteraciones de la 1 a la $((h-1)k+1)m$ se comunican los multiplicadores desde el procesador en el que se ubican, que denotaremos como P_z, al resto de procesadores. A partir de la iteración $((h-1)k+1)m+1$ y hasta la etapa $(hk-1)m$ se comunican los multiplicadores desde el procesador en el que se ubican, P_z, a los procesadores P_{z+1}, P_{z+2}, …, P_k. Desde la iteración $(hk-1)m+1$, inclusive ésta, ya no es necesario comunicar ningún multiplicador.

Como hemos indicado anteriormente la operación de comunicación es una comunicación de un procesador al resto, de donde, y sin más que tener en cuenta el coste de esta comunicación recogido en (3.5), se tiene la siguiente expresión que recoge el coste de la comunicación llevada a cabo:

$$\sum_{j=1}^{((h-1)k+1)m} \left((t_s + (n-j)\,t_w)\,log\ k\right) + \sum_{i=1}^{k-2} \sum_{j=((h-1)k+i)m+1}^{((h-1)k+(i+1))m} \left((t_s + (n-j)\,t_w)\,log(k-i)\right).$$

Indicar que para la resolución de esta ecuación se ha realizado una acotación en el factor

log(k-i) pasando a considerarlo como *log k*, es decir, dependiente exclusivamente del número de procesadores. Así el tiempo de comunicación es aproximadamente

$$T_{COMUNICACIÓN\_BCC} \approx (n-m)\log k\, t_s + \frac{1}{2}\left(n^2 - m^2 + m - n\right)\log k\, t_w \,. \tag{4.18}$$

La expresión del tiempo de algoritmo secuencial (ver (2.25)), así como las expresiones de los tiempos de cálculo y de comunicación que acabamos de mostrar, determinan que la eficiencia de esta distribución cuando el orden de la matriz es suficientemente elevado es

$$E_{BCC} \approx 1 \,. \tag{4.19}$$

Este resultado evidencia que esta distribución es la que proporciona el máximo valor de la eficiencia, sin desmerecer a la partición *BFC* con tamaño de bloque óptimo ya que su eficiencia estaba entorno a 0.9.

A continuación, vamos a determinar qué valor del tamaño de bloque es el que nos proporciona ese resultado tan prometedor, para a partir de ahí realizar los experimentos que nos permitan validar este pronóstico.

Tamaño de Bloque Óptimo

Una vez analizada la distribución *BCC* y conocido su tiempo de cálculo y de comunicación es interesante determinar aquel tamaño de bloque que nos permita obtener un menor tiempo de ejecución. Para ello, procedemos igual que en *BFC* realizando un análisis para los dos entornos (*IBM* y *CLÚSTER),* para el mismo rango de valores de *n* (960-4320) y para el mismo número de procesadores (4, 8 y 16).

Los resultados obtenidos en esta distribución son que el tamaño de bloque óptimo en todos los casos es 1, es decir, un caso particular de *BCC*, al cual denominamos *Cíclico por Columnas*. Dado que esta distribución en concreto la vamos a tratar al final de la sección 4.2 (subsección 4.2.3), dejaremos su análisis como mejor distribución por columnas para ese momento.

4.2.2 Bloques de Columnas

Esta subsección detalla el rendimiento de la distribución *BCC*, cuando el tamaño de bloque toma el valor *n/k*, en este caso hablamos de la distribución de la matriz *A* mediante *Bloques de Columnas (BC)*. En esta partición las columnas de la matriz *A* se dividen en bloques de *n/k* columnas consecutivas, *m=n/k*, como se muestra en figura 4.7. De ese modo cada procesador dispone de un único bloque (*h*=1).

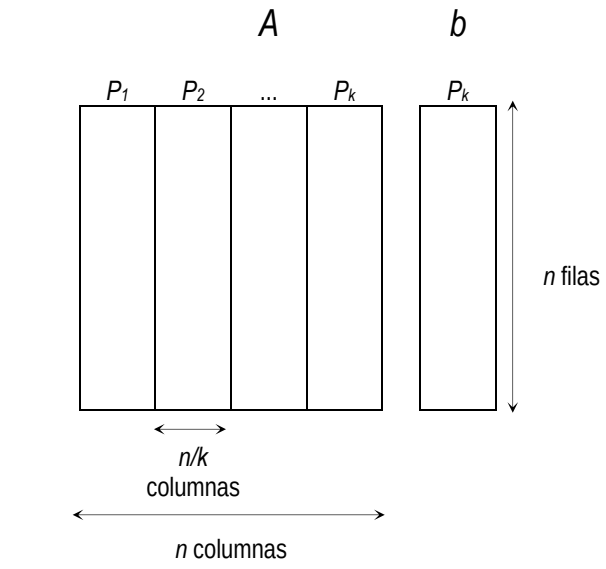


Figura 4.7 Bloques de Columnas (*BC*)

Para obtener el tiempo de cálculo de este caso particular de *BC* sustituiremos *m* por *n/k* en (4.17), siendo éste

$$T_{CÁLCULO\_BC} = \left(\frac{n^3}{k} - \frac{1}{3}\frac{n^3}{k^3} + \frac{1}{2}n^2 - \frac{n^2}{k} + \frac{1}{3}\frac{n}{k} - \frac{1}{2}n \right) t_c \,. \tag{4.20}$$

Si sustituimos de nuevo *m* por *n/k* en la expresión del tiempo de comunicaciones de *BCC* (ver (4.18)) resulta el siguiente coste de comunicación

$$T_{COMUNICACIÓN\_BC} \approx \left(n - \frac{n}{k} \right) \log k \; t_s + \frac{1}{2}\left(n^2 - \frac{n^2}{k^2} + \frac{n}{k} - n \right) \log k \; t_w \,. \tag{4.21}$$

Así, el tiempo de ejecución paralelo total de esta distribución es

$$T_{PARALELO\_BC} \approx \frac{n^3}{k} t_c + n \log k \; t_s + \frac{1}{2} n^2 \log k \; t_w \,. \tag{4.22}$$

La expresión de la eficiencia para valores elevados de *n* se recoge en (4.23), como podemos

observar, tal expresión depende exclusivamente del número de procesadores, de igual forma que en la misma distribución orientada a filas,

$$E_{BC} \approx \frac{2k^2}{3k^2 - 1}, \tag{4.23}$$

siendo una eficiencia aproximada de 2/3.

Análisis de la Escalabilidad

El producto tiempo-procesador para esta formulación paralela es $n^3\ t_c + n\ k\ log\ k\ t_s + (n^2/2)\ k\ log\ k\ t_w$, coste que coincide, para esa misma distribución, con la eliminación gaussiana (ver [Grama *et al* 03]).

Si en la expresión del producto tiempo-procesador del algoritmo se ignoran los costes de comunicación, la expresión pasa a ser n^3. Así, el coste del algoritmo paralelo es 3/2 el tiempo del algoritmo secuencial. Por consiguiente, tiene un límite superior de 2/3 en la eficiencia del algoritmo paralelo al igual que ocurría en *BF*. Esta ineficiencia de nuevo es debida al tiempo de inactividad de los procesadores, el cual viene motivado por el mal balanceado de la carga. Este problema se puede subsanar si las columnas de la matriz se distribuyen entre los procesadores de un modo cíclico. En la subsección 4.2.3 abordaremos este tipo de distribución, *Cíclica por Columnas*.

La función de sobrecarga relativa a la comunicación (ver (3.11)) para este algoritmo es

$$T_{O\_BC} \approx n\ k\ log\ k\ t_s + (n^2/2)\ k\ log\ k\ t_w.$$

En lo que respecta al término de la isoeficiencia debido a t_s tenemos que

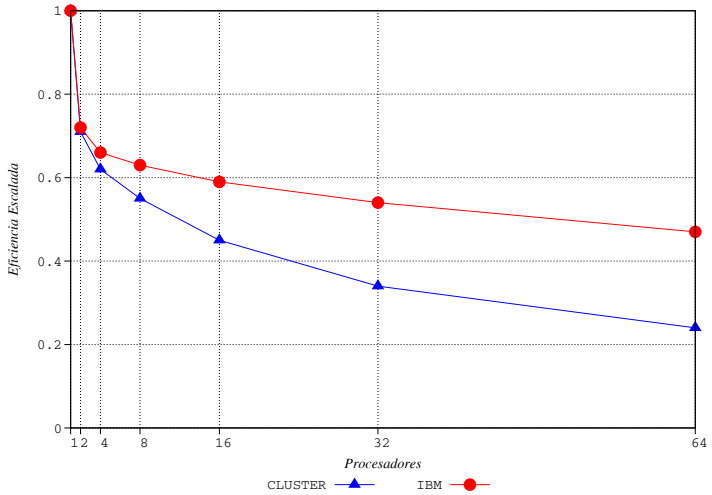
$$\begin{aligned} n^3 &\propto K\, n\, k\, log\ k\ t_s \\ n^3 = W &\propto K^{\frac{3}{2}} k^{\frac{3}{2}} log^{\frac{3}{2}} k\ \ t_s^{\frac{3}{2}}. \end{aligned} \tag{4.24}$$

De manera similar, para determinar el término de la isoeficiencia debido a t_w, n^3 tiene que ser proporcional a $(n^2/2)\ k\ log\ k\ t_w$. Por tanto,

$$\begin{aligned} n^3 &\propto K(n^2/2)\, k\, log\, k\, t_w\ , \\ n^3 = W &\propto K^2 (1/2)^3 k^3\, log^3\, k\ t_w{}^3\ . \end{aligned} \tag{4.25}$$

De (4.24) y (4.25) se obtiene que la función de isoeficiencia asintótica debida a la sobrecarga de comunicación es $\Theta(k^3\ log^3\ k)$.

A través de la siguiente gráfica se muestra la eficiencia escalada teórica de *BC*, donde se puede apreciar la evolución de la eficiencia a medida que *k* y *W* aumentan en la misma medida. Si la comparamos con la eficiencia escalada de *BF* (ver gráfica 4.4) se advierte que existe una caída más acentuada en el caso que nos ocupa, esto viene motivado por la utilización de operaciones colectivas de comunicación frente a la comunicación entre vecinos ya que el tiempo de cálculo es muy similar en ambas distribuciones. La eficiencia desciende hasta 0.47 y 0.24 en el *IBM* y en el *CLÚSTER* respectivamente, lo que nos permite observar a su vez que la diferencia entre los dos entornos de experimentación es más marcada que en *BF*, concretamente 0.23 en *BC* frente a 0.10 en *BF*.



Gráfica 4.8 *BC*: Eficiencia escalada

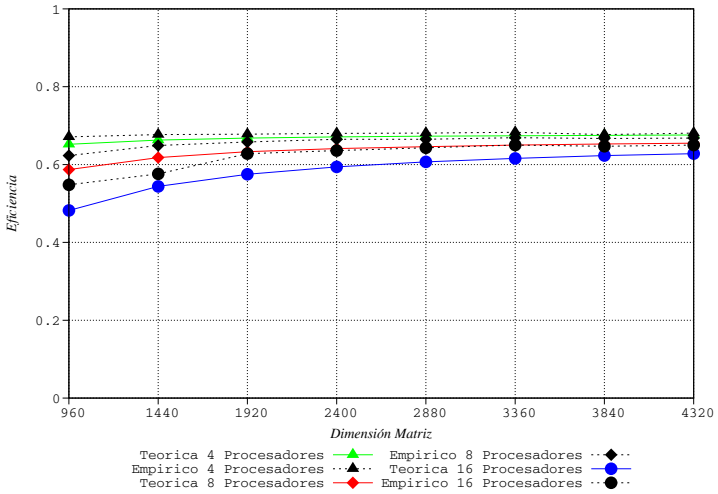
Experimentos

En la subsección 4.1.2 la distribución se realizaba por bloques de filas (*BF*), de tal manera que todos los elementos de una misma fila estaban contenidos en el mismo procesador. La bondad de esta forma de proceder está en el modelo de comunicaciones, siempre entre vecinos y de baja intensidad.

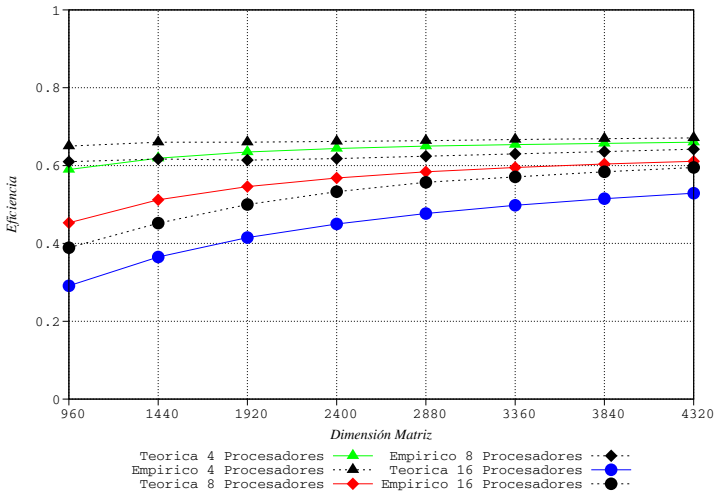
En la distribución que nos ocupa (*BC*), y en la siguiente (*CF*), los elementos de una columna están en el mismo procesador y para llevar a cabo cada etapa del algoritmo es necesario acceder a cierta información residente en otros procesadores mediante la propagación de dicha información. Concretamente es necesario difundir los multiplicadores computados en la etapa actual a todos aquellos procesadores que contengan columnas activas, es decir, columnas situadas a la derecha de la que está siendo tratada. Tenemos, por tanto, un cambio en el modelo de comunicaciones, que ahora viene caracterizado por operaciones colectivas de difusión. En la distribución *BC* supone una difusión en un

único sentido, desde el procesador activo (el que realiza el cálculo de multiplicadores) hacia los procesadores situados a su derecha, es decir, hacia los procesadores con un índice superior.

A continuación pasamos a comparar el modelo teórico con los resultados empíricos obtenidos en los entornos descritos en el capítulo 3 a través de las gráficas 4.9 y 4.10.



Gráfica 4.9 *BC*: Eficiencia teórica vs. empírica para *IBM SP2*



Gráfica 4.10 *BC*: Eficiencia teórica vs. empírica para *CLÚSTER*

A diferencia de lo que ocurría en *BFC* y en *BF*, se observa que en *BC* las expectativas teóricas son superadas por los valores empíricos tanto en el *IBM* como en el *CLÚSTER*. Esto es debido a que en la ecuación analítica del modelo teórico hay un factor dependiente del número de procesadores y de la iteración en curso, que ha sido acotado superiormente haciéndolo depender exclusivamente del número de procesadores. Así, a medida que el número de procesadores crece, la expresión del modelo teórico es más pesimista, lo que queda claramente reflejado en las gráficas 4.9 y 4.10. En el caso del *CLÚSTER* esta acotación tiene una mayor repercusión debido a que el valor de t_w es mayor que en el *IBM*.

A pesar del hecho anteriormente referido, el grado de coincidencia entre las estimaciones y los datos extraídos de los experimentos para n=4320 oscilan entre 96.6% y 99.3% en el *IBM* y entre 88.9% y 98.4% en el *CLÚSTER*, correspondiendo en ambos casos el menor porcentaje a 16 procesadores. Notar que en ambos entornos la eficiencia empírica tiende a 2/3 tal y como apuntaba (4.23).

En la siguiente tabla se muestra la eficiencia teórica, calculada con (2.25), (4.17) y (4.18), y sustituyendo en las dos últimas m por n/k, para un valor elevado de n como es 2^{15}, para los dos entornos de experimentación y para 4, 8, 16, 32, 64 y 128 procesadores.

k	*IBM SP2*	*CLÚSTER*
4	0.680	0.678
8	0.668	0.662
16	0.662	0.646
32	0.654	0.615
64	0.638	0.555
128	0.603	0.453

Tabla 4.17 *BC*: Estimación de la eficiencia para $n=2^{15}$

En la tabla anterior podemos comprobar como la eficiencia comienza con valores entorno a 0.67 en ambos casos, disminuyendo a medida que el número de procesadores aumenta, siendo este deterioro mayor en el *CLÚSTER* que en el *IBM*.

4.2.3 Cíclica por Columnas

Si en la distribución *BCC* consideramos que existen n/k bloques de columnas de tamaño 1 (figura 4.8) estaremos ante la distribución denominada *Cíclica por Columnas* (*CF*). Recordemos que en la subsección 4.2.1 quedó patente que la distribución por columnas que ofrece un menor tiempo de ejecución es la que corresponde a un tamaño de bloque igual a 1; esto es, *CC*, dejando para la presente subsección su análisis que realizaremos de manera similar al llevado a cabo en *BC*.

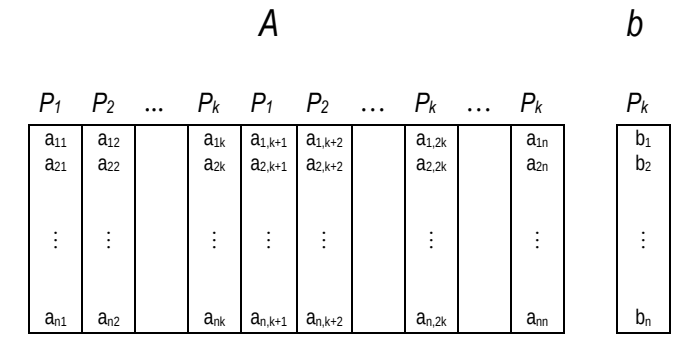


Figura 4.8 Cíclica por Columnas (*CC*)

En primer lugar, obtendremos la expresión del tiempo de cálculo de este tipo de distribución, reemplazando *m* por 1 en (4.17), resultando ser

$$T_{CÁLCULO\_CC} = \left(\frac{2}{3}\frac{n^3}{k} + n^2 - \frac{3}{2}\frac{n^2}{k} - \frac{1}{6}nk\right)t_c \,. \tag{4.26}$$

De igual forma, al sustituir *m* por 1 en (4.18), se obtiene el coste de comunicación en la distribución que nos ocupa, siendo éste

$$T_{COMUNICACIÓN\_CC} \approx (n-1)\log k\, t_s + \frac{1}{2}\left(n^2 - n\right)\log k\, t_w \,. \tag{4.27}$$

En conclusión, los términos de mayor grado correspondientes a t_c, t_s y t_w en el tiempo de ejecución total para este tipo de distribución nos proporcionan la siguiente aproximación

$$T_{PARALELO\_CC} \approx \frac{2}{3}\frac{n^3}{k} t_c + n\log k\, t_s + \frac{1}{2}n^2 \log k\, t_w \,. \tag{4.28}$$

Como ya apuntábamos, la distribución *BCC* con el tamaño de los bloques de columnas igual a 1 proporciona el máximo valor de la eficiencia, esto es,

$$E_{CC} \approx 1 \,. \tag{4.29}$$

Tal hecho se puede constatar al calcular la eficiencia para tamaños de problema suficientemente elevados utilizando (4.26), (4.27) y la expresión del tiempo del algoritmo secuencial

recogida en el capítulo 2 (ver (2.25)). Posteriormente, a través de los experimentos demostraremos que esto es efectivamente así.

Análisis de la Escalabilidad

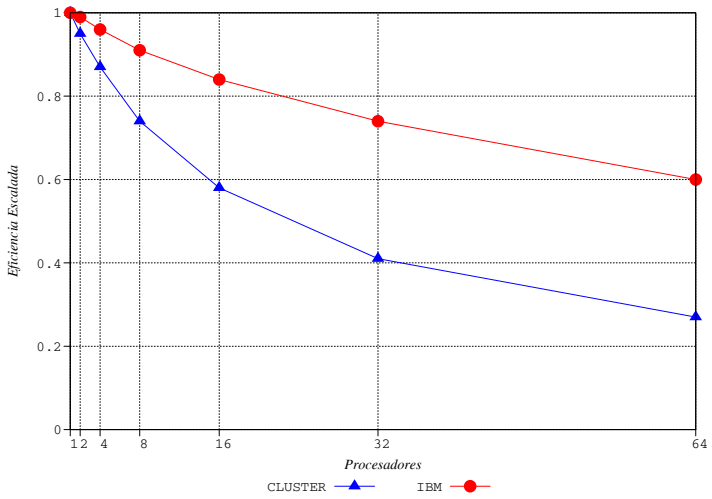
Con esta distribución, la diferencia de carga entre cada procesador es a lo sumo de una columna, esto es $\Theta(n)$ operaciones aritméticas, y dado que se realizan n iteraciones a lo largo del proceso, la sobrecarga acumulada debido al balanceado de la carga es sólo $\Theta(n^2)$ comparado con $\Theta(n^3)$ de *BC*.

La función de sobrecarga del sistema paralelo actual debida a la comunicación es

$$n\ k \log k\ t_s + (n^2/2)\ k \log k\ t_w,$$

al igual que sucedía en la distribución *BC*. Por ello el análisis de la escalabilidad realizado en la subsección 4.2.2 es válido en este apartado, lo que supone que la función de isoeficiencia total es

$$W \propto max\left\{ k^{\frac{3}{2}} \log^{\frac{3}{2}} k,\ k^3 \log^3 k \right\} = k^3 \log^3 k\ . \qquad (4.30)$$

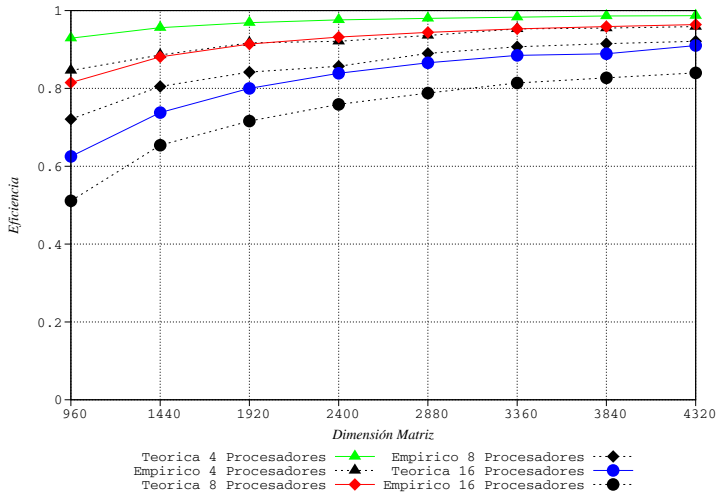


Gráfica 4.11 *CC*: Eficiencia escalada

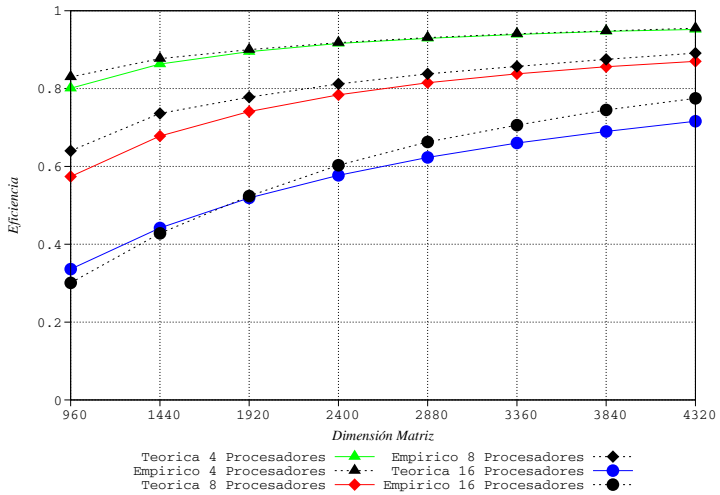
En la gráfica anterior aparece la eficiencia escalada correspondiente a esta partición de la matriz y donde podemos observar que la eficiencia en el *CLÚSTER* se degrada hasta 0.27, un valor similar al de *BC* (0.24), mientras que en el *IBM* la eficiencia termina siendo 0.60 en CC frente al 0.47 de BC. En cualquiera de los dos entornos se observa que el declive de la eficiencia es más uniforme en la distribución cíclica que en la de bloques (para más detalle ver Anexo B).

Experimentos

Al igual que ocurría en *BF*, la distribución por bloques de las columnas carece de un reparto de la carga bien balanceado, y de manera similar a como ocurría en filas, con la asignación *CC* se pretende subsanar ese hecho, llevando a cabo una distribución cíclica. Sin embargo, tal y como apunta la eficiencia para valores suficientemente grandes de *n*, los pronósticos en este caso son muy diferentes de los que mostramos en *CF*, ya que el equilibrio de la carga en este caso no supone un incremento de las comunicaciones. Veamos a través de las gráficas 4.12 y 4.13 si se confirman las expectativas teóricas.



Gráfica 4.12 *CC*: Eficiencia teórica vs. empírica para *IBM SP2*



Gráfica 4.13 *CC*: Eficiencia teórica vs. empírica para *CLÚSTER*

Teniendo en cuenta lo anterior, podemos observar un comportamiento similar al que se había detectado en las distribuciones *BFC* y *BF*, en el sentido de que las expectativas teóricas para el *IBM* no se ven superadas por las empíricas, no ocurriendo lo mismo en el *CLÚSTER*. En este caso la acotación a la que nos referíamos tras las gráficas 4.9 y 4.10, tiene una trascendencia insignificante dado que sólo en las últimas k - 2 iteraciones del proceso de eliminación estamos recurriendo a dicha cota superior, mientras que en *BC* se utilizaba para $n - 2(n/k)$ iteraciones.

En cuanto a la similitud entre las previsiones teóricas y la realidad sobre las máquinas, indicar que el grado de sintonía es muy elevado oscilando para un tamaño de matriz de 4320 entre 97% y 92% (*IBM*) y 99.6% y 92% (*CLÚSTER*), confirmándose por tanto la tendencia a una eficiencia de valor 1 para *CC*.

En la misma línea de las distribuciones anteriores y como consecuencia de que el modelo teórico se ve respaldado por el modelo empírico, mostramos a través de la siguiente tabla las eficiencias teóricas esperadas para un número de procesadores entre 4 y 128 procesadores y para un tamaño de problema 2^{15}.

k	*IBM SP2*	*CLÚSTER*
4	0.998	0.994
8	0.996	0.981
16	0.988	0.951
32	0.972	0.887
64	0.935	0.766
128	0.861	0.585

Tabla 4.18 *CC*: Estimación de la eficiencia para $n=2^{15}$

La eficiencia ronda 1.00 para 4 procesadores y disminuye hasta 0.861 y 0.585 para *IBM* y *CLÚSTER* respectivamente, lo cual viene motivado principalmente por la diferencia existente entre los valores de t_w de los dos entornos.

4.2.4 Conclusiones de las Distribuciones Orientadas a Columnas

El modelo de comunicación de las distribuciones orientadas a columnas consiste en la difusión de los multiplicadores desde el procesador que los calcula al resto de los procesadores activos. El volumen de datos transferidos no es función del tamaño de bloque de columnas fijado; sin embargo, el número de procesadores participes en la propagación sí que depende del tamaño de bloque, habiendo más procesadores implicados cuanto menor sea el tamaño del problema; en cualquier caso, esa dependencia es pequeña.

Por otro lado, la distribución de la carga de trabajo es más equilibrada en la distribución cíclica (menor tamaño de bloque) que en la de bloques (mayor tamaño de bloque).

Debido a todo ello, hemos procedido a calcular el tamaño de bloque que proporciona el menor tiempo de cálculo sin verse perjudicado por un incremento excesivo de las comunicaciones. Dichos cómputos nos llevan a que ese tamaño corresponde con el menor posible, esto es, una única columna.

Las estimaciones del tiempo de ejecución sitúan a la distribución *Cíclica por Columnas* como aquella que minimiza el tiempo de ejecución total y por tanto, maximiza la eficiencia, siendo ésta cercana al valor 1. Además, hemos procedido a analizar la situación opuesta, es decir, aquella distribución por columnas cuyo tamaño de bloque es el mayor posible (n/k); comprobando que su tiempo de ejecución es mayor y que su eficiencia está entorno a 2/3. En cuanto a la métrica de la escalabilidad, la distribución cíclica es la más escalable.

Las implementaciones realizadas arrojan unos tiempos de ejecución y, en consecuencia, unas eficiencias empíricas que ratifican el comportamiento estimado para ambas distribuciones, dado el elevado grado de similitud existente.

4.3 Conclusiones de las Particiones Unidimensionales

Durante este capítulo hemos planteado diferentes divisiones por bloques de filas y de columnas de la matriz *A* de un sistema de ecuaciones y su posterior asignación cíclica entre los procesadores con objeto de aplicar la eliminación de Neville. Estas distintas formulaciones de la eliminación de Neville presentan comportamientos diversos en base al reparto de la carga computacional y al modelo de comunicaciones necesario en cada caso.

Los estudios analíticos realizados han concluido que la partición que presenta un menor tiempo de ejecución total es la distribución que consiste en dividir la matriz en bloques de una única columna y asignar éstos de forma cíclica entre los procesadores. Como consecuencia de esto, esta formulación es la que ofrece la mayor eficiencia (cercana a 1); sin embargo, no es la división más escalable, dado que al aumentar el número de procesadores requiere un mayor incremento en el tamaño de la matriz para mantener la eficiencia constante que las distribuciones por filas: *BFC* y *BF*, siendo *BFC* la más escalable.

5

Particiones Bidimensionales

En el presente capítulo abordaremos una distribución de la matriz de datos en la que ésta es dividida en submatrices cuadradas y estas submatrices, o bloques, serán asignadas a los procesadores. Este tipo de reparto se denomina *Partición Bidimensional*, el cual, dependiendo del tamaño de las submatrices, proporciona diferentes fraccionamientos de la matriz. Al igual que en el capítulo anterior, analizaremos en primer lugar la formulación genérica, considerando el orden de las submatrices como una variable, para posteriormente llevar a cabo el estudio de dos casos particulares de la misma.

5.1 Particiones Bidimensionales por Bloques Cíclicos

En una distribución bidimensional, la matriz $A=(a_{ij})_{1\le i,j\le n}$ es dividida en submatrices cuadradas, $A=(A_{ij})_{1\le i,j\le q}$, de dimensión *m x m* con $m \in [1, n/\sqrt{k}]$, siendo *k* el número de procesadores utilizado.

$$A = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1q} \\ A_{21} & A_{22} & \dots & A_{2q} \\ \dots & \dots & \dots & \dots \\ A_{q1} & A_{q2} & \dots & A_{qq} \end{pmatrix}$$

Figura 5.1 División de la matriz *A* en *qxq* bloques cuadrados

Teniendo en cuenta la división realizada sobre la matriz *A*, vamos a considerar que los procesadores están conectados formando una malla cuadrada de *k* procesadores, al procesador situado en la fila *i* y columna *j* lo denotaremos P_{ij}, con $1\le i,j\le\sqrt{k}$. La división realizada sobre la matriz *A* se traslada de una forma natural a una malla cuadrada de procesadores.

En el caso que el orden de las submatrices *(m)* en las que se divide la matriz fuese $n/\sqrt{k}$, cada procesador tendría asignada una única submatriz, existiendo una correspondencia absoluta entre el procesador y la submatriz que contiene. Esto quiere decir que el procesador P_{ij} almacenará la submatriz A_{ij} con $1\le i,j\le\sqrt{k}$. Este tipo de distribución bidimensional se denomina *Partición Bidimensional por Bloques (PBB)*.

Por otro lado, si el orden de las submatrices es 1, entonces distribuimos las filas de éstas, o lo que es lo mismo, los elementos, entre las filas de procesadores de manera cíclica y posteriormente, serán las columnas de submatrices (o de elementos) las que se repartirán cíclicamente entre las columnas de procesadores. En este caso cada procesador tendrá asignados ($n/\sqrt{k}$) *x* ($n/\sqrt{k}$) bloques de tamaño 1 y dicha asignación recibe el nombre de *Partición Bidimensional Cíclica (PBC)*.

Podemos considerar una distribución genérica que englobe a las dos anteriores particiones, denominada *Partición Bidimensional por Bloques Cíclicos (PBBC)*, que consiste en dividir la matriz en submatrices de tamaño *m* x *m* y distribuir dichos bloques de manera cíclica, al igual que en la *PBC* se hace con las submatrices de orden 1. Consideraremos que cada procesador contiene *h x h* bloques de tamaño *m* x *m*, donde $m = n / (h\sqrt{k})$, $h \in [1, n/\sqrt{k}]$ y *n* divisible por $h\sqrt{k}$. Observar que las dos primeras distribuciones mencionadas corresponden con los límites inferior y superior del número de bloques en una dimensión *(h)* de la *PBBC.*

En esta sección analizaremos el rendimiento y la escalabilidad del algoritmo de Neville cuando éste se aplica a la resolución de un sistema de ecuaciones lineales en el que la matriz de coeficientes es dividida en bloques, los cuales se distribuyen de forma cíclica entre los procesadores, esto es, cuando consideramos una *PBBC*, determinando cuál es el valor de *m* que proporciona el menor tiempo de ejecución.

Tal como ya hemos indicado en el caso unidimensional, consideraremos que el proceso de eliminación de Neville sobre la matriz real *A* (no singular) podrá llevarse a cabo sin cambios de filas.

En la figura *5.3* se muestra como, una vez dividida la matriz *A* en *q* x *q* submatrices de tamaño *m* x *m*, se distribuyen éstas entre los *k* procesadores de una malla cuadrada de acuerdo a la *PBBC*, esto es, distribuimos las filas de submatrices de manera cíclica entre las filas de procesadores seguido por las columnas o viceversa. De eso modo, la fila 1 de submatrices se ubicará en la fila 1 de procesadores, la fila 2 se situará en la fila 2 de procesadores y así sucesivamente hasta la fila $\sqrt{k}$ de bloques que se distribuirá a lo largo de la fila $\sqrt{k}$ de procesadores. Este proceso se repite hasta distribuir las *q* filas de submatrices para posteriormente hacer lo mismo con las *q* columnas de bloques entre las columnas de la malla de procesadores.

Con respecto a los elementos del vector *b*, éstos se dividirán en *q* subvectores, $b=(b_i)_{1\le i\le q}$, de dimensión *m* con $m \in [1, n / \sqrt{k}]$, distribuyéndose los mismos de manera cíclica entre los procesadores P_{ij}, con $1\le i \le\sqrt{k}$ y $j = \sqrt{k}$. En la figura 5.3 también se presenta como se asignan los *q* subvectores del vector *b* de términos independientes.

$$b=\begin{pmatrix} b_1 \\ b_2 \\ ... \\ b_q \end{pmatrix}$$

Figura 5.2 División del vector *b* en *q* subvectores

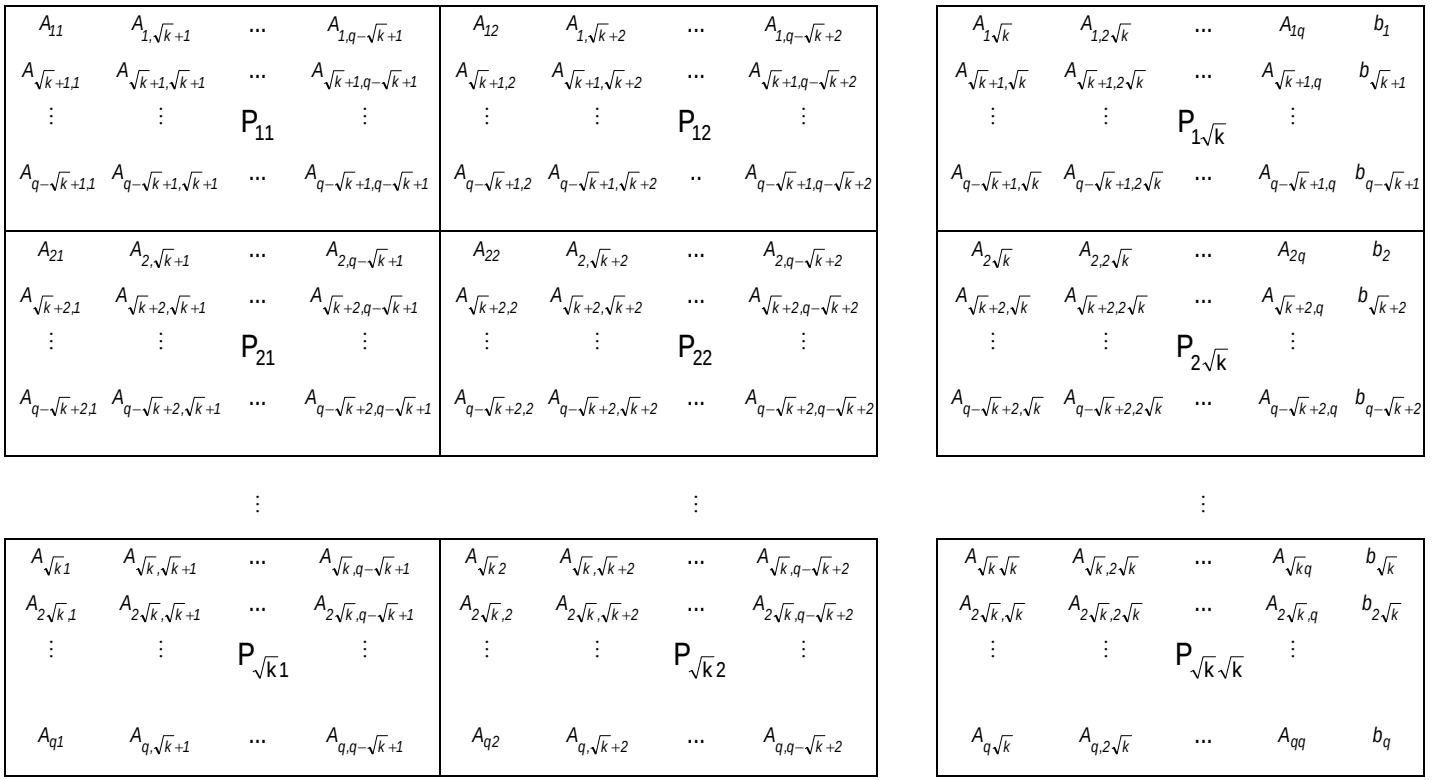


Figura 5.3 Particiones Bidimensionales por Bloques Cíclicos (*PBBC*)

A continuación vamos a describir los pasos que tienen lugar en la *j*-ésima etapa del algoritmo de eliminación de Neville con la distribución que vamos a analizar:

1. Cada procesador activo P_{lp} enviará los elementos a_{it}, con $i,t \geq j$, desde la última fila de cada una de sus submatrices al procesador inmediatamente inferior. En el caso de los procesadores situados en la última fila de la malla de procesadores, éstos enviarán dichos elementos a los procesadores de la fila 1 (figura 5.5a). En la figura 5.4 se muestra el contenido de un procesador P_{lp} genérico.
2. Aquellos procesadores que contengan los elementos a_{ij}, con $i>j$, de la matriz serán los encargados de calcular los multiplicadores de la etapa *j*. Sea *z* la columna de la malla de procesadores donde están ubicados estos procesadores, en ese caso los procesadores activos P_{lz}, con $1 \leq l \leq \sqrt{k}$, calcularán dichos multiplicadores (figura 5.5b).
3. Transferir los multiplicadores desde el procesador P_{lz} a los procesadores activos situados en su misma fila, esto es, a P_{lr} con $1 \leq l, r \leq \sqrt{k}$ (figura 5.6a).
4. Cada procesador realizará las actualizaciones de todos sus elementos de la matriz *A* que correspondan a filas y columnas con un índice superior a *j* (figura 5.6b).

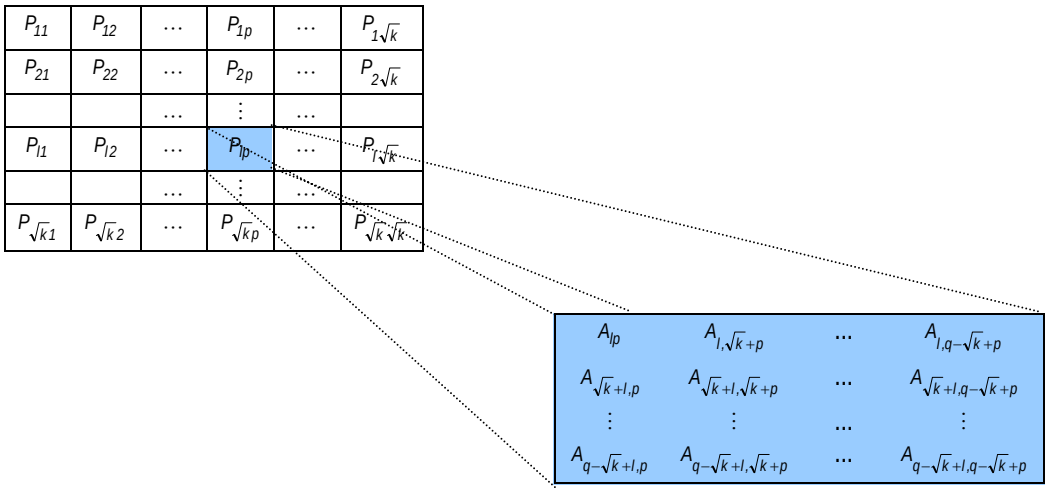
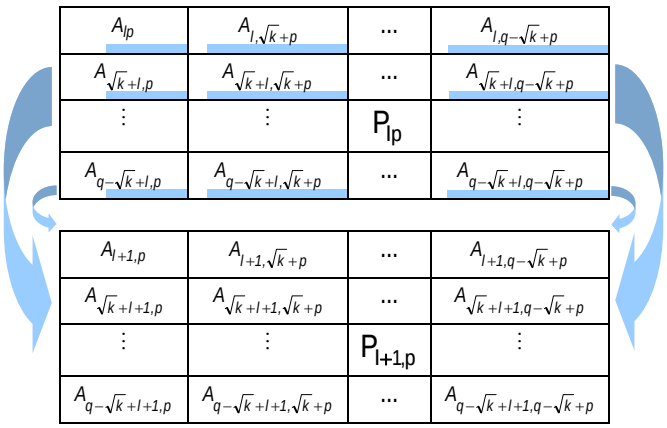


Figura 5.4 *PBBC*: Submatrices en el procesador P_{lp}



(a)

Cada procesador activo P_{lp} envía las últimas filas de cada submatriz al procesador inmediatamente inferior

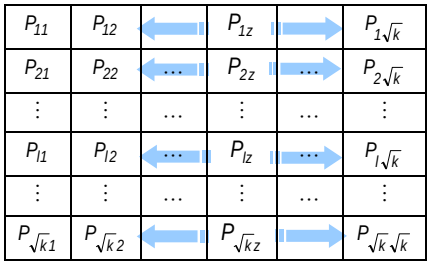
columna *j* de la matriz *A*

P_{11}	P_{12}	…	P_{1z}	…	$P_{1\sqrt{k}}$
P_{21}	P_{22}	…	P_{2z}	…	$P_{2\sqrt{k}}$
⋮	⋮	…	⋮	…	⋮
P_{l1}	P_{l2}	…	P_{lz}	…	$P_{l\sqrt{k}}$
⋮	⋮	…	⋮	…	⋮
$P_{\sqrt{k}1}$	$P_{\sqrt{k}2}$	…	$P_{\sqrt{k}z}$	…	$P_{\sqrt{k}\sqrt{k}}$

(b)

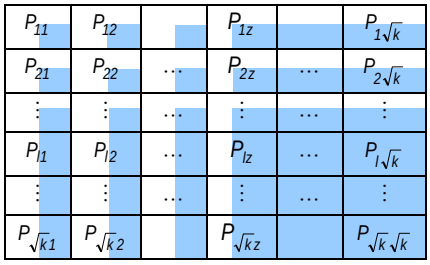
Supuesto que los elementos a_{ij} estén situados en la columna *z* de procesadores, éstos realizarán el cálculo de los multiplicadores: $a_{ij} = a_{ij} / a_{i-1,j}$ *para* $j+1 \leq i \leq n$.

Figura 5.5 Pasos 1 y 2 de la distribución *PBBC* durante la iteración *j*



(a)

Cada procesador activo de la columna *z* transmite sus multiplicadores a los procesadores activos situados en su misma fila, esto es, P_{lz} envía multiplicadores a P_{lr} con $1 \leq l, r \leq \sqrt{k}$



(b)

Actualización de la parte activa de la matriz

$a_{ti} = a_{ti} - (a_{tj} / a_{t-1,j})\, a_{t-1,i}$ *para* $j+1 \leq i \leq n$ y $j+1 \leq t \leq n$.

Figura 5.6 Pasos 3 y 4 de la distribución *PBBC* durante la iteración *j*

Seguidamente se muestra el algoritmo de la *PBBC* donde aparecen con detalle todos los pasos que hemos reseñado anteriormente, y que hemos ilustrado a través de las figuras 5.5 y 5.6, además de las operaciones que afectan al vector *b* de términos independientes.

```
1.  para j := 1 hasta n – 1 hacer                    /* bucle externo */
2.      en paralelo hacer
3.          para todo P_{l p} , con 1 ≤ l < √k  y 1 ≤ p ≤ √k , hacer
4.              para v := 0 hasta h – 1 hacer
5.                  t := (v * √k  + l ) * m;
6.                  si t ≥ j entonces
7.                      para w := 0 hasta h – 1 hacer
8.                          para r := ( p – 1) * m + 1 hasta p * m hacer
9.                              s := r + w * √k  * m ;
10.                             si s ≥ j entonces
11.                                 enviar A[t, s] de P_{l p} a P_{l+1, p} ;
12.                         fin para        /* línea 8 */
13.             fin para        /* línea 4 */
14.         para todo P_{l p} , con l = √k  y 1 ≤ p ≤ √k , hacer
15.             para v := 0 hasta h – 2 hacer
16.                 t := (v + 1) * √k  * m;
17.                 si t ≥ j entonces
18.                     para w := 0 hasta h – 1 hacer
19.                         para r := ( p – 1) * m + 1 hasta p * m hacer
20.                             s := r + w * √k  * m;
21.                             si s ≥ j entonces
22.                                 enviar A[t, s] de P_{l p} a P_{1 p} ;
23.                         fin para     /* línea 19 */
24.             fin para        /* línea 15 */
25.         fin en        /* línea 2 */
26.     en paralelo hacer
27.         para todo  P_{l √k}  , con 1 ≤ l < √k , hacer
28.             para v := 0 hasta h – 1 hacer
29.                 t := (v * √k  + l ) * m;
30.                 si t ≥ j entonces
31.                     enviar b[t] de P_{l p} a P_{l+1, p} ;
32.             fin para        /* línea 28 */
33.         para  P_{√k √k}  hacer
34.             para v := 0 hasta h – 2 hacer
35.                 t := (v + 1) * √k  * m;
36.                 si t ≥ j entonces
37.                     enviar b[t] de  P_{√k √k} a  P_{1√k} ;
38.             fin para        /* línea 34 */
39.     fin en        /* línea 26 */
```

```
40.  z := (( j – 1 ) div m) mod √k + 1;          /* P_lz , con 1 ≤ l ≤ √k , contiene la columna j */
41.    en paralelo hacer
42.      para todo P_lz , con 1 ≤ l ≤ √k , hacer
43.        para v := h – 1 descendiendo hasta 0 hacer
44.          para q := l * m descendiendo hasta ( l – 1 ) * m + 1 hacer
45.            i := q + v * √k * m;
46.            si ( i > j ) entonces
47.              A[i, j] := A[i, j] / A[i – 1, j];      /* cálculo de multiplicadores */
48.              enviar A[i, j] a los procesadores de la fila l de procesadores
49.            fin si      /* línea 46 */
50.          fin para     /* línea 44 */
51.    fin en       /* línea 41 */
52.    en paralelo hacer
53.      para todo P_lp , con 1 ≤ l, p ≤ √k , hacer
54.        para v := h – 1 descendiendo hasta 0 hacer
55.          para q := l * m descendiendo hasta ( l – 1 ) * m + 1
56.            t := q + v √k m;
57.            si ( t > j) entonces
58.              para w := 0 hasta h – 1 hacer
59.                para r := ( p – 1 ) * m + 1 hasta p * m hacer
60.                  s := r + w * √k * m;
61.                  si ( s > j ) entonces
62.                    A[t, s] := A[t, s] – A[t, j] * A[t – 1,s];      /* paso de eliminación */
63.                fin para      /* línea 59 */
64.          fin para      /* línea 55 */
65.    fin en       /* línea 52 */
66.    en paralelo hacer
67.      para todo P_l√k , con 1 ≤ l ≤ √k hacer
68.        para v := h – 1 descendiendo hasta 0 hacer
69.          para q := l * m descendiendo hasta ( l – 1 ) * m + 1
70.            t := q + v * √k * m;
71.            si ( t > j) entonces
72.              b[t] := b[t] – A[t, j] * b[t – 1];          /* paso de eliminación */
73.          fin para      /* línea 69 */
74.    fin en       /* línea 66 */
75.  fin para      /* línea 1 */
```

Algoritmo 5.1 Eliminación de Neville mediante *PBBC*

Del mismo modo que en el capítulo anterior, vamos a estimar el tiempo de ejecución de esta distribución genérica para así poder llevar a cabo el cálculo del tamaño de bloque óptimo, el análisis de la eficiencia, de la escalabilidad y la comparación entre los resultados obtenidos en los experimentos y las estimaciones teóricas.

Tiempo de Cálculo

En el algoritmo anterior se realizan dos pasos de computación, los cuales se ilustraron en las figuras 5.5b y 5.6b. El primero de ellos consiste en el cálculo de los multiplicadores (línea 47 del Algoritmo 5.1), mientras que el segundo paso de computación se encarga de actualizar la parte activa de la matriz (línea 62 del Algoritmo 5.1).

Para determinar el número de multiplicadores calculados en esta formulación paralela podemos observar que tiene un comportamiento similar al algoritmo de la *Partición Unidimensional Bloques de Filas Cíclicos (BFC)* ya que en el algoritmo que nos ocupa, las filas de la matriz *A* son también agrupadas en bloques y posteriormente distribuidas cíclicamente entre las filas de procesadores, en este caso, $\sqrt{k}$ filas de procesadores. Recordar que en *BFC* las filas de la matriz *A* se repartían entre *k* procesadores.

En *PBBC* los procesadores que más cocientes realizan en cada iteración son los situados en la última fila (fila $\sqrt{k}$) de la malla de procesadores. En la siguiente tabla se muestran los multiplicadores que deben calcular los procesadores P_{ij}, con $i = \sqrt{k}$ y $1 \leq j \leq \sqrt{k}$, a lo largo de las iteraciones del proceso de eliminación. Podemos observar como en la primera iteración del proceso, el procesador 1 de la fila $\sqrt{k}$ de procesadores, calculará exactamente *hm* multiplicadores. En la segunda iteración del proceso, cuando se elimina la variable x_2 de las ecuaciones *n*, *n*-1, …, 3; el procesador de la fila $\sqrt{k}$ al que le corresponda calculará de nuevo *hm* multiplicadores, repitiéndose esta misma situación hasta la iteración ($\sqrt{k}$ -1)*m.* A lo largo de las *m* iteraciones siguientes se da la circunstancia de que el número de multiplicadores que se deben calcular va disminuyendo una unidad en cada iteración del proceso, debido a que x_j se encuentra almacenada en algún procesador de la fila $\sqrt{k}$. Tras ese bloque de *m* iteraciones se vuelve a la situación anterior, es decir, un bloque de ($\sqrt{k}$ -1)*m* iteraciones en las que se calcula un número constante de multiplicadores, esto es, *hm-m*. Estas dos situaciones se irán alternando hasta concluir el proceso de eliminación.

ITERACIONES	NÚMERO DE MULTIPLICADORES CALCULADOS
$1,2,\ldots,(\sqrt{k}-1)\,m$	$hm, hm, \ldots, hm$
$(\sqrt{k}-1)m+1,(\sqrt{k}-1)\,m+2,\ldots,\sqrt{k}m$	hm-1, hm-2, …, hm-m
$\sqrt{k}m+1,\sqrt{k}m+2,\ldots,(2\sqrt{k}-1)\,m$	hm-m, hm-m, …, hm-m
$(2\sqrt{k}-1)\,m+1,(2\sqrt{k}-1)\,m+2,\ldots,2\sqrt{k}m$	hm-(m+1), hm-(m+2), hm-2m
$2\sqrt{k}m+1,2\sqrt{k}m+2,\ldots,(3\sqrt{k}-1)\,m$	hm-2m, hm-2m, …, hm-2m
$(3\sqrt{k}-1)\,m+1,(3\sqrt{k}-1)\,m+2,\ldots,3\sqrt{k}m$	hm-(2m+1), hm-(2m+2), …, hm-3m
$3\sqrt{k}m+1,3\sqrt{k}m+2,\ldots,(4\sqrt{k}-1)\,m$	hm-3m, hm-3m, …, hm-3m
…	…
$(h-1)\sqrt{k}m+1,(h-1)\sqrt{k}m+2,\ldots,(h\sqrt{k}-1)\,m$	$m, m, \ldots, m$
$(h\sqrt{k}-1)m+1,(h\sqrt{k}-1)\,m+2,\ldots,h\sqrt{k}m-1$	m-1, m-2, …, 1

Tabla 5.1 *PBBC*: Número de multiplicadores calculados por los procesadores de la fila $\sqrt{k}$

Los siguientes sumatorios recogen la situación mostrada en la tabla anterior:

$$\sum_{i=0}^{h-1}\sum_{j=i\sqrt{k}m+1}^{((i+1)\sqrt{k}-1)m}(hm-im)+\sum_{i=1}^{h}\sum_{j=\left(i\sqrt{k}-1\right)m+1}^{i\sqrt{k}m}\left(hm-\left(j-i\left(\sqrt{k}-1\right)m\right)\right),$$

siendo el número total de cocientes

$$\frac{1}{2}h^2m^2\sqrt{k}+\frac{1}{2}hm^2\sqrt{k}-\frac{1}{2}hm^2-\frac{1}{2}hm.$$

Como hemos mencionado anteriormente, el segundo paso de computación consiste en la actualización por parte de cada procesador de la parte activa de la matriz. Para precisar el coste de este paso de computación, conviene indicar que la distribución de las columnas de la matriz *A* en esta formulación guarda similitud con la distribución de las columnas en la *Partición Unidimensional Bloques de Columnas Cíclicas (BCC)*, al igual que la distribución de las filas de la matriz *A* es parecida a la distribución de las filas en *BFC*. La tabla 5.2 muestra como se distribuyen las filas (o columnas) de *A* entre las filas (o columnas) de la malla de procesadores.

FILAS/COLUMNAS DE PROCESADORES	FILAS/COLUMNAS DE LA MATRIZ A
Fila 1 / Columna 1	$1, 2, \ldots, m$
	$\sqrt{k}\,m+1, \sqrt{k}\,m+2, \ldots, (\sqrt{k}+1)\,m$
	...
	$(h-1)\sqrt{k}\,m+1, (h-1)\sqrt{k}\,m+2, \ldots, ((h-1)\sqrt{k}+1)m$
Fila 2 / Columna *2*	$m+1, m+2, \ldots, 2m$
	$(\sqrt{k}+1)\,m+1, (\sqrt{k}+1)\,m+2, \ldots, (\sqrt{k}+2)\,m$
	...
	$((h-1)\sqrt{k}+1)\,m+1, ((h-1)\sqrt{k}+1)\,m+2, \ldots, ((h-1)\sqrt{k}+2)\,m$
...	...
Fila $\sqrt{k}$ / Columna $\sqrt{k}$	$(\sqrt{k}-1)\,m+1, (\sqrt{k}-1)m+2, \ldots, \sqrt{k}\,m$
	$(2\sqrt{k}-1)\,m+1, (2\sqrt{k}-1)\,m+2, \ldots, 2\sqrt{k}\,m$
	...
	$(h\sqrt{k}-1)\,m+1, (h\sqrt{k}-1)\,m+2, \ldots, h\sqrt{k}\,m$

Tabla 5.2 *PBBC*: Distribución de filas/columnas entre los *k* procesadores

El procesador que realiza mayor número de actualizaciones en cada iteración del algoritmo *PBBC* es el situado en la fila $\sqrt{k}$ y columna $\sqrt{k}$. Así, en las iteraciones de la 1 a la $(\sqrt{k}-1)\,m$ se actualizarán en este procesador siempre *hm* filas *de hm* elementos cada una. En el siguiente bloque de *m* iteraciones, desde la iteración $(\sqrt{k}-1)\,m+1$ hasta la $\sqrt{k}\,m$, se va disminuyendo unidad a unidad el número de filas a actualizar al igual que el número de elementos (columnas). En el tercer bloque, en que se consideran $(\sqrt{k}-1)\,m$ iteraciones, esto es, desde la iteración $\sqrt{k}\,m+1$ hasta la iteración $(2\sqrt{k}-1)\,m$, se actualizan siempre *hm-m* filas y columnas.

En el resto de los bloques de iteraciones se van repitiendo estas dos últimas situaciones de manera que entre las iteraciones $(h-1)\sqrt{k}\,m+1$ y $(h\sqrt{k}-1)m$ se actualizan *m* filas, mientras que en la iteración $(h\sqrt{k}-1)m+1$ se actualizan *m*-1 filas, en la iteración $(h\sqrt{k}-1)m+2$ se actualizan *m*-2 filas y así sucesivamente hasta la iteración $h\sqrt{k}\,m-1$, en la cual, se actualiza una fila de un único elemento.

Teniendo en cuenta que los cálculos se realizan simultáneamente en los procesadores y que el procesador situado en la fila y columna $\sqrt{k}$ es el que más actualizaciones efectúa, vamos a contabilizar el número de operaciones aritméticas que lleva a cabo este procesador.

La situación anteriormente descrita se puede reflejar en los sumandos que aparecen a continuación. Por un lado

$$\sum_{i=0}^{h-1} \sum_{j=i\sqrt{k}\,m+1}^{\left((i+1)\sqrt{k}-1\right)m} (h\,m - i\,m)^2 \ ,$$

donde se contabilizará el número de restas que tienen lugar en los bloques de $(\sqrt{k}$ -1$)\ m$ iteraciones en los cuales siempre se actualiza el mismo número de filas y de columnas, número que varía dependiendo del bloque de iteraciones en el que nos encontremos. Por otro

$$\sum_{i=1}^{h} \sum_{j=\left(i\sqrt{k}-1\right)m+1}^{i\sqrt{k}\,m} \left(hm - \left(j - i\left(\sqrt{k}-1\right)m\right)\right)^2 \ ,$$

que registrará el número de restas que se llevan a cabo en los bloques de *m* iteraciones en los cuales el número de filas y de columnas a actualizar va disminuyendo de unidad en unidad conforme avanzan las iteraciones. El número total de restas (al igual que el de productos) es el resultado de sumar los dos sumatorios anteriores, siendo éste

$$\left(2m^2h^2\sqrt{k} + 3m^2h\sqrt{k} - 3m^2h + m^2\sqrt{k} - m^2 - 3hm + 1\right)\frac{hm}{6} \ .$$

La siguiente expresión recoge el coste de los dos pasos de computación analizados, es decir, el número total de divisiones, restas y productos, siendo por tanto el tiempo total de cálculo:

$$T_{CÁLCULO\_PBBC} = \left(\frac{1}{3}hm^3\sqrt{k} + m^3h^2\sqrt{k} + \frac{2}{3}m^3h^3\sqrt{k} - h^2m^3 - \frac{1}{6}hm - h^2m^2 - \frac{1}{3}hm^3 + \frac{1}{2}h^2m^2\sqrt{k} + \right.$$
$$\left. + \frac{1}{2}hm^2\sqrt{k} - \frac{1}{2}hm^2\right)t_{c.}$$

Sustituyendo *h* por su valor, es decir, $n / (m\sqrt{k})$, se obtiene la siguiente expresión del tiempo de cálculo, siendo ésta una función dependiente de *n*, *k*, *m* y t_c:

$$T_{CÁLCULO\_PBBC} = \left(\frac{1}{3}nm^2 + \frac{n^2m}{\sqrt{k}} + \frac{2}{3}\frac{n^3}{k} - \frac{n^2m}{k} - \frac{1}{6}\frac{n}{\sqrt{k}} - \frac{n^2}{k} - \frac{1}{3}\frac{nm^2}{\sqrt{k}} + \frac{1}{2}\frac{n^2}{\sqrt{k}} + \frac{1}{2}nm - \frac{1}{2}\frac{nm}{\sqrt{k}}\right)t_c. \qquad (5.1)$$

Tiempo de Comunicación

En esta formulación paralela del algoritmo son necesarios dos tipos de comunicaciones, tal como hemos ilustrado en las figuras 5.5a y 5.6a. La primera de ellas tiene como objetivo transferir los elementos que un procesador activo P_{ij} , con $1 \leq i,j \leq \sqrt{k}$, necesita para el cálculo de multiplicadores (líneas 11 y 22 del Algoritmo 5.1) y para la actualización de sus elementos. Esta comunicación tiene lugar desde el procesador $P_{i-1,j}$. La segunda comunicación es la encargada de difundir los multiplicadores desde el procesador P_{lz}, en el que se ubican, a los procesadores P_{lr} , con $1 \leq l,r \leq \sqrt{k}$, que los requieren (línea 48 del Algoritmo 5.1).

En el primer tipo de comunicación, un procesador enviará la última fila de cada submatriz que almacena al procesador inmediatamente inferior, siempre y cuando el índice de dicha fila sea igual o superior al número de iteración en el que nos encontremos. En este caso, los procesadores que más filas comunican son los procesadores de la penúltima fila y dentro de éstos, el que más elementos transmite es el procesador $P_{\sqrt{k}-1,\sqrt{k}}$.

En la siguiente tabla se muestran el número de filas de submatrices enviadas por el procesador $P_{\sqrt{k}-1,\sqrt{k}}$ al procesador $P_{\sqrt{k}\sqrt{k}}$. Conviene clarificar que si hay que enviar la última fila de las submatrices $A_{\sqrt{k}-1,\sqrt{k}}, A_{\sqrt{k}-1,2\sqrt{k}}, \ldots, A_{\sqrt{k}-1,q}$, consideraremos que se enviará una fila formada por los elementos que en cada etapa del proceso de eliminación sean necesarios.

ITERACIONES	NÚMERO DE FILAS COMUNICADAS
$1,2,\ldots,(\sqrt{k}-1)\,m$	$h, h, \ldots, h$
$(\sqrt{k}-1)\,m+1,(\sqrt{k}-1)\,m+2,\ldots,(2\sqrt{k}-1)\,m$	h-1, h-1, …, h-1
$(2\sqrt{k}-1)\,m+1,(2\sqrt{k}-1)\,m+2,\ldots,(3\sqrt{k}-1)\,m$	h-2, h-2, …, h-2
$(3\sqrt{k}-1)\,m+1,(3\sqrt{k}-1)\,m+2,\ldots,(4\sqrt{k}-1)\,m$	h-3, h-3, …, h-3
…	…
$((h-1)\sqrt{k}-1)\,m+1,((h-1)\sqrt{k}-1)\,m+2,\ldots,(h\sqrt{k}-1)\,m$	1

Tabla 5.3 *PBBC*: Número de filas de submatrices comunicadas en cada etapa por el procesador $P_{\sqrt{k}-1,\sqrt{k}}$

En la tabla 5.3, las iteraciones se han agrupado en bloques en función del número de filas transmitidas por el procesador $P_{\sqrt{k}-1,\sqrt{k}}$ al procesador $P_{\sqrt{k}\sqrt{k}}$. Cada una de esas filas estará formada inicialmente por *hm* elementos pero a medida que se van sucediendo las iteraciones este número irá disminuyendo. Esta variación de los elementos a comunicar en cada iteración tiene un comportamiento similar al número de multiplicadores a calcular (ver tabla 5.1).

Una vez que sabemos el número de filas que comunica el procesador $P_{\sqrt{k}-1,\sqrt{k}}$ en función de las iteraciones, podemos obtener el tiempo de comunicaciones también en función de éstas.

Para obtener el tiempo de comunicaciones vamos a utilizar dos sumandos, el primero de ellos, contabiliza el número de elementos que transmite el procesador $P_{\sqrt{k}-1,\sqrt{k}}$ en las iteraciones que van de $i\sqrt{k}\,m+1$ a $((i+1)\sqrt{k}-1)m$ con $0 \le i \le h-1$ y el segundo, registra el número de elementos que el procesador $P_{\sqrt{k}-1\sqrt{k}}$ transfiere en las iteraciones que van de $(i\sqrt{k}-1)m+1$ a $i\sqrt{k}\,m$ con $1 \le i \le h-1$. El motivo es que en el primer grupo se transmiten el mismo número de filas y el mismo número de elementos en cada bloque de iteraciones fijado; mientras que en el segundo, el número de elementos va disminuyendo dependiendo de la iteración.

El tipo de comunicación que realizamos es un envío entre procesadores vecinos, siendo el coste de dicha comunicación $t_s + m\,t_w$, donde *m* es el tamaño del mensaje, esto es, el número de filas por el número de elementos de cada fila

$$\sum_{i=0}^{h-1}\sum_{j=i\sqrt{k}\,m+1}^{((i+1)\sqrt{k}-1)m}\left(t_s+((h-i)(hm-im))t_w\right)+\sum_{i=1}^{h-1}\sum_{j=(i\sqrt{k}-1)m+1}^{i\sqrt{k}\,m}\left(t_s+\left((h-i)\left(hm-\left(j-i\left(\sqrt{k}-1\right)m\right)+1\right)\right)t_w\right).$$

El segundo tipo de comunicación consiste en un envío de uno a todos para difundir los multiplicadores. Siendo *j* la iteración del proceso de Neville y *z* la columna de los procesadores que contienen la fila *j* de la matriz *A*, cada procesador activo P_{lz}, con $1 \le l \le \sqrt{k}$, transferirá sus multiplicadores a los procesadores activos situados en su misma fila. Para estimar el coste de este tipo de comunicación nos centraremos en aquellos procesadores que más multiplicadores calculan en cada etapa y que serán por tanto los que más multiplicadores propaguen. Como ya hemos visto en la sección *Tiempo de Cálculo*, estos procesadores a los que nos referimos están situados en la fila de procesadores $\sqrt{k}$. El número de multiplicadores comunicados por un procesador perteneciente a la fila $\sqrt{k}$ se muestra en la siguiente tabla.

ITERACIONES	NÚMERO DE MULTIPLICADORES COMUNICADOS
$1,2,...,\left(\sqrt{k}-1\right)m$	*hm*, *hm*, …, *hm*
$(\sqrt{k}-1)\,m+1,(\sqrt{k}-1)\,m+2,...,\sqrt{k}\,m$	*hm*-1, *hm*-2, …, *hm-m*
$\sqrt{k}\,m+1,\sqrt{k}\,m+2,...,(2\sqrt{k}-1)\,m$	*hm-m*, *hm-m*, …, *hm-m*
$(2\sqrt{k}-1)\,m+1,(2\sqrt{k}-1)\,m+2,...,2\sqrt{k}\,m$	*hm-m*-1, *hm-m*-2, ..., *hm*-2*m*
…	…
$(h-1)\sqrt{k}\,m+1,(h-1)\sqrt{k}\,m+2,...,(h\sqrt{k}-1)\,m$	*m*, *m*, ..., *m*

Tabla 5.4 *PBBC*: Número de multiplicadores comunicados en cada etapa

Los siguientes sumatorios recogen una cota superior del coste de la difusión de los multiplicadores:

$$\sum_{i=0}^{h-1}\sum_{j=i\sqrt{k}\,m+1}^{((i+1)\sqrt{k}-1)m}\left(\left(t_s+t_w\left(hm-im\right)\right)\log\sqrt{k}\right)+\sum_{i=1}^{h-1}\sum_{j=\left(i\sqrt{k}-1\right)m+1}^{i\sqrt{k}\,m}\left(\left(t_s+t_w\left(hm-\left(j-i\left(\sqrt{k}-1\right)m\right)\right)\right)\log\sqrt{k}\right).$$

La acotación previa viene de considerar que durante todo el proceso hay $\sqrt{k}$ procesadores implicados en la propagación cuando esto realmente sucede durante todas las iteraciones salvo las $m(\sqrt{k}-2)$ últimas en las que tiene lugar dicha comunicación.

Sumando los costes de los dos tipos de comunicación y sustituyendo *h* por $n/(m\sqrt{k})$, tenemos el siguiente tiempo de comunicaciones:

$$T_{COMUNICACIÓN\_PBBC}\approx\left(n\log\sqrt{k}+n-m-m\log\sqrt{k}\right)t_s+\left(\frac{1}{3}\frac{n^3}{mk}+\frac{1}{2}\frac{n^2}{\sqrt{k}}-\frac{3}{4}\frac{n^2}{k}+\frac{1}{6}nm-\frac{1}{4}\frac{nm}{\sqrt{k}}+\frac{1}{4}\frac{n^2}{mk}-\frac{1}{4}\frac{n}{\sqrt{k}}+\right.$$
$$\left.+\frac{1}{2}\frac{n^2\log\sqrt{k}}{\sqrt{k}}+\frac{1}{2}nm\log\sqrt{k}-\frac{1}{2}\frac{nm\log\sqrt{k}}{\sqrt{k}}-\frac{1}{2}\frac{n\log\sqrt{k}}{\sqrt{k}}-\frac{1}{2}m^2\log\sqrt{k}+\frac{1}{2}m\log\sqrt{k}\right)t_w. \qquad (5.2)$$

Así, la expresión de la eficiencia teórica de *PBBC* para valores suficientemente grandes de *n*, resulta ser

$$E_{PBBC}\approx\frac{2\,m\,t_c}{2\,m\,t_c+t_w}. \qquad (5.3)$$

Hacer notar que esta expresión es la misma que resultó para *BFC* (ver (4.3)), dependiente igualmente de la constante de cálculo y de la constante de comunicación t_w, así como del tamaño de bloque utilizado (m), que a su vez es función de n y k. Conviene observar que en este caso m representa el orden de las submatrices consideradas, formadas por $m \times m$ elementos.

Tamaño de Bloque Óptimo

A través del presente apartado mostraremos, fijados n y k, el valor de m (orden de las submatrices en las que se divide la matriz A) que minimiza el tiempo de ejecución del proceso de eliminación de Neville; lo denominaremos $m_{óptimo}$. Para ello, procederemos de igual forma que en la subsección 4.1.1; esto es, sustituiremos las variables dependientes de la experimentación (t_c, t_s y t_w) por sus estimaciones, quedándonos una función dependiente de n, k y m. Posteriormente, fijados los valores de n y k, derivaremos la expresión con respecto a m y aquellos valores que anulen la derivada junto con los límites inferior y superior de m (1 y $n/\sqrt{k}$), serán los potenciales tamaños de bloque óptimo.

En la tabla 5.5, se recoge el valor de $m_{óptimo}$ para una serie de valores de n que van desde 960 a 4320, para 4, 9 y 16 procesadores y para el entorno de pruebas *IBM.* También presentamos las estimaciones del tiempo de ejecución y de la eficiencia una vez fijados n, k y m.

El valor $m_{óptimo}$ que aparece en la tabla 5.5 representa el orden exacto que deben tener las submatrices en las que se divide la matriz A, para de ese modo obtener el menor tiempo de ejecución. Como es fácilmente comprobable, estos valores de m no proporcionan una distribución uniforme, esto es, las submatrices en las que se dividiría A no serían del mismo tamaño y los procesadores no tendrían el mismo número de bloques. Por ello, hemos procedido a calcular valores cercanos a $m_{óptimo}$, tales que permitan que los procesadores tengan la misma carga de trabajo, para poder llevar a cabo los experimentos, siendo estos valores $m_{inferior}$ y $m_{superior,}$ definidos como en (4.4) y (4.5).

k	n	$m_{óptimo}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
4	960	47.647	2.819	0.836
	1440	58.496	9.138	0.871
	1920	67.711	21.200	0.890
	2400	75.854	40.838	0.902
	2880	83.226	69.880	0.911
	3360	90.011	110.145	0.918
	3840	96.330	163.450	0.924
	4320	102.267	231.608	0.928
9	960	33.555	1.396	0.751
	1440	41.205	4.379	0.808
	1920	47.718	10.003	0.838
	2400	53.476	19.096	0.858
	2880	58.692	32.479	0.871
	3360	63.492	50.972	0.882
	3840	67.963	75.389	0.890
	4320	72.164	106.545	0.897
16	960	27.269	0.876	0.673
	1440	33.483	2.648	0.751
	1920	38.795	5.947	0.793
	2400	43.497	11.243	0.819
	2880	47.757	19.004	0.838
	3360	51.678	29.692	0.852
	3840	55.331	43.772	0.862
	4320	58.763	61.705	0.871

Tabla 5.5 *PBBC*: Valores de $m_{óptimo}$ para *IBM SP2*

En las tablas 5.6, 5.7 y 5.8 se recogen los valores de $m_{inferior}$ y $m_{superior}$ correspondientes a cada valor de $m_{óptimo}$ que se mostraba en la tabla 5.5. En el centro de cada fila, correspondiente a un valor de n concreto, aparece $m_{óptimo}$ mientras que por encima figura $m_{inferior}$ y por debajo, se muestra $m_{superior}$.

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	48	2.819	0.836
	47.647	2.819	0.836
	48	2.819	0.836
1440	48	9.159	0.869
	55.496	9.138	0.871
	60	9.138	0.871
1920	64	21.203	0.890
	67.711	21.200	0.890
	80	21.230	0.889
2400	75	40.839	0.902
	75.854	40.838	0.902
	80	40.844	0.902
2880	80	69.884	0.911
	83.226	69.880	0.911
	90	69.898	0.911
3360	84	110.165	0.918
	90.011	110.145	0.918
	105	110.247	0.917
3840	96	163.450	0.924
	96.330	163.450	0.924
	96	163.450	0.924
4320	90	231.739	0.928
	102.267	231.608	0.928
	108	231.632	0.928

Tabla 5.6 *PBBC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *IBM SP2 y* 4 procesadores

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	32	1.396	0.751
	33.555	1.396	0.751
	40	1.399	0.749
1440	40	4.379	0.808
	41.205	4.379	0.808
	48	4.386	0.806
1920	40	10.024	0.837
	47.718	10.003	0.838
	64	10.062	0.833
2400	50	19.101	0.857
	53.476	19.096	0.858
	80	19.291	0.849
2880	48	32.554	0.869
	58.692	32.479	0.871
	60	32.480	0.871
3360	56	51.014	0.881
	63.492	50.972	0.882
	70	50.997	0.881
3840	64	75.403	0.890
	67.963	75.389	0.890
	80	75.490	0.889
4320	72	106.545	0.897
	72.164	106.545	0.897
	72	106.545	0.897

Tabla 5.7 *PBBC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *IBM SP2 y* 9 procesadores

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	24	0.877	0.672
	27.259	0.876	0.673
	30	0.877	0.672
1440	30	2.651	0.751
	33.483	2.648	0.751
	36	2.649	0.751
1920	32	5.964	0.791
	38.795	5.947	0.793
	40	5.948	0.793
2400	40	11.249	0.819
	43.497	11.243	0.819
	50	11.259	0.818
2880	48	19.004	0.838
	47.757	19.004	0.838
	48	19.004	0.838
3360	42	29.773	0.849
	51.678	29.692	0.852
	56	29.704	0.851
3840	48	43.825	0.861
	55.331	43.772	0.862
	60	43.789	0.862
4320	54	61.730	0.871
	58.763	61.705	0.871
	60	61.706	0.871

Tabla 5.8 *PBBC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *IBM SP2 y* 16 procesadores

Sin más que observar los resultados obtenidos, podemos comprobar que:

- fijado un valor de n y k, los tiempos y la eficiencia obtenidos para $m_{inferior,}$ $m_{óptimo\ y}$ $m_{superior}$, son muy próximos, en algunos casos idénticos.
- conforme n y k crecen, la eficiencia aumenta, alcanzando los valores más elevados (por encima de 0.9) cuando n=4320 y k=4.

En la misma línea, las tablas 5.9-5.11 mostrarán los valores de $m_{óptimo}$ para 4, 9 y 16 procesadores en el segundo entorno de experimentación, esto es, *CLÚSTER*.

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	96	8.055	0.695
	97.862	8.055	0.695
	120	8.098	0.691
1440	120	25.235	0.749
	120.928	25.235	0.749
	144	25.322	0.746
1920	120	57.490	0.779
	140.556	57.349	0.781
	160	57.445	0.780
2400	150	108.949	0.803
	157.916	108.923	0.804
	200	109.481	0.799
2880	160	184.552	0.820
	173.643	184.449	0.820
	180	184.469	0.820
3360	168	288.675	0.832
	188.122	288.387	0.833
	210	288.662	0.832
3840	192	425.254	0.843
	201.610	425.479	0.843
	240	426.141	0.841
4320	180	600.523	0.850
	214.284	599.249	0.852
	216	599.252	0.852

Tabla 5.9 *PBBC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *CLÚSTER y* 4 procesadores

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	64	4.355	0.571
	67.741	4.353	0.572
	80	4.371	0.569
1440	80	13.018	0.645
	84.136	13.014	0.646
	96	13.045	0.644
1920	80	28.987	0.687
	98.090	28.837	0.691
	128	29.098	0.684
2400	100	53.957	0.721
	110.427	53.895	0.722
	160	54.781	0.710
2880	120	90.238	0.745
	121.599	90.236	0.745
	160	90.988	0.739
3360	112	140.271	0.761
	131.881	139.890	0.763
	140	139.941	0.763
3840	128	205.072	0.777
	141.455	204.874	0.778
	160	205.177	0.777
4320	144	287.246	0.790
	150.449	287.195	0.790
	160	287.296	0.790

Tabla 5.10 *PBBC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *CLÚSTER y* 9 procesadores

n	$m_{inferior}$ / $m_{óptimo}$ / $m_{superior}$	$T_{TEÓRICO}$	$E_{TEÓRICA}$
960	48	2.921	0.479
	53.987	2.915	0.480
	60	2.920	0.479
1440	60	8.358	0.565
	67.420	8.341	0.567
	72	8.346	0.566
1920	60	18.241	0.614
	78.863	18.052	0.621
	80	18.052	0.621
2400	75	33.367	0.656
	88.980	33.239	0.658
	100	33.299	0.657
2880	90	55.127	0.686
	98.140	55.076	0.687
	120	55.356	0.683
3360	105	84.728	0.709
	106.569	84.726	0.709
	120	84.868	0.708
3840	96	123.767	0.724
	114.415	123.342	0.727
	120	123.373	0.727
4320	120	172.074	0.742
	121.785	172.070	0.742
	135	172.267	0.741

Tabla 5.11 *PBBC*: $m_{inferior}$, $m_{óptimo}$ y $m_{superior}$ para *CLÚSTER y* 16 procesadores

Lo ya comentado para el *IBM* es extrapolable al caso del *CLÚSTER*, si bien en este último las eficiencias que se alcanzan son algo menores. Así podemos observar como para n=4320 y k=4 se alcanza la eficiencia más elevada (entorno a 0.850).

Realizando una comparativa más detallada de ambos entornos observamos que el tiempo de ejecución del algoritmo en el *IBM* es aproximadamente un tercio del tiempo en el *CLÚSTER*. Si desglosáramos el tiempo total del algoritmo en sus tres partes (tiempo de cálculo, tiempo de comunicación entre vecinos y tiempo de difusión) comprobaríamos que

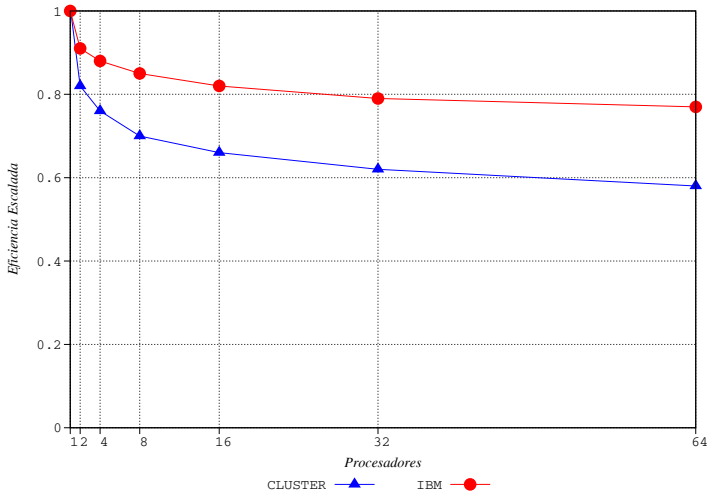
- el tiempo de cálculo en el *CLÚSTER* es aproximadamente 2.5 veces mayor que en el *IBM*, lo cual es una consecuencia directa de la diferencia existente entre los valores estimados para la constante t_c de ambos entornos,

- el tiempo de la comunicación entre vecinos es el quíntuple en el *CLÚSTER* que en el *IBM* y
- el tiempo de la difusión viene a ser 9 veces mayor en el *CLÚSTER*.

Como consecuencia de lo anterior, hay diferencias en las eficiencias siendo 0.193 la más elevada (ver el caso n=960 y k=16).

Análisis de la Escalabilidad

La mayor o menor facilidad de nuestro sistema paralelo para mantener la eficiencia constante la mostraremos a través de la eficiencia escalada (gráfica 5.1). Dicha gráfica evidencia que redoblar n y W hace debilitarse la eficiencia de una forma relativamente suave en el *IBM* y algo más fuerte en el *CLÚSTER*. A partir de los datos que presentamos en el Anexo B, podemos comprobar como en el *IBM* se pasa de una eficiencia 1.00 para el par (n,k) =(960,1) a una eficiencia 0.77 para n=3841 y k=64, esto es, una pérdida de 0.23 de eficiencia. En el *CLÚSTER*, el detrimento es de 0.43 al pasar de 1.00 a 0.57.

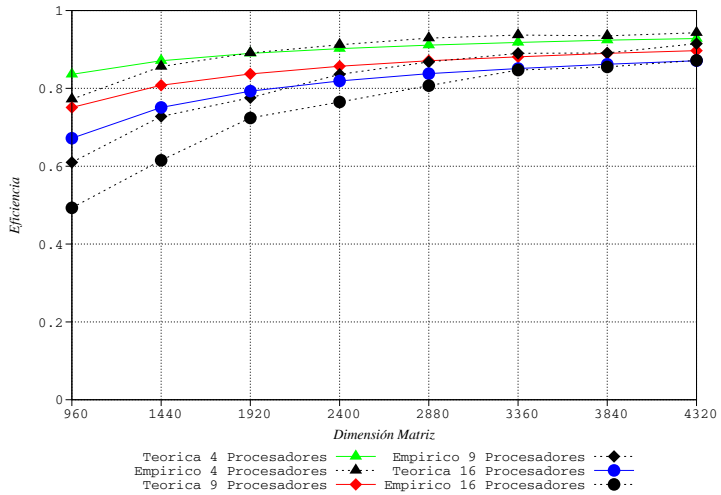


Gráfica 5.1 *PBBC*: Eficiencia escalada

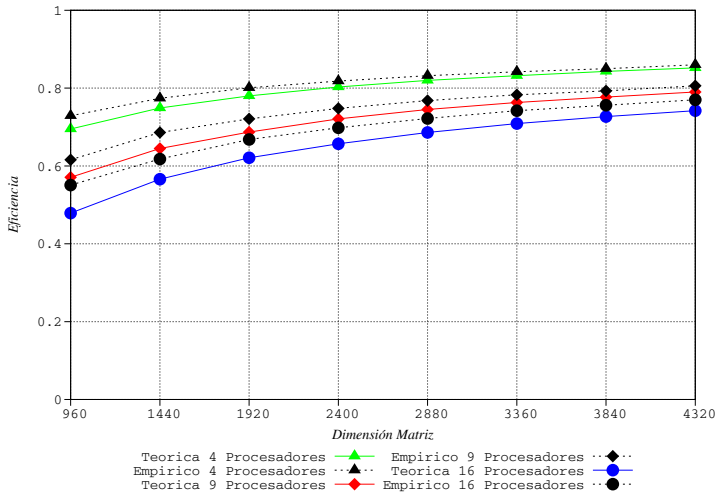
Experimentos

Una vez obtenidas las estimaciones del menor tiempo de ejecución y de su correspondiente eficiencia, en esta sección vamos a contrastar esos datos con los obtenidos tras implementar el Algoritmo 5.1 en los entornos de experimentación descritos en el capítulo 3. Las pruebas se realizaron para los valores de *n* y *k* mencionados en el apartado *Tamaño de Bloque Óptimo* y como orden de las submatrices *(m)* utilizamos $m_{pseudoóptimo}$, (ver (4.6)), esto es, el tamaño de bloque más cercano (mayor o menor) a $m_{óptimo}$ tal que permita una distribución equitativa de la matriz en los procesadores.

Seguidamente, a través de gráficas, vamos a cotejar los datos estimados y los datos derivados de las pruebas.



Gráfica 5.2 *PBBC* con $m_{pseudoóptimo}$: Eficiencia teórica vs. empírica para *IBM SP2*



Gráfica 5.3 *PBBC* con $m_{pseudoóptimo}$: Eficiencia teórica vs. empírica para *CLÚSTER*

Como se puede observar en la gráfica 5.3 (*CLÚSTER*), la tendencia del modelo teórico es seguida fielmente desde el principio por el modelo empírico y a medida que aumenta el orden de la matriz, la proximidad entre los dos modelos claramente es mayor.

En el caso de la gráfica 5.2 (*IBM*), el camino marcado por las expectativas teóricas también es seguido por los resultados derivados de los experimentos, aunque a una mayor distancia que en el caso anterior para órdenes pequeños de la matriz. Sin embargo, para las matrices de mayor tamaño como 3840 o 4320, el grado de proximidad llega a ser del 99% y 100%, mientras que en el *CLÚSTER* es del 96%. Lo que nos confirma que esta distribución proporciona una eficiencia muy elevada, superior a 0.9 y 0.8 en *IBM* y *CLÚSTER*, respectivamente.

Se evidencia por tanto que el modelo teórico es prácticamente igual al modelo empírico, y por ello estamos en condiciones de mostrar como sería el comportamiento de esta distribución para un valor elevado de *n* y para 4, 9, ... y 121 procesadores.

k	*IBM SP2*	*CLÚSTER*
4	0.974	0.945
9	0.963	0.922
16	0.955	0.904
25	0.947	0.889
36	0.941	0.875
121	0.915	0.820

Tabla 5.12 *PBBC*: Estimación de la eficiencia para $n=2^{15}$

La eficiencia para 4 procesadores es muy elevada en ambos entornos y disminuye cuando el número de procesadores aumenta. Si comparamos la información recogida en esta tabla con sus homólogas de *BFC* y *BCC*, vemos claramente que la eficiencia disminuye muy levemente en *PBBC*. Tanto es así que en el *IBM* se pierde solamente 0.059 frente a las pérdidas que tienen lugar en *BFC* y en *BCC*, 0.146 y 0.137 respectivamente. La misma situación se produce en el *CLÚSTER* siendo el detrimento de la eficiencia mayor que en el *IBM*, esto es, 0.125, 0.253 y 0.409, en *PBBC*, *BFC* y *BCC* respectivamente.

5.2 Particiones Bidimensionales por Bloques

En esta sección vamos a analizar un caso particular de *PBBC*, concretamente aquel en el que el orden de las submatrices en las que se divide la matriz *A* es $n/\sqrt{k}$. Tal y como se indica en la figura 5.7, la matriz *A* quedará dividida en *k* submatrices de dimensión $n/\sqrt{k}\ x\ n/\sqrt{k}$. A la hora de distribuir los bloques entre los procesadores resulta ser que cada submatriz A_{ij} se situará en el procesador P_{ij}, con $1 \le i,j \le \sqrt{k}$. Del mismo modo, el vector *b* se fracciona en $\sqrt{k}$ subvectores de tamaño $n/\sqrt{k}$ (figura 5.7) y, los subvectores b_i con $1 \le i,j \le \sqrt{k}$ se distribuyen entre los procesadores de la columna $\sqrt{k}$ de procesadores. La situación queda finalmente como se muestra en la figura 5.8.

$$A = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1\sqrt{k}} \\ A_{21} & A_{22} & \dots & A_{2\sqrt{k}} \\ \dots & \dots & \dots & \dots \\ A_{\sqrt{k}1} & A_{\sqrt{k}2} & \dots & A_{\sqrt{k}\sqrt{k}} \end{pmatrix}, \; b = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_{\sqrt{k}} \end{pmatrix}$$

Figura 5.7 División de la matriz *A* en *k* bloques cuadrados y del vector *b* en $\sqrt{k}$ subvectores

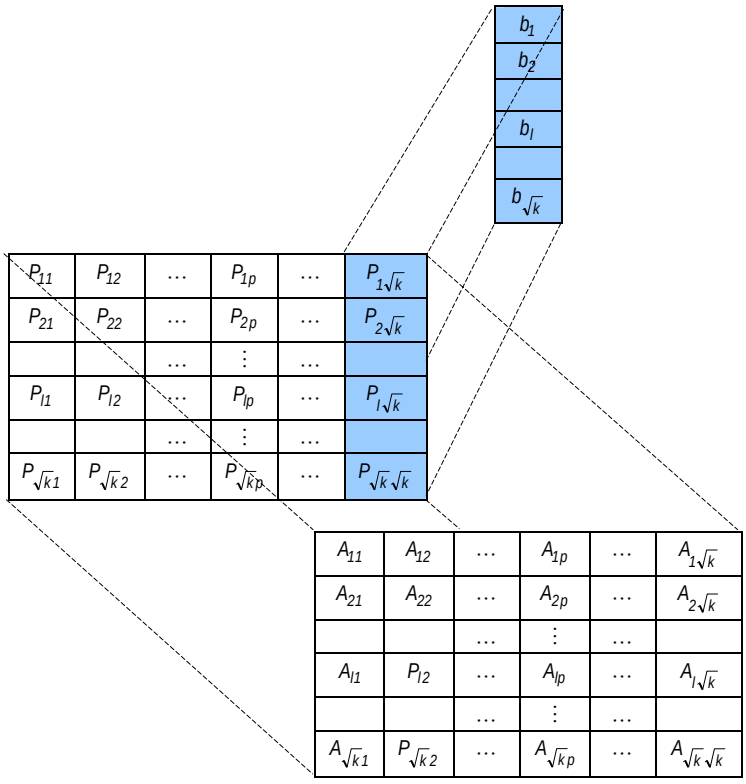


Figura 5.8 Particiones Bidimensionales por Bloques (*PBB*)

A continuación vamos a obtener las expresiones del tiempo de cálculo y del tiempo de comunicaciones de *PBB*, para así proceder a un análisis detallado de este tipo de partición.

Sustituiremos por tanto *m* por $n/\sqrt{k}$ en (5.1) y (5.2) resultando

$$T_{CÁLCULO\_PBB} = \left(2\frac{n^3}{k} - \frac{4}{3}\frac{n^3}{k^{3/2}} - \frac{3}{2}\frac{n^2}{k} - \frac{1}{6}\frac{n}{\sqrt{k}} + \frac{n^2}{\sqrt{k}} \right) t_c \tag{5.4}$$

y

$$T_{COMUNICACIÓN\_PBB} \approx \left(n\log\sqrt{k} + n - \frac{n}{\sqrt{k}} - \frac{n\log\sqrt{k}}{\sqrt{k}} \right) t_s + \left(\frac{n^2}{\sqrt{k}} + \frac{n^2\log\sqrt{k}}{\sqrt{k}} - \frac{n^2}{k} - \frac{n^2\log\sqrt{k}}{k} \right) t_w \ . \tag{5.5}$$

De ese modo, el tiempo de ejecución paralelo total es de manera aproximada,

$$T_{PARALELO\_PBB} \approx 2\frac{n^3}{k} t_c + n\log\sqrt{k}\, t_s + \frac{n^2\log\sqrt{k}}{\sqrt{k}} t_w \ . \tag{5.6}$$

Si calculamos la expresión de la eficiencia, para *n* tendiendo a infinito, a partir de (5.4), de (5.5) y de la expresión del tiempo del algoritmo secuencial de Neville, tenemos

$$E_{PBB} \approx \frac{\sqrt{k}}{3\sqrt{k} - 2} \ . \tag{5.7}$$

Observar que esta expresión depende del número de procesadores que integran las filas o columnas de la malla de procesadores y es aproximadamente 1/3.

Análisis de la Escalabilidad

Si consideramos (5.6) prescindiendo de los costes de comunicación, veremos que el coste de esta distribución es igual a $2n^3t_c$, el cual es tres veces más que el coste del algoritmo secuencial (ver (2.25)). Por tanto, existe una cota superior de 1/3 en la eficiencia del algoritmo paralelo.

La ineficiencia de esta formulación es debida a la inactividad de los procesadores, resultante de la mala distribución de la carga. Este problema puede ser aliviado si la matriz es dividida, dentro de las *Particiones Bidimensionales*, de una manera cíclica. De ese modo, la mayor diferencia de carga computacional entre dos procesadores en una iteración cualquiera será de una fila y una columna.

Cuando veamos la distribución cíclica, comprobaremos que ciertamente se obtiene un mejor reparto de la carga de trabajo entre los procesadores, pero eso conlleva un incremento elevado de las comunicaciones. De ahí que en *PBBC* se obtenga un valor de m ($m_{óptimo}$) entre 1 y $n/\sqrt{k}$,y diferente de ellos, que proporciona un equilibrio entre el reparto de la carga y el volumen de las comunicaciones.

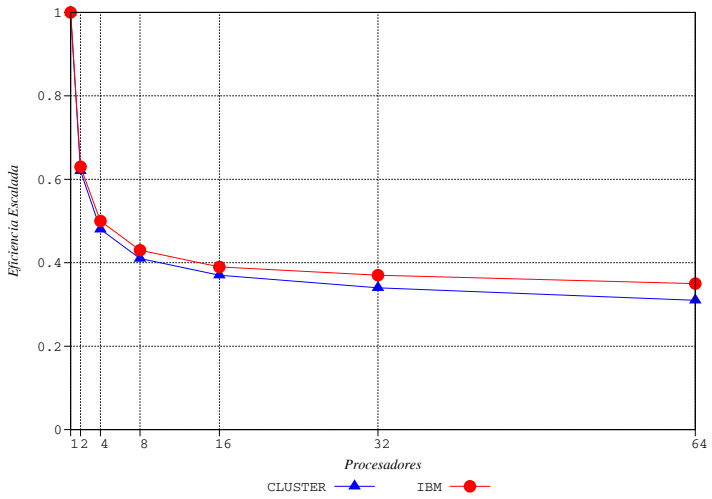
A continuación analizaremos la parte de la función de isoeficiencia asociada a las comunicaciones. Si calculamos el término de la isoeficiencia debido a t_s resulta

$$n^3 \propto K\, n k \log\sqrt{k}\; t_s ,$$
$$n^3 = W \propto K^{\frac{3}{2}} k^{\frac{3}{2}} \log^{\frac{3}{2}}\sqrt{k}\; t_s^{\frac{3}{2}} . \qquad (5.8)$$

De igual modo, para determinar el término de la isoeficiencia debido a t_w,

$$n^3 \propto K(n^2/\sqrt{k})\, k \log\sqrt{k}\; t_w ,$$
$$n^3 = W \propto K^3 k^{\frac{3}{2}} \log^3\sqrt{k}\; t_w{}^3 . \qquad (5.9)$$

De (5.8) y (5.9) obtenemos que la función de isoeficiencia asintótica debida a la sobrecarga de comunicación es $\theta\left(k^{\frac{3}{2}} \log^3\sqrt{k}\right)$.

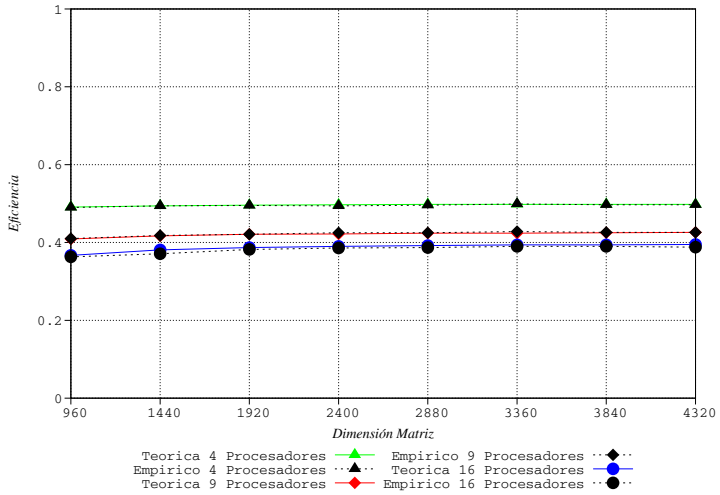


Gráfica 5.4 *PBB*: Eficiencia escalada

Con la gráfica 5.4 concluimos el estudio de la escalabilidad para *PBB* y a través de ella podemos ver claramente que la eficiencia cae drásticamente en el tramo de 1 a 8 procesadores para posteriormente mostrar un descenso muy leve en ambos entornos. Los valores exactos por los que discurre la eficiencia escalada pueden verse en el Anexo B.

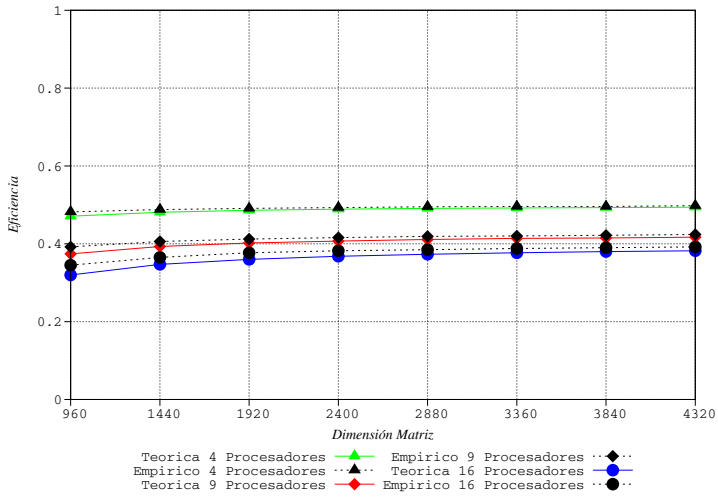
Experimentos

Una vez concluidos los experimentos, los resultados que arrojan son comparados con los del modelo teórico a través de las siguientes gráficas.



Gráfica 5.5 *PBB*: Eficiencia teórica vs. empírica para *IBM SP2*

Como se puede comprobar en las gráficas 5.5 y 5.6, en esta distribución se logra un ajuste casi perfecto, independientemente del número de procesadores y del tamaño de la matriz, corroborando así las previsiones teóricas que indicaban que la eficiencia estaría alrededor de 0.33.



Gráfica 5.6 *PBB*: Eficiencia teórica vs. empírica para *CLÚSTER*

Teniendo en cuenta el buen ajuste entre el modelo teórico y el modelo empírico, vamos a obtener estimaciones de la eficiencia para valores más elevados de n y k, esto es, $n=2^{15}$ y k= 4, 9, 16, 25, 36, 121.

k	*IBM SP2*	*CLÚSTER*
4	0.500	0.499
9	0.428	0.427
16	0.399	0.398
25	0.384	0.381
36	0.374	0.371
121	0.353	0.346

Tabla 5.13 *PBB*: Estimación de la eficiencia para $n=2^{15}$

donde se refleja que el comportamiento en ambos entornos de pruebas es prácticamente igual. La eficiencia comienza siendo 0.5 para 4 procesadores y desciende a 0.35 para el caso de 121 procesadores.

5.3 Particiones Bidimensionales Cíclicas

En esta sección vamos a analizar el otro caso extremo de *PBBC*, concretamente aquel que corresponde con el límite inferior de *m*, es decir, la matriz *A* se divide en submatrices de orden *1*, quedando por tanto dividida en n^2 submatrices. Como podemos ver en la figura 5.9, la submatriz A_{ij} está integrada exclusivamente por el elemento a_{ij} con $1 \leq i,j \leq n$. Esta distribución se denomina *Partición Bidimensional Cíclica (PBC)*.

$$A = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{pmatrix} \equiv \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

Figura 5.9 División de la matriz *A* en *nxn* bloques cuadrados

Si procedemos a distribuir los bloques, en este caso de tamaño 1 x 1, entre los procesadores de la malla de procesadores, llegaríamos a una distribución de la matriz *A* que corresponde con la mostrada en la figura 5.3, teniendo en cuenta que la submatriz A_{ij} estaría compuesta por un único elemento y éste es a_{ij}.

En el caso que nos ocupa, sustituyendo *m* por 1 en (5.1) y (5.2), obtendremos el tiempo de cálculo y el tiempo de comunicaciones de *PBC*, los cuales se muestran a continuación:

$$T_{CÁLCULO\_PBC} = \left(\frac{2}{3}\frac{n^3}{k} + \frac{3}{2}\frac{n^2}{\sqrt{k}} - 2\frac{n^2}{k} + \frac{5}{6}n - \frac{n}{\sqrt{k}} \right) t_c \tag{5.10}$$

y

$$T_{COMUNICACIÓN\_PBC} \approx \left(n\log\sqrt{k} + n - 1 - \log\sqrt{k} \right) t_s + \left(\frac{1}{2}\frac{n^2}{\sqrt{k}} + \frac{1}{3}\frac{n^3}{k} + \frac{1}{6}n - \frac{1}{2}\frac{n^2}{k} - \frac{1}{2}\frac{n}{\sqrt{k}} + \frac{1}{2}n\log\sqrt{k} - \right.$$
$$\left. - \frac{n\log\sqrt{k}}{\sqrt{k}} + \frac{1}{2}\frac{n^2}{\sqrt{k}}\log\sqrt{k} \right) t_w \,. \tag{5.11}$$

Así, el tiempo de ejecución paralelo total es de manera aproximada,

$$T_{PARALELO\_PBC} \approx \frac{2}{3}\frac{n^3}{k} t_c + n \log \sqrt{k}\, t_s + \frac{1}{3}\frac{n^3}{k} t_w \ . \tag{5.12}$$

En lo que respecta a la eficiencia cuando *n* tiene a infinito, ésta depende exclusivamente de la constantes t_c y t_w y es exactamente igual que la obtenida para *CF*, esto es

$$E_{PBC} \approx \frac{2t_c}{2t_c + t_w} \ . \tag{5.13}$$

Los valores que proporciona esta expresión son: 0.21 para el *IBM* y 0.055 para el *CLÚSTER*.

A través de (5.10) y (5.11) hemos obtenido las estimaciones para el tiempo de ejecución y teniendo en cuenta la expresión del tiempo del algoritmo secuencial hemos calculado las eficiencias estimadas que mostramos en la siguiente tabla:

k	*n*	$T_{TEÓRICO}$	$E_{TEÓRICA}$
4	960	11.311	0.208
	1440	38.013	0.209
	1920	89.953	0.210
	2400	175.535	0.210
	2880	303.165	0.210
	3360	481.247	0.210
	3840	718.187	0.210
	4320	1022.000	0.210
9	960	5.103	0.205
	1440	17.028	0.208
	1920	40.182	0.209
	2400	78.301	0.209
	2880	135.21	0.209
	3360	214.377	0.210
	3840	319.804	0.210
	4320	455.139	0.210
16	960	2.933	0.201
	1440	9.683	0.205
	1920	22.759	0.207
	2400	44.26	0.208
	2880	76.288	0.209
	3360	120.945	0.209
	3840	180.331	0.209
	4320	256.548	0.209

Tabla 5.14 *PBC*: Estimaciones del tiempo de ejecución y de la eficiencia para *IBM SP2*

Como se puede observar, los pronósticos para *PBC* son poco satisfactorios, igual que sucedía en *CF*. Todo ello es debido, tanto en una distribución como en otra, al elevado volumen de comunicación que se debe realizar.

Análisis de la Escalabilidad

La ineficiencia de *PBB* relativa al balanceado de la carga se ve corregida en esta distribución, pero como contrapartida, se eleva enormemente el volumen de comunicaciones haciendo que éstas sean de orden n^3. En *PBBC* tienen lugar dos tipos de comunicaciones, entre vecinos y difusiones, siendo las primeras las que tienen un aumento considerable en el caso particular *PBC*. Las operaciones colectivas de comunicación no sufren un aumento en cuanto al número de elementos a difundir sino que hay más procesadores implicados en la comunicación que es, precisamente, el peor escenario para este tipo de comunicaciones.

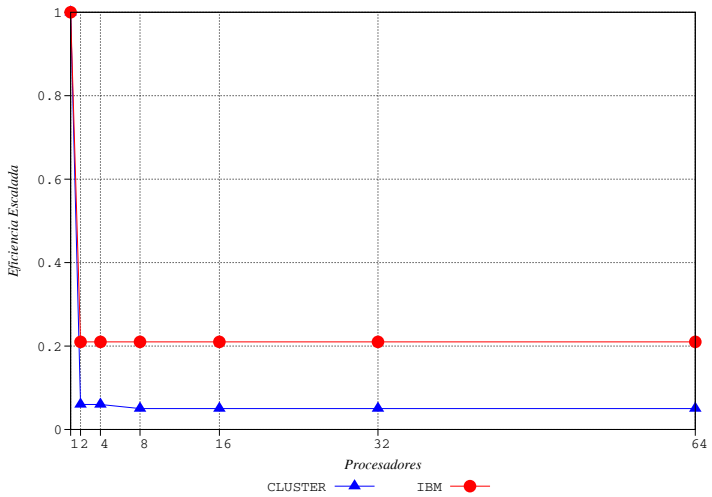
Para realizar el análisis de la función de isoeficiencia de *PBC* partimos de la función de sobrecarga relativa a la comunicación de esta distribución la cual es aproximadamente

$$n k \log \sqrt{k}\, t_s + n^3 t_w .$$

El término de la isoeficiencia debido a t_s es igual al obtenido en *PBB* y está recogido en (5.8). Puesto que W es n^3, el término $n^3 t_w$ siempre está balanceado con W. Este término es independiente de k y no contribuye a la función de isoeficiencia. Si calculamos el término de la isoeficiencia relativo al mayor término positivo n^2 de t_w, obtendremos que éste corresponde con la expresión recogida en (5.9). Teniendo en cuenta el análisis anterior, podemos concluir que la función de isoeficiencia para *PBC* es

$$\theta\left(k^{\frac{3}{2}} \log^3 \sqrt{k} \right).$$

Concluiremos el presente análisis con la gráfica 5.7 donde se muestra la eficiencia escalada correspondiente a esta distribución. Los datos que nos han permitido obtener la gráfica 5.7 pueden verse en el Anexo B, en cualquier caso, es evidente al observarla que para un cierto valor de n y k=2 las eficiencias son 0.21 y 0.06 en el *IBM* y en el *CLÚSTER* respectivamente y que estas eficiencias se mantienen constantes cuando k y W se van multiplicando por el mismo factor.



Gráfica 5.7 *PBC*: Eficiencia escalada

Experimentos

Dado el poco interés de esta distribución debido a su exigua eficiencia, no expondremos gráficamente las eficiencias obtenidas tras los experimentos, nos limitaremos a dejar constancia de que las expectativas teóricas se vieron confirmadas.

5.4 Conclusiones de las Particiones Bidimensionales

Al comienzo del presente capítulo hemos propuesto una división de la matriz *A* de coeficientes en submatrices cuadradas, para posteriormente distribuir éstas de forma cíclica en una malla cuadrada de procesadores. Hemos comprobado como el orden de las submatrices tiene un fuerte impacto tanto en el reparto de la carga de trabajo entre los procesadores como en el coste de las comunicaciones. De hecho, no coincidirán el buen balanceado de la carga con el mínimo tiempo en las comunicaciones; es más, ambos factores entrarán en competencia. Por ello, estamos obligados a adoptar compromisos razonables en ambos casos y calcular aquel orden de la submatriz que suponga el mejor tiempo de cálculo minimizando el efecto perjudicial en las comunicaciones. Esto nos lleva a estimar dicho tamaño de bloque óptimo para diferentes valores de *n* y *k*. Los pronósticos de la eficiencia para la distribución

por bloques cíclicos, considerando tal tamaño de submatriz, establecen que ésta es muy elevada, alcanzando valores de 0.9 y 0.85 dependiendo del entorno de experimentación analizado.

Además hemos analizado dos casos particulares de la distribución *PBBC* que corresponden con el peor balanceado de la carga y el menor tiempo de comunicaciones (*PBB*) y con el mejor reparto de la carga computacional y el peor tiempo de comunicaciones (*PBC*). El estudio de estas dos particiones de la matriz *A* nos confirma que sus estimaciones del tiempo de ejecución, y como consecuencia de ello, de la eficiencia, son peores que en el caso mencionado en el párrafo anterior, siendo 0.5 el máximo valor de la eficiencia en *PBB* y 0.2, su homólogo en *PBC*.

En cuanto a la métrica de la escalabilidad, la asignación por bloques cíclicos (*PBBC*), considerando un tamaño de bloque óptimo, se demuestra como la más escalable comparándola tanto con *PBB* y *PBC* como con las distribuciones unidimensionales analizadas a lo largo del capítulo 4.

6

Conclusiones y Líneas Futuras

A lo largo de la memoria de la tesis se ha mostrado un estudio en profundidad de diferentes estrategias paralelas orientadas a bloques para resolver el método de eliminación de Neville. Dichos estudios teóricos tenían como finalidad analizar las prestaciones de cada uno de los algoritmos en base a tres métricas: tiempo de ejecución, eficiencia y escalabilidad.

A continuación presentaremos algunas de las conclusiones más destacadas para los distintos tipos de distribuciones consideradas.

6.1 Conclusiones

La eliminación de Neville puede abordarse en multicomputadores con distintas organizaciones de los algoritmos por bloques. En este trabajo hemos estudiado las prestaciones tanto teóricas como experimentales de los dos tipos de distribuciones más habituales: *Particiones Unidimensionales*, por bloques de filas o bien por bloques de columnas, y *Particiones Bidimensionales*.

La posibilidad de subdividir la matriz de diferentes formas da también lugar a diferentes prestaciones por lo que las distintas organizaciones del algoritmo pueden ser más o menos adecuadas en función del *hardware* disponible y del uso que se le quiera dar.

Con el fin de comprobar la robustez de los modelos teóricos desarrollados y validar su comportamiento frente a los modelos experimentales, hemos comenzado por modelizar los dos entornos sobre los cuales íbamos a realizar nuestros experimentos, un *IBM SP2* y un *CLÚSTER.*

Los entornos utilizados para la experimentación efectuada en este trabajo corresponden a la clase *MIMD*, y deben ser considerados como sistemas de memoria distribuida, siendo el paradigma de paso de mensaje la técnica más apropiada para su programación. Además, y con el fin de realizar un uso lo más eficiente posible de los recursos de comunicación y reducir el tiempo de comunicación, hemos considerado la segmentación como técnica de ruta.

Los resultados experimentales muestran la validez de la aproximación teórica y permiten obtener una idea aproximada pero bastante fiel de los resultados que se obtendrían en otro sistema *hardware* que se pudiera modelar mediante el modelo de paso de mensajes, con distintos parámetros de los probados aquí. En este sentido se dispone de una herramienta de análisis con gran capacidad de predicción.

En lo que respecta a la *Partición Unidimensional* de los datos, hemos planteado diferentes divisiones por bloques de filas y de columnas de la matriz *A* de un sistema de ecuaciones y su posterior asignación cíclica entre los procesadores con objeto de aplicar la eliminación de Neville. Estas distintas formulaciones de la eliminación de Neville presentan comportamientos diversos en base al reparto de la carga computacional y al modelo de comunicaciones necesario en cada caso.

Los estudios analíticos realizados han concluido que la partición que presenta un menor tiempo de ejecución total es la distribución que consiste en dividir la matriz en bloques de una única columna y asignar éstos de forma cíclica entre los procesadores. Como consecuencia de esto, esta formulación es la que ofrece la mayor eficiencia (cercana a 1); sin embargo, no es la división más escalable, dado que al aumentar el número de procesadores requiere un mayor incremento en el tamaño de la matriz para mantener la eficiencia constante que las distribuciones por filas: *BFC* y *BF*, siendo *BFC* la más escalable.

Al abordar las *Particiones Bidimensionales,* hemos propuesto una división de la matriz *A* de coeficientes en submatrices cuadradas, para posteriormente distribuir éstas de forma cíclica en una

malla cuadrada de procesadores. Hemos comprobado como el orden de las submatrices tiene un fuerte impacto tanto en el reparto de la carga de trabajo entre los procesadores como en el coste de las comunicaciones. Por ello, hemos calculado aquel orden de la submatriz que suponga el menor tiempo de ejecución total. Esto nos lleva a estimar dicho tamaño de bloque óptimo para diferentes valores de *n* y *k*. Los pronósticos de la eficiencia para la distribución por bloques cíclicos, considerando tal tamaño de submatriz, establecen que ésta es muy elevada, alcanzando valores de 0.9 y 0.85 dependiendo del entorno de experimentación analizado.

Asimismo, hemos analizado dos casos particulares de la distribución *PBBC* que corresponden con un orden de submatriz igual a 1 (*PBC*) e igual a $n/\sqrt{k}$ (*PBB*), siendo *n* el orden de la matriz A de coeficientes y *k* el número de procesadores. El estudio de estas dos particiones de la matriz *A* nos confirma que sus estimaciones del tiempo de ejecución, y como consecuencia de ello, de la eficiencia son peores que en el caso mencionado en el párrafo anterior.

Si el foco del análisis es la escalabilidad, la asignación por bloques cíclicos (*PBBC*), considerando un tamaño de bloque óptimo, se demuestra como la más escalable comparándola tanto con *PBB* y *PBC* como con las distribuciones unidimensionales analizadas.

Por todo lo dicho podemos concluir que, en función del objetivo prioritario o las características técnicas del entorno de experimentación, uno de los algoritmos analizados será el más apropiado. Así, si el objetivo es la escalabilidad, la distribución por bloques cíclicos es la mejor, pero si se busca una eficiencia máxima lo será el basado en una distribución por bloques de columnas. Cuando se pretenda maximizar ambas métricas nuevamente el planteamiento por bloques cíclicos es el más apropiado, para tamaños de problema razonables. Finalmente, si la topología de la red es muy simple (línea o anillo) y/o el subsistema de comunicaciones lento y con tendencia a la saturación, el idóneo será el algoritmo por bloques de filas dado que todas las comunicaciones que se hacen en este modelo son del tipo entre vecinos.

6.2 Producción Científica

El trabajo desarrollado a lo largo de estos años en el campo del álgebra lineal numérica y la computación de altas prestaciones, ha sido presentado ante la comunidad científica a través de diferentes publicaciones recogidas en revistas de reconocido prestigio o a través de congresos de carácter internacional, los cuales recordaremos brevemente en la presente sección.

Así, las publicaciones [Alonso *et al* 99b], [Alonso *et al* 01], [Alonso *et al* 02], [Alonso *et al* 03] y [Alonso *et al* 04] fueron evaluados por la Comisión Nacional Evaluadora de la Actividad Investigadora (*CNEAI*) que en su reunión de 6 de junio de 2005 otorgó una valoración positiva a la autora de la presente memoria.

Por otro lado, y en especial a lo que se refiere a la parte experimental, nuestras investigaciones se han podido llevar a cabo gracias a nuestro esfuerzo personal (dada la limitación de entornos adecuados en nuestra universidad) y a la colaboración con distintas entidades, bien a través de proyectos u otro tipo de colaboraciones.

6.2.1 Publicaciones de Investigación

El trabajo recopilado en la presente memoria de tesis ha dado lugar a distintas publicaciones, una de ellas, pendiente de publicación. A continuación mostramos una relación de las mismas.

- Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J.; Scalability Blocking Neville Elimination Algorithm for Exploiting Cache Memories, Applied Mathematics and Computation, DOI:10.1016/j.amc.2008.06.023, 2008.

- Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J.; Scalability of Neville Elimination using Checkerboard Partitioning, International Journal of Computer Mathematics, Vol. 85(3&4), pp. 309-317, 2008.

- Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J.; Analyzing Scalability of Neville Elimination, Journal of Mathematical Chemistry, Vol. 40(1), pp. 49-61, 2006.

- Abascal, P.; Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J.; Analyzing the Efficiency of Block-Cyclic Checkerboard Partitioning in Neville Elimination, Lecture Notes in Computer Science, Vol. 3019, pp. 963-968, Springer-Verlag, 2004.

- Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J.; Neville Elimination: A Study of the Efficiency using Checkerboard Partitioning. Linear Algebra and its Applications, Vol. 393, pp. 3-14, 2004.

- Alonso, P.; Cortina, R.; Díaz, I.; Hernández, V.; Ranilla, J.; A Simple Cost-Optimal Parallel Algorithm to Solve Linear Equation Systems, INFORMATION: An International Interdisciplinary Journal, Vol. 6(3), pp. 297-304, 2003.

- Alonso, P.; Cortina, R.; Díaz, I.; Hernández, V.; Ranilla, J.; A Columnwise Block Striping in Neville Elimination, Lecture Notes in Computer Science, Vol. 2328, pp. 379-386, Springer-Verlag, 2002.

- Alonso, P.; Cortina, R.; Hernández, V.; Ranilla, J.; A Study of the Performance of Neville Elimination using Two Kinds of Partitioning Techniques, Linear Algebra and its Applications, Vol. 332-334, pp. 111-117, 2001.

- Alonso, P.; Cortina, R.; Ranilla, J.; Block-Striped Partitioning and Neville Elimination, Lecture Notes in Computer Science, Vol. 1685, pp. 1073-1077, Springer-Verlag, 1999.

6.2.2 Congresos

La difusión de los diferentes trabajos ha tenido lugar en congresos internacionales como:

- International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2006 y CMMSE 2005).
- International Conference of Computational Methods in Sciences and Engineering (ICMMSE 2005).
- International Meeting on High Performance Computing for Computational Science (VECPAR 2004)
- International Conference on Parallel Processing and Applied Mathematics (PPAM 2003 y PPAM 2001)
- Workshop on High Performance Scientific and Engineering Computing with Applications (HPSECA 2001)
- International European Conference on Parallel Computing (Euro-Par 1999)
- Internacional Linear Algebra Society Conference (ILAS 1999)

6.2.3 Proyectos

El trabajo ha estado integrado a lo largo de los años en diferentes proyectos, comenzando por tres proyectos de investigación de la Universidad de Oviedo desde el año 1998 al 2001, siendo éstos:

- Eliminación de Neville y sus Aplicaciones: un Análisis basado en Computación de Altas Prestaciones (ENYCAP), 1998.
- Computación de Altas Prestaciones y Eliminación de Neville (HPCNE), 1999.
- Computación en Paralelo: Particiones Bidimensionales en la Eliminación de Neville (PCNE), 2001.

Por otro lado, y con el fin de aplicar nuestros conocimientos en el campo de la computación paralela a la Organización y Recuperación de Información, formamos parte del equipo de investigadores de los siguientes proyectos del Ministerio de Educación y Ciencia:

- Organización y Recuperación de Información Paralela e Inteligente (ORIPI), 2004-2007.

- Sistema Flexible para la Reducción de Atributos en Problemas de Categorización de Alta Dimensionalidad (SERAPAD), 2007-2010.

Con la intención de poder disponer de recursos necesarios para un adecuado desarrollo de nuestro trabajo desde un punto de vista empírico, entre 1998 y 2004, se estableció con el Centro de Supercomputación de Cataluña (CESCA) un Proyecto Académico, el cual nos ha permitido utilizar sus instalaciones en la implementación de una gran cantidad de algoritmos:

- Eliminación de Neville y sus Aplicaciones: un Análisis basado en Computación de Altas Prestaciones.

Además, y considerando las carencias de infraestructuras que para cálculo científico existían en nuestra universidad, hemos participado de diversos proyectos para el establecimiento y actualización de un Cluster de Modelización Científica:

- Cluster Universitario de Modelización Científica (Ministerio de Educación y Ciencia - Ayudas FEDER a la realización de proyectos de infraestructura científica), 2005-2006.

- Ampliación del Cluster de Modelización Científico-Técnica del Campus de Mieres (Universidad de Oviedo – Principado de Asturias), 2008.

Por último, también queremos reseñar que en colaboración con investigadores de otras universidades formamos parte del equipo del siguiente proyecto, patrocinado por el Ministerio de Educación y Ciencia para Ayuda a la realización de Acciones Complementarias Nacionales:

- Red de Computación de Altas Prestaciones sobre Arquitecturas Paralelas Heterogéneas (CAPAP-H), 2007-2009.

6.3 Líneas Futuras

Como líneas futuras de trabajo y de estudio estamos abordando en la actualidad las siguientes:

- *Análisis del comportamiento de un modelo general:* En colaboración con el profesor Antonio Vidal, proponemos una organización del algoritmo de eliminación de Neville por bloques para computadores que sigan el modelo de paso de mensajes, llevando a cabo

un análisis general basado en cotas superiores para las tres métricas, tiempo, *speedup*/eficiencia y escalabilidad. Este análisis se particularizará para las distribuciones por bloques más usuales y se comparará con el caso experimental en dos tipos de máquinas representativas del modelo de paso de mensajes: una red de estaciones de trabajo y un multicomputador. Dedicaremos especial atención al estudio de la escalabilidad de los algoritmos, tratando de determinar cual es la relación entre el tamaño del bloque y las prestaciones obtenidas en esta métrica.

- *Adecuación de la Eliminación de Neville para el uso del nivel 3 de BLAS:* Las subrutinas del nivel 3 de *BLAS* son las que obtienen un mayor rendimiento en las computadoras actuales, dado que aprovechan al máximo las ventajas de las jerarquías de memoria. Por ello, procederemos a reorganizar el algoritmo de eliminación de Neville orientándolo a bloques y, de esta forma, reducir el número de accesos/referencias a memoria central. El resultado, como en el nivel 3 de *BLAS*, será un incremento sustancial de la cantidad de operaciones de cómputo que se realizarían por cada referencia a memoria central.

- *Paralelización de algoritmos de simulación de fracturas:* Clásicamente, la simulación de fracturas se aborda a través de distintos métodos numéricos del álgebra lineal, entre ellos la factorización de Cholesky adaptada al caso de matrices dispersas. El elevado coste del proceso de resolución de este tipo de problemas, ha hecho que nos plantearemos la paralelización del procedimiento, para mejorar los tiempos de ejecución y/o permitir la manipulación de un número más elevado de datos. Este trabajo de investigación lo estamos realizando en colaboración con algunos investigadores del Instituto de Física de Cantabria (*IFCA*).

- *Optimización de algoritmos numéricos de uso común en Química:* Entre las múltiples operaciones que deben realizar los paquetes de *software* de modelización molecular se encuentran tanto la resolución de sistemas de ecuaciones lineales como la inversión de matrices cuadradas. Más en particular, el cálculo de la inversa de matrices positivas es una tarea muy común en la Química Computacional. Por un lado, con los métodos químico-cuánticos es posible calcular la energía electrónica o bien analizar las densidades de carga electrónicas, pero siempre es condición necesaria invertir previamente la matriz de solapamiento (S) del conjunto de funciones de base auxiliares. Por otro lado, la inversión de la matriz Hessiana de un sistema molecular (H, la matriz de las derivadas segundas de la energía potencial respecto a las coordenadas atómicas) es un paso obligado en la optimización eficiente de geometrías moleculares y en los cálculos de modos normales. A medida que los sistemas moleculares que son sometidos a la simulación computacional van siendo más y más grandes, hasta llegar a los cientos y miles de átomos, la disponibilidad de métodos directos para invertir las correspondientes matrices S y H podría ser crucial para superar los cuellos de botella de otros algoritmos tradicionales. En los últimos meses, hemos iniciado diversos contactos con profesores del Departamento de Química Física y Analítica de la Universidad de Oviedo, con el objeto de intentar mejorar el rendimiento de algunos de los algoritmos habitualmente usados utilizando estrategias de computación paralela.

- *Análisis de las prestaciones de diferentes estrategias de pivotaje:* Dentro del exhaustivo estudio que venimos haciendo del proceso de eliminación de Neville en el campo de la computación de altas prestaciones, pretendemos abordar, en colaboración con el profesor Juan M. Peña, un completo análisis de la eficacia y precisión del método cuando sobre él se efectúan algunas estrategias de pivotaje especialmente adecuadas cuando se trabaja sobre un multicomputador.

Anexo A

Resultados Empíricos

Los tiempos de ejecución empleados por las implementaciones de los diferentes algoritmos que figuran en el presente documento, así como el tiempo de ejecución del algoritmo secuencial, son mostrados a lo largo de este anexo. Dichos tiempos corresponden a los tamaños de matriz y número de procesadores que figuran en los análisis de los capítulos 4 y 5, así como a dos entornos de experimentación representativos del modelo de paso de mensajes: una red de estaciones de trabajo (*CLÚSTER*) y un multicomputador (*IBM SP2*). Recordemos que con objeto de refrendar el modelo teórico, las constantes de cálculo y de comunicación de ambos entornos han sido estimadas, siendo éstas:

ENTORNO	t_c	t_s	t_w
IBM SP2	1.6×10^{-8} s	3.1×10^{-5} s	1.2×10^{-7} s
CLÚSTER	3.8×10^{-8} s	8.8×10^{-5} s	1.3×10^{-6} s

A partir de dichos resultados empíricos se ha calculado la eficiencia correspondiente así como su relación con la eficiencia estimada, obtenida ésta a partir de la aplicación al modelo teórico de las estimaciones de las constantes vinculadas a los entornos de experimentación. Todos estos datos han dado lugar a las gráficas que confrontan sendas eficiencias en los capítulos 4 y 5 del presente documento. Los datos obtenidos se muestran mediante tablas, donde se recoge la siguiente información:

- k: número de procesadores,
- n: orden de la matriz,
- $T_{SECUENCIAL\_EMPÍRICO}$: tiempo de ejecución, en segundos, de la implementación del algoritmo secuencial,
- $T_{PARALELO\_EMPÍRICO}$: tiempo de ejecución, en segundos, de la implementación de la distribución mostrada,
- $m_{pseudoóptimo}$: tamaño de bloque, cercano al óptimo, utilizado en cada experimento,
- $E_{EMPÍRICA}(E_E)$: eficiencia calculada a partir de $T_{SECUENCIAL\_EMPÍRICO}$ y $T_{PARALELO\_EMPÍRICA}$,
- $E_{TEÓRICA}(E_T)$: estimación de la eficiencia,
- E_T-E_E: diferencia entre $E_{TEÓRICA}$ y $E_{EMPÍRICA}$.

A.1 IBM SP2 del CESCA

A.1.1 Secuencial

k	n	$T_{SECUENCIAL\_EMPÍRICO}$
1	960	9.502
1	1440	31.859
1	1920	75.211
1	2400	147.028
1	2880	252.986
1	3360	402.792
1	3840	598.366
1	4320	851.263

A.1.2 Particiones Unidimensionales

A.1.2.1 Bloques de Filas Cíclicos (*BFC*)

k	*n*	$m_{pseudoóptimo}$	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	40	2.913	0.815	0.831	0.016
	1440	45	9.362	0.851	0.861	0.010
	1920	60	21.549	0.872	0.879	0.007
	2400	60	41.454	0.887	0.891	0.004
	2880	72	70.473	0.897	0.900	0.003
	3360	70	110.747	0.909	0.907	-0.002
	3840	80	164.458	0.910	0.913	0.003
	4320	90	233.344	0.912	0.918	0.006
8	960	30	1.744	0.681	0.757	0.076
	1440	36	5.095	0.782	0.799	0.017
	1920	40	11.483	0.819	0.824	0.005
	2400	50	21.941	0.838	0.840	0.002
	2880	45	37.248	0.849	0.854	0.005
	3360	42	58.364	0.863	0.862	-0.001
	3840	48	85.956	0.870	0.871	0.001
	4320	54	121.466	0.876	0.878	0.002
16	960	20	0.916	0.648	0.673	0.025
	1440	18	2.827	0.704	0.722	0.018
	1920	30	6.366	0.738	0.758	0.020
	2400	30	12.129	0.758	0.781	0.023
	2880	30	20.381	0.776	0.797	0.021
	3360	35	32.011	0.786	0.810	0.024
	3840	40	46.169	0.810	0.821	0.011
	4320	45	64.564	0.824	0.829	0.005

A.1.2.2 Bloques de Filas (*BF*)

k	*n*	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	3.628	0.655	0.666	0.011
	1440	11.969	0.665	0.672	0.007
	1920	28.047	0.670	0.674	0.004
	2400	54.567	0.674	0.676	0.002
	2880	93.804	0.674	0.677	0.003
	3360	148.618	0.677	0.677	0.000
	3840	222.527	0.672	0.679	0.007
	4320	314.869	0.676	0.678	0.002
8	960	1.980	0.600	0.640	0.040
	1440	6.220	0.640	0.652	0.012
	1920	14.481	0.649	0.657	0.008
	2400	27.980	0.657	0.660	0.003
	2880	48.047	0.658	0.662	0.004
	3360	75.996	0.662	0.663	0.001
	3840	113.045	0.662	0.664	0.002
	4320	160.592	0.662	0.665	0.003
16	960	1.078	0.551	0.610	0.059
	1440	3.271	0.609	0.632	0.023
	1920	7.633	0.616	0.642	0.026
	2400	14.574	0.630	0.648	0.018
	2880	25.014	0.632	0.652	0.020
	3360	38.979	0.646	0.654	0.008
	3840	58.256	0.642	0.656	0.014
	4320	82.139	0.648	0.657	0.009

A.1.2.3 Bloques de Columnas (*BC*)

k	*n*	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	3.539	0.671	0.652	-0.019
	1440	11.764	0.677	0.663	-0.014
	1920	27.713	0.678	0.668	-0.010
	2400	54.019	0.680	0.671	-0.009
	2880	92.898	0.681	0.673	-0.008
	3360	147.315	0.683	0.674	-0.009
	3840	221.043	0.677	0.675	-0.002
	4320	312.512	0.681	0.676	-0.005
8	960	1.907	0.623	0.587	-0.036
	1440	6.134	0.649	0.618	-0.031
	1920	14.285	0.658	0.633	-0.025
	2400	27.618	0.665	0.641	-0.024
	2880	47.533	0.665	0.646	-0.019
	3360	75.269	0.669	0.650	-0.019
	3840	112.062	0.667	0.653	-0.014
	4320	159.344	0.668	0.655	-0.013
16	960	1.084	0.548	0.482	-0.066
	1440	3.455	0.576	0.544	-0.032
	1920	7.489	0.628	0.575	-0.053
	2400	14.435	0.636	0.594	-0.042
	2880	24.595	0.643	0.607	-0.036
	3360	38.702	0.650	0.616	-0.034
	3840	57.828	0.647	0.623	-0.024
	4320	81.846	0.650	0.628	-0.022

A.1.2.4 Cíclico por Columnas (*CC*)

k	*n*	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	2.809	0.846	0.929	0.083
	1440	8.990	0.886	0.956	0.070
	1920	20.513	0.917	0.969	0.052
	2400	39.894	0.921	0.976	0.055
	2880	67.559	0.936	0.980	0.044
	3360	105.650	0.953	0.983	0.030
	3840	156.566	0.955	0.986	0.031
	4320	221.873	0.959	0.987	0.028
8	960	1.648	0.721	0.815	0.094
	1440	4.949	0.805	0.881	0.076
	1920	11.166	0.842	0.914	0.072
	2400	21.435	0.857	0.932	0.075
	2880	35.533	0.890	0.944	0.054
	3360	55.504	0.907	0.953	0.046
	3840	81.734	0.915	0.959	0.044
	4320	115.549	0.921	0.964	0.043
16	960	1.162	0.511	0.625	0.114
	1440	3.043	0.654	0.738	0.084
	1920	6.566	0.716	0.800	0.084
	2400	12.107	0.759	0.839	0.080
	2880	20.066	0.788	0.866	0.078
	3360	30.911	0.814	0.885	0.071
	3840	45.189	0.827	0.889	0.062
	4320	63.354	0.840	0.910	0.070

A.1.3 Particiones Bidimensionales

A.1.3.1 Bloques Cíclicos (*PBBC*)

k	n	$m_{pseudoóptimo}$	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	48	3.078	0.772	0.836	0.064
	1440	60	9.298	0.856	0.871	0.015
	1920	64	21.094	0.891	0.890	-0.001
	2400	75	40.315	0.912	0.902	-0.010
	2880	80	68.103	0.929	0.911	-0.018
	3360	84	107.476	0.937	0.918	-0.019
	3840	96	160.032	0.935	0.924	-0.011
	4320	108	225.720	0.943	0.928	-0.015
9	960	32	1.732	0.610	0.751	0.141
	1440	40	4.860	0.728	0.808	0.080
	1920	40	10.770	0.776	0.837	0.061
	2400	50	19.531	0.836	0.857	0.021
	2880	60	32.369	0.868	0.871	0.003
	3360	70	50.294	0.890	0.881	-0.009
	3840	64	74.600	0.891	0.890	-0.001
	4320	72	103.326	0.915	0.897	-0.018
16	960	30	1.205	0.493	0.672	0.179
	1440	36	3.238	0.615	0.751	0.136
	1920	40	6.494	0.724	0.793	0.069
	2400	40	12.018	0.765	0.819	0.054
	2880	48	19.596	0.807	0.838	0.031
	3360	56	29.710	0.847	0.851	0.004
	3840	60	43.739	0.855	0.862	0.007
	4320	60	61.037	0.872	0.871	-0.001

A.1.3.2 Bloques (*PBB*)

k	*n*	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	4.844	0.490	0.491	0.001
	1440	16.128	0.494	0.494	0.000
	1920	37.949	0.495	0.496	0.001
	2400	74.365	0.494	0.497	0.003
	2880	127.522	0.496	0.498	0.002
	3360	201.657	0.499	0.498	-0.001
	3840	300.797	0.497	0.498	0.001
	4320	427.755	0.497	0.498	0.001
9	960	2.573	0.410	0.409	-0.001
	1440	8.4583	0.418	0.417	-0.001
	1920	19.831	0.421	0.421	0.000
	2400	38.420	0.425	0.422	-0.003
	2880	66.112	0.425	0.424	-0.001
	3360	104.648	0.428	0.424	-0.004
	3840	155.887	0.426	0.425	-0.001
	4320	221.731	0.426	0.426	0.000
16	960	1.635	0.363	0.367	0.004
	1440	5.369	0.371	0.381	0.010
	1920	12.314	0.382	0.387	0.005
	2400	23.785	0.386	0.390	0.004
	2880	40.816	0.387	0.392	0.005
	3360	64.453	0.390	0.394	0.004
	3840	95.952	0.390	0.394	0.004
	4320	137.185	0.388	0.395	0.007

A.2 CLÚSTER

A.2.1 Secuencial

k	n	$T_{SECUENCIAL\_EMPÍRICO}$
1	960	22.260
1	1440	75.147
1	1920	178.148
1	2400	347.837
1	2880	600.997
1	3360	954.358
1	3840	1425.006
1	4320	2033.234

A.2.2 Particiones Unidimensionales

A.2.2.1 Bloques de Filas Cíclicos (*BFC*)

k	*n*	$m_{pseudoóptimo}$	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	80	7.413	0.751	0.698	-0.053
	1440	120	24.452	0.768	0.744	-0.024
	1920	120	56.280	0.791	0.772	-0.019
	2400	150	107.831	0.806	0.792	-0.014
	2880	144	183.705	0.818	0.808	-0.010
	3360	168	288.354	0.827	0.820	-0.007
	3840	192	426.849	0.835	0.830	-0.005
	4320	180	603.776	0.842	0.838	-0.004
8	960	60	4.227	0.658	0.597	-0.061
	1440	90	13.658	0.688	0.649	-0.039
	1920	80	31.117	0.716	0.685	-0.031
	2400	100	59.306	0.733	0.711	-0.022
	2880	120	100.912	0.744	0.729	-0.015
	3360	105	156.969	0.760	0.746	-0.014
	3840	120	231.781	0.768	0.759	-0.009
	4320	135	327.429	0.776	0.770	-0.006
16	960	60	2.267	0.614	0.504	-0.110
	1440	45	7.635	0.615	0.555	-0.060
	1920	60	17.587	0.633	0.594	-0.039
	2400	75	33.709	0.645	0.624	-0.021
	2880	90	57.019	0.659	0.644	-0.015
	3360	70	87.666	0.680	0.662	-0.018
	3840	80	128.873	0.691	0.680	-0.011
	4320	90	181.474	0.700	0.693	-0.007

A.2.2.2 Bloques de Filas (*BF*)

k	*n*	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	8.691	0.640	0.632	-0.008
	1440	28.665	0.655	0.649	-0.006
	1920	67.277	0.662	0.657	-0.005
	2400	130.532	0.666	0.662	-0.004
	2880	224.497	0.669	0.665	-0.004
	3360	355.701	0.671	0.667	-0.004
	3840	530.167	0.672	0.670	-0.002
	4320	753.815	0.674	0.670	-0.004
8	960	4.699	0.592	0.578	-0.014
	1440	15.319	0.613	0.608	-0.005
	1920	35.418	0.629	0.623	-0.006
	2400	68.116	0.638	0.632	-0.006
	2880	116.452	0.645	0.639	-0.006
	3360	184.050	0.648	0.643	-0.005
	3840	273.732	0.651	0.647	-0.004
	4320	388.705	0.654	0.649	-0.005
16	960	2.381	0.584	0.504	-0.080
	1440	8.241	0.570	0.553	-0.017
	1920	18.799	0.592	0.579	-0.013
	2400	35.847	0.606	0.596	-0.010
	2880	60.755	0.618	0.607	-0.011
	3360	95.549	0.624	0.615	-0.009
	3840	141.620	0.629	0.622	-0.007
	4320	200.545	0.634	0.627	-0.007

A.2.2.3 Bloques de Columnas (*BC*)

k	*n*	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	8.564	0.650	0.590	-0.060
	1440	28.455	0.660	0.619	-0.041
	1920	67.464	0.660	0.635	-0.025
	2400	131.413	0.662	0.644	-0.018
	2880	226.251	0.664	0.650	-0.014
	3360	357.894	0.667	0.654	-0.013
	3840	532.809	0.669	0.657	-0.012
	4320	756.890	0.671	0.660	-0.011
8	960	4.558	0.610	0.453	-0.157
	1440	15.223	0.617	0.512	-0.105
	1920	36.240	0.614	0.546	-0.068
	2400	70.393	0.618	0.568	-0.050
	2880	120.447	0.624	0.584	-0.040
	3360	189.437	0.630	0.595	-0.035
	3840	280.135	0.636	0.604	-0.032
	4320	396.149	0.642	0.611	-0.031
16	960	3.574	0.389	0.291	-0.098
	1440	10.399	0.452	0.365	-0.087
	1920	22.276	0.500	0.415	-0.085
	2400	40.775	0.533	0.450	-0.083
	2880	67.481	0.557	0.477	-0.080
	3360	104.406	0.571	0.498	-0.073
	3840	152.523	0.584	0.515	-0.069
	4320	213.715	0.595	0.529	-0.066

A.2.2.4 Cíclico por Columnas (*CC*)

k	*n*	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	6.707	0.830	0.801	-0.029
	1440	21.418	0.877	0.863	-0.014
	1920	49.491	0.900	0.895	-0.005
	2400	94.711	0.918	0.916	-0.002
	2880	161.301	0.931	0.929	-0.002
	3360	253.485	0.941	0.939	-0.002
	3840	375.884	0.948	0.947	-0.001
	4320	532.271	0.955	0.952	-0.003
8	960	4.347	0.640	0.574	-0.066
	1440	12.765	0.736	0.678	-0.058
	1920	28.611	0.778	0.741	-0.037
	2400	53.563	0.812	0.784	-0.028
	2880	89.629	0.838	0.815	-0.023
	3360	139.137	0.857	0.838	-0.019
	3840	203.471	0.875	0.856	-0.019
	4320	285.304	0.891	0.870	-0.021
16	960	4.621	0.301	0.336	0.035
	1440	10.981	0.428	0.442	0.014
	1920	21.238	0.524	0.519	-0.005
	2400	36.061	0.603	0.577	-0.026
	2880	56.679	0.663	0.623	-0.040
	3360	84.4641	0.706	0.660	-0.046
	3840	119.481	0.745	0.690	-0.055
	4320	164.004	0.775	0.716	-0.059

A.2.3 Particiones Bidimensionales

A.2.3.1 Bloques Cíclicos (*PBBC*)

k	n	$m_{pseudoóptimo}$	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	96	7.629	0.729	0.695	-0.034
	1440	120	24.269	0.774	0.749	-0.025
	1920	160	55.585	0.801	0.780	-0.021
	2400	150	106.251	0.818	0.803	-0.015
	2880	180	180.549	0.832	0.820	-0.012
	3360	168	283.310	0.842	0.832	-0.010
	3840	192	418.952	0.850	0.843	-0.007
	4320	216	590.984	0.860	0.852	-0.008
9	960	64	4.015	0.616	0.571	-0.045
	1440	80	12.172	0.686	0.645	-0.041
	1920	80	27.462	0.721	0.687	-0.034
	2400	100	51.656	0.748	0.721	-0.027
	2880	120	86.934	0.768	0.745	-0.023
	3360	140	135.345	0.783	0.763	-0.020
	3840	128	199.679	0.793	0.777	-0.016
	4320	144	280.205	0.806	0.790	-0.016
16	960	60	2.526	0.551	0.479	-0.072
	1440	72	7.604	0.618	0.566	-0.052
	1920	80	16.673	0.668	0.621	-0.047
	2400	100	31.127	0.698	0.657	-0.041
	2880	90	52.016	0.722	0.686	-0.036
	3360	105	80.373	0.742	0.709	-0.033
	3840	120	117.717	0.756	0.727	-0.029
	4320	120	165.022	0.770	0.742	-0.028

A.2.3.2 Bloques (*PBB*)

k	*n*	$T_{PARALELO\_EMPÍRICO}$	$E_{EMPÍRICA}$ (E_E)	$E_{TEÓRICA}$ (E_T)	E_T-E_E
4	960	11.538	0.482	0.471	-0.011
	1440	38.510	0.488	0.481	-0.007
	1920	90.651	0.491	0.486	-0.005
	2400	176.278	0.493	0.489	-0.004
	2880	303.722	0.495	0.491	-0.004
	3360	481.316	0.496	0.492	-0.004
	3840	717.678	0.496	0.493	-0.003
	4320	1020.770	0.498	0.494	-0.004
9	960	6.305	0.392	0.374	-0.018
	1440	20.568	0.406	0.393	-0.013
	1920	48.006	0.412	0.402	-0.010
	2400	92.892	0.416	0.407	-0.009
	2880	159.512	0.419	0.411	-0.008
	3360	252.317	0.420	0.414	-0.006
	3840	375.369	0.422	0.415	-0.007
	4320	533.255	0.424	0.417	-0.007
16	960	4.037	0.345	0.320	-0.025
	1440	12.873	0.365	0.347	-0.018
	1920	29.563	0.377	0.360	-0.017
	2400	56.930	0.382	0.368	-0.014
	2880	97.480	0.385	0.373	-0.012
	3360	153.817	0.388	0.377	-0.011
	3840	228.382	0.390	0.380	-0.010
	4320	323.984	0.392	0.382	-0.010

Anexo B

Eficiencia Escalada

A través del presente anexo vamos a exponer los datos que han permitido generar las gráficas de la eficiencia escalada que figuran a lo largo de los capítulos 4 y 5. En estas gráficas se refleja la evolución de la eficiencia a medida que el número de procesadores (*k*) y el número de operaciones básicas requeridas por el algoritmo secuencial (*W*) crecen en la misma cuantía, concretamente hemos optado por multiplicar *W* por 2, al igual que el número de procesadores, lo que equivale a decir que el orden de la matriz es multiplicado por la raíz cúbica de 2. De ese modo, hemos calculado la eficiencia escalada teórica desde *n*=960 y *k*=1 hasta *n*=3841 y *k*=64 para cada distribución de los datos analizada así como para cada entorno de experimentación. Los datos calculados se muestran mediante tablas, estando cada una de ellas compuesta por seis columnas donde se recoge la siguiente información:

- k: número de procesadores,

- n: orden de la matriz,

- W: número de operaciones básicas requeridas por el algoritmo secuencial conocido más rápido para solucionar el problema en un único procesador. En el caso de la eliminación de Neville corresponde a la siguiente expresión $\frac{2}{3}n^3$,

- $T_{SECUENCIAL\_TEÓRICO}$: estimación del tiempo de ejecución, en segundos, del algoritmo de Neville secuencial,

- $T_{PARALELO\_TEÓRICO}$: estimación del tiempo de ejecución total paralelo, en segundos, de la distribución mostrada y

- $E_{ESCALADA\_TEÓRICA}$: estimación de la eficiencia escalada, calculada de acuerdo a la siguiente expresión

B.1 PARTICIONES UNIDIMENSIONALES

B.1.1 BLOQUES DE FILAS CÍCLICOS (*BFC*)

IBM SP2

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	9.44	9.43	1.00
2	1210	1.18×10^{9}	18.88	10.39	0.91
4	1524	2.36×10^{9}	37.76	10.91	0.86
8	1920	4.72×10^{9}	75.54	11.45	0.82
16	2420	9.44×10^{9}	151.11	12.09	0.78
32	3049	1.89×10^{10}	302.27	12.85	0.73
64	3841	3.78×10^{10}	604.66	13.79	0.68

CLÚSTER

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	22.41	22.18	1.00
2	1210	1.18×10^{9}	44.84	27.69	0.80
4	1524	2.36×10^{9}	89.69	29.87	0.74
8	1920	4.72×10^{9}	179.41	32.67	0.68
16	2420	9.44×10^{9}	358.88	35.91	0.62
32	3049	1.89×10^{10}	717.90	39.63	0.56
64	3841	3.78×10^{10}	1436.07	43.60	0.51

B.1.2 BLOQUES DE FILAS (*BF*)

IBM SP2

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	9.44	9.43	1.00
2	1210	1.18×10^{9}	18.88	13.07	0.72
4	1524	2.36×10^{9}	37.76	14.03	0.67
8	1920	4.72×10^{9}	75.54	14.35	0.66
16	2420	9.44×10^{9}	151.11	14.57	0.65
32	3049	1.89×10^{10}	302.27	14.82	0.64
64	3841	3.78×10^{10}	604.66	15.17	0.62

CLÚSTER

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	22.41	22.40	1.00
2	1210	1.18×10^{9}	44.84	31.62	0.71
4	1524	2.36×10^{9}	89.69	34.44	0.65
8	1920	4.72×10^{9}	179.41	35.95	0.62
16	2420	9.44×10^{9}	358.88	37.61	0.60
32	3049	1.89×10^{10}	717.90	39.95	0.56
64	3841	3.78×10^{10}	1436.07	43.57	0.51

B.1.3 CÍCLICO POR FILAS (*CF*)

IBM SP2

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	9.44	9.43	1.00
2	1210	1.18×10^{9}	18.88	44.94	0.21
4	1524	2.36×10^{9}	37.76	44.94	0.21
8	1920	4.72×10^{9}	75.54	45.01	0.21
16	2420	9.44×10^{9}	151.11	45.16	0.21
32	3049	1.89×10^{10}	302.27	45.31	0.21
64	3841	3.78×10^{10}	604.66	45.53	0.21

CLÚSTER

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	22.41	22.40	1.00
2	1210	1.18×10^{9}	44.84	406.62	0.06
4	1524	2.36×10^{9}	89.69	406.60	0.06
8	1920	4.72×10^{9}	179.41	407.07	0.06
16	2420	9.44×10^{9}	358.88	408.37	0.05
32	3049	1.89×10^{10}	717.90	409.63	0.05
64	3841	3.78×10^{10}	1436.07	411.42	0.05

B.1.4 BLOQUES DE COLUMNAS (*BC*)

IBM SP2

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	9.44	9.43	1.00
2	1210	1.18×10^{9}	18.88	13.08	0.72
4	1524	2.36×10^{9}	37.76	14.21	0.66
8	1920	4.72×10^{9}	75.54	14.91	0.63
16	2420	9.44×10^{9}	151.11	15.88	0.59
32	3049	1.89×10^{10}	302.27	17.48	0.54
64	3841	3.78×10^{10}	604.66	20.29	0.47

CLÚSTER

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	22.41	22.40	1.00
2	1210	1.18×10^{9}	44.84	31.62	0.71
4	1524	2.36×10^{9}	89.69	35.98	0.62
8	1920	4.72×10^{9}	179.41	41.01	0.55
16	2420	9.44×10^{9}	358.88	49.67	0.45
32	3049	1.89×10^{10}	717.90	65.29	0.34
64	3841	3.78×10^{10}	1436.07	93.42	0.24

B.1.5 CÍCLICO POR COLUMNAS (*CC*)

IBM SP2

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	9.44	9.43	1.00
2	1210	1.18×10^{9}	18.88	9.58	0.98
4	1524	2.36×10^{9}	37.76	9.84	0.96
8	1920	4.72×10^{9}	75.54	10.33	0.91
16	2420	9.44×10^{9}	151.11	11.24	0.84
32	3049	1.89×10^{10}	302.27	12.85	0.74
64	3841	3.78×10^{10}	604.66	15.70	0.60

CLÚSTER

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	22.41	22.40	1.00
2	1210	1.18×10^{9}	44.84	23.51	0.95
4	1524	2.36×10^{9}	89.69	25.76	0.87
8	1920	4.72×10^{9}	179.41	30.22	0.74
16	2420	9.44×10^{9}	358.88	38.71	0.58
32	3049	1.89×10^{10}	717.90	54.32	0.41
64	3841	3.78×10^{10}	1436.07	82.53	0.27

B.2 PARTICIONES BIDIMENSIONALES

B.2.1 BLOQUES CÍCLICOS (*PBBC*)

IBM SP2

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	9.44	9.43	1.00
2	1210	1.18×10^{9}	18.88	10.37	0.91
4	1524	2.36×10^{9}	37.76	10.78	0.87
8	1920	4.72×10^{9}	75.54	11.15	0.85
16	2420	9.44×10^{9}	151.11	11.51	0.82
32	3049	1.89×10^{10}	302.27	11.89	0.79
64	3841	3.78×10^{10}	604.66	12.31	0.77

CLÚSTER

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	22.41	22.17	1.00
2	1210	1.18×10^{9}	44.84	27.22	0.81
4	1524	2.36×10^{9}	89.69	29.65	0.75
8	1920	4.72×10^{9}	179.41	31.81	0.70
16	2420	9.44×10^{9}	358.88	34.01	0.65
32	3049	1.89×10^{10}	717.90	36.30	0.61
64	3841	3.78×10^{10}	1436.07	38.83	0.57

B.2.2 BLOQUES (*PBB*)

IBM SP2

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	9.44	9.43	1.00
2	1210	1.18×10^{9}	18.88	15.05	0.63
4	1524	2.36×10^{9}	37.76	19.07	0.49
8	1920	4.72×10^{9}	75.54	22.00	0.43
16	2420	9.44×10^{9}	151.11	24.20	0.39
32	3049	1.89×10^{10}	302.27	25.86	0.36
64	3841	3.78×10^{10}	604.66	27.19	0.35

CLÚSTER

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	22.41	22.40	1.00
2	1210	1.18×10^{9}	44.84	36.22	0.62
4	1524	2.36×10^{9}	89.69	46.49	0.48
8	1920	4.72×10^{9}	179.41	54.43	0.41
16	2420	9.44×10^{9}	358.88	60.90	0.37
32	3049	1.89×10^{10}	717.90	66.36	0.34
64	3841	3.78×10^{10}	1436.07	71.32	0.31

B.2.3 CÍCLICAS (*PBC*)

IBM SP2

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	9.44	9.43	1.00
2	1210	1.18×10^{9}	18.88	44.99	0.21
4	1524	2.36×10^{9}	37.76	45.04	0.21
8	1920	4.72×10^{9}	75.54	45.16	0.21
16	2420	9.44×10^{9}	151.11	45.37	0.21
32	3049	1.89×10^{10}	302.27	45.57	0.21
64	3841	3.78×10^{10}	604.66	45.80	0.21

CLÚSTER

k	*n*	*W*	$T_{SECUENCIAL\_TEÓRICO}$	$T_{PARALELO\_TEÓRICO}$	$E_{ESCALADA\_TEÓRICA}$
1	960	5.90×10^{8}	22.41	22.40	1.00
2	1210	1.18×10^{9}	44.84	406.97	0.06
4	1524	2.36×10^{9}	89.69	407.30	0.05
8	1920	4.72×10^{9}	179.41	408.08	0.05
16	2420	9.44×10^{9}	358.88	409.59	0.05
32	3049	1.89×10^{10}	717.90	410.84	0.05
64	3841	3.78×10^{10}	1436.07	412.21	0.05

Referencias Bibliográficas

[Agerwala *et al* 95] Agerwala, T.; Martin, J.L.; Mirza, J.H.; Sadler, D.C.; Dias, D.M.; Snir, M., SP2 System Architecture, IBM Systems Journal: Scalable Parallel Computing, 34(2), pp. 152-184, 1995.

[Abascal *et al* 04] Abascal, P.; Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J., Analyzing the Efficiency of Block-Cyclic Checkerboard Partitioning in Neville Elimination, Lecture Notes in Computer Science, Vol. 3019, pp. 963-968, 2004.

[Almasi *et al* 94] Almasi, G.S.; Gottlieb, A., Highly Parallel Computing, The Benjamin/Cummings Publishing Company Inc., second edition, 1994.

[Alonso 95] Alonso, P., Eliminación de Neville y Análisis del Error, Tesis Doctoral, Departamento de Matemáticas de la Universidad de Oviedo, 1995.

[Alonso *et al* 97] Alonso, P.; Gasca, M.; Peña, J.M., Backward Error Analysis of Neville Elimination, Applied Numerical Mathematics Vol. 23, pp. 193-204, 1997.

[Alonso *et al* 98] Alonso, P.; Gasca, M.; Peña, J.M., Estudio del Error Progresivo en la Eliminación de Neville, Revista de la Real Academia de Ciencias Exactas, Físicas y Naturales de Madrid, Vol. 92, nº 1, pp. 1-8, 1998.

[Alonso *et al* 99a] Alonso, P.; Peña, J.M., Development of Block and Partitioned Neville Elimination, C.R. Acad. Sci. Paris Sér. I Math. 329, nº 12, pp. 1091-1096,1999.

[Alonso *et al* 99b] Alonso, P.; Cortina, R.; Ranilla, J., Block-Striped Partitioning and Neville Elimination, Lecture Notes in Computer Science, Vol. 1685, pp. 1073-1077, 1999.

[Alonso *et al* 01] Alonso, P.; Cortina, R.; Hernández, V.; Ranilla, J., A Study of the Performance of Neville Elimination using Two Kinds of Partitioning Techniques, Linear Algebra and its Applications, Vol. 332-334, pp. 111-117, 2001.

[Alonso *et al* 02] Alonso, P.; Cortina, R.; Díaz, I.; Hernández, V.; Ranilla, J., A Columnwise Block Striping in Neville Elimination, Lecture Notes in Computer Science, Vol. 2328, pp. 379-386, 2002.

[Alonso *et al* 03] Alonso, P.; Cortina, R.; Díaz, I.; Hernández, V.; Ranilla, J., A Simple Cost-Optimal Parallel Algorithm to Solve Linear Equation Systems, INFORMATION: An International Interdisciplinary Journal, 6(3), pp. 297-304, 2003.

[Alonso *et al* 04] Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J., A Study of the Efficiency using Checkerboard Partitioning, Linear Algebra and its Applications, Vol. 393, pp. 3-14, 2004.

[Alonso *et al* 06] Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J., Analyzing Scalability of Neville Elimination, Journal of Mathematical Chemistry, Vol. 40(1), pp. 49-61, 2006.

[Alonso *et al* 08a] Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J., Scalability of Neville Elimination using Checkerboard Partitioning, International Journal of Computer Mathematics, Vol. 85(3&4), pp. 309-317, 2008.

[Alonso *et al* 08b] Alonso, P.; Cortina, R.; Díaz, I.; Ranilla, J., Blocking Neville Elimination Algorithm for Exploiting Cache Memories, Applied Mathematics and Computation, DOI:10.1016/j.amc.2008.06.023, 2008.

[Amdahl 67] Amdahl, G.M., Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, In AIFPS Conference Proceedings 30, pp. 483-485, 1967.

[Ando 87] Ando, T., Totally Positive Matrices, Linear Algebra and its Applications, Vol. 9, pp. 165-219, 1987.

[Bapat *et al* 97] Bapat, R.B.; Raghavann, T.E.S., Nonnegative Matrices and Applications, Cambridge University Press, New York, 1997.

[Berman *et al* 94] Berman, A.; Plemmons, R.J., Nonnegative Matrices in the Mathematical Sciences, SIAM, Philadelphia, 1994.

[Bomans *et al* 90] Bomans, L.; Roose, D.; Hempel, R., The Argonne/Gmd Macros in Fortran for Portable Parallel, Programming and their Implementation on the Intel Ipsc/2. Parallel Computing, 15 (1-3), pp. 119-132, 1990.

[Boor *et al* 77] de Boor, C.; Pinkus, A., Backward Error Analysis for Totally Positive Matrices Linear Systems, Numer. Math. Vol. 27, pp. 485-490, 1977.

[Brenti 95] Brenti, F., Combinatorics and Total Positivity, J. Combin. Theory Ser. A, 71 , pp. 175– 218, 1995.

[Brenti 96] Brenti, F., The Applications of Total Positivity to Combinatorics, and Conversely, en [Gasca *et al* 96a], pp. 451-473.

[Calkin *et al* 94] Calkin, R.; Hempel, R.; Hoppe, H.C.; Wypior, P., Portable Programming with the Parmacs Message-Passing Library, Parallel Computing 20 (4), pp. 615-632, 1994.

[Carnicer *et al* 93] Carnicer, J.M.; Peña, J.M., Shape Preserving Representations and Optimality of the Bernstein Basis, Adv. Comput. Math. 1, pp. 173-196, 1993.

[Carnicer *et al* 94] Carnicer, J.M.; Peña, J.M., Totally Positive Bases for Shape Preserving Curve Design and Optimality of B-Splines, Comput. Aided Geom. Design, 11, pp. 633-654, 1994.

[Carnicer *et al* 96] Carnicer, J.M.; Peña, J.M., Total Positivity and Optimal Bases, en [Gasca *et al* 96a], pp. 133-155.

[Cortes *et al* 07] Cortes, V.; Peña, J. M., Sign Regular Matrices and Neville Elimination, Linear Algebra Appl. 421(1), pp. 53-62, 2007.

[Cryer 73] Cryer, C.W., The LU-Factorizations of Totally Positive Matrices, Linear Algebra and its Applications, Vol 7, pp. 83-92, 1973.

[Curtis *et al* 98] Curtis, E.; Ingerman, D.V.; Morrow, J., Circular Planar Graphs and Resistor Networks, Linear Algebra and its Applications, Vol. 283, pp. 115-150, 1998.

[Chapoton *et al* 02] Chapoton, F.; Fomin, S.; Zelevinsky, A., Polytopal Realizations of Generalized Associahedra, Bulletin Canadien de Mathématiques, Vol 45, nº 4, pp. 537-566, 2002.

[Davis *et al* 88] Davis, Timothy A.; Davidson, Edward S., Pairwise Reduction for the Direct, Parallel Solution of Sparse, Unsymmetric Sets of Linear Equations. IEEE Trans. Comput. 37 (12), pp. 1648-1654, 1988 .

[Delgado *et al* 07] Delgado, J.; Peña, J.M., A Corner Cutting Algorithm for Evaluating Rational Bézier Surfaces and the Optimal Stability of the Basis, SIAM J. Sci. Comput. 29(4), pp. 1668-1682, 2007.

[Demmel 97] Demmel, J., Applied Numerical Linear Algebra, SIAM, 1997.

[Demmel *et al* 99] Demmel, J., Gu, M.; Eisenstat, S.; Slapnicar, I.; Veselic, K.; Drmac, Z., Computing the Singular Value Decomposition with High Relative Accuracy. Linear Algebra Appl. 299(1-3), pp. 21-80, 1999.

[Demmel *et al* 04] Demmel, J.; Koev, P., Accurate SVDs of Weakly Diagonally Dominant M-Matrices, Numer. Math. 98(1), pp. 99-104, 2004.

[Demmel *et al* 05] Demmel, J.; Koev, P., The Accurate and Efficient Solution of a Totally Positive Generalized Vandermonde Linear System, SIAM J. Matrix Anal. Appl. 27, pp. 142-152, 2005.

[Dongarra *et al* 93] Dongarra, J.J.; Hempel, R.; Hey, A.J.G.; Walter, D.W., A Proposal for a User-Level Message Passing Interface in a Distributed Memory Environment. Technical report TM-12231, Oak Ridge National Laboratory, 1993.

[Dongarra et al 94] Dongarra, J.; Sunderam, V.S.; Geist, G.A.; Manchek, R., The PVM Concurrent Computing System: Evolution, Experiences and Trends, Parallel Computing 20(4), pp. 531-545, 1994.

[Dopico *et al* 06] Dopico, F. M.; Koev, P., Accurate Symmetric Rank Revealing and Eigen Decompositions of Symmetric Structured Matrices, SIAM J. Matrix Anal. Appl. 28(4), pp. 1126-1156, 2006.

[Doss *et al* 93] Doss, N.; Gropp, W.; Lusk, E.; Skjellum, A., A Model Implementation of MPI. Technical Report, Argonne National Laboratory, 1993.

[Edrei 52] Edrei, A., On the Generating Function of Totally Positive Sequences, II, J. d´Anal. Math 2, pp. 104-109, 1952.

[Fallat *et al* 00] Fallat, S.M; Johnson, C.R.; Smith, R.L., The General Totally Positive Matrix Completion Problem with Few Unspecified Entries, Electronic J. Linear Algebra 7, pp. 1-20, 2000.

[Fallat 01] Fallat, S.M., Bidiagonal Factorizations of Totally Nonnegative Matrices, Amer. Math. Monthly, 108, pp. 697–712, 2001.

[Fekete *et al* 12] Fekete, M.; Pólya, G., Über ein Problem von Laguerre, Rend. C. M. Palermo 34, pp. 89-120, 1912.

[Flynn 66] Flynn, M. J., Very High-Speed Computing Systems, Proceedings of the IEEE Vol. 54, Num. 12, pp. 1901-1909, 1966.

[Fomin *et al* 00] Fomin, S.; Zelevinsky, A., Total Positivity: Test and Parametrizations, Mathematical Intelligencer 22, pp. 23-33, 2000.

[Fomin 01] Fomin, S., Loop-Erased Walks and Total Positivity, Transactions of the American Mathematical Society, Volume 353, Number 9, pp. 3563-3583, 2001.

[Gallivan *et al* 90] Gallivan, K. A.; Plemmons, R. J.; Sameh, A. H., Parallel Algorithms for Dense Linear Algebra Computations. SIAM Rev. 32(1), pp. 54-135, 1990.

[Gantmacher *et al* 35] Gantmacher, F.R.; Krein, M.G., Sur les Matrices Oscillatoires, C.R. Acad. Sci. París, 201, pp. 577-579, 1935.

[Gantmacher *et al* 37] Gantmacher, F.R.; Krein, M.G., Sur les Matrices Complètement Non Negatives et Oscillatoires, Compositio Math. 4, pp. 445-476, 1937.

[Gasca *et al* 87] Gasca, M.; Mühlbach, G., Generalized Schur-Complements and a Test for Total Positivity, Appl. Numer. Math., 3, pp 215-232, 1987.

[Gasca *et al* 92a] Gasca, M.; Peña, J.M., Total Positivity and Neville Elimination, Linear Algebra and its Applications, Vol. 165, pp. 25-44, 1992.

[Gasca *et al* 92b] Gasca, M.; Michelli, C.A.; Peña, J.M., Almost Strictly Totally Positive Matrices, Numer. Algorithms 2, pp. 225-236, 1992.

[Gasca *et al* 93] Gasca, M.; Peña, J.M., Total positivity, QR-factorization and Neville Elimination, SIAM J. Matrix Anal. Appl. 14, pp. 1132-1140, 1993.

[Gasca *et al* 94a] Gasca, M.; Peña, J.M., A Matricial Description of Neville Elimination with Applications to Total Positivity, Linear Algebra and its Applications, Vol. 202, pp. 33-54, 1994.

[Gasca *et al* 94b] Gasca, M.; Peña, J.M., Corner Cutting Algorithms and Totally Positive Matrices, en P.J. Laurent, A. Le Mehaute, L.L Schumaker (Eds.), Curves and Surfaces II, 177-184, A.K. Peters, Wellesley, M.A., 1994.

[Gasca *et al* 95] Gasca, M.; Peña, J.M., Neville Elimination and Approximation Theory, En Approximation Theory Wavelets and Applications, ed. S.P. Singh Kluwer Academic Publishers, pp. 131-151, 1995.

[Gasca *et al* 96a] Gasca, M.; Michelli, C.A. (Eds.), Total Positivity and its Applications, Mathematics and its Applications, vol. 359, Kluwer Acad. Publ., Dordrecht, 1996.

[Gasca *et al* 96b] Gasca, M.; Peña, J.M., On Factorizations of Totally Positive Matrices, Total Positivity and its Applications, Gasca, M., Michelli, C.A. (Eds.), Kluwer Acad. Publ., pp. 109-132, 1996.

[Gasca *et al* 06] Gasca, M.; Peña, J. M., Characterizations and Decompositions of Almost Strictly Positive Matrices. SIAM J. Matrix Anal. Appl. 28(1), pp. 1-8, 2006.

[Gasca *et al* 00] Gasca, M.; Mühlbach, G., Elimination Techniques: from Extrapolation to Totally Positive Matrices and CAGD, Journal of Computational and Applied Mathematics 122, pp. 37-50, 2000.

[Geist *et al* 91] Geist, G.A.; Heath, M.T.; Peyton, B.W.; Worley, P.H., A User´s Guide to PICL: a Portable Instrumented Communication Library. Technical Report TM-11616, Oak Ridge National Laboratory, 1991.

[Geist *et al* 94] Geist, G.A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Manchek, R.; Sunderam, V., PVM: A User´s Guide and Tutorial for Networked Parallel Computing, MIT Press, 1994.

[Gemignani 08] Gemignani, L., Neville Elimination for Rank-Structured Matrices, Linear Algebra Appl. 428(4), pp. 978-991, 2008.

[Golub *et al* 96] Golub, G.H.; Van Loan, C.F., Matrix Computations, Johns Hopkins University Press, 1996.

[Grama *et al* 93] Grama, A., Gupta, A. Kumar V., Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures, IEEE Parallel and Distributed Technology; 1(3), pp. 12-21, 1993.

[Grama *et al* 00] Grama, A., Gupta, A., Han, E., Kumar V., Parallel Algorithm Scalability Issues in Petaflops Architectures, Ultrascale Computing, 2000.

[Grama *et al* 03] Grama, A., Gupta, A., Karypis, G., Kumar, V., Introduction to Parallel Computing, Pearson Education Limited, 2003.

[Gropp *et al* 96a] Gropp, W.D.; Lusk, E., User´s Guide for MPICH, a Portable Implementation of MPI, Mathematics and Computer Science Division, Argonne National Laboratory, ANL-96/6, 1996.

[Gropp *et al* 96b] Gropp, W.D.; Lusk, E.; Doss, N.; Skjellnum, A., A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard, Parallel Computing 22(6), pp. 789-828, 1996.

[Gupta *et al* 94] Gupta, A.; Kumar, V., A Scalable Parallel Algorithm for Sparse Matrix Factorizations, Technical Report 94-19, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994. Una version corta aparece Supercomputing´94 Proceedings. TR disponible in users/kumar at anonymous FTP site ftp.cs.umn.edu.

[Gupta 95] Gupta, A., Analysis and Design of Scalable Parallel Algorithms for Scientific Computing, University of Minnesota, 1995.

[Gupta *et al* 95] Gupta, A.; Kumar, V.; Sameh, A., Performance and Scalability of Preconditioned Conjugated Gradient Methods on Parallel Computers, IEEE Transactions on Parallel and Distributed Systems 6(5), pp. 455-469, 1995.

[Gustafson 88] Gustafson, J.L., Reevaluating Amdahl´s Law, Communications of ACM, 33(5), pp. 532-533, 1988.

[Higham 02] Higham, N.J., Accuracy and Stability of Numerical Algorithms, SIAM, 2002.

[Hockney *et al* 94] Hockney, R.; Berry, M., Public International Benchmarks for Parallel Computers: Parkbench Committee Report, Scientific Programming, 3(2), pp. 101-146, 1994.

[Hwang 93] Hwang, K., Advanced Computer Architecture: Parallelism, Scalability, Programmability. McGraw-Hill, New York, NY, 1993.

[Karlin 68] Karlin, S., Total Positivity, Vol. I, Stanford University Press, Stanford, 1968.

[Koev 07] Koev, P., Accurate Computations with Totally Nonnegative Matrices. SIAM J. Matrix Anal. Appl. 29(3), pp. 731-751, 2007.

[Kumar *et al* 87] Kumar, V., Rao, V.N., G., Parallel Depth-First Search, part II: Analysis, International Journal of Parallel Programming Vol. 16, Num. 6, pp. 501-519, 1987.

[Kumar *et al* 91] Kumar, V.; Singh, V., Scalability of Parallel Algorithms for the All-Pairs Shortest Path Problem, Journal of Parallel and Distributed Computing, 13(2), pp. 124-138, 1991. Una version corta aparece en los Proceedings of the International Conference on Parallel Processing, 1990.

[Li 96] Li, X. S., Sparse Gaussian Elimination on High Performance Computers, Tesis Doctoral, University of California at Berkeley, 1996.

[Li *et al* 99] Li, K.; Pan, Y.; Shen, H., Zheng, S.Q., A Study of Average-Case Speedup and Scalability of Parallel Computations on Static Networks, Mathematical and Computer Modelling 29, pp. 83-94, 1999.

[Mainar *et al* 99] Mainar, E.; Peña, J.M., Corner Cutting Algorithms Associated with Optimal Shape Preserving Representations, Comput. Aided Geom. Design 16, nº 9, 883-906, 1999.

[Martín *et al* 96] Martín, I.; Tirado, F.; Vázquez, L., Some Aspects about the Scalability of Scientific Applications on Parallel Architectures, Parallel Computing 22, pp. 1169-1195, 1996.

[Martín *et al* 97] Martín, I.; Tirado, F., Relationships Between Efficiency and Execution Time of Full Multigrid Methods on Parallel Computers, IEEE Transactions on Parallel and Distributed Systems, Vol. 8, Num. 6, 1997.

[Moler87] C. Moler, Another Look at Amdahl´s Law, Technical Report TN-02-0587-0288, Intel Scientific Computers,1987.

[MPI Forum 95] Message Passing Interface Forum. MPI: A Message Passing Interface Standard, Junio 1995.

[Mühlbach *et al* 87] Mühlbach, G.; Gasca, M., A Test for Strict Total Positivity via Neville Elimination, F. Uhlig, R. Grone (Eds.), Current Trends in Matrix Theory, North-Holland, Amsterdam, pp. 225-232, 1987.

[Pacheco 96] Pacheco, P., Parallel Programming with MPI, Morgan Kauffmann, 1996.

[Parasoft] Parasoft Corporation, Pasadena, CA, USA. Express User´s Guide, 1992.

[Patel *et al* 00] Patel, J.N.; Khokhar, A.A.; Jamieson L.H., Scalability of 2-D Wavelet Transform Algorithms: Analytical and Experimental Results on MPPs, IEEE Transactions on Signal Processing, Vol. 48, Num. 12, 2000.

[Peña 95] Peña, J.M., Matrices with Sign Consistency of a Given Order, SIAM Journal on Matrix Analysis and Applications, Vol. 16, Num. 4, pp. 1100-1106, 1995.

[Peña 98] Peña, J.M., On the Relationship between Graphs and Totally Positive Matrices, SIAM Journal on Matrix Analysis and Applications, Vol. 19, Num. 2, pp. 369-377, 1998.

[Peña 99] Peña, J. M.; Shape Preserving Representations in Computer Aided-Geometric Design, Nova Science Publishers, New York, 1999.

[Peña 01] Peña, J.M., Determinantal Criteria for Total Positivity, Proceedings of the Eighth Conference of the International Linear Algebra Society (Barcelona, 1999), Linear Algebra and its Applications, Vol. 332-334, 2001.

[Peña 03] Peña, J.M., Simultaneous Backward Stability of Gauss and Gauss-Jordan Elimination, Numer. Linear Algebra Appl. 10, nº 4, 317-321, 2003.

[Petitet 96] Petitet, A. Algorithmic Redistribution Methods for Block Cyclic Decompositions, Tesis Doctoral, University of Tennessee, 1996.

[Petitet *et al* 99] Petitet, A.P., Dongarra, J.J., Algorithmic Redistribution Methods for Block Cyclic Decompositions, IEEE Transactions on Parallel and Distributed Computing, Volume 10(12), pp. 201-220, 1999.

[Pinkus 96] Pinkus, A., Spectral Properties of Totally Positive Kernels and Matrices, en [Gasca *et al* 96a], pp. 477-511.

[Prieto *et al* 03] Prieto, M.; Montero, R.S.; Llorente, I.M.; Tirado, F., A Parallel Multigrid Solver for Viscous Flows on Anisotropic Structured Grids, Parallel Computing 29, pp. 907-923, 2003.

[Ranka *et al* 90] Ranka, S.; Sahni, S., Hypercube Algorithms for Image Processing and Pattern Recognition, Springer-Verlag, New York, NY, 1990.

[Rathe *et al* 99] Rathe, U.W.; Sanders, P.; Knight, P.L., A Case Study in Scalability: An ADI Method for Two-Dimensional Time-Dependent Dirac Equation, Parallel Computing 25, pp. 525-533, 1999.

[Schoenberg 30] Schoenberg, I. J., Über Varationsverminderende lineare Transformationen, Math. Z. 32, pp. 321-328, 1930.

[Schoenberg 88] Schoenberg, I. J., Selected Papers, Vol. 2, Birkhäuser, 1988.

[Singh *et al* 91] Singh, V.; Kumar, V.; Agha, G.; Tomlinson, C., Scalability of Parallel Sorting on Mesh Multicomputers, International Journal of Parallel Programming, 10(2), 1991.

[Skjellum *et al* 90] Skjellum, A.; Leung, A., Zipcode: A Portable Multicomputer Communication Library atop the Reactive Kernel, In D. W. Walker and Q. F. Stout editors, Proceedings of the Fifth Distributed Memory Concurrent Computing Conference, pp. 767-776, IEEE Press, 1990.

[Skjellum *et al* 92] Skjellum, A.; Smith, S.; Still, C.; Leung, A.; Morari, M., The Zipcode Message Passing System, Technical Report, Lawrence Livermore National Laboratory, 1992.

[Skjellum *et al* 94] Skjellum, A.; Smith, S.G.; Doss, N.E.; Leung, A.P.; Morari, M., The Design and Evolution of Zipcode, Parallel Computing 20(4), pp. 656-596, 1994.

[Snell *et al* 96] Snell, Q.O.; Mikler, A.; Gustafson, J.L., NetPIPE: A Network Protocol Independent Performance Evaluator, Proceedings of the IASTED International Conference on Intelligent Information Management and Systems, June 1996.

[Sorensen 85] Sorensen, D. C., Analysis of Pairwise Pivoting in Gaussian Elimination. IEEE Trans. Comput. 34 (3), pp 274-278, 1985.

[Stanley 89] Stanley, R.P., Log-Concave and Unimodal Sequences in Algebra, Combinatorics and Geometry, Ann., New York Acad. Sci. 576, pp. 500-534, 1989.

[Stembridge 91] Stembridge, J.R., Immanants of Totally Positive Matrices are Nonnegative, Bull. London Math. Soc. 23, pp. 422-428, 1991.

[Stummel 85] Stummel, F., Forward Error Analysis of Gaussian Elimination, Part I: Error and residual estimates, Numer. Math. 46, pp. 365-395, 1985.

[Sturmfels 88] Sturmfels, B., Totally Positive Matrices and Cyclic Polytopes, Linear Algebra and its Applications, Vol. 107, pp. 275-281, 1988.

[Sun *et al* 94] Sun, X-H; Rover, D.T., Scalability of Parallel Algorithm-Machine Combinations, IEEE Transactions on Parallel and Distributed Systems, 5(6), pp. 599-613, 1994.

[Tiskin 07] Tiskin, A., Communication-Efficient Parallel Generic Pairwise Elimination, Future Generation Computer Systems 23, pp. 179-188, 2007.

[Wilkinson 63] Wilkinson, J.H., Rounding Errors in Algebraic Processes, Notes on Applied Science 32, Her Majesty´s Stationery Office, 1963.

[Wilkinson 65] Wilkinson, J.H., The Algebraic Eigenvalue Problem, Oxford University Press, 1965.