BIBLIOTECA VIRTUAL
MIGUEL DE CERVANTES

Universidad de Alicante

Análisis de eficiencia de algoritmos BSP para la resolución de sistemas lineales tridiagonales

Antonio Zamora Gómez

Tesis de Doctorado

Facultad: Escuela Politécnica Superior

Director: Dr. Joan Josep Climent Coloma

2000

Universidad de Alicante

Departamento de Ciencia de la Computación e Inteligencia Artificial

Análisis de eficiencia de algoritmos BSP para la resolución de sistemas lineales tridiagonales

TESIS DOCTORAL

Presentada por:
Antonio Zamora Gómez

Dirigida por:
Joan Josep Climent Coloma

Universidad de Alicante

Departamento de Ciencia de la Computación e Inteligencia Artificial

Análisis de eficiencia de algoritmos BSP para la resolución de sistemas lineales tridiagonales

Memoria presentada para optar al grado de Doctor Ingeniero en Informática por Antonio Zamora Gómez.

Alicante, 14 de Diciembre de 1999

Don Joan Josep Climent Coloma, Profesor Titular de Universidad del Departamento de Ciencia de la Computación e Inteligencia Artificial de la Universidad de Alicante,

CERTIFICA:

Que la presente memoria *Análisis de eficiencia de algoritmos BSP para la resolución de sistemas lineales tridiagonales*, ha sido realizada bajo su dirección, en el Departamento de Ciencia de la Computación e Inteligencia Artificial de la Universidad de Alicante, por el licenciado Don Antonio Zamora Gómez, y constituye su tesis para optar al grado de Doctor Ingeniero en Informática. Para que conste, en cumplimiento de la legislación vigente, autoriza la presentación de la referida tesis doctoral ante la comisión de Doctorado de la Universidad de Alicante, firmando el presente certificado.

Alicante, 14 de Diciembre de 1999.

Fdo.: Joan Josep Climent Coloma

A Carlos y Alberto, esos diablillos.

A Yolanda, por su amor y ayuda.

A mis padres, por el apoyo que me han dado.

Quiero expresar mi más sincero agradecimiento:

Al profesor Joan Josep Climent Coloma por sus constantes orientaciones científicas, por mostrarme día a día qué significa tener capacidad de trabajo, por su incondicional ayuda sin la cual habría sido imposible la realización de esta memoria y sobre todo por su amistad.

Al profesor Leandro Tortosa, compañero de fatigas, por haber estado siempre ahí, por haber hecho mucho más agradable el trabajo y por haberme brindado su amistad desde el primer día.

A los miembros del Grupo de Computación Paralela de la Universidad de Alicante por el buen ambiente que se respira.

A los técnicos Ginés López, Francisco Martínez y Francisco Maciá por su decisiva y pronta ayuda en la resolución de los problemas surgidos en el uso de las distintas máquinas.

Esta tesis doctoral ha sido subvencionada por el proyecto número PB98-0977 de la Dirección General de Enseñanza Superior e Investigación Científica.

Índice

Índice de figuras **xii**

Índice de tablas **xviii**

Prólogo **xix**

1 Computación Paralela **1**

1.1 Antecedentes de la computación paralela 2

1.2 Modelos de computación paralela 3

1.2.1 El modelo PRAM (1978) 4

1.2.2 El modelo BSP (1990) 5

1.2.3 El modelo LogP (1993) 7

1.2.4 Data-Parallel 8

1.2.5 Modelos de paso de mensajes (1989)(1992) 8

1.3 Plataformas de computación paralela 9

1.4 Medidas de paralelismo 16

1.5 El modelo BSP 17

1.5.1 Los parámetros del modelo BSP 18

1.5.2 El modelo de programación 20

1.5.3 El modelo de coste 24

1.5.4 Librerías BSP disponibles 32

2 Método de las particiones superpuestas 47

2.1 Preliminares 47

2.2 Descripción del método. 48

2.3 Precisión del método 57

2.4 Paralelización del método. 70

2.5 Algoritmos BSP 76

2.6 Resultados numéricos 81

3 Métodos bidireccionales 111

3.1 Factorización LDU de matrices tridiagonales 112

3.2 Método bidireccional para dos procesadores 114

3.2.1 Descripción del método 114

3.2.2 Algoritmo BSP para dos procesadores 122

3.3 Método bidireccional para un número par de procesadores 125

3.3.1 Descripción del método 125

3.3.2 Algoritmos BSP para un número par de procesadores 129

3.4 Resultados numéricos 141

4 Método de Wang **171**

4.1 Descripción del método 171

4.2 Paralelización del método 181

4.3 Algoritmos BSP 187

4.4 Coste computacional de los algoritmos basados en el método de Wang . . . 192

4.5 Resultados numéricos 195

5 Comparación entre métodos **223**

5.1 IBM SP2 225

5.1.1 *Switch* 225

5.1.2 *Ethernet* 229

5.2 *Cluster* de PC's 229

5.3 CRAY T3D 236

5.4 CRAY T3E 245

Conclusiones y líneas futuras **255**

Bibliografía **257**

Índice de figuras

1.1 Computador BSP: modelo arquitectónico. 6

1.2 Multiprocesador con memoria distribuida. 10

1.3 Topología de anillo. 12

1.4 Topología de malla bidimensional. 12

1.5 Topología de malla tridimensional. 13

1.6 Estructura básica de un IBM SP2. 15

1.7 Ejemplos de distintos tipos de patrones de comunicación que definen una h-relación. 19

1.8 Espacio generado por los parámetros p, l y g. 20

1.9 Segmento de un programa BSP. 22

1.10 Fases de una barrera de sincronización organizada en forma de árbol 23

1.11 Curva teórica de g para los valores $n_{\frac{1}{2}} = 12$ y $g_\infty = 0.23\mu s$. 28

1.12 Patrón de comunicación todos a todos para 4 procesadores. 29

1.13 Comparación de los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación trid y de la curva teórica de Miller en un IBM SP2 con 4 procesadores interconectados mediante *switch*. 34

1.14 Comparación de los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación trid y de la curva teórica de Miller en un IBM SP2 con 4 procesadores interconectados mediante *switch*. 35

1.15 Comparación de los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación trid y de la curva teórica de Miller en un IBM SP2 con 2 procesadores interconectados mediante *switch*. 36

1.16 Comparación de los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación trid y de la curva teórica de Miller en un IBM SP2 con 2 procesadores interconectados mediante *switch*. 37

1.17 Comparación de los valores $g(x)$ obtenidos experimentalmente con los patrones de comunicación *local*, trid y *all-to-all* en un IBM SP2 con 4 procesadores interconectados mediante *switch*. La curva teórica de Miller está ajustada al modelo trid. 38

1.18 Comparación de los valores $g(x)$ obtenidos experimentalmente con los patrones de comunicación *local*, trid y *all-to-all* en un IBM SP2 con 4 procesadores interconectados mediante *switch*. La curva teórica de Miller está ajustada al modelo trid. 39

2.1 Partición natural y superpuesta de A y $\boldsymbol{d}$ para $p = 4$. 56

2.2 Obtención de la solución general a partir de las soluciones parciales para $p = 4$. 57

2.3 Valor de m en función de δ para distintos valores de ϵ. 69

2.4 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 2 procesadores interconectados mediante *switch*. 84

2.5 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 4 procesadores interconectados mediante *switch*. 86

2.6 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 6 procesadores interconectados mediante *switch*. 88

2.7 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 2 procesadores interconectados mediante *ethernet*. 90

2.8 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 4 procesadores interconectados mediante *ethernet*. 92

2.9 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 6 procesadores interconectados mediante *ethernet*. 94

2.10 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un *cluster* de PC's con 2 procesadores. 96

2.11 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un *cluster* de PC's con 4 procesadores. 98

2.12 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un *cluster* de PC's con 6 procesadores. 100

2.13 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con *switch*. 102

2.14 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con *ethernet*. 103

2.15 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un *cluster* de PC's. 104

3.1 Cálculo en paralelo de los elementos de D para una matriz con $n = 8$. . . . 118

3.2 Cálculo en paralelo de la solución del sistema (2.4) para una matriz con $n = 8$. 119

3.3 Obtención de la solución general a partir de las soluciones parciales para $p = 8$. 130

3.4 Esquema de comunicación para 8 procesadores del método descrito en el algoritmo 3.2. 132

3.5 Tiempos teóricos y experimentales del algoritmo 3.1 (TW) en un IBM SP2 con 2 procesadores interconectados mediante *switch*. 144

3.6 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con 4 procesadores interconectados mediante *switch*. . . . 146

3.7 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con 6 procesadores interconectados mediante *switch*. . . . 148

3.8 Tiempos teóricos y experimentales del algoritmo 3.1 (TW) en un IBM SP2 con 2 procesadores interconectados mediante *ethernet*. 150

3.9 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con 4 procesadores interconectados mediante *ethernet*. . . 152

3.10 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con 6 procesadores interconectados mediante *ethernet*. . . 154

3.11 Tiempos teóricos y experimentales del algoritmo 3.1 (TW) en un *cluster* de PC's para 2 procesadores. 157

3.12 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un *cluster* de PC's para 4 procesadores. 159

3.13 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un *cluster* de PC's para 6 procesadores. 161

3.14 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con *switch*. . 162

3.15 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con *ethernet*. 163

3.16 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un *cluster* de PC's. . . . 164

4.1 Matriz de coeficientes para el caso $n = 16$. 175

4.2 Transformación de la matriz de coeficientes después de la eliminación de los elementos situados por debajo de la diagonal principal, para el caso $n = 16$. 176

4.3 Transformación de la matriz de coeficientes después de la eliminación los elementos situados por encima de la diagonal principal, para el caso $n = 16$. 177

4.4 Transformación de la matriz de coeficientes después de anular los elementos b_4, b_8 y b_{12}, para el caso $n = 16$. 178

4.5 Transformación de la matriz de coeficientes después de anular los elementos f_{4i+j}, con $i = 1, 2, 3$ y $j = 1, 2, 3, 4$, para el caso $n = 16$. 179

4.6 Transformación de la matriz de coeficientes después de anular los elementos g_{4i+j}, con $i = 0, 1, 2, 3$ y $j = 1, 2, 3$, para el caso $n = 16$. 180

4.7 Transformación de la matriz de coeficientes después de eliminar los nuevos elementos no nulos, para el caso $n = 16$. 181

4.8 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 2 procesadores interconectados mediante *switch*. . . . 199

4.9 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 4 procesadores interconectados mediante *switch*. . . . 201

4.10 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 6 procesadores interconectados mediante *switch*. . . . 203

4.11 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3(WANG2) en un IBM SP2 con 2 procesadores interconectados mediante *ethernet*. . . 205

4.12 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 4 procesadores interconectados mediante *ethernet*. . . 207

4.13 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 6 procesadores interconectados mediante *ethernet*. . . 209

4.14 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un *cluster* de PC's para 2 procesadores. 211

4.15 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un *cluster* de PC's para 4 procesadores. 213

4.16 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un *cluster* de PC's para 6 procesadores. 215

4.17 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con *switch*. . . . 216

4.18 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con *ethernet*. . . 217

4.19 Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un *cluster* de PC's. 218

5.1 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 2 procesadores interconectados mediante *switch*, para $16384 \leq n \leq 524288$. 226

5.2 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 4 procesadores interconectados mediante *switch*, para $16384 \leq n \leq 524288$. 227

5.3 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 6 procesadores interconectados mediante *switch*, para $16128 \leq n \leq 516096$. 228

5.4 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 2 procesadores interconectados mediante *ethernet*, para $16384 \leq n \leq 524288$. 230

5.5 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 4 procesadores interconectados mediante *ethernet*, para $16384 \leq n \leq 524288$. 231

5.6 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 6 procesadores interconectados mediante *ethernet*, para $16128 \leq n \leq 516096$. 232

5.7 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un *cluster* de PC's, para 2 procesadores y $4096 \leq n \leq 65536$. 233

5.8 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un *cluster* de PC's, para 4 procesadores y $4096 \leq n \leq 65536$. 234

5.9 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un *cluster* de PC's, para 6 procesadores y $4032 \leq n \leq 64512$. 235

5.10 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 2 procesadores y $16384 \leq n \leq 524288$. 237

5.11 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 4 procesadores y $16384 \leq n \leq 524288$. 238

5.12 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 8 procesadores y $16384 \leq n \leq 524288$. 239

5.13 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 16 procesadores y $16384 \leq n \leq 524288$. 240

5.14 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 32 procesadores y $16384 \leq n \leq 524288$. 241

5.15 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 64 procesadores y $16384 \leq n \leq 524288$. 242

5.16 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 128 procesadores y $16384 \leq n \leq 524288$. 243

5.17 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 128 procesadores y $16384 \leq n \leq 524288$. 244

5.18 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para 2 procesadores y $16384 \leq n \leq 524288$. 247

5.19 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para 4 procesadores y $16384 \leq n \leq 524288$. 248

5.20 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para 8 procesadores y $16384 \leq n \leq 524288$. 249

5.21 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para 16 procesadores y $16384 \leq n \leq 524288$. 250

5.22 Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para 32 procesadores y $16384 \leq n \leq 524288$. 251

Índice de tablas

1.1 Valores de parámetros BSP. 31

1.2 Megaflops obtenidos para un IBM SP2 y un *cluster* de PC's. 31

1.3 Coste de comunicación en flops de una palabra de 32 bits para 4 procesadores en un IBM SP2 utilizando *switch* y distintos patrones de comunicación. 33

1.4 Valores de g_∞ y $n_{\frac{1}{2}}$ para los patrones *local*, trid y *all-to-all* en un IBM SP2 con 4 procesadores interconectados mediante *switch*. 33

1.5 Funciones de la librería Green BSP. 41

1.6 Núcleo de la librería BSPLib. 43

2.1 Rango de la diagonal dominanza δ dependiendo del máximo error tolerado ϵ y del número de ecuaciones solapadas m. 70

2.2 Valores de parámetros BSP. 81

2.3 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 2 procesadores interconectados mediante *switch*, para $128 \leq n \leq 524288$. 83

2.4 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 4 procesadores interconectados mediante *switch*, para $128 \leq n \leq 524288$. 85

2.5 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 6 procesadores interconectados mediante *switch*, para $126 \leq n \leq 516096$. 87

2.6 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 2 procesadores interconectados mediante *ethernet*, para $128 \leq n \leq 524288$. 89

2.7 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 4 procesadores interconectados mediante *ethernet*, para $128 \leq n \leq 524288$. 91

2.8 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 6 procesadores interconectados mediante *ethernet*, para $126 \leq n \leq 516096$. 93

2.9 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un *cluster* de PC's, para 2 procesadores y $128 \leq n \leq 65536$. 95

2.10 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un *cluster* de PC's, para 4 procesadores y $128 \leq n \leq 65536$. 97

2.11 Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un *cluster* de PC's, para 6 procesadores y $126 \leq n \leq 64512$. 99

2.12 Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 *switch*. 105

2.13 Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 *ethernet*. 106

2.14 Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un *cluster* de PC's. 107

2.15 *Speed-up* (S_p) y eficiencia (E_p) de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 y un *cluster* de PC's. 108

2.16 Valores de parámetros BSP. 109

2.17 *Speed-up* (S_p) y eficiencia (E_p) de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un CRAY T3D y un CRAY T3E. 110

3.1 Valores de parámetros BSP. 142

3.2 Tiempos teóricos y experimentales del algoritmo 3.1 (TW), medidos en un IBM SP2 con 2 procesadores interconectados mediante *switch*, para $128 \leq n \leq 524288$. 143

3.3 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2), medidos en un IBM SP2 con 4 procesadores interconectados mediante *switch*, para $128 \leq n \leq 524288$. 145

3.4 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2), medidos en un IBM SP2 con 6 procesadores interconectados mediante *switch*, para $126 \leq n \leq 516096$. 147

3.5 Tiempos teóricos y experimentales del algoritmo 3.1 (TW), medidos en un IBM SP2 con 2 procesadores interconectados mediante *ethernet*, para $128 \leq n \leq 524288$. 149

3.6 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2), medidos en un IBM SP2 con 4 procesadores interconectados mediante *ethernet*, para $128 \leq n \leq 524288$. 151

3.7 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2), medidos en un IBM SP2 con 6 procesadores interconectados mediante *ethernet*, para $126 \leq n \leq 516096$. 153

3.8 Tiempos teóricos y experimentales del algoritmo 3.1 (TW) medidos en un *cluster* de PC's, para 2 procesadores y $128 \leq n \leq 65536$. 156

3.9 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) medidos en un *cluster* de PC's, para 4 procesadores y $128 \leq n \leq 65536$. . . 158

3.10 Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) medidos en un *cluster* de PC's, para 6 procesadores y $126 \leq n \leq 64512$. . . 160

3.11 Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 *switch*. 165

3.12 Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 *ethernet*. 166

3.13 Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 3.2 (TW1) y 3.3 (TW2) en un *cluster* de PC's. 167

3.14 *Speed-up* (S_p) y eficiencia (E_p) de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 y un *cluster* de PC's. 168

3.15 Valores de parámetros BSP. 169

3.16 *Speed-up* (S_p) y eficiencia (E_p) de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un CRAY T3D y un CRAY T3E. 170

4.1 Valores de parámetros BSP. 195

4.2 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con 2 procesadores interconectados mediante *switch*, para $128 \leq n \leq 524288$. 198

4.3 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con 4 procesadores interconectados mediante *switch*, para $128 \leq n \leq 524288$. 200

4.4 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con 6 procesadores interconectados mediante *switch*, para $126 \leq n \leq 516096$. 202

4.5 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con 2 procesadores interconectados mediante *ethernet*, para $128 \leq n \leq 524288$. 204

4.6 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con 4 procesadores interconectados mediante *ethernet*, para $128 \leq n \leq 524288$. 206

4.7 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con 6 procesadores interconectados mediante *ethernet*, para $126 \leq n \leq 516096$. 208

4.8 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) medidos en un *cluster* de PC's, para 2 procesadores y $128 \leq n \leq 65536$. 210

4.9 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) medidos en un *cluster* de PC's, para 4 procesadores y $128 \leq n \leq 65536$. 212

4.10 Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) medidos en un *cluster* de PC's, para 6 procesadores y $126 \leq n \leq 64512$. 214

4.11 Tiempos teóricos de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) medidos en un CRAY T3D, para 256 procesadores y $512 \leq n \leq 524288$. 219

4.12 *Speed-up* (S_p) y eficiencia (E_p) de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 y un *cluster* de PC's. 220

4.13 Valores de parámetros BSP. 221

4.14 *Speed-up* (S_p) y eficiencia (E_p) de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D y un CRAY T3E. 222

5.1 Valores de parámetros BSP. 224

5.2 Número óptimo de procesadores en un CRAY T3D, para $16384 \leq n \leq 524288$. 246

5.3 Número óptimo de procesadores en un CRAY T3E, para $16384 \leq n \leq 524288$. 253

Prólogo

La resolución de sistemas de ecuaciones lineales tridiagonales en paralelo es un problema extensamente analizado y estudiado durante las dos últimas décadas. La razón para este interés es doble, por un lado está el punto de vista académico: el problema presenta una doble vertiente de desafío por el poco paralelismo inherente que muestra al tiempo que de sencillez y simplicidad en su tratamiento; por el otro lado está el punto de vista de la ingeniería pues son muchas las aplicaciones de diversos campos en las que se hace necesaria la resolución de sistemas tridiagonales, tal es el caso del método iterativo para la resolución de ecuaciones diferenciales parciales ADI (*Alternating Direction Implicit*), el ajuste de curvas mediante *splines* cúbicos, el tratamiento estadístico de los fotones en los rayos láser o la modelización del comportamiento eléctrico de las neuronas. La mayoría de los algoritmos (y aplicaciones para arquitecturas concretas) propuestos para resolver este tipo de sistemas en paralelo están basados en los trabajos desarrollados en las décadas de los sesenta y setenta.

Hockney y Golub [62] proponían en 1965 el primer método específico para resolver sistemas tridiagonales: reducción cíclica (*Cyclic Reduction*). Aunque originalmente el método no era paralelo, fue precursor de una amplia familia de algoritmos paralelos basados en el mismo. El primer algoritmo paralelo para la resolución de sistemas tridiagonales, propuesto por Stone [96] en 1973, es el método *Recursive Doubling* que utiliza transformaciones (basadas en las propiedades asociativa y conmutativa de la suma y el producto) distintas a las usadas por el método de la reducción cíclica y es el embrión de otra familia de algoritmos para la resolución de sistemas tridiagonales en paralelo.

Otra familia de algoritmos, conocida como divide y vencerás (*Divide and Conquer*), tiene su origen en los dos algoritmos paralelos para la resolución de sistemas tridiagonales propuestos por Sameh y Kuck [93] en 1978; la idea en la que se fundamenta esta familia

de algoritmos consiste en la división de un problema en diversos subproblemas más pequeños, la solución del problema original se obtiene combinando las soluciones de estos subproblemas. Posteriormente, en 1981, Wang [105] propone un nuevo método (divide y vencerás) para la resolución de sistemas tridiagonales que particiona el sistema en bloques consecutivos de ecuaciones y utiliza la eliminación gaussiana para la diagonalización de éstos. Entre otros métodos no pertenecientes a ninguna de las tres familias mencionadas anteriormente, cabe mencionar el método de las particiones superpuestas (*Overlapped Partitions Method*) propuesto en 1993 por Larriba, Jorba y Navarro [72].

Existen diversos modelos que a lo largo de los años se han ido proponiendo para hacer factible una computación paralela de propósito general. Hasta la fecha no existe un modelo único que fundamente el desarrollo de la computación paralela al igual que el modelo von Neumann lo ha hecho en la computación secuencial. El modelo BSP (*Bulk Synchronous Parallel*) propuesto por Valiant [101] en 1990, es uno de los que más seriamente se ha considerado en los últimos años como fundamento de una computación paralela de propósito general; se caracteriza por disponer de un modelo de coste que permite obtener una predicción sobre el tiempo de ejecución de un algoritmo concreto en un entorno de computación concreto. Uno de los objetivos de este trabajo ha sido el análisis del modelo de coste para lo cual se ha medido, en distintas máquinas paralelas (IBM SP2 y *cluster* de PC's), la desviación del tiempo de ejecución real sobre el tiempo de ejecución previsto en diversos algoritmos BSP para la resolución de sistemas tridiagonales. Otro objetivo ha sido proponer un algoritmo BSP para la resolución de sistemas tridiagonales estrictamente diagonal dominantes con buenos resultados de tiempo, objetivo que se ha visto cumplido con la formalización de un nuevo método.

La presente memoria está estructurada en cinco capítulos; en el capítulo 1 se describen algunos de los modelos de computación paralela que se han propuesto a lo largo de los años, con especial dedicación al modelo BSP. En el capítulo 2 se describe el método de las particiones superpuestas para el que se proponen dos algoritmos BSP y se comparan entre sí. La paralelización de la factorización LDU de matrices tridiagonales fundamenta el algoritmo BSP bidireccional para dos procesadores que se formula en el capítulo 3 en el que, además, se propone un nuevo método bidireccional para un número par de procesadores basado en el método bidireccional para dos procesadores y en el método de las particiones superpuestas; asimismo, se plantean dos algoritmos BSP para el nuevo método y se comparan entre sí. El método de las particiones de Wang [105] es un clásico

y rápido (véase Larriba [71]) método para la resolución de sistemas tridiagonales que se describe en el capítulo 4; en este capítulo se proponen dos algoritmos BSP para el mismo, uno de los cuales es una modificación de dicho método que supone una mejora cuando el número de procesadores es grande. En la última sección del capítulo se comparan entre sí ambos algoritmos BSP. En el capítulo 5 y último se comparan entre sí todos los algoritmos BSP descritos y analizados en los capítulos anteriores y se obtiene el óptimo para cada una de las situaciones que han sido objeto de estudio. Es de reseñar el buen comportamiento del nuevo método propuesto en el capítulo tercero frente al método de Wang y al método (secuencial) de eliminación de Gauss para sistemas tridiagonales, especialmente en un CRAY T3D y en un CRAY T3E, máquinas en las que se han obtenido sólo tiempos teóricos haciendo uso del modelo de coste BSP.

Capítulo 1

Computación Paralela

Cada día se está exigiendo mayor potencia de cálculo en los computadores, no sólo a nivel científico sino a otros: industrial, comercial, etc. Cabe esperar que esta demanda se irá incrementando en el futuro, sobre todo en aplicaciones de simulación y de tiempo real. La velocidad en la computación tiene unos límites, debido a restricciones tecnológicas y lógicas, a los que poco a poco se va llegando. Las soluciones que se plantean ante esta demanda (véase Bilardi y Preparata [12]) se pueden concretar básicamente en dos líneas de actuación:

Software. Las soluciones de tipo *software* consisten en la mejora de los algoritmos para que estos se ejecuten requiriendo el menor tiempo posible. Ahora bien, los algoritmos poseen una complejidad y una necesidad de recursos que limitan este tipo de solución.

Hardware. Las soluciones de tipo *hardware* consisten en mejoras en la tecnología de computadores que deriven en una mayor rapidez en la ejecución de instrucciones.

Una solución alternativa, compatible con las líneas de actuación anteriores, es el **paralelismo**, que por un lado abarca la réplica de unidades de tratamiento de información con el objetivo de repartir tareas entre las mismas (espacial o temporalmente) y por otro la adaptación de los algoritmos a estas arquitecturas paralelas.

1.1 Antecedentes de la computación paralela

En 1936 Turing [99] demostró que, al menos teóricamente, era posible diseñar una máquina secuencial de propósito general capaz de llevar a cabo eficientemente cualquier cálculo que pudiese realizar una máquina secuencial de propósito específico, diseñada por tanto para resolver de forma muy eficiente un determinado tipo de cálculos. Ocho años más tarde von Neumann [23] realiza su propuesta de computador secuencial de propósito general que refleja los principios del trabajo de Turing en un modelo práctico. En este modelo, conocido con el nombre de "Computador von Neumann", se han basado la mayoría de computadores secuenciales construidos desde finales de los años cuarenta hasta hoy.

La universalidad proporcionada por el modelo von Neumann ha permitido y estimulado el desarrollo de lenguajes de alto nivel, lo que, unido a la estabilidad del modelo, ha fomentado la creación de aplicaciones de alto nivel portables, ejecutables por tanto en cualquier computador secuencial con un buen rendimiento.

La búsqueda de un mayor rendimiento en las aplicaciones ha impulsado la utilización de la computación paralela en muchas áreas, sin duda, es en el álgebra lineal numérica donde más esfuerzos se han dedicado al desarrollo de algoritmos paralelos; en las dos últimas décadas los trabajos sobre algoritmos paralelos para la resolución de sistemas lineales (tanto por métodos directos como por métodos iterativos), para el problema de mínimos cuadrados y el cálculo de valores propios son abundantes (véanse Gallivan, Plemmons y Sameh [43], Golub y Ortega [49], Ortega [91] y Ortega, Voigt y Romine [92]).

La transición desde la computación secuencial a la computación paralela no se está produciendo sin dificultades. En la mayoría de computadores paralelos que se construyeron en los años ochenta sólo era posible obtener un rendimiento aceptable de las aplicaciones explotando los detalles de la arquitectura de la máquina, esto tiene como consecuencia que las aplicaciones paralelas producidas para una determinada máquina no se pueden adaptar con facilidad a otras. La diversidad de arquitecturas paralelas rápidamente cambiantes y lo laboriosa que resultaba la creación de las aplicaciones paralelas en esa época, impidió el crecimiento de la computación en paralelo. Esa situación ha mejorado sustancialmente en la actual década.

Desde principios de los noventa las diferentes clases de computadores paralelos (prin-

cipalmente máquinas paralelas de memoria compartida o distribuida y redes de estaciones de trabajo) están convergiendo a una estructura cada vez más parecida y se está evolucionando hacia un modelo arquitectónico estándar, consistente en un conjunto de pares procesador-memoria interconectados mediante una red de comunicación que permite el empleo de un espacio de direcciones global. Las razones que explican esta convergencia son de muy diversa índole, pero fundamentalmente de tipo tecnológico y económico. Una causa es la rápida evolución de los microprocesadores que tiene lugar con un altísimo coste económico, coste que se ve compensado por el elevado número de ventas de los mismos. El mercado de los supercomputadores paralelos es más pequeño por lo que la evolución de los procesadores no se ve compensada económicamente, de ahí que a principios de los noventa muchos fabricantes (Intel, Meiko, IBM o CRAY) comenzaran a utilizar los microprocesadores existentes en el mercado para construir sus máquinas paralelas en lugar de diseñar (con el coste que ello supone) procesadores específicos para las mismas. Mientras que la velocidad del procesador aumenta entre el 50% y el 100% cada año, los tiempos de acceso a memoria decrecen lentamente, por ello los procesadores actuales necesitan estructuras de caché muy sofisticadas que ajusten la diferencia entre el tiempo de ciclo del procesador y el de acceso a memoria. Esta última causa además del motivo económico han sido el móvil para que definitivamente se haya abandonado la línea de construcción de ordenadores masivamente paralelos y se tienda a converger hacia estructuras de memoria distribuida.

El proceso de convergencia arquitectónica conlleva la confianza de que en un futuro no muy lejano se empiece a ver el crecimiento de una industria de *hardware* y *software* paralelos similar a la existente para computación secuencial. Uno de los principales objetivos de esta industria debe ser crear *software* escalable, es decir, ejecutable en cualquier arquitectura con un gran rendimiento sin que este sufra menoscabo cuando aumente el tamaño de problema o el número de procesadores. El éxito en este reto provocará la extensión de la computación paralela más allá del ámbito de las aplicaciones científicas para convertirse en la forma habitual de computación.

1.2 Modelos de computación paralela

Existen diversos modelos que a lo largo de los años se han ido proponiendo para hacer factible una computación paralela de propósito general. Hasta la fecha no existe

un modelo único que fundamente el desarrollo de la computación paralela al igual que el modelo von Neumann lo ha hecho en la computación secuencial. En esta sección se describen brevemente algunos de estos modelos, utilizados para el análisis teórico y diseño de algoritmos paralelos.

1.2.1 El modelo PRAM (1978)

El modelo PRAM (*Parallel Random Acces Machine*) es uno de los más conocidos y utilizados históricamente para el análisis de la complejidad de los algoritmos paralelos.

En este modelo, se considera un conjunto ilimitado de procesadores, cada uno de ellos con una pequeña memoria local, que ejecutan todas las instrucciones sincronizadamente y tienen acceso a una memoria común. En una unidad de tiempo cada procesador puede realizar cualquier subconjunto de las siguientes operaciones:

- leer dos valores de la memoria global,
- realizar una de las operaciones básicas sobre los dos valores leidos,
- escribir un valor de vuelta en la memoria.

La comunicación entre procesadores se realiza exclusivamente a través de la memoria global, mediante la lectura de posiciones de memoria en las que previamente ha escrito otro procesador. Dependiendo de forma de lectura o escritura de dos o más procesadores en las mismas posiciones de memoria concurrentemente, el modelo PRAM puede ser clasificado en cuatro subgrupos:

- **CRCW** (*Concurrent Read Concurrent Write*), todos los procesadores pueden leer y escribir en las mismas posiciones de memoria concurrentemente, durante una unidad de tiempo.
- **CREW** (*Concurrent Read Exclusive Write*), durante una unidad de tiempo, varios procesadores pueden leer determinadas posiciones de memoria pero sólo uno puede escribir en ellas.

- **ERCW** (*Exclusive Read Concurrent Write*), durante una unidad de tiempo, varios procesadores pueden escribir en determinadas posiciones de memoria pero sólo un procesador puede leerlas.
- **EREW** (*Exclusive Read Exclusive Write*). Este es el modelo más restrictivo, sólo un procesador puede leer determinadas posiciones de memoria y sólo un procesador puede escribir en ellas, durante una unidad de tiempo.

La complejidad de un algoritmo PRAM viene dada en términos del número de fases o pasos de que consta y del número máximo de procesadores necesario en cada uno de los pasos.

Aun cuando es un modelo enormemente atractivo para el análisis y diseño de algoritmos paralelos es poco realista, fundamentalmente por la suposición de que todos los procesadores operan de forma síncrona y que 1a comunicación entre los procesadores se lleva a cabo, por medio de la memoria global, sin coste alguno. A pesar de su utilidad teórica, el modelo PRAM es difícilmente implementable en arquitecturas con memoria distribuida, existen variaciones del modelo que han tratado de eliminar restricciones para obtener un modelo más práctico pero conservando gran parte de su simplicidad. Para mayor información, véanse Aggarwal, Chandra y Snir [1, 2], Fortune y Wyllie [42], Gibbons [48] y Karp, Luby y Meyer auf der Heide [67].

1.2.2 El modelo BSP (1990)

El modelo BSP (*Bulk Synchronous Parallel*) es uno de los que más seriamente se ha considerado en los últimos años como fundamento de una computación paralela de propósito general. Fue propuesto por Valiant [101] en 1990, en un intento de lograr un modelo puente entre teoría y práctica mucho más realista que el modelo PRAM.

Un computador BSP consiste en un conjunto de pares procesador-memoria, una red de comunicación global y un mecanismo para la sincronización de todos los procesadores o un subconjunto de ellos, véase la figura 1.1.

El modelo utiliza cuatro parámetros para caracterizar a un computador paralelo:

- p es el número de procesadores.

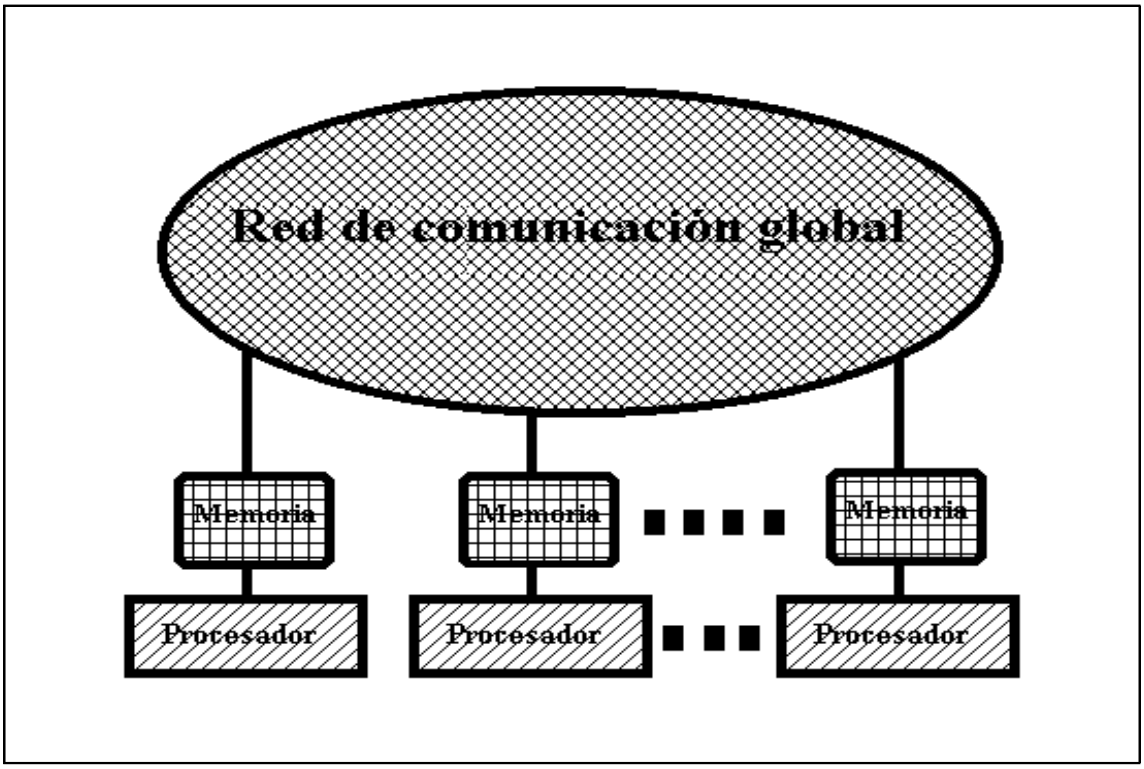


Figura 1.1: *Computador BSP: modelo arquitectónico.*

- s es la velocidad de computación de cada procesador.
- l es el tiempo necesario para realizar una sincronización entre los procesadores.
- g es la razón entre la capacidad total de computación por segundo y la capacidad total de comunicación por segundo.

El parámetro l está relacionado con la **latencia** de la red, es decir, con el tiempo necesario para realizar un acceso a una memoria remota en una situación de continuo tráfico en la red. El parámetro g corresponde a la frecuencia con que se pueden realizar accesos a memoria remota, en las máquinas con valores altosde g los accesos a memoria no local deben llevarse a cabo con poca frecuencia.

En el diseño de algoritmos BSP es conveniente asegurar que por cada petición de acceso a memoria no local que haga un procesador, éste sea capaz de ejecutar g operaciones sobre datos locales. El rendimiento de la red de conexión global se describe en función

de los valores de l y g, por lo que el modelo ofrece la posibilidad de diseñar algoritmos parametrizados por esos valores e implementables eficientemente en un amplio rango de arquitecturas con valores diversos de l y g.

En la sección1.5 se describirá este modelo con mayor detalle.

1.2.3 El modelo LogP (1993)

El modelo LogP, propuesto por Culler, Karp, Patterson, Sahay, Schauser, Santos, Subramonian y voon Eicken [34] en los últimos años, intenta reflejar las tendencias tecnológicas de los computadores paralelos: multiprocesador con memoria distribuida en el que los procesadores se comunican mediante paso de mensajes, y pretende ser útil en el desarrollo de algoritmos paralelos portables y eficientes así como aportar ideas para el diseño de máquinas paralelas. Está caracterizado por cuatro parámetros que modelan el ancho de banda de comunicación, el tiempo de comunicación y la eficiencia de simultanear comunicación y cálculo:

- p es el número de procesadores.
- L es una cota superior de la latencia que se produce al realizar una comunicación punto a punto de un mensaje corto, desde un procesador origen a otro destino.
- o (*overhead*) se define como el tiempo en que los procesadores están ocupados en la transmisión o recepción de un mensaje y no pueden realizar ninguna otra tarea.
- $g(gap)$ es el mínimo intervalo de tiempo requerido en un procesador entre dos envíos o recepciones consecutivas, su recíproco corresponde al ancho de banda disponible por cada procesador.

La elección de estos parámetros pretende recoger de forma precisa las características del funcionamiento de las máquinas reales, esto hace que la utilización del modelo para el desarrollo y análisis de algoritmos no sea sencilla. En [11], Bilardi, Herley, Pietracaprina, Pucci y Spirakis realizan una comparación cuantitativa entre los modelos LogP y BSP y señalan que ambos modelos pueden simularse eficientemente entre sí por lo que es preferible el modelo BSP debido a su mayor simplicidad.

1.2.4 Data-Parallel

Dentro del campo de la computación paralela escalable, el paradigma *data-parallel* constituye otra alternativa importante a la construcción de programas. Existe un buen número de lenguajes de programación y elegantes teorías desarrolladas en torno al mismo, véase por ejemplo Skillicorn [94].

Data-parallel, más que una base teórica para el análisis de la complejidad de los algoritmos paralelos es un modelo de programación que se basa en la distribución de los datos entre los distintos procesadores y en la generación automática por parte del compilador de la comunicación necesaria para llevar a cabo los cálculos, resulta particularmente apropiado para problemas en los que la localización es crucial. Un buen ejemplo práctico de lenguaje de programación basado en este modelo lo constituye *High Performance Fortran*, véase Koelbel, Loveman, Schreiber, Steele y Zosel [68].

1.2.5 Modelos de paso de mensajes (1989)(1992)

Los modelos de paso de mensajes están basados en primitivas de comunicación de los procesadores mediante pares de envío-recepción, no disponen de un modelo analítico sencillo con el que estudiar o poder predecir el coste de los algoritmos paralelos, son difíciles de usar y los programas difíciles de depurar. Los más utilizados son PVM (*Parallel Virtual Machine*) y MPI (*Message Passing Interface*).

La librería de paso de mensajes PVM permite que una red heterogénea de computadores aparezca como un simple recurso: una máquina virtual. Su desarrollo se inició en 1989 en el ORNL (*Oak Ridge National Laboratory*), Tennessee, MI; en los últimos años ha sido implementada para la mayoría de sistemas multiprocesadores y ha constituido prácticamente el modo estándar de escribir programas paralelos en sistemas de memoria distribuida, por lo que tiene alcanzado el objetivo de conseguir códigos paralelos portables. Para un mayor detalle pueden consultarse Beguelin, Dongarra, Geist, Jiang, Manchek y Sunderam [7, 8, 9] y Geist [44].

El principal objetivo de MPI es aportar un estándar para escribir programas paralelos portables con paso de mensajes. En su diseño, iniciado en 1992, se tuvo en cuenta los rasgos de un buen número de sistemas de paso de mensajes existentes. MPI permite la

compilación de librerías separadas y está diseñado para porderse ejecutar en la mayoría de plataformas paralelas. Véanse Geist, Kohl y Papadopoulos [45] y Gropp, Lusk y Skjellum [55].

1.3 Plataformas de computación paralela

Los multiprocesadores se pueden clasificar de varias formas, atendiendo a diversos criterios. Una importante clasificación (debida a Flynn [41]) se basa en la posibilidad de procesar uno o más flujos de instrucciones y/o datos, pueden darse cuatro posibilidades:

SISD (*Simple Instructión Simple Data*). Un flujo simple de instrucciones opera sobre un flujo simple de datos, este es el modelo clásico de von Neumann.

SIMD (*Simple Instructión Multiple Data*). Un flujo simple de instrucciones opera sobre múltiples flujos de datos. Todos los procesadores ejecutan la misma instrucción aunque sobre distintos datos. A este tipo pertenecen fundamentalmente los procesadores matriciales y los seccionados (pipelinizados). Permiten un gran número de procesadores.

MIMD (*Multiple Instructión Multiple Data*). Cada procesador ejecuta su propio código sobre datos distintos a los de otros procesadores. A este tipo corresponden los multiprocesadores paralelos.

MISD (*Multiple Instructión Simple Data*). Es teóricamente equivalente al tipo SISD.

Otra clasificación importante, atendiendo a la situación física de la memoria y a la forma en que los procesadores acceden a ella, distingue dos grandes clases de computadores paralelos: los de memoria compartida (*shared memory*) y los de memoria distribuida (*distributed memory*).

Memoria compartida. Todos los procesadores tienen acceso independiente a una memoria común, cada uno de ellos posee una pequeña memoria local para almacenar código y resultados intermedios. La comunicación entre los distintos procesadores se realiza a través de la memoria común. La principal ventaja de este sistema es

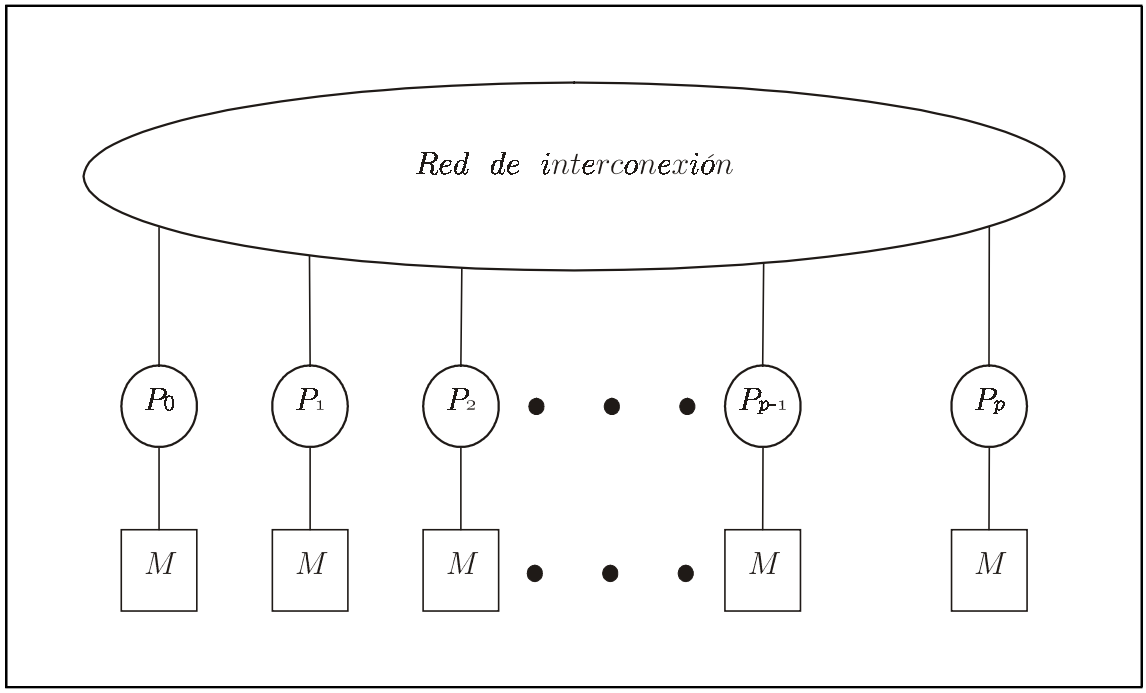


Figura 1.2: *Multiprocesador con memoria distribuida.*

que permite comunicaciones muy rápidas entre los procesadores, sin embargo pueden producirse conflictos en el acceso a los datos, así como tiempos de contención cuando un procesador deba esperar a que otro haya leído o actualizado un dato. Este tipo de problemas se incrementa al aumentar el número de procesadores.

Memoria distribuida. Cada procesador posee su propia memoria local inaccesible a los demás. Los procesadores están conectados entre sí, a efectos de intercambio de datos, mediante una red de interconexión (véase la figura 1.2). La comunicación entre los distintos procesadores debe realizarse mediante paso de mensajes, lo que conlleva un retraso en el tiempo de ejecución similar al tiempo de contención.

Por lo general los elementos de memoria de los multiprocesadores con memoria distribuida tienen menor capacidad que el único elemento de memoria de un multiprocesador con memoria compartida. Esto implica que los datos necesarios para resolver un problema deben distribuirse de forma adecuada entre los procesadores, intentando minimizar la duplicación de datos y equilibrar la carga entre los distintos procesadores.

Un aspecto fundamental en los computadores paralelos es la **topología del sistema**: forma en que están conectados los procesadores. Teniendo en cuenta que la comunicación debe realizarse mediante paso de mensajes a través de la red de interconexión, es importante que la longitud del camino más largo que conecta dos procesadores sea lo más pequeña posible, dicha longitud se conoce como **diámetro del sistema**. También son importantes los llamados **algoritmos de encaminamiento** que marcan los caminos que deben seguir los mensajes para evitar las colisiones, mensajes que básicamente son de tres tipos:

- **punto a punto**: de un procesador a otro procesador,
- **difusiones**: mensajes que desde un procesador deben llegar a todos los demás, y
- **reducciones**: mensajes que desde todos los procesadores deben llegar a uno concreto.

De entre las distintas topologías que se pueden considerar, las más utilizadas son las de anillo y malla. En la topología de anillo cada procesador está conectado a sus dos vecinos, para conectar p procesadores se requieren exclusivamente p líneas por lo que es sencilla y fácilmente ampliable. Puede resultar poco eficiente debido a que el diámetro es grande: $p-1$. Dependiendo de que las líneas permitan la comunicación en un sentido o en los dos, el anillo puede ser unidireccional o bidireccional. La figura 1.3 muestra estas dos topologías.

La topología de malla comprende muy distintos tipos configuración. En una malla bidimensional las líneas de interconexión corresponden a las fibras de la misma y los procesadores a los cruces de la malla, como se muestra en la figura 1.4. Cada procesador está conectado a sus procesadores vecinos, obteniéndose así con pocas conexiones diámetros pequeños. Superponiendo varias mallas bidimensionales y conectando los procesadores correspondientes se obtiene una malla tridimensional. Una variante consiste en conectar los procesadores de la primera fila con los correspondientes de la última fila y los procesadores de la primera columna con los correspondientes de la última columna, obteniéndose una estructura toroidal (véase la figura 1.5) en la que se reduce el diámetro .

A continuación se proporciona una breve descripción de algunos de los supercomputadores paralelos más usados.

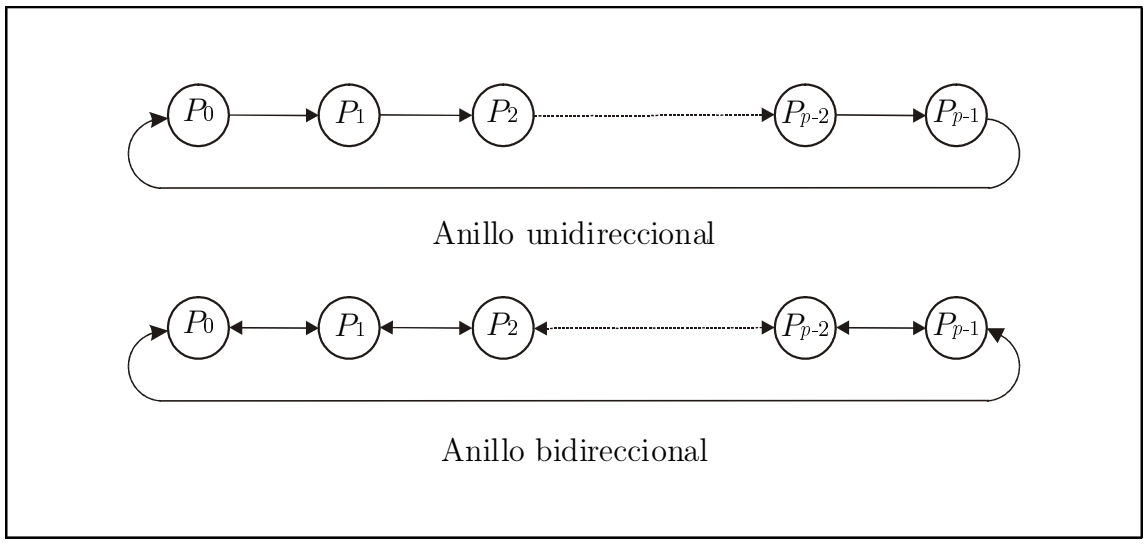


Figura 1.3: *Topología de anillo.*

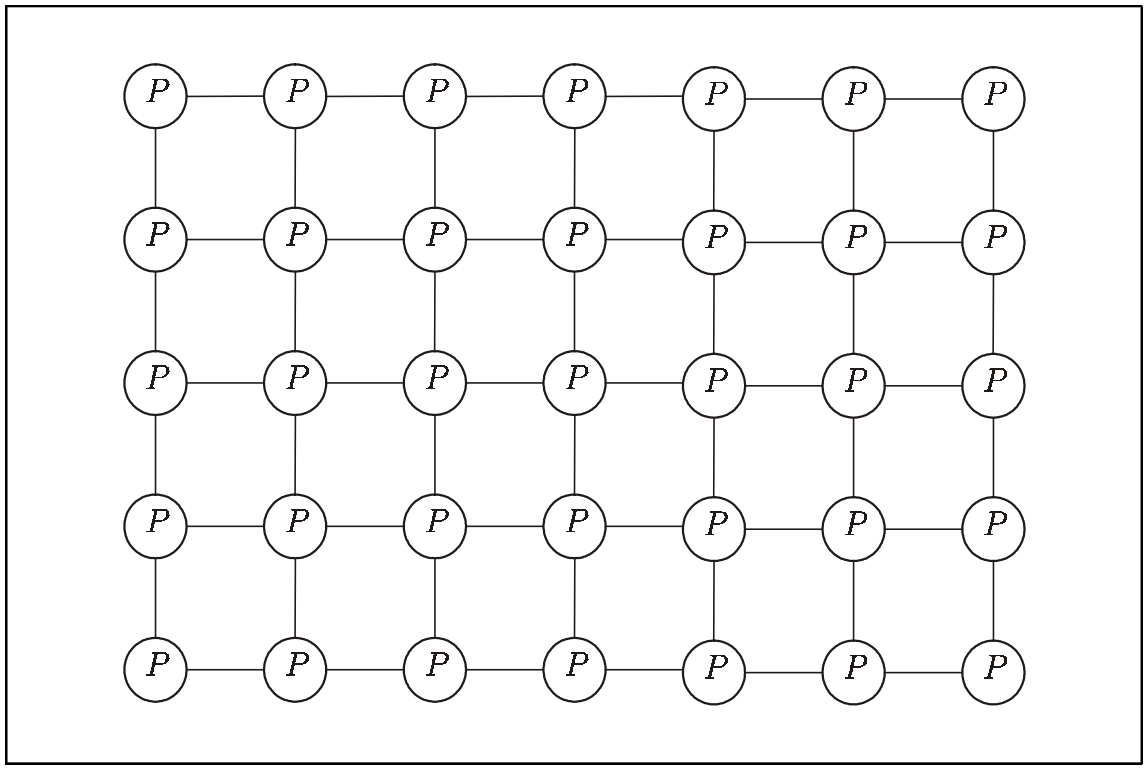


Figura 1.4: *Topología de malla bidimensional.*

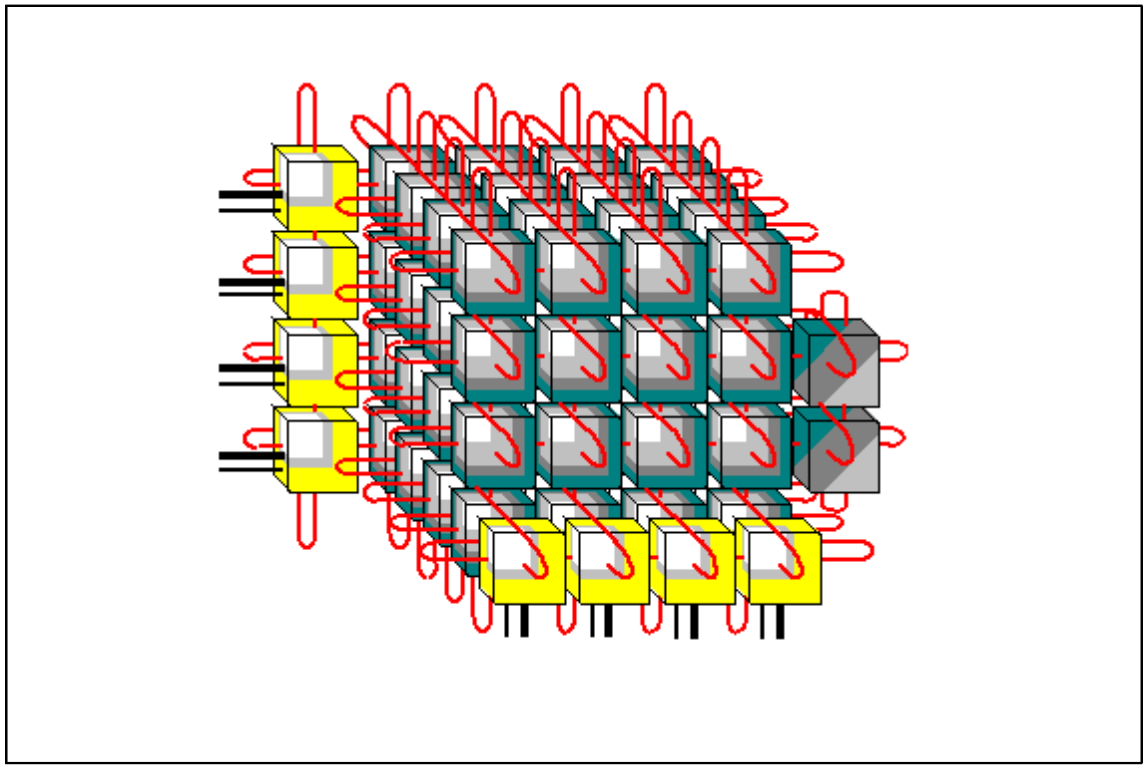


Figura 1.5: *Topología de malla tridimensional.*

IBM SP2. Los supercomputadores de cálculo paralelo IBM SP2 son sistemas multiprocesadores que pertenecen a la familia de productos RISC System/6000 de IBM, están basados en la tecnología de estación de trabajo que utilizan los procesadores POWER y POWER2 y se caracterizan entre otros aspectos por su escalabilidad (el sistema puede ampliarse en potencia incorporando más nodos conforme aumenten las necesidades) y su compatibilidad con las aplicaciones existentes para AIX en sistemas inferiores de la familia RS/6000.

Se trata de computadores paralelos de memoria distribuida en los que los nodos están interconectados a través de un sistema de comunicación formado por la red de transmisión de datos (*High Performance Switch* - HPS) y los adaptadores que conectan los nodos al *switch*, estos adaptadores disponen de un microprocesador que realiza parte del trabajo necesario para llevar a cabo el intercambio de mensajes entre los procesadores y de alguna memoria para *buffers*.

En la figura 1.6 se detalla la estructura básica de un IBM SP2 en la que pueden

destacarse los siguientes elementos.

- La estación de control, desde la cual se instala, administra y monitoriza todo el sistema.
- El *frame* (marco), que contiene los distintos elementos que componen el sistema, excepto la estación de control. Un *frame* puede contener hasta 16 nodos, pudiendo interconectarse con otros *frames* para crear sistemas de hasta 128 nodos.
- Los nodos, cada uno de los cuales es un ordenador completo. Cada *frame* puede almacenar hasta dieciséis nodos *thin*, ocho *wide* y cuatro 604 *high*. Mientras un nodo *thin* o *wide* sólo puede incorporar un procesador del tipo POWER, un nodo 604 *high* tiene capacidad para almacenar hasta ocho procesadores PowerPC 604. De la misma forma, el número de *slots* de expansión y de discos internos que se pueden instalar en un nodo depende también de su tamaño.
- El *High Performance Switch* que interconecta los nodos de un *frame* proporcionando un ancho de banda de hasta 110 Megabytes por segundo entre cada par de nodos.
- Una red local ethernet, que interconecta los nodos y la estación de control (en este caso con un menor ancho de banda que el proporcionado por el HPS).
- Una línea serie RS-232 que conecta la estación de control con cada uno de los *frames* con el fin de gestionar el hardware (parada y encendido de nodos, monitorización, etc). Existe una línea serie por cada *frame*.

Silicon Graphics POWER CHALLENGE. Los supercomputadores POWER CHALLENGE son multiprocesadores de memoria compartida que usan procesadores RISC superescalares R10000 y R8000. La línea de productos POWER CHALLENGE de Silicon Graphics ofrece sistemas desde 2 a 12 nodos como el POWER CHALLENGE L (con un precio bajo o moderado) hasta supercomputadores con hasta 288 procesadores como el POWER CHALLENGE array.

CRAY T3E. Los sistemas paralelos escalables CRAY T3E están construidos sobre la arquitectura de la primera generación de sistema escalable de CRAY, el CRAY T3D. Se trata de un multiprocesador de memoria compartida . Los microprocesadores son DEC Alpha EV5 con un rendimiento pico de 600 Mflops. Cada procesador tiene su propia memoria local con una capacidad entre 64 Mbytes y 2 Gbytes. Un subsistema

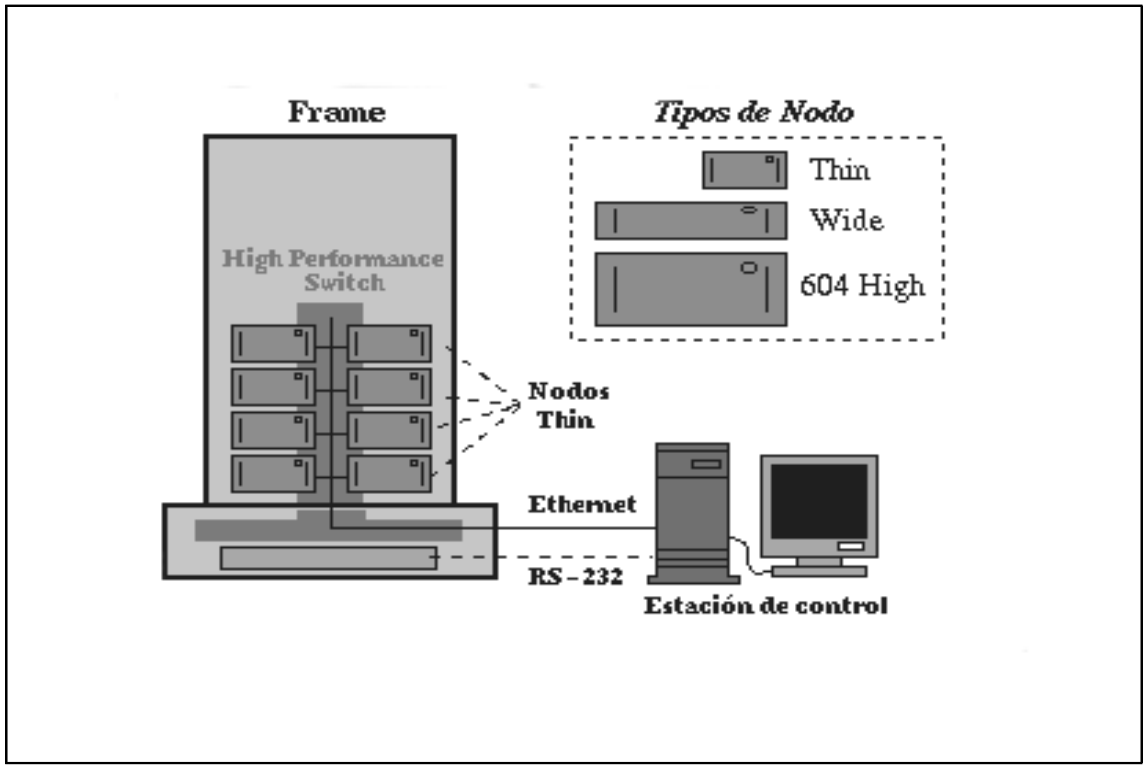


Figura 1.6: *Estructura básica de un IBM SP2.*

de memoria direccionable globalmente, hace accesibles todas las memorias para cualquier procesador en un sistema CRAY T3E. Los procesadores están conectados mediante una red con topología en toro.

CONVEX Exemplar SPP1000. Un sistema Exemplar SPP1000 consiste en un conjunto de hipernodos (de 1 a 16). Cada hipernodo a su vez contiene de 4 a 8 procesadores HP PA-RISC 7100 y una memoria local (de 256 Mbytes hasta 2Gbytes). Los procesadores se organizan por pares en bloques funcionales. Dentro de cada hipernodo las (hasta) 4 unidades funcionales acceden a memoria a través de una red de interconexión que permite la conexión directa de todos los nodos con el resto (*crossbar*). La conexión entre hipernodos se lleva a cabo mediante una colección de anillos. Se tienen por tanto dos costes de acceso a memoria distintos según que un procesador acceda a una memoria local a su hipernodo o remota situada en un hipernodo distinto.

1.4 Medidas de paralelismo

En esta sección se definen algunos de los conceptos y parámetros que se utilizan para evaluar el rendimiento de los algoritmos paralelos. Se supone que el sistema en el que se ejecuta el algoritmo paralelo dispone de p procesadores.

El **grado de paralelismo** de un algoritmo numérico es el número de operaciones que pueden realizarse en paralelo, este parámetro es específico del algoritmo y no depende del número de procesadores del sistema. Relacionado con este concepto está el de **granularidad** que es el tamaño de las tareas realizables en paralelo de forma independiente.

Los parámetros más usados para medir el paralelismo de un algoritmo son el incremento de velocidad y la eficiencia del mismo. Se llama **incremento de velocidad** S_p (*speed-up*) de un algoritmo paralelo al cociente

$$S_p = \frac{\text{tiempo de ejecución en un solo procesador}}{\text{tiempo de ejecución en } p \text{ procesadores}},$$

en la práctica $S_p \leq p$.

Relacionada con la medida anterior está la **eficiencia** E_p de un algoritmo paralelo con respecto a sí mismo, que mide el grado de utilización de los procesadores del sistema al ejecutar en éste el algoritmo paralelo, se define como el cociente

$$E_p = \frac{S_p}{p}$$

y en la práctica es menor o igual que 1.

El objetivo, a la hora de diseñar un algoritmo paralelo, es conseguir la mayor eficiencia posible. En la práctica, la eficiencia disminuye cuando se incrementa el número de procesadores, entre los factores que más influyen en la reducción de la misma se pueden destacar la pérdida de paralelismo del algoritmo, el tiempo requerido para la comunicación y sincronización entre los procesadores y el desequilibrio de la carga computacional (en la distribución de las tareas entre los distintos procesadores debe procurarse que todos tengan una cantidad de trabajo similar, a fin de evitar que algunos procesadores se mantengan inactivos).

Aunque son más significativos los valores del incremento de velocidad y la eficiencia con respecto al mejor algoritmo secuencial que con respecto al propio algoritmo ejecutado

en un solo procesador, dichos valores se suelen calcular aplicando las definiciones dadas anteriormente ya que en muchos problemas resulta difícil (si no imposible) determinar cuál es el mejor algoritmo secuencial.

1.5 El modelo BSP

El modelo BSP (*Bulk Synchronous Parallel*) es una generalización del modelo PRAM inicialmente propuesta por Valiant [101] en 1990. Durante los últimos nueve años, la aproximación del modelo BSP a un modelo de propósito general para la computación paralela ha sido desarrollado por McColl [15, 16, 52, 59, 83, 84, 85, 86] en estrecha colaboración con Valiant. Su trabajo en algoritmos BSP, arquitecturas y lenguajes hace suponer que BSP proporciona un robusto modelo para la computación paralela y ofrece *software* paralelo con la doble perspectiva de escalabilidad e independencia de la arquitectura paralela.

Como ya se ha comentado en la subsección 1.2.2, un computador BSP consiste en un conjunto de procesadores cada uno de ellos con una memoria local, una red de comunicación global que permite la entrega de mensajes punto a punto y un mecanismo que permite la sincronización eficiente de todos los procesadores o un subconjunto de ellos. No existe ningún tipo de facilidad especial para combinar, replicar o difundir mensajes. El mecanismo de sincronización debe garantizar que todos los procesadores, o un subconjunto de ellos, alcancen un punto específico de la computación antes de que ninguno continúe con la computación, no tiene por qué ser una parte física del computador, puede ser realizada por el *software*.

Existe una amplia gama de sistemas informáticos que están conformados acorde al modelo BSP, como por ejemplo,

- un simple procesador con cache y memoria *off-chip*,
- redes de estaciones de trabajo o PC's con PVM, MPI o cualquier otro *software* de paso de mensajes,
- sistemas de multiprocesadores con memoria distribuida como IBM SP2, Intel Paragon, Meiko CS2 y Parsytec GC/PowerPlus,

- sistemas de multiprocesadores con memoria compartida, con la debida asignación de memoria para la actual distribución física de la misma, como el multiprocesador INTEL Pentium Pro, Silicon Graphics Origin (o Power Challenge), CRAY T3D/T3E y Convex SPP.

1.5.1 Los parámetros del modelo BSP

Es necesario identificar los parámetros que caractericen a un computador BSP. Obviamente, el número de procesadores y la velocidad de computación de cada uno de ellos son parámetros clave. Si se define la unidad de tiempo como el tiempo necesario para realizar una operación básica (a menudo se toma una operación aritmética: suma, resta, multiplicación o división, en coma flotante) entonces se puede tomar como **unidad de la velocidad de computación** de un procesador el número de operaciones básicas que éste puede realizar en un segundo.

Asimismo, la capacidad y velocidad de la red de comunicación son elementos vitales en la caracterización de un computador BSP. Para facilitar la comparación entre distintos computadores BSP, se puede medir el tiempo necesario para la comunicación en unidades de la velocidad de computación. Otro factor a tener en cuenta es el tiempo necesario para la sincronización entre los procesadores, que también se puede en unidades a la velocidad de computación.

El modelo BSP utiliza cuatro parámetros para caracterizar a un computador paralelo. La definición de los parámetros puede variar según los distintos autores, quizás la más extendida es la de McColl [86] que los define así:

- p es el número de procesadores.
- s es la velocidad de computación de cada procesador, se suele medir en Megaflops (millones de operaciones aritméticas en coma flotante por segundo).
- l es el número mínimo de unidades de tiempo necesarias para realizar una sincronización entre los procesadores. El parámetro corresponde a la latencia de la red de comunicación.
- g es la razón entre la capacidad total de computación por segundo y la capacidad

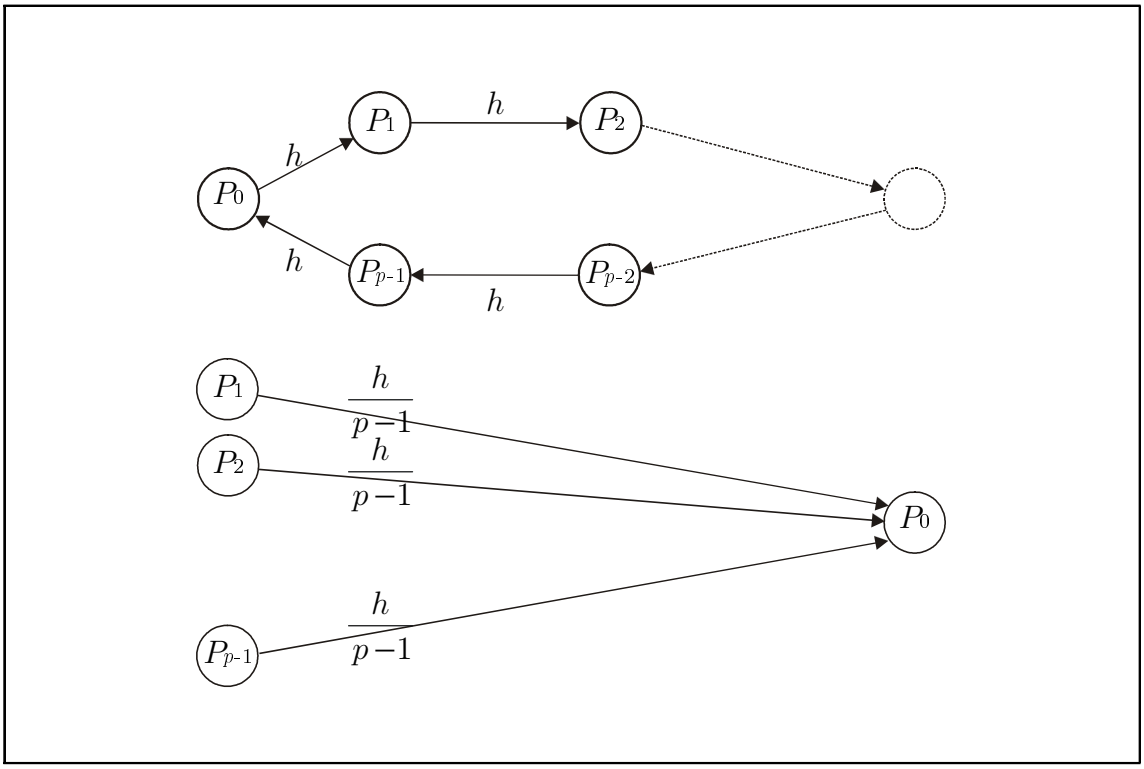


Figura 1.7: *Ejemplos de distintos tipos de patrones de comunicación que definen una h-relación.*

total de comunicación por segundo, esto es, el cociente entre el número total de operaciones básicas que se pueden realizar en un segundo por todos los procesadores y el número total de unidades de datos que se pueden repartir por la red de comunicación en un segundo.

Para explicar el significado del parámetro g, se introduce el concepto de h-relación. Una **h-relación** es un patrón de comunicación global en el que cada procesador puede enviar o recibir a lo sumo h unidades de datos (las unidades de datos pueden ser, por ejemplo, palabras de 32 bits). El coste de una h-relación se define como hg. El modelo BSP no distingue entre enviar h mensajes de tamaño una unidad de datos o enviar un sólo mensaje de tamaño h unidades de datos, en ambos casos el coste es hg. En la figura 1.7 se muestran dos ejemplos de patrones de comunicación que definen una h-relación: en el primero se produce una comunicación cíclica en la que cada procesador envía y recibe h unidades de datos; en el segundo, $p-1$ procesadores envían a otro, común a todos, $\frac{h}{p-1}$

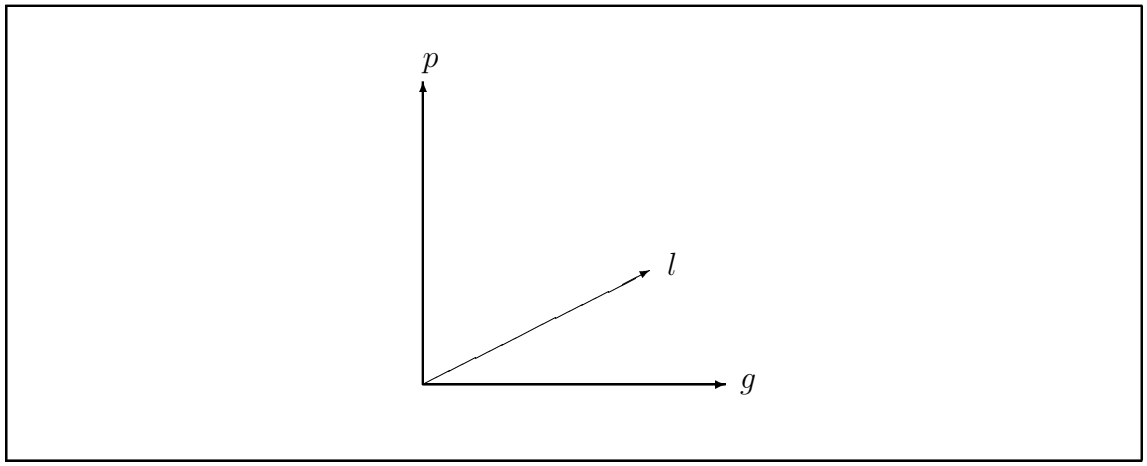


Figura 1.8: *Espacio generado por los parámetros p, l y g.*

unidades de datos.

Como ya se ha comentado anteriormente, para facilitar la comparación entre distintos computadores BSP los parámetros g y l se toman normalizados con respecto a s y por ello este último parámetro se omite a veces. Los restantes parámetros, p, l y g, generan un espacio (véase la figura 1.8) en el que cualquier computador BSP puede ser localizado en un punto.

1.5.2 El modelo de programación

El modelo formulado originalmente por Valiant [101] no hace ninguna presunción con respecto al tipo de máquina que representa el modelo BSP. En la práctica la implementación del modelo se ha realizado con posterioridad y se organiza en el formato de programación SPMD (simple programa, múltiples datos) en el que un sólo texto de programa se ejecuta en paralelo por un número arbitrario de procesadores. En el formato SIMD (simple instrucción, múltiples datos) todos los procesadores utilizan la misma instrucción al mismo tiempo mientras que en el formato SPMD todos los procesadores ejecutan el mismo programa pero no necesariamente la misma instrucción al mismo tiempo.

Sincronizaciones y superpasos

Un programa BSP no es más que una secuencia de instrucciones dividida en una serie de fases con la necesaria comunicación. Esta propuesta de programación paralela es aplicable a todo tipo de arquitecturas paralelas: máquinas con memoria distribuida, con memoria compartida, redes de estaciones de trabajo, etc. y proporciona una base consistente y general con la que desarrollar *software* portable para máquinas paralelas escalables.

Para garantizar cierto grado de determinismo, el modelo BSP utiliza sincronizaciones con el siguiente doble propósito:

- Asegurar que todos los procesadores llegan al mismo punto de ejecución del programa antes de que ningún procesador pueda continuar, lo que a menudo es llamado barrera de sincronización. El modelo propuesto por Valiant [101] supone que algunos de los procesadores pueden quedar excluidos de la sincronización (sincronización de un subconjunto de procesadores).

- Asegurar que todas las comunicaciones pendientes se realicen.

El conjunto de instrucciones entre dos sincronizaciones sucesivas se denomina superpaso. Durante un superpaso, el proceso paralelo corre independientemente, cada procesador trabaja con los datos de que disponía al iniciar dicho superpaso y puede iniciar requerimientos de lectura o escritura sobre datos remotos que podrán utilizarse en futuros superpasos. La figura 1.9 muestra un segmento de un programa BSP.

No existe ninguna suposición específica sobre la técnica que deben utilizar los computadores BSP para la sincronización entre procesadores. En general, una barrera de sincronización consta de dos fases:

- **Fase de reducción**, en la que todos los procesadores señalan que están listos para sincronizar (han llegado al mismo punto de ejecución del programa).

- **Fase de expansión** (*broadcast*), en la que, tras la fase de reducción, cada procesador señala que puede continuar.

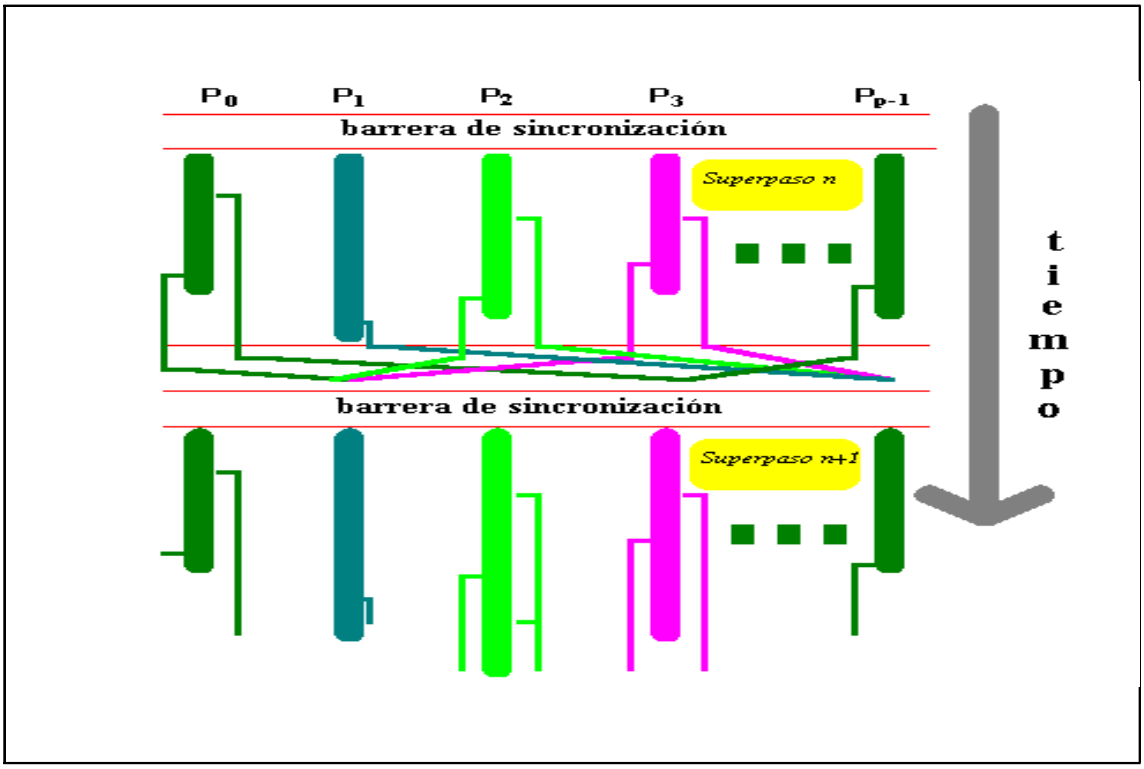


Figura 1.9: *Segmento de un programa BSP.*

Una barrera de sincronización puede ser implementada mediante *hardware* o mediante *software*:

- Mediante *hardware* cada procesador emite una señal cuando esta listo para sincronizarse, esa señal es recogida en un circuito eléctrico que determina el momento en el que (una vez completada la fase de reducción) cada procesador debe recibir la señal que le permite continuar con la ejecución del programa. Una solución sencilla consiste en conectar en forma de árbol diferentes puertas AND. Las señales de la fase de reducción, cuando los diferentes procesadores están listos para sincronizarse, se propagan a través del árbol de abajo hacia arriba. Una vez completada esta fase, se expande una señal desde la raíz del árbol hacia abajo.

- Mediante *software*, la sincronización se obtiene pasando mensajes entre los procesadores. Las dos fases de la barrera de sincronización, reducción y expansión, pueden realizarse de diversas formas. En general, la organización de los procesadores en forma de árbol resulta eficiente; esta organización implica que en la fase de reducción

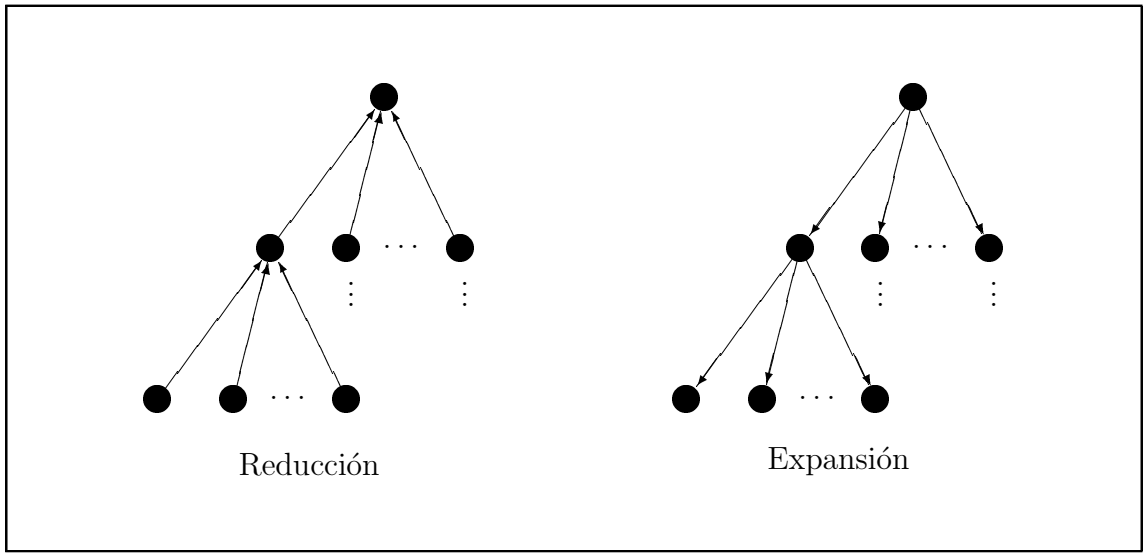


Figura 1.10: *Fases de una barrera de sincronización organizada en forma de árbol*

cada procesador envía un mensaje al nodo del que pende y en la fase de expansión cada procesador envía un mensaje a los nodos que penden de él.

En la figura 1.10 se muestra la organización en forma de árbol de una barrera de sincronización.

Comunicación

Los procesadores pueden leer o escribir en la memoria de otros procesadores durante la ejecución de un superpaso, este paradigma se conoce como DRMA (*Direct Remote Memory Acces*). No existe ninguna suposición adicional acerca de la llegada de datos, la única garantía que existe es que los datos llegarán antes del comienzo del siguiente superpaso; se establece para ello una regla de no interferencia que puede enunciarse de este modo: *Los datos utilizados en una comunicación y los que han cambiado por que se ha operado sobre ellos son disjuntos.* Los programas que respeten esta regla de no interferencia, tienen garantizado el 100% de determinismo en la comunicación entre procesadores.

Existe otra forma más determinista de comunicación basada en el paso de mensajes, aunque siempre dentro de las hipótesis del modelo BSP, que es conocida como BSMP (*Bulk*

Synchronous Message Passing) Al contrario que en el modo DRMA, en el que el procesador receptor no tiene en cuenta en los cálculos locales del superpaso la comunicación entrante del superpaso, en el modo BSMP los mensajes se envían y reciben explícitamente garantizándose que un mensaje enviado en un superpaso llega a su destino antes del inicio del siguiente superpaso; los mensajes son situados en cada procesador en una cola de entrada, pudiendo este recuperar el primer mensaje de la cola mediante la ejecución del comando apropiado.

1.5.3 El modelo de coste

Los parámetros definidos en la sección 1.5.1 cuantifican el rendimiento de un computador BSP y dejan la puerta abierta al análisis del coste de los programas BSP. Los parámetros s, g y l pueden ser utilizados para predecir el rendimiento de un programa BSP en cualquier computador BSP y para cualquier tamaño de problema (por supuesto, en la práctica existen limitaciones físicas).

Visto desde el nivel superior, un programa BSP está compuesto por una secuencia de superpasos por lo que el coste de ejecución del programa es la suma de los costes de cada uno de los superpasos. El coste de un superpaso es la suma de tres términos:

- El coste **aritmético**: máximo coste de computación local de cada procesador.
- El coste de **comunicación**: coste de comunicación global de la h-relación del superpaso.
- El coste de **sincronización** del final del superpaso.

Para un computador BSP con p procesadores (que se representan por P_i, para $i = 0, 1, \ldots, p-1$), el coste de un superpaso se puede expresar como

$$\begin{aligned} C &= \max_{0 \le i \le p-1} w_i + \left(\max_{0 \le i \le p-1} h_i \right) g + l \\ &= w + hg + l \end{aligned} \tag{1.1}$$

donde w_i es el coste de computación del procesador P_i en el superpaso y h_i es el máximo entre el número de unidades de datos que el procesador P_i envía o el número de unidades de datos que el procesador P_i recibe en el superpaso.

Se observa que el coste de un superpaso depende del procesador más lento, de la cantidad más alta de datos que un procesador envía o recibe y del coste de sincronización. Para que el coste de un programa BSP sea significativo y se pueda comparar entre distintas arquitecturas paralelas, los valores de w, g y l se expresan en función de la unidad de velocidad de computación (a menudo en flops).

La existencia de un modelo de coste maleable y preciso hace posible realmente el diseño de programas BSP, esto significa que consciente y justificadamente se puedan tomar decisiones sobre distintas implementaciones para un mismo problema. Por ejemplo, a la vista de la expresión (1.1) está claro que para crear programas BSP eficientes deben tenerse en cuenta las siguientes estrategias:

- La computación en cada superpaso debe estar balanceada ya que la sincronización no se realiza hasta que acaba el procesador que más tarda en realizar la computación que se le haya asignado.
- La comunicación entre procesadores debe estar balanceada, evitando grandes variaciones en los valores de h_i.
- El número de superpasos debe ser minimizado debido al coste que supone cada sincronización.

El modelo de coste puede ser utilizado también para predecir el coste de ejecución en distintos computadores BSP. Los valores de w y h para cada superpaso y el número de superpasos pueden obtenerse mediante inspección del programa BSP, sujeta a las mismas limitaciones que se tengan a la hora de determinar el coste de un programa secuencial. Los valores de p, g y l pueden ser sustituidos en la fórmula de coste del programa BSP y estimar así el tiempo que tardará antes de ejecutarlo en la máquina paralela.

Existen otras propuestas del modelo de coste BSP en las que se afina en algunos detalles. Por ejemplo,

- si se consideran comunicación y computación solapadas, el coste de un superpaso viene expresado por una fórmula del tipo

$$\max(w, hg) + l;$$

- si se considera que en el coste de una h-relación influye el tiempo utilizado para enviar datos y para recibirlos (lo que en definitiva supone que una h-relación es un patrón de comunicación global en el que cada procesador puede enviar y recibir a lo sumo h unidades de datos), el coste de un superpaso viene expresado por una fórmula del tipo

$$w + \max_{0 \leq i \leq p-1} (h_i^{enviadas} + h_i^{recibidas})g + l.$$

Estas diferenciaciones sobre el modelo de coste sólo producen pequeñas variaciones en el coste de un programa BSP, por lo que en esta memoria se ha optado por utilizar la expresión (1.1).

Una omisión más importante en el modelo de coste es la restricción que supone la cantidad de memoria requerida por cada procesador. El modelo obliga a balancear las comunicaciones y a minimizar las sincronizaciones pero deja rienda suelta al uso (y abuso) de la memoria, pudiendo darse el caso de que en un programa se haya hecho una distribución uniforme de la computación y la comunicación pero que la eficiencia del mismo se vea mermada por que alguno de los procesadores no tenga suficientes recursos de memoria. Se está investigando una extensión del modelo de coste en el que se tenga en cuenta la cantidad de memoria asociada a cada procesador, para obtener mayor información véase Hill, McColl y Skillicorn [59].

El modelo de coste también permite el diseño de algoritmos BSP, no sólo programas. En este caso el objetivo es obtener soluciones óptimas con respecto a la computación total, la comunicación total y en número total de superpasos para el posible rango de valores de p.

Un modelo refinado para el parámetro g

En el modelo original de Valiant [101] los parámetros s, l y g son constantes. Ya se ha comentado (página 19) que el modelo de coste no distingue entre el coste de un procesador enviando h mensajes de tamaño una unidad de datos o un mensaje de tamaño h unidades de datos, en ambos casos se tiene una h-relación de coste hg. Aunque este modelo de coste es bastante preciso para tamaños de mensaje grandes, no se ajusta al caso en que se producen comunicaciones de mensajes de pequeño tamaño debido al tiempo

de inicialización (*start-up*) en la transmisión de mensajes. Para tamaños pequeños de mensaje este tiempo representa un alto porcentaje del tiempo de comunicación mientras que para mensajes grandes es irrelevante.

Miller [89], utilizando una técnica introducida por Hockney [63], refina el modelo de coste original de manera que se incluya el efecto de la granularidad de los mensajes en el coste de comunicación. Para ello, se define el valor de g como una función que depende del tamaño x de mensaje,

$$g(x) = \left(\frac{n_{\frac{1}{2}}}{x} + 1 \right) g_\infty, \tag{1.2}$$

donde g_∞ es el coste asintótico de comunicación para tamaños de mensaje muy grandes y $n_{\frac{1}{2}}$ es el tamaño de mensaje que produce la mitad del ancho de banda asintótico, o sea $g(n_{\frac{1}{2}}) = 2g_\infty$. En la figura 1.11 se muestra un ejemplo de la curva (conocida como curva de Miller) que se obtiene con la expresión (1.2) para los valores $n_{\frac{1}{2}} = 12$ unidades de datos y $g_\infty = 0.23\mu s$.

Estimación de los valores de los parámetros BSP

Los valores de los parámetros BSP de una máquina paralela concreta, se pueden obtener utilizando programas de prueba (proceso conocido como *benchmarking*) que evalúen el rendimiento de las partes del sistema que interesen.

En la estimación del parámetro s influye fuertemente el tipo de cálculo que se realice en el programa de prueba, se suele tomar la media aritmética de dos estimaciones para este parámetro:

- $\lfloor s \rfloor$, que mide el coste de un producto escalar en el que se realizan $O(n)$ operaciones aritméticas sobre estructuras de tamaño n y proporciona una cota inferior del valor de s;

- $\lceil s \rceil$, que mide el coste del producto de dos matrices densas en el que se realizan $O(n^3)$ operaciones aritméticas sobre estructuras de tamaño n^2 y proporciona una cota superior del valor de s.

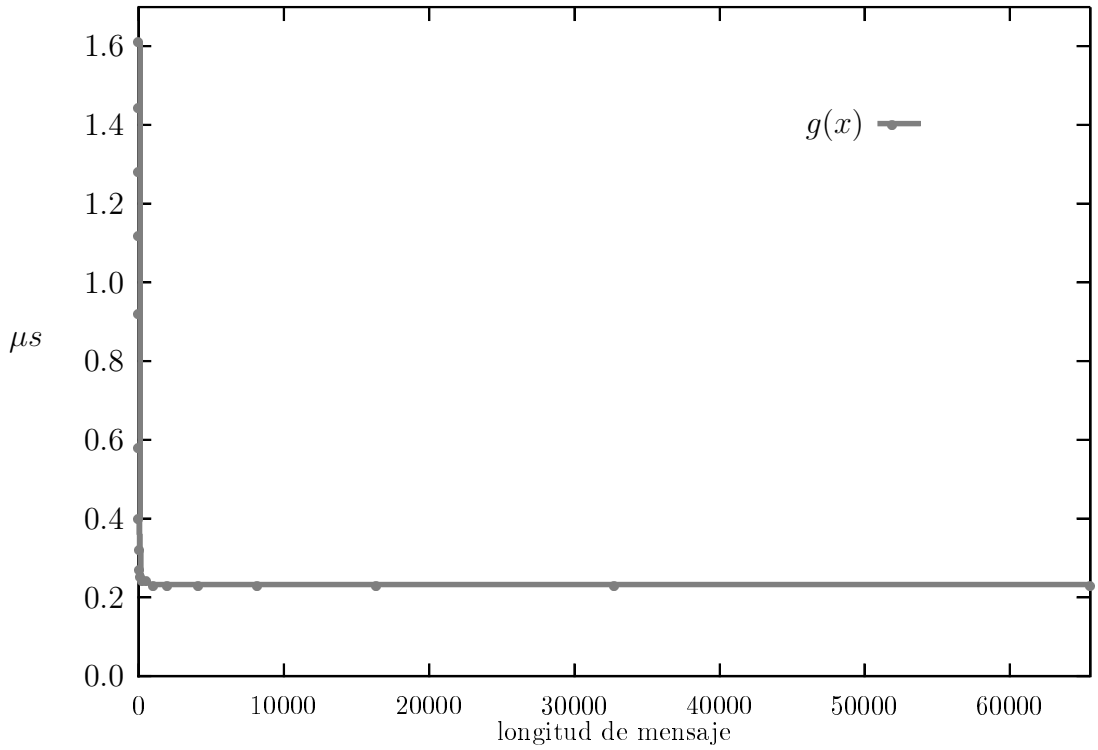


Figura 1.11: *Curva teórica de g para los valores* $n_{\frac{1}{2}} = 12$ *y* $g_\infty = 0.23\mu s$.

Si se quieren obtener predicciones no muy alejadas de la realidad, es mejor estimar el valor de s con el tipo de cálculo predominante en el algoritmo. Las predicciones más precisas se obtienen utilizando el valor de s efectivo para cada algoritmo concreto, este valor se puede obtener dividiendo el número de operaciones aritméticas que se realicen en el algoritmo secuencial y el tiempo real que se tarde en ejecutar el mismo en secuencial.

El parámetro l puede ser estimado realizando un cierto número de veces una sincronización entre todos los procesadores y tomando como valor efectivo del mismo el promedio de los valores obtenidos o bien el valor mínimo de todos ellos.

Si se realiza una estimación de g con un patrón de comunicación particularmente sencillo: una 1-relación con una comunicación cíclica entre procesadores vecinos, se obtiene una cota superior del valor efectivo de g. En el otro extremo, si se realiza una estimación de g con un patrón de comunicación en el que todos los procesadores comunican a todos h unidades de datos (véase la figura 1.12), se obtiene una cota inferior del valor efectivo de

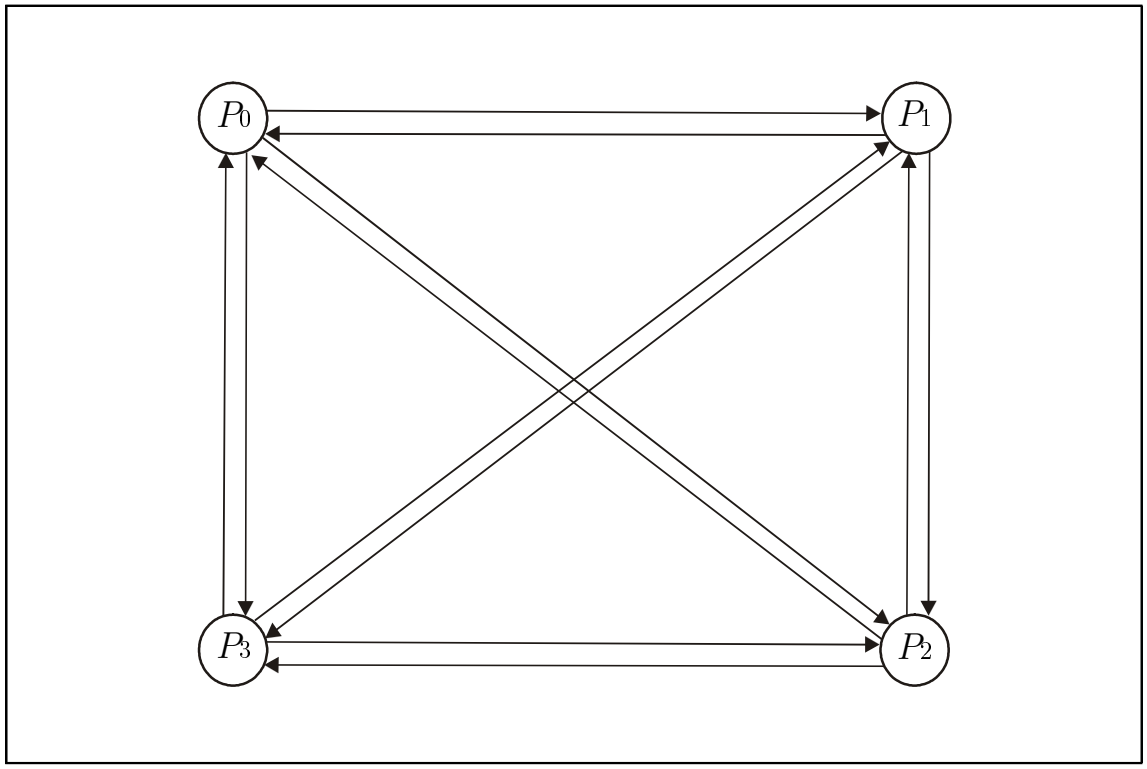


Figura 1.12: *Patrón de comunicación todos a todos para 4 procesadores.*

g. Para una mayor precisión en la predicción, la estimación de g se debe realizar teniendo en cuenta el patrón o los patrones de comunicación que se utilicen en el programa.

Parámetros BSP en las plataformas de experimentación

Los resultados experimentales de los métodos que se expondran en los siguientes capítulos han sido obtenidos en dos máquinas: el IBM SP2 de la Universidad de Alicante y el *cluster* de PC's del grupo de Computación Paralela de esa universidad.

El IBM SP2 está formado por 8 nodos *thin* cuyas características son:

- Procesador: POWER2 Super Chip 120 MHz
- L1 cache: 128 Kilobytes datos/32 Kilobytes instrucciones
- RAM: 256 Megabytes

- Anchura del bus de memoria: 256 bits
- Velocidad del bus: 160 Megabytes/seg
- Sistema operativo: AIX 4.1.5

El *cluster* está formado por 7 PC's, conectados mediante un *switch* con un ancho de banda de hasta 100 Megabytes/seg, cuyas características son:

- Procesador: Pentium 120 MHz
- RAM: 64 Megabytes
- Sistema operativo: LINUX 2.2.5

Los parámetros obtenidos se muestran en la tabla 1.1. Se ha utilizado la librería BSPLib (v1.3) que proporciona una utilidad: `bsp_probe`, para el cálculo de parámetros. Como unidad de datos se ha considerado palabras de 32 bits.

El cálculo de s se ha realizado teniendo en cuenta el tipo de cálculo predominante en los algoritmos que se describirán en sucesivos capítulos, por lo que se ha medido el número de Megaflops que puede realizar el procesador más lento resolviendo un sistema tridiagonal por el método de eliminación de Gauss. Con `bsp_probe` se obtienen resultados para productos escalares y producto de dos matrices densas; se ha observado que en el IBM SP2 no hay mucha diferencia entre ambos valores, en cambio en el *cluster* sí. Además el valor obtenido para el método de eliminación de Gauss para sistemas tridiagonales está más próximo al obtenido para el producto escalar. Los valores obtenidos se muestran en la tabla 1.2.

En el cálculo de g se ha considerado el patrón de comunicación que básicamente siguen las tres métodos que se describirán y que consiste en una expansión (*broadcast*) y una reducción; en lo sucesivo se hará referencia a él como patrón **trid**. Se ha tenido en cuenta la granularidad de los mensajes transmitidos, para ello se ha medido el tiempo total que tarda la máquina en comunicar todos los mensajes para la máxima h-relación soportada, varian el tamaño de mensaje desde bloques de tamaño 2 hasta el máximo tamaño para la h-relación. Dividiendo el tiempo total que se tarda en comunicar todas las palabras por el número total de palabras comunicadas se obtiene el tiempo $g(x)$ que tarda en

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	26
	2	3294	9.5	25
	4	5366	12.4	25
	6	8164	12.5	25

(a) *IBM SP2 switch*

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	8
	2	20235	709.7	3
	4	54163	1362.6	9
	6	121958	3211.2	9

(b) *IBM SP2 ethernet*

s	p	l	g	$n_{\frac{1}{2}}$
16.4	1	23	0.2	22
	2	2556	6.9	5
	4	5152	7.4	4
	6	7538	6.8	4

(c) *Cluster de PC's*

Tabla 1.1: *Valores de parámetros BSP.*

	Escalar	Gauss trid.	Matricial
IBM SP2	43.73	44.98	49.39
CLUSTER	11.95	16.44	42.72

Tabla 1.2: *Megaflops obtenidos para un IBM SP2 y un cluster de PC's.*

comunicarse una palabra para cada uno de los tamaños de bloque x. Con `bsp_probe` se obtienen resultados para dos patrones de comunicación: comunicación cíclica (*local shift*) y una comunicación todos a todos (*all-to-all*). Sobre los valores obtenidos para el patrón de comunicación específico (trid) de los métodos que se estudiarán, se ha realizado un ajuste teórico haciendo uso de la expresión (1.2), introducida por Miller. Los valores obtenidos para g_∞ (normalizados con respecto a s y expresados en flops) y $n_{\frac{1}{2}}$ figuran en la tabla 1.4.

En la tabla 1.3 se muestran los valores obtenidos (normalizados con respecto a s y expresados en flops) para 4 procesadores en un IBM SP2 utilizando *switch*. En la columna trid se han escrito los valores obtenidos para el patrón de comunicación de los métodos de resolución de sistemas tridiagonales que se describirán en sucesivos capítulos.

En las figuras 1.13 y 1.14 se muestran los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación trid y de la curva teórica de Miller en un IBM SP2 con 4 procesadores interconectados mediante *switch*. Se observa que para tamaños de mensaje pequeños el valor experimental está por encima del teórico, y que ambos valores se van aproximando a medida que aumenta el tamaño de mensaje. En este caso se produce un pequeño crecimiento en el coste experimental de comunicación de una palabra de 32 bits para tamaños de mensaje desde 64 hasta 256 palabras, como se puede apreciar en la figura 1.14(a).

Para 2 y 6 procesadores, en el IBM SP2 *switch*, se obtienen resultados análogos, con mayor o menor desviación de los valores experimentales sobre la curva teórica de Miller en tamaños de mensaje pequeños. A modo de ejemplo, en las figuras 1.15 y 1.16 se muestran los resultados para 2 procesadores en un IBM SP2 *switch*. Finalmente, en las figuras 1.17 y 1.18 se muestran los valores $g(x)$ experimentales, recogidos en la tabla 1.3, de los tres patrones en un IBM SP2 para 4 procesadores interconectados mediante *switch*. La curva teórica de Miller que aparece corresponde al patrón trid, para los otros patrones se puede realizar un mejor ajuste teórico tomando los valores g_∞ y $n_{\frac{1}{2}}$ de la tabla 1.4.

1.5.4 Librerías BSP disponibles

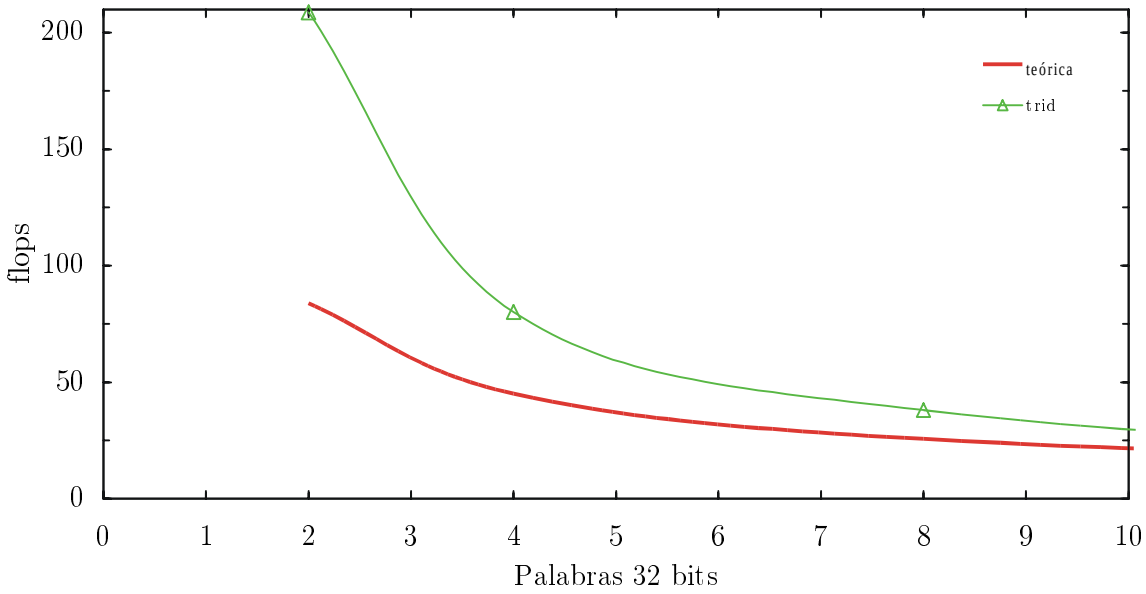
Desde que Valiant [101] propuso el modelo BSP en 1990, varias son las aproximaciones que se han realizado a la implementación de computadores BSP por medio de librerías.

Tamaño	Teórico	*local*	trid	*all-to-all*
2	83.91	678.32	208.71	63.75
4	45.06	259.97	80.06	24.68
8	25.64	116.46	38.05	18.84
16	15.93	52.27	18.83	14.68
32	11.07	26.82	11.36	13.57
64	8.64	16.96	8.24	12.34
128	7.43	16.35	11.06	22.97
256	6.82	10.71	7.62	16.40
512	6.52	8.18	6.56	12.32
1024	6.37	6.04	6.26	10.59
2048	6.29	5.57	6.25	10.49
4096	6.25	5.57	6.24	10.11
8192	6.23	5.32	6.23	11.11
16384	6.23	5.10	6.23	10.11
32768	6.22	5.00	6.22	10.11
65536	6.22	5.00	6.22	10.11
131072	6.22	5.00	6.22	10.11

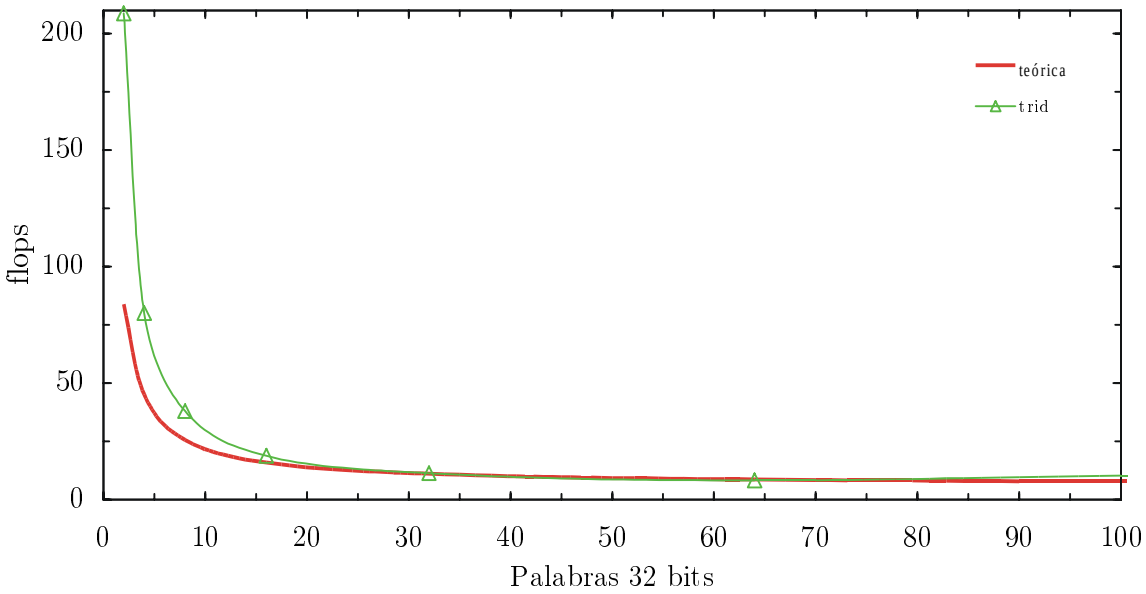
Tabla 1.3: *Coste de comunicación en flops de una palabra de 32 bits para 4 procesadores en un IBM SP2 utilizando switch y distintos patrones de comunicación.*

	local	trid	*all-to-all*
g_∞	5.00	6.20	10.11
$n_{\frac{1}{2}}$	269	25	11

Tabla 1.4: *Valores de g_∞ y $n_{\frac{1}{2}}$ para los patrones local,* trid y *all-to-all* en un IBM SP2 con 4 procesadores interconectados mediante *switch*.

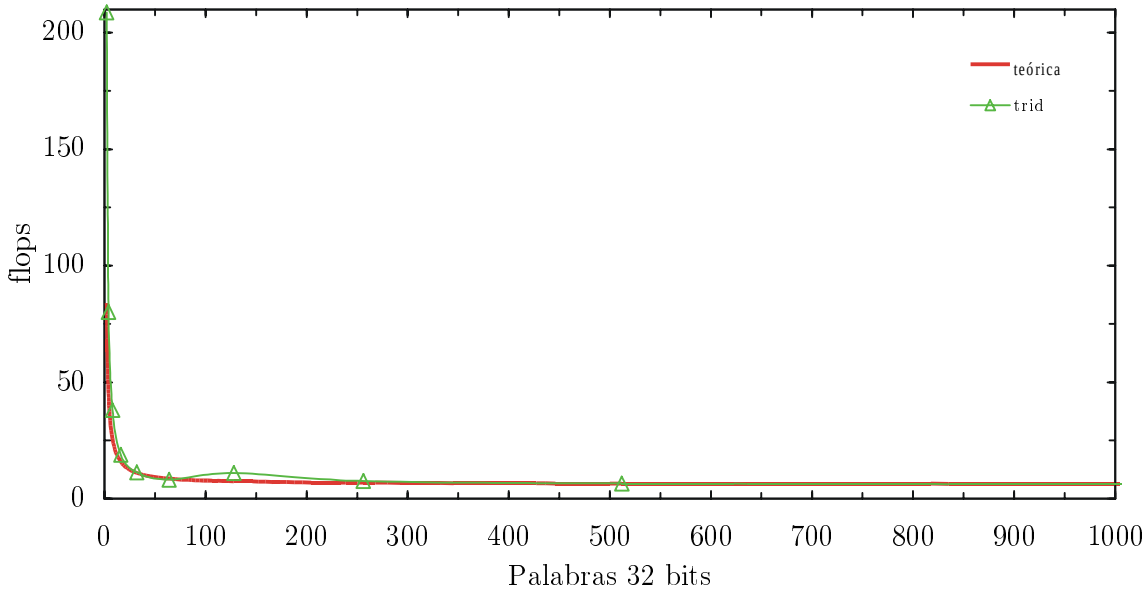


(a) *Tamaños de mensaje x con $0 \leq x \leq 10$*

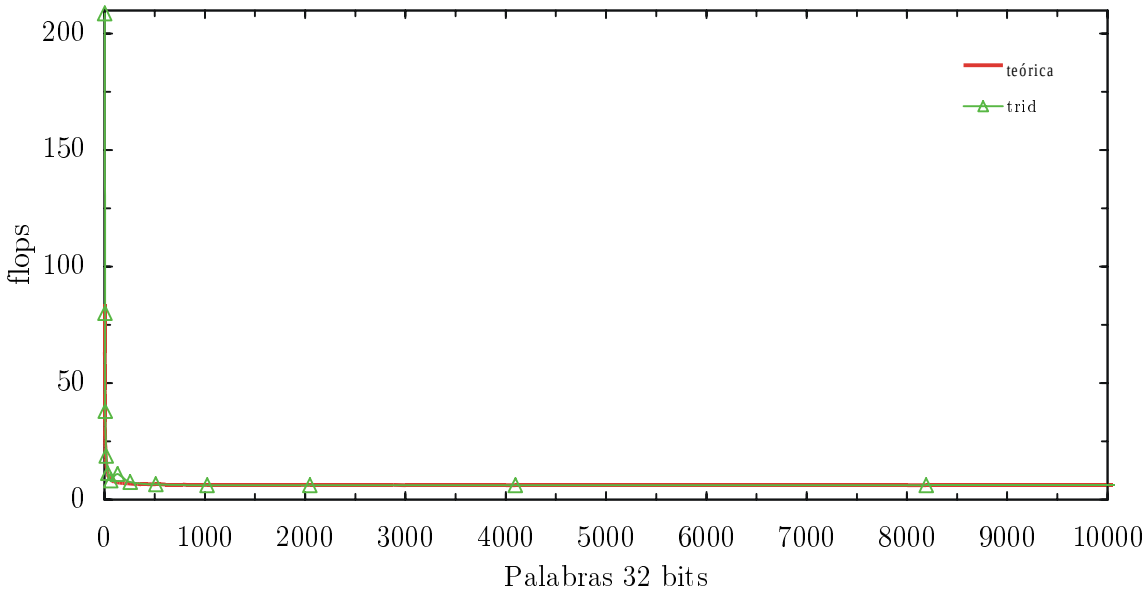


(b) *Tamaños de mensaje x con $0 \leq x \leq 100$*

Figura 1.13: *Comparación de los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación* trid y de la curva teórica de Miller en un IBM SP2 con 4 procesadores interconectados mediante *switch*.

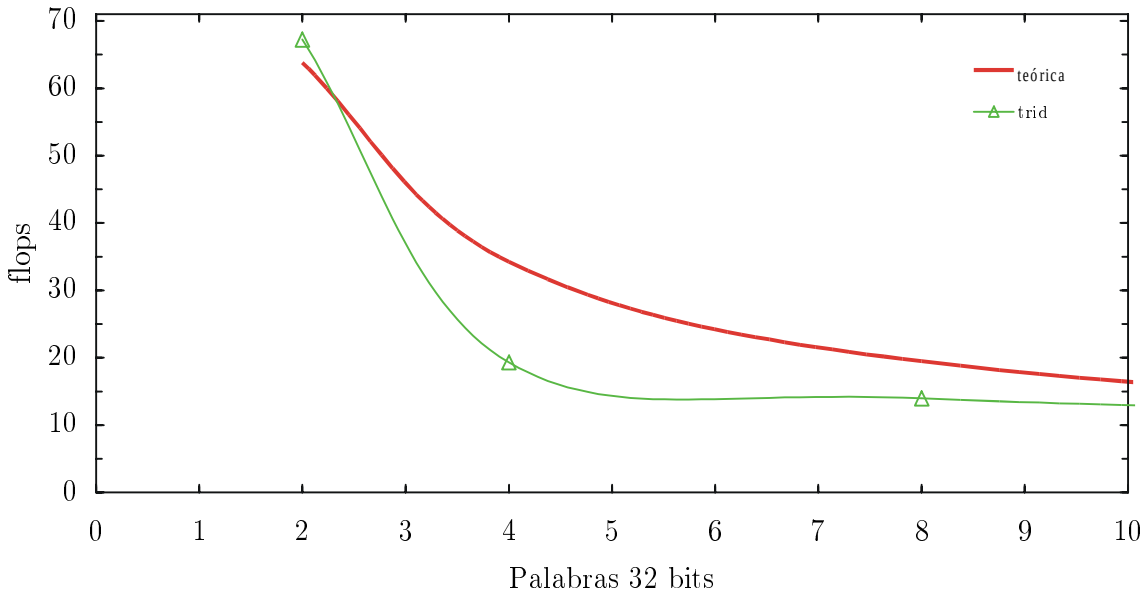


(a) *Tamaños de mensaje x con $0 \leq x \leq 1000$*

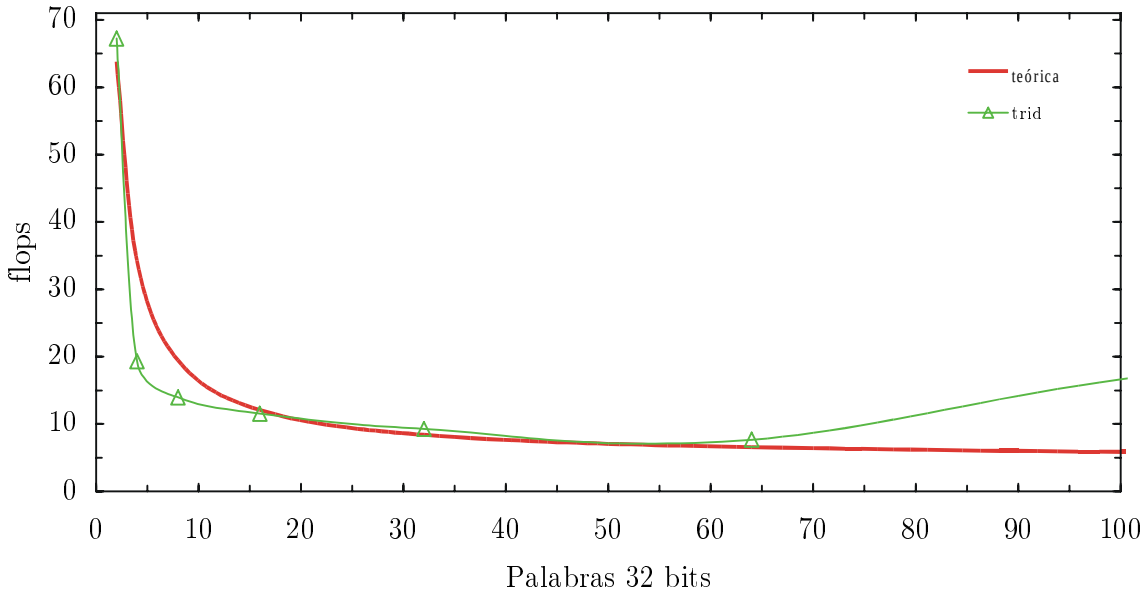


(b) *Tamaños de mensaje x con $0 \leq x \leq 10000$*

Figura 1.14: *Comparación de los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación* trid y de la curva teórica de Miller en un IBM SP2 con 4 procesadores interconectados mediante *switch*.

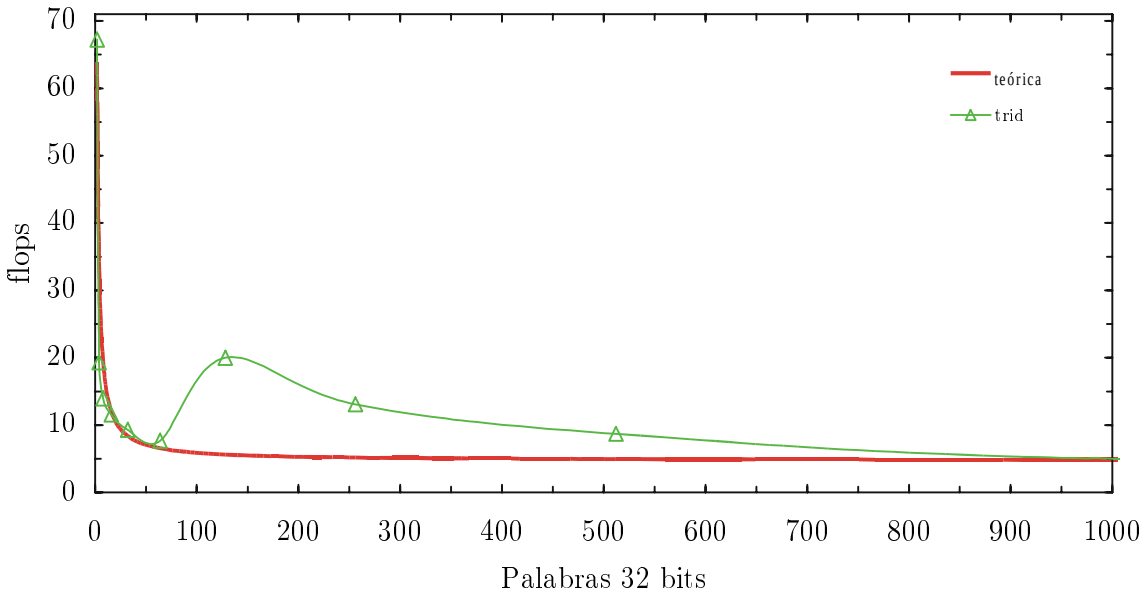


(a) *Tamaños de mensaje x con $0 \leq x \leq 10$*

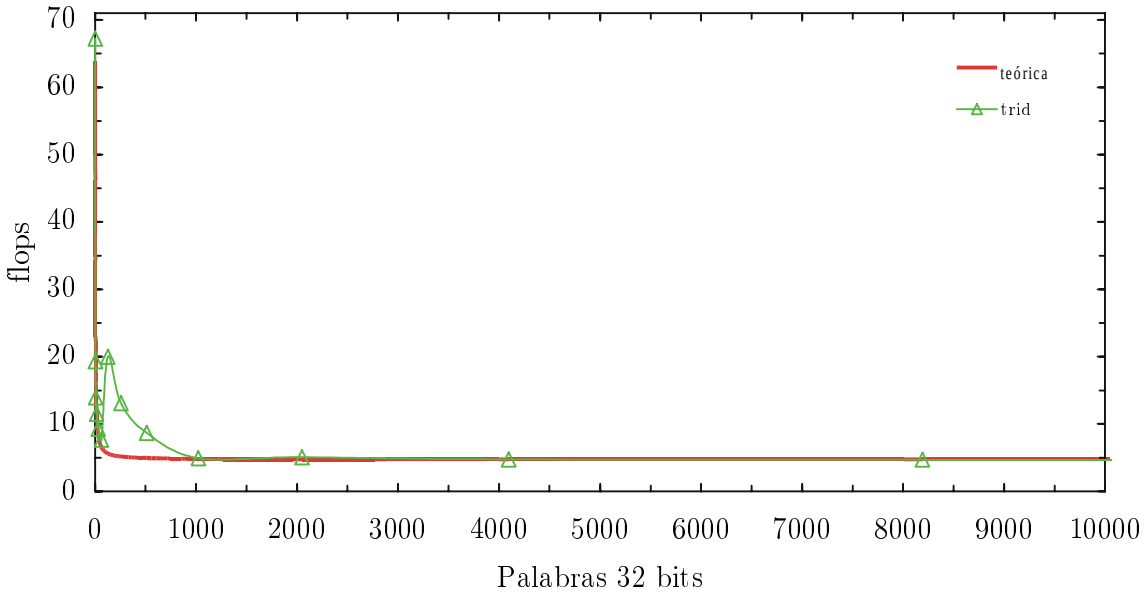


(b) *Tamaños de mensaje x con $0 \leq x \leq 100$*

Figura 1.15: *Comparación de los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación* trid y de la curva teórica de Miller en un IBM SP2 con 2 procesadores interconectados mediante *switch*.

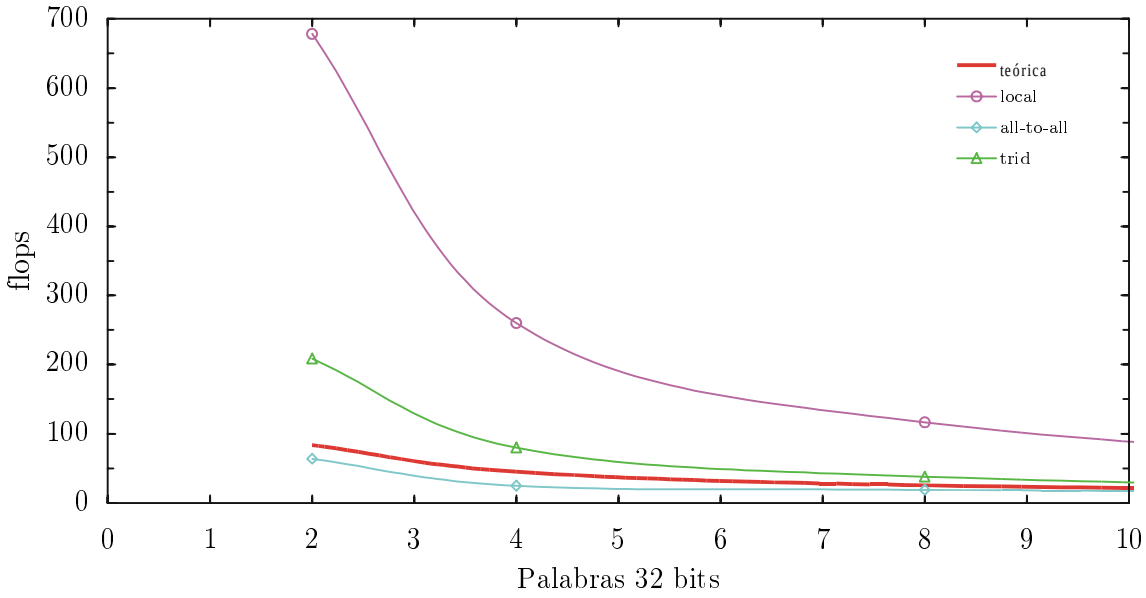


(a) *Tamaños de mensaje x con $0 \leq x \leq 1000$*

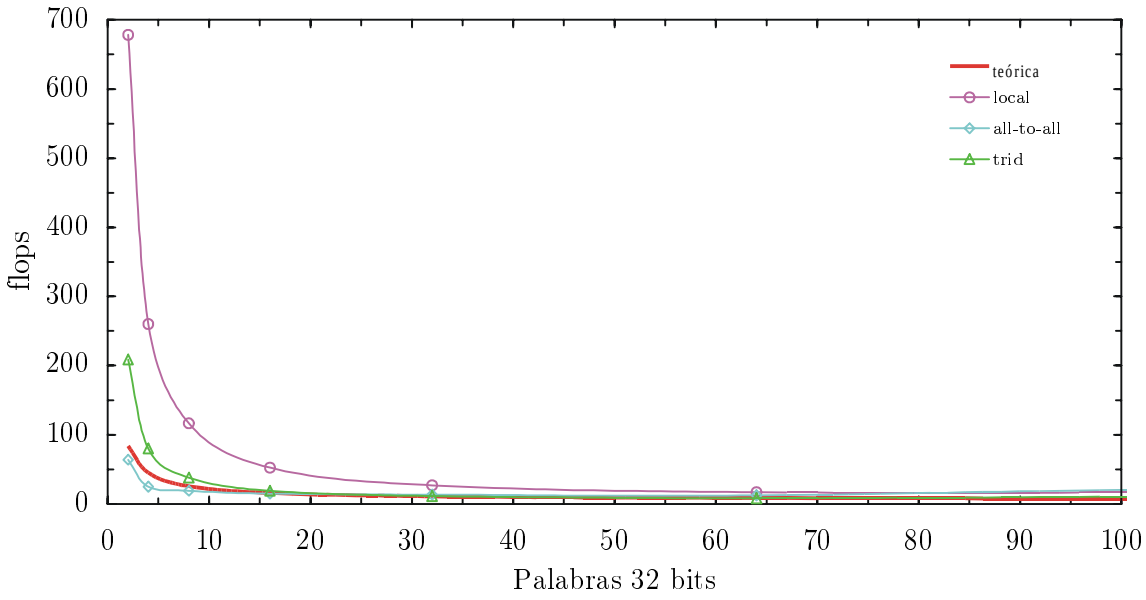


(b) *Tamaños de mensaje x con $0 \leq x \leq 10000$*

Figura 1.16: *Comparación de los valores $g(x)$ obtenidos experimentalmente con el patrón de comunicación* trid y de la curva teórica de Miller en un IBM SP2 con 2 procesadores interconectados mediante *switch*.

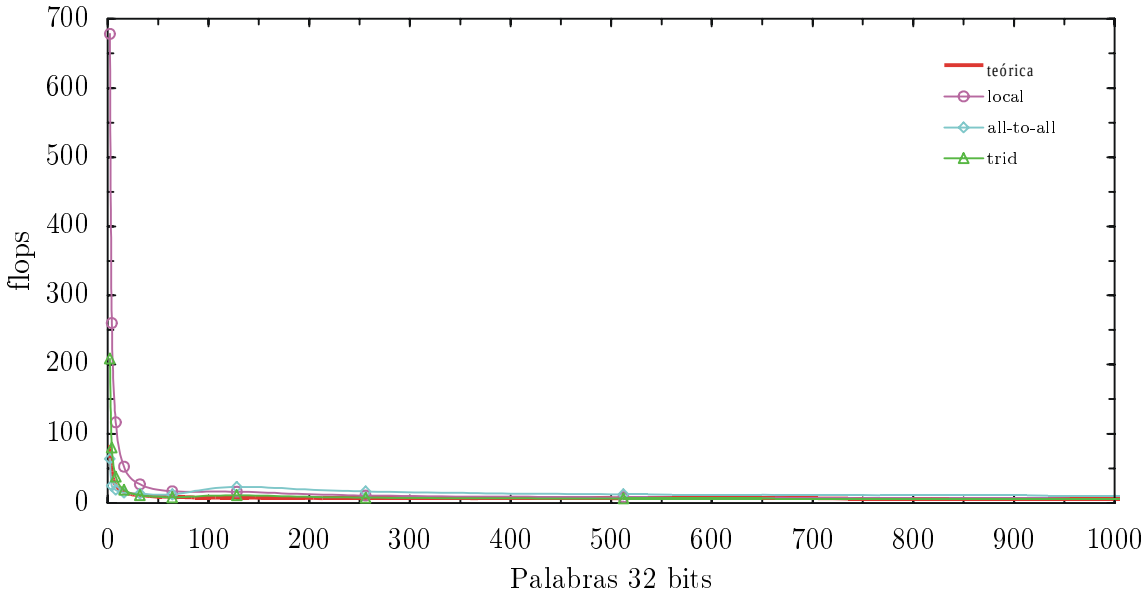


(a) *Tamaños de mensaje x con $0 \leq x \leq 10$*

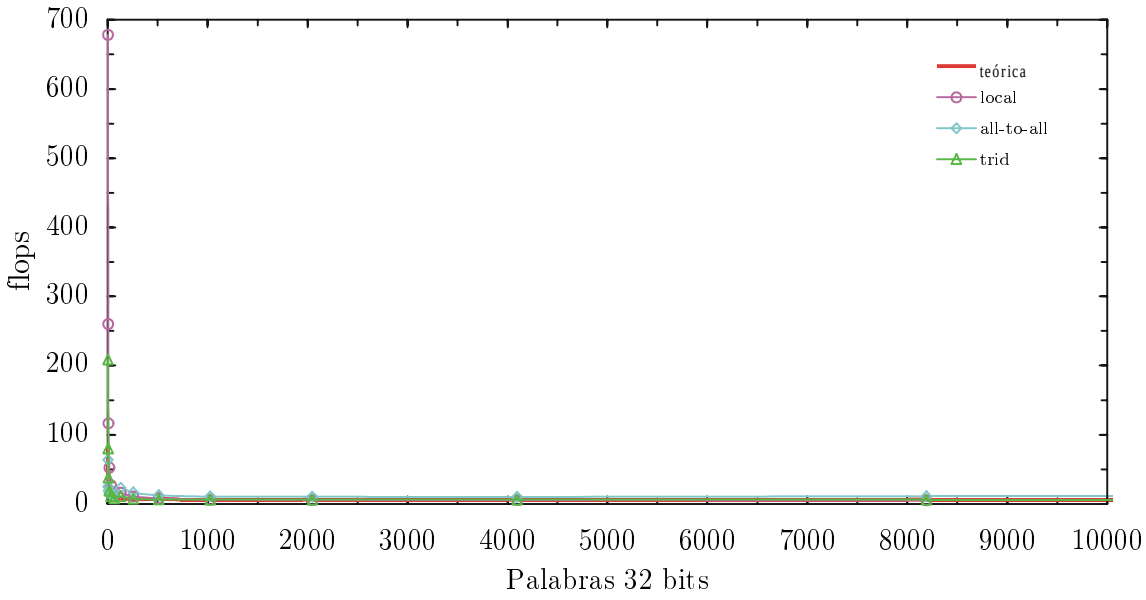


(b) *Tamaños de mensaje x con $0 \leq x \leq 100$*

Figura 1.17: *Comparación de los valores $g(x)$ obtenidos experimentalmente con los patrones de comunicación local,* trid *y all-to-all* en un IBM SP2 con 4 procesadores interconectados mediante *switch*. La curva teórica de Miller está ajustada al modelo trid.



(a) *Tamaños de mensaje x con $0 \leq x \leq 1000$*



(b) *Tamaños de mensaje x con $0 \leq x \leq 10000$*

Figura 1.18: *Comparación de los valores $g(x)$ obtenidos experimentalmente con los patrones de comunicación local,* trid *y all-to-all* en un IBM SP2 con 4 procesadores interconectados mediante *switch*. La curva teórica de Miller está ajustada al modelo trid.

Muchas de estas librerías son una extensión de algún lenguaje de programación ya existente como C/C++ o FORTRAN. Esta extensión consiste generalmente en funciones generadoras de tareas, funciones de comunicación entre procesadores y funciones de sincronización. Las librerías permiten al programador controlar plenamente la comunicación y la sincronización entre los procesadores lógicos sin necesidad de conocer la asignación física que se realiza a cada procesador lógico ni el tipo de arquitectura de la máquina en la que se ejecuta el programa, hecho que es conocido como programación BSP en modo directo.

Las librerías BSP que se describen a continuación han sido implementadas para una amplia gama de plataformas de computación paralela respetando el modo directo de programación.

Librería Oxford BSP

El modelo de Valiant [101] fue desarrollado más extensamente por McColl [83, 84], consecuencia de ello, en 1993, se llevo a cabo la primera librería para crear programas BSP. Esta primera librería (véanse Miller [88, 89] y Miller y Reed [90]) contiene sólo un pequeño conjunto de funciones, dos de ellas se encargan de delimitar el programa BSP: `bsp_start` que contiene un parámetro indicativo del número de procesadores a utilizar por el programa y `bsp_finish` que indica el final del programa. Otras dos se encargan de la comunicación: `bsp_fetch` y `bsp_store`. Es la única forma de comunicación posible. Los programas deben respetar la regla de no interferencia. Finalmente, otras dos funciones se encargan de la delimitación de los superpasos: `bsp_sstep` y `bsp_sstep_end`. La librería tambien soporta dos patrones de comunicación: expansión y reducción. Es utilizable desde C o FORTRAN en numerosas plataformas como CRAY T3D, SGI Power Challenge, IBM SP1 y cluster de estaciones de trabajo RS/6000 utilizando TCP/IP. La librería sólo soporta el modo de comunicación DRMA y se supone que la localización de las variables es la misma para cada procesador, esto conlleva que todas las variables son estáticas.

Librería BSP++

Es una extensión de la librería Oxford BSP, creada en Oxford en 1994, que soporta porgramación orientada a objetos. Tiene pocas características que distingan su lenguaje

Función	Tarea
`void bspSendPkt()`	Envía un paquete a otro procesador.
`bspPkt *bspGetPkt()`	Recibe un paquete enviado en el anterior superpaso.
`void bspSynch()`	Lleva a cabo una sincronización global.
`int bspGetNumProcs()`	Devuelve el número de procesadores.
`int bspGetNumPkts()`	Devuelve el número de paquetes enviados al procesador en el anterior superpaso, a los que no se ha accedido todavía.
`int bspGetNumStep()`	Devuelve el número del actual superpaso.

Tabla 1.5: *Funciones de la librería Green BSP.*

del de otras, debido al incremento en la complejidad del modelo de coste que acarrea (véase Lecomber [75]).

Librería Green BSP

Esta librería, desarrollada en torno a 1995 en *University of Central Florida* (Orlando), utiliza el paso de mensajes para la comunicación, trabaja sólo con el paradigma de comunicación BSMP que garantiza la llegada de los mensajes antes del inicio del siguiente superpaso, soporta diversos sistemas operativos y librerías en C. Consta de las siete funciones que se incluyen en la tabla 1.5, los paquetes consisten en un array de un tamaño prefijado (generalmente 16 bytes) por lo que los datos que vayan a ser enviados a otros procesador deben ser particionados. Véase Goudreau, Lang, Rao y Tsantilas [53] para obtener más detalles.

Libreria BSP Worldwide Standard (BSPLib)

Esta librería, fruto de la colaboración entre las personas que han estado detrás del diseño y desarrollo de las librerías Oxford BSP y Green BSP, es el resultado de un considerable esfuerzo por conseguir un estándar de programación en BSP. En el proyecto

participan miembros de las Universidades de Oxford, Harvard, Suffolk, Central Florida, California (Berkeley), Columbia, Utrech y del NEC Research Institute of Pricenton. La primera propuesta [56] fue presentada en abril de 1996 y la última versión [38] en septiembre de 1998.

Debido a su origen (surge a partir de las liberías Oxford BSP y Green BSP), soporta los dos paradigmas de comunicación: DRMA y BSMP. BSPlib proporciona además del conjunto de operaciones básicas que se muestra en la tabla 1.6 otras funciones que permiten realizar comunicaciones para algunos patrones especiales, estas funciones no se consideran como parte del núcleo de la librería por que se pueden obtener sencillamente a partir de las operaciones básicas.

Los procesos son generados por medio de `bsp_begin` y `bsp_end`. Si estas instrucciones son la primera y última del programa, éste se ejecuta desde el principio en la forma SPMD. BSPLib proporciona un modo alternativo consistente en que el procesador principal inicia secuencialmente la ejecución del programa y determina el número de procesadores necesarios para la computación en paralelo (para ello se hace uso de la función `bsp_init`); a continuación se generan los procesos necesarios utilizando `bsp_begin` y continúa la ejecución en la forma SPMD hasta que `bsp_end` es ejecutado por todos los procesadores; en este punto todos los procesadores terminan su tarea excepto el procesador principal que puede continuar la ejecución del programa secuencialmente.

La librería está disponible para C y FORTRAN, su última versión (v1.4) proporciona cuatro tipos de inplementación para los siguientes tipos de máquina:

(i) Máquina con memoria distribuida que utiliza su propia librería de paso de mensajes o MPI.

(ii) Máquina con memoria distribuida que utiliza primitivas de comunicación unidireccionales, como la librería Cray SHMEM de T3E.

(iii) Multiprocesador con memoria compartida que utiliza sus propias primitivas de concurrencia o semáforos System V.

(iv) Redes de estaciones de trabajo que utilizan el protocolo de comunicación TCP/IP o UDP/IP.

Ha sido probada satisfactoriamente en las siguientes plataformas

Clase	Operación	Significado
Inicialización	`bsp_init`	Simula procesos dinámicos
	`bsp_begin`	Inicia un programa.
	`bsp_end`	Finaliza un programa.
Indagación	`bsp_pid`	Identifica el procesador.
	`bsp_nprocs`	Número de procesadores.
	`bsp_time`	Tiempo local.
Sicronización	`bsp_sync`	Barrera de sincronización.
DRMA	`bsp_pushregister`	Crea una zona de memoria globalmente visible.
	`bsp_popregister`	Borra una zona de memoria globalmente visible.
	`bsp_put`	Envía a memoria remota.
	`bsp_get`	Recibe de memoria remota.
BSMP	`bsp_set_tag_size`	Elige el tamaño de etiqueta.
	`bsp_send`	Envía a una cola remota.
	`bsp_get_tag`	Empareja etiqueta y mensaje.
	`bsp_move`	Extrae de la cola.
Parada	`bsp_abort`	Un procesador para a todos.
Alto rendimiento	`bsp_hpput`	Versión de las primitivas de
	`bsp_hpget`	comunicación sin *buffer* local
	`bsp_hpmove`	ni remoto.

Tabla 1.6: *Núcleo de la librería BSPLib.*

- Silicon Grafics Power Challenge (Irix 6.x)
- Silicon Grafics Origin 2000 (Irix 6.x)
- IBM SP2
- Cray T3D
- Cray T3E
- Parsytec Explorer
- Transtec Paramid
- Convex SPP
- Hitachi SR2001
- Cluster de estaciones de trabajo Silicon Grafics (Irix 5.x) o (Irix 5.x, -n32 *objects*)
- SunOS 4.1.x
- Cluster de estaciones de trabajo IBM RS/6000
- Cluster de estaciones de trabajo Solaris
- Cluster de PC's bajo LINUX o Windows NT.

Para obtener mayor información véanse Bisseling, Goudreau, Hill, Lang, McColl, Rao, Stefanescu, Suel y Tsantilas [15], Donaldson, Hill y McEwan [38] y Goudreau, Hill, Lang, McColl, Rao, Stefanescu, Suel y Tsantilas [52].

Libreria PUB

La librería PUB (*Paderborn University BSP-Library*) ha sido desarrollada en la Univesidad de Paderborn (Alemania) recientemente: enero de 1999. Ofrece las dos posibilidades de comunicación entre procesadores: paso de mensajes y acceso a memoria remota, y proporciona algunas operaciones de comunicación colectiva como difusión y reducción de mensajes. Está inplementada en C y, para ser más flexible, permite crear objetos BSP

independientes, cada uno de ellos representando un computador BSP virtual; además, se puede utilizar BSPLib para programar con PUB.

La librería está disponible para varias plataformas

- Parsytec Gigacluster - GCel (Parix)
- Parsytec GC/PowerPlus (Parix)
- Parsytec CC (Embedded Parix)
- IBM SP2 (MPL o MPI)
- Cluster de estaciones de trabajo Solaris (TCP/IP)
- Cluster de PC's bajo LINUX (TCP/IP)

y puede adaptarse fácilmente a cualquier plataforma que disponga de MPI. Contiene también, un simulador BSP para poder ejecutar programas paralelos y depurarlos en una estación de trabajo o PC dotados de sistema operativo Sun Solaris, Windows 32 o IBM OS/2. Una comparación con otras librerías BSP y más detalles sobre la misma pueden encontrarse en Bonorden, Juurlink, von Otte y Rieping [18].

Capítulo 2

Método de las particiones superpuestas

2.1 Preliminares

En general, se utilizará la siguiente notación para una matriz tridiagonal A de tamaño $n \times n$

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix}. \tag{2.1}$$

Se dice que una matriz tridiagonal A de tamaño $n \times n$ es **estrictamente diagonal dominante** si

$$|a_i| > |b_i| + |c_i|, \text{ para } i = 1, 2, \ldots, n; \tag{2.2}$$

donde $c_1 = b_n = 0$.

Si $b_i \neq 0$ y $c_{i+1} \neq 0$, para $i = 1, 2, \ldots, n-1$ la matriz es **irreducible**.

Es conocido que si A es una matriz tridiagonal, estrictamente diagonal dominante, entonces es no singular y se puede realizar la eliminación Gaussiana sin intercambio de filas en cualquier sistema lineal cuya matriz de coeficientes sea A (véase por ejemplo Burden y Faires [21], páginas 366 y 372).

Para una matriz A de tamaño $n \times n$ tridiagonal e irreducible, se define la **diagonal dominanza** δ como

$$\delta = \min_{1 \leq i \leq n} \left\{ \frac{|a_i|}{|b_i| + |c_i|} \right\}. \tag{2.3}$$

Obviamente, si A es estrictamente diagonal dominante, entonces $\delta > 1$ por la expresión (2.2.)

Si $\boldsymbol{x}$ es la solución de un sistema $A\boldsymbol{x} = \boldsymbol{d}$ y se toma otro vector $\boldsymbol{s}$ como solución del mismo, la cantidad de **error** cometido viene determinada, por ejemplo, por $\|\boldsymbol{e}\|_\infty$, donde

$$\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{s}.$$

2.2 Descripción del método.

Se considera el problema general de obtener la solución del sistema

$$A\boldsymbol{x} = \boldsymbol{d}, \tag{2.4}$$

donde A es una matriz tridiagonal, estrictamente diagonal dominante e irreducible, dada por la expresión (2.1) y

$$\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

es el vector de términos independientes.

Se supone que existen dos números naturales k y p tales que $k = \dfrac{n}{p}$ y se considera en A la siguiente partición por bloques

$$A = \begin{bmatrix} A_0 & B_0 & & & \\ C_1 & A_1 & B_1 & & \\ & \ddots & \ddots & \ddots & \\ & & C_{p-2} & A_{p-2} & B_{p-2} \\ & & & C_{p-1} & A_{p-1} \end{bmatrix} \tag{2.5}$$

donde cada uno de los bloques diagonales

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

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

$$C_{i+1} = \left[\begin{array}{cccc|c} 0 & 0 & \ldots & 0 & c_{(i+1)k+1} \\ \hline & & & & 0 \\ & & O & & \vdots \\ & & & & 0 \\ & & & & 0 \end{array}\right]$$

y superdiagonal

$$B_i = \left[\begin{array}{c|ccc} 0 & & & \\ 0 & & O & \\ \vdots & & & \\ 0 & & & \\ \hline b_{(i+1)k} & 0 & \ldots & 0 \quad 0 \end{array}\right]$$

es una matriz de tamaño $k \times k$ con un sólo elemento no nulo.

En los vectores $\boldsymbol{x}$ y $\boldsymbol{d}$ se considera una partición por bloques conforme con la realizada en la matriz de coeficientes A, es decir

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{p-2} \\ \boldsymbol{x}_{p-1} \end{bmatrix} \quad \text{y} \quad \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_{p-2} \\ \boldsymbol{d}_{p-1} \end{bmatrix}, \tag{2.6}$$

con

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

En consecuencia, el sistema (2.4) se puede escribir como

$$\begin{bmatrix} A_0 & B_0 & & & \\ C_1 & A_1 & B_1 & & \\ & \ddots & \ddots & \ddots & \\ & & C_{p-2} & A_{p-2} & B_{p-2} \\ & & & C_{p-1} & A_{p-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{p-2} \\ \boldsymbol{x}_{p-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_{p-2} \\ \boldsymbol{d}_{p-1} \end{bmatrix}. \tag{2.7}$$

Si se denota por $\boldsymbol{s}_i$ la solución del sistema

$$A_i \boldsymbol{x}_i = \boldsymbol{d}_i, \quad i = 0, 1, \ldots, p-1, \tag{2.8}$$

se obtiene el vector

$$\boldsymbol{s} = \begin{bmatrix} \boldsymbol{s}_0 \\ \boldsymbol{s}_1 \\ \vdots \\ \boldsymbol{s}_{p-2} \\ \boldsymbol{s}_{p-1} \end{bmatrix},$$

que no coincide exactamente con el vector solución del sistema (2.4), ya que en su obtención se ha omitido alguna información del sistema (los bloques subdiagonales y superdiagonales); aunque se pueden utilizar las diferentes soluciones de los subsistemas independientes (2.8) para formar la solución del sistema (2.4).

Cada una de estas soluciones $\boldsymbol{s}_i$, para $i = 0, 1, \ldots, p-1$, contiene una cierta cantidad de error. Como se puede observar en el siguiente ejemplo, dependiendo del valor particular de la diagonal dominanza de A y del tamaño de los bloques, este error se puede hacer muy pequeño en las componentes centrales de $\boldsymbol{s}_i$.

Ejemplo 2.1 *En el sistema (2.4), se considera* $n = 24$, $a_i = 30$, *para* $i = 1, 2, \ldots, n$, $b_i = -c_{i+1} = 1$, *para* $i = 1, 2, \ldots, n-1$ *y* $d_i = c_i + a_i + b_i$, *para* $i = 1, 2, \ldots, n$ *(con* $b_n = c_1 = 0$*).*

Por la forma en que se ha construido, resulta inmediato que la solución $\boldsymbol{x}$ *del sistema tiene todas sus componentes iguales a 1.*

Para $p = 4$ *(y por tanto* $k = 6$*) el sistema se puede expresar como*

$$\begin{bmatrix} A_0 & B_0 & & \\ C_1 & A_1 & B_1 & \\ & C_2 & A_2 & B_2 \\ & & C_3 & A_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ \boldsymbol{d}_3 \end{bmatrix}.$$

El sistema $A_0\boldsymbol{x}_0 = \boldsymbol{d}_0$, *esto es*

$$\begin{bmatrix} 30 & 1 & 0 & 0 & 0 & 0 \\ -1 & 30 & 1 & 0 & 0 & 0 \\ 0 & -1 & 30 & 1 & 0 & 0 \\ 0 & 0 & -1 & 30 & 1 & 0 \\ 0 & 0 & 0 & -1 & 30 & 1 \\ 0 & 0 & 0 & 0 & -1 & 30 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 31 \\ 30 \\ 30 \\ 30 \\ 30 \\ 30 \end{bmatrix},$$

tiene solución

$$\boldsymbol{s}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0.9989 \\ 1.0333 \end{bmatrix}.$$

Los sistemas $A_1\boldsymbol{x}_1 = \boldsymbol{d}_1$ *y* $A_2\boldsymbol{x_2} = \boldsymbol{d_2}$ *tienen la misma matriz de coeficientes*

$$A_1 = A_2 = \begin{bmatrix} 30 & 1 & 0 & 0 & 0 & 0 \\ -1 & 30 & 1 & 0 & 0 & 0 \\ 0 & -1 & 30 & 1 & 0 & 0 \\ 0 & 0 & -1 & 30 & 1 & 0 \\ 0 & 0 & 0 & -1 & 30 & 1 \\ 0 & 0 & 0 & 0 & -1 & 30 \end{bmatrix}$$

y el mismo vector de términos independientes

$$\boldsymbol{d}_1 = \boldsymbol{d}_2 = \begin{bmatrix} 30 \\ 30 \\ 30 \\ 30 \\ 30 \\ 30 \end{bmatrix},$$

por tanto sus vectores solución coinciden

$$\boldsymbol{s}_1 = \boldsymbol{s}_2 = \begin{bmatrix} 0.9667 \\ 0.9989 \\ 1 \\ 1 \\ 0.9989 \\ 1.0333 \end{bmatrix}.$$

El sistema $A_3\boldsymbol{x}_3 = \boldsymbol{d}_3$, *o sea*

$$\begin{bmatrix} 30 & 1 & 0 & 0 & 0 & 0 \\ -1 & 30 & 1 & 0 & 0 & 0 \\ 0 & -1 & 30 & 1 & 0 & 0 \\ 0 & 0 & -1 & 30 & 1 & 0 \\ 0 & 0 & 0 & -1 & 30 & 1 \\ 0 & 0 & 0 & 0 & -1 & 30 \end{bmatrix} \begin{bmatrix} x_{19} \\ x_{20} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix} = \begin{bmatrix} 30 \\ 30 \\ 30 \\ 30 \\ 30 \\ 29 \end{bmatrix},$$

tiene solución

$$\boldsymbol{s}_3 = \begin{bmatrix} 0.9667 \\ 0.9989 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Si se toma

$$\boldsymbol{s} = \begin{bmatrix} \boldsymbol{s}_0 \\ \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \boldsymbol{s}_3 \end{bmatrix}$$

como solución del sistema $A\boldsymbol{x} = \boldsymbol{d}$, *para* $i = 1, 2, 3$ *en las componentes* s_{6i}, s_{6i+1} *se produce un error igual a* 3.33×10^{-2}, *en* s_{6i-1}, s_{6i+2} *el error que se comete es* 0.11×10^{-2} *(menor) y en el resto de componentes no se produce error.*

En cambio, si en el sistema (2.4) se considera $a_i = 20$ *para* $i = 1, 2, \ldots, 24$, $b_i = -c_{i+1} = 1$, *para* $i = 1, 2, \ldots, 23$ *y* $d_i = c_i + a_i + b_i$, *para* $i = 1, 2, \ldots, 24$ *(con* $b_{24} = c_1 = 0$*) se obtienen las siguientes soluciones de los sistemas* $A_i\boldsymbol{x}_i = \boldsymbol{d}_i$, *para*

$i = 0, 1, 2, 3$;

$$\boldsymbol{s}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1.0001 \\ 0.9975 \\ 1.0499 \end{bmatrix}, \quad \boldsymbol{s}_1 = \boldsymbol{s}_2 = \begin{bmatrix} 0.9501 \\ 0.9975 \\ 0.9999 \\ 1.0001 \\ 0.9975 \\ 1.0499 \end{bmatrix}, \quad \boldsymbol{s}_3 = \begin{bmatrix} 0.9501 \\ 0.9975 \\ 0.9999 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

En esta ocasión, el error se produce en las tres últimas componentes de $\boldsymbol{s}_0$*, en las tres primeras de* $\boldsymbol{s}_3$ *y en todas las componentes de los vectores* $\boldsymbol{s}_1$ *y* $\boldsymbol{s}_2$*.*

Por último, si en el sistema (2.4) se considera $a_i = 4$*, para* $i = 1, 2, \ldots, 24$*, y el resto de valores como en los dos casos anteriores, se obtienen los siguientes vectores solución de los sistemas* $A_i \boldsymbol{x}_i = \boldsymbol{d}_i$*, para* $i = 0, 1, 2, 3$*;*

$$\boldsymbol{s}_0 = \begin{bmatrix} 0.9998 \\ 1.0007 \\ 0.9969 \\ 1.0132 \\ 0.9443 \\ 1.2361 \end{bmatrix}, \quad \boldsymbol{s}_1 = \boldsymbol{s}_2 = \begin{bmatrix} 0.7637 \\ 0.9450 \\ 0.9837 \\ 1.0101 \\ 0.9435 \\ 1.2359 \end{bmatrix}, \quad \boldsymbol{s}_3 = \begin{bmatrix} 0.7639 \\ 0.9443 \\ 0.9868 \\ 0.9969 \\ 0.9993 \\ 0.9998 \end{bmatrix}.$$

En este caso, se comete error en todas las componentes de los vectores $\boldsymbol{s}_i$*, para* $i = 0, 1, 2, 3$*.*

En los tres casos el tamaño de los bloques de la partición no ha variado, en cambio sí ha variado la diagonal dominanza que en el primer caso es igual a 15, en el segundo 10 y en el último 2.

Si ahora se considera en el primer caso $p = 6$ *(y por tanto* $k = 4$*), la diagonal dominanza es la misma pero el tamaño de los bloques es distinto. El sistema (2.4) se*

puede expresar como

$$\begin{bmatrix} A_0 & B_0 & & & & \\ C_1 & A_1 & B_1 & & & \\ & C_2 & A_2 & B_2 & & \\ & & C_3 & A_3 & B_3 & \\ & & & C_4 & A_4 & B_4 \\ & & & & C_5 & A_5 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ \boldsymbol{d}_3 \\ \boldsymbol{d}_4 \\ \boldsymbol{d}_5 \end{bmatrix}.$$

Se obtienen los siguientes vectores solución de los sistemas $A_i\boldsymbol{x}_i = \boldsymbol{d}_i$, *para* $i = 0, 1, \ldots, 5$;

$$\boldsymbol{s}_0 = \begin{bmatrix} 1 \\ 1 \\ 0.9989 \\ 1.0333 \end{bmatrix}, \quad \boldsymbol{s}_1 = \boldsymbol{s}_2 = \boldsymbol{s}_3 = \boldsymbol{s}_4 = \begin{bmatrix} 0.9667 \\ 0.9989 \\ 0.9989 \\ 1.0333 \end{bmatrix}, \quad \boldsymbol{s}_5 = \begin{bmatrix} 0.9667 \\ 0.9989 \\ 1 \\ 1 \end{bmatrix}.$$

Ahora, el error se produce en las dos últimas componentes de $\boldsymbol{s}_0$, *en las dos primeras de* $\boldsymbol{s}_5$ *y en todas las componentes de las vectores centrales* $\boldsymbol{s}_i, i = 1, 2, 3, 4$. ■

Para minimizar y acotar el error, el método OPM (*Overlapped Partitions Method*) propone considerar una nueva partición (véase Larriba, Jorba, y Navarro [72]) basada en la partición natural dada en la expresión (2.7), añadiendo $2m$ filas (respectivamente, componentes) a cada uno de los bloques diagonales de la matriz de coeficientes (respectivamente, vector de términos independientes) y m filas (respectivamente, componentes) al primer y último bloque de la matriz de coeficientes (respectivamente, vector de términos independientes). Como consecuencia, cada uno de los nuevos bloques está solapado con sus adyacentes (véase la figura 2.1) y se obtiene un nuevo conjunto de subsistemas

$$\hat{A}_i\hat{\boldsymbol{x}}_i = \hat{\boldsymbol{d}}_i, \quad i = 0, 1, \ldots, p-1, \tag{2.9}$$

donde

- $\hat{A}_0$ es la submatriz de A de tamaño $(k+m)\times(k+m)$ formada por las filas y columnas $1, 2, \ldots, k+m$ y $\hat{\boldsymbol{d}}_0$ es el vector formado por las componentes $1, 2, \ldots, k+m$ de $\boldsymbol{d}$;

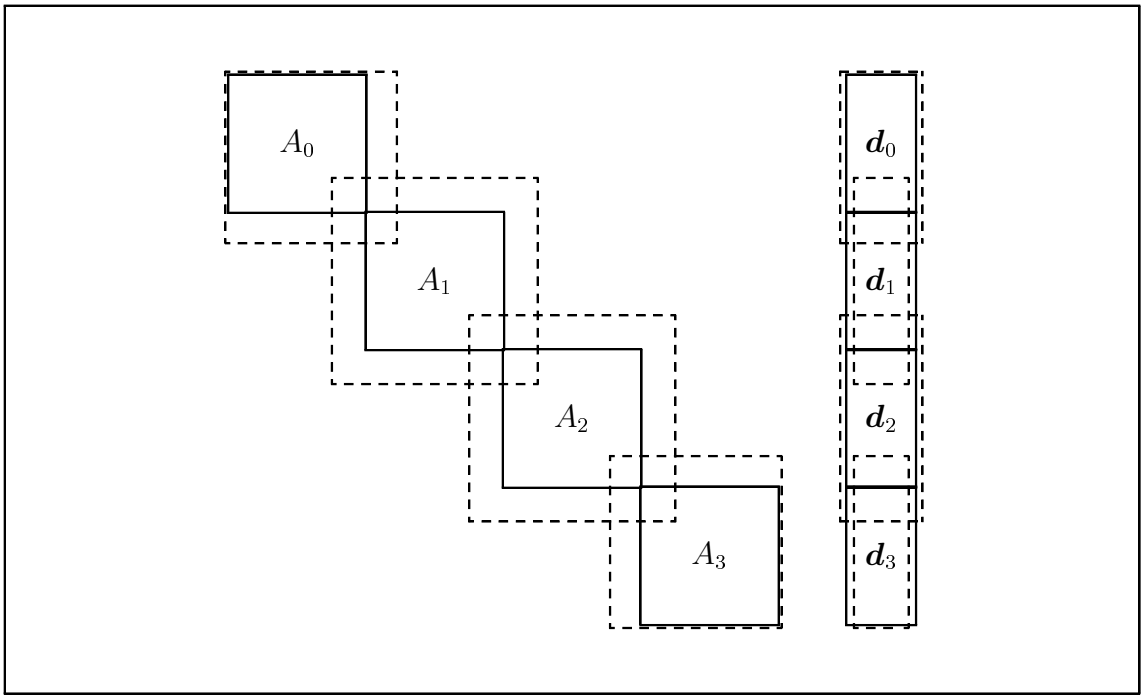


Figura 2.1: *Partición natural y superpuesta de A y $\boldsymbol{d}$ para $p = 4$.*

- $\hat{A}_i$, para $i = 1, 2, \ldots, p-2$, es la submatriz de A de tamaño $(k+2m) \times (k+2m)$ formada por las filas y columnas $ik - m + 1, ik - m + 2, \ldots, (i+1)k + m$ y $\hat{\boldsymbol{d}}_i$ es el vector formado por las componentes $ik - m + 1, ik - m + 2, \ldots, (i+1)k + m$ de $\boldsymbol{d}$;

- $\hat{A}_{p-1}$ es la submatriz de A de tamaño $(k+m) \times (k+m)$ formada por las filas y columnas $j = n - k - m + 1, n - k - m + 2, \ldots, n$ y, finalmente, $\hat{\boldsymbol{d}}_{p-1}$ es el vector formado por las componentes $n - k - m + 1, n - k - m + 2, \ldots, n$ de $\boldsymbol{d}$.

En la figura 2.1 se muestran, para $p = 4$, los bloques A_i, $\boldsymbol{d}_i$ marcados con trazo continuo y los bloques $\hat{A}_i$, $\hat{\boldsymbol{d}}_i$ con trazo discontinuo.

Para obtener la solución del sistema (2.4), se resuelven los subsistemas (2.9) por eliminación Gaussiana (no hay que realizar intercambio de filas por ser la matriz A estrictamente diagonal dominante), de cada subsistema intermedio se eligen las k componentes centrales del vector solución, del primer subsistema se toman las k primeras y del último las k últimas. En la figura 2.2 se muestra, para $p = 4$, la formación de la solución del sistema (2.4) a partir de las soluciones de los subsistemas (2.9).

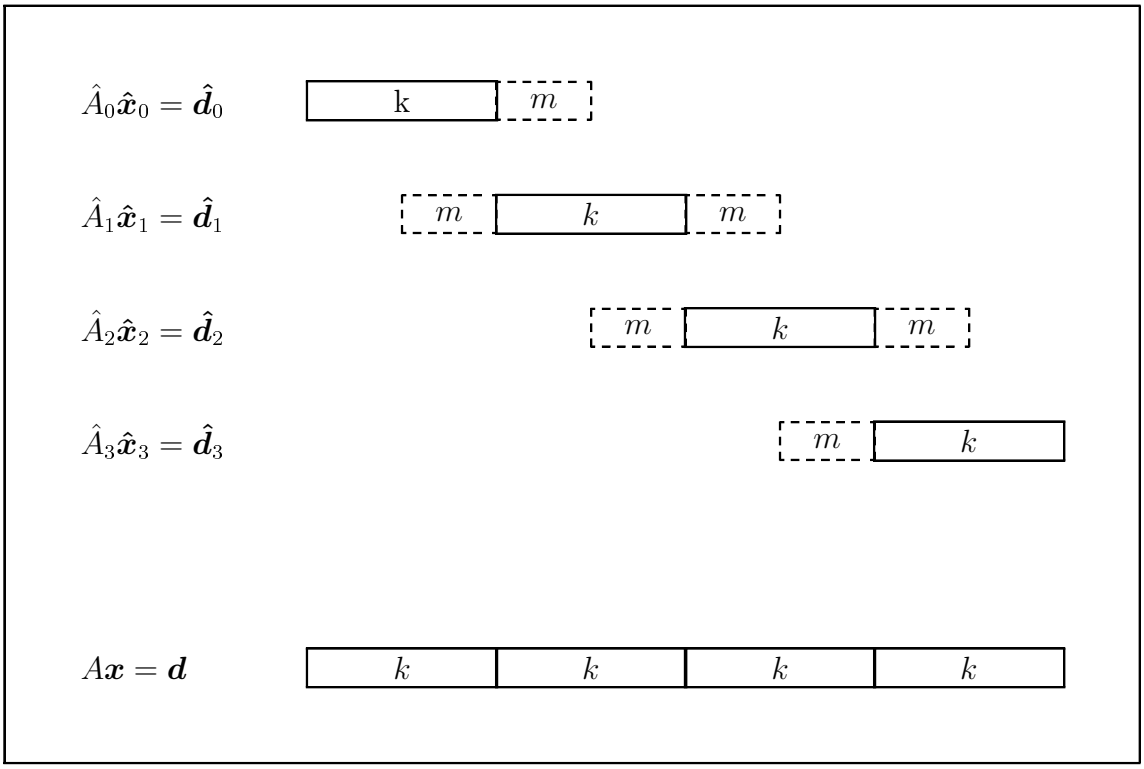


Figura 2.2: *Obtención de la solución general a partir de las soluciones parciales para $p = 4$.*

2.3 Precisión del método

En la sección 2.2 no se ha especificado nada acerca de la forma de calcular m, que representa el número de ecuaciones superpuestas entre dos bloques contiguos. En esta sección, el parámetro m se obtiene como una función de la diagonal dominanza δ de la matriz de coeficientes y el máximo error permitido ϵ.

Teorema 2.1 *Si $\boldsymbol{x}$ es la solución del sistema (2.4) se tiene*

$$\|\boldsymbol{x}\|_\infty \leq \frac{\max\limits_{1\leq i\leq n}\left\{\left|\dfrac{d_i}{a_i}\right|\right\}}{1-\delta^{-1}}, \tag{2.10}$$

con δ la diagonal dominanza de la matriz A.

Demostración: La matriz A se puede descomponer como $A = D + B$, donde

$$D = \begin{bmatrix} a_1 & & & & \\ & a_2 & & & \\ & & \ddots & & \\ & & & a_{n-1} & \\ & & & & a_n \end{bmatrix} \quad \text{y} \quad B = \begin{bmatrix} 0 & b_1 & & & \\ c_2 & 0 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & 0 & b_{n-1} \\ & & & c_n & 0 \end{bmatrix}.$$

El hecho de que la matriz A sea estrictamente diagonal dominante implica que es no singular por lo que D también lo es, con

$$D^{-1} = \begin{bmatrix} \frac{1}{a_1} & & & & \\ & \frac{1}{a_2} & & & \\ & & \ddots & & \\ & & & \frac{1}{a_{n-1}} & \\ & & & & \frac{1}{a_n} \end{bmatrix},$$

de manera que A se puede escribir como

$$A = D(I + D^{-1}B)$$

y así

$$\boldsymbol{x} = A^{-1}\boldsymbol{d} = (I + D^{-1}B)^{-1}D^{-1}\boldsymbol{d},$$

con lo que

$$\|\boldsymbol{x}\|_\infty \leq \left\|(I + D^{-1}B)^{-1}\right\|_\infty \left\|D^{-1}\boldsymbol{d}\right\|_\infty. \tag{2.11}$$

Ahora bien

$$\left\|(I + D^{-1}B)^{-1}\right\|_\infty \leq \frac{1}{1 - \|D^{-1}B\|_\infty} = \frac{1}{1 - \delta^{-1}} \tag{2.12}$$

ya que

$$\begin{aligned}\left\|D^{-1}B\right\|_\infty &= \max_{1\le i\le n}\left\{\frac{|b_i|+|c_i|}{|a_i|}\right\}\\ &= \frac{1}{\min\limits_{1\le i\le n}\left\{\dfrac{|a_i|}{|b_i|+|c_i|}\right\}}\\ &= \delta^{-1},\end{aligned}$$

mientras que

$$\left\|D^{-1}\boldsymbol{d}\right\|_\infty = \max_{1\le i\le n}\left\{\left|\frac{d_i}{a_i}\right|\right\}. \tag{2.13}$$

Por tanto, de las expresiones (2.11), (2.12) y (2.13) se obtiene la desigualdad (2.10). ■

Lema 2.1 *Si* $D = \mathrm{diag}(A)$ *y*

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

es la solución del sistema (2.4), se tiene

$$|x_i| \le \frac{\delta^{-h+1}}{1-\delta^{-2}}\left\|D^{-1}\boldsymbol{d}\right\|_\infty, \ \textit{para } i = 1, 2, \ldots, n;$$

donde

(i) $h = i$, *si* $d_2 = d_3 = \cdots = d_n = 0$.

(ii) $h = n - i + 1$, *si* $d_1 = d_2 = \cdots = d_{n-1} = 0$.

(iii) $h = \min\{i, n-i+1\}$, *si* $d_2 = d_3 = \cdots = d_{n-1} = 0$.

Demostración: De la expresión (2.13) se tiene

$$\boldsymbol{x} = (D^{-1}\boldsymbol{d}) - N(D^{-1}\boldsymbol{d}) + N^2(D^{-1}\boldsymbol{d}) - N^3(D^{-1}\boldsymbol{d}) + \cdots$$

(i) Obsérvense los siguientes productos.

$$D^{-1}\boldsymbol{d} = \begin{bmatrix} * \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$N(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & & \\ * & 0 & * & & & & \\ & * & 0 & * & & & \\ & & * & 0 & * & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & & * & 0 \end{bmatrix} \begin{bmatrix} * \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$N^2(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & & \\ * & 0 & * & & & & \\ & * & 0 & * & & & \\ & & * & 0 & * & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & & * & 0 \end{bmatrix} \begin{bmatrix} 0 \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} * \\ 0 \\ * \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$N^3(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & & \\ * & 0 & * & & & & \\ & * & 0 & * & & & \\ & & * & 0 & * & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & & * & 0 \end{bmatrix} \begin{bmatrix} * \\ 0 \\ * \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ * \\ 0 \\ * \\ \vdots \\ 0 \end{bmatrix}$$

Debido a la especial estructura de la matriz N, se tiene que el primer término de la serie (2.3) que afecta a la componente x_i está en $N^{i-1}(D^{-1}\boldsymbol{d})$. Nótese que los términos $N^i(D^{-1}\boldsymbol{d})$, $N^{i+2}(D^{-1}\boldsymbol{d}),\ldots$ no contribuyen en el cálculo de la componente x_i, mientras que los términos $N^{i+1}(D^{-1}\boldsymbol{d})$, $N^{i+3}(D^{-1}\boldsymbol{d}),\ldots$ sí. Como consecuencia

$$\begin{aligned}
|x_i| &\leq \left\|N^{i-1}(D^{-1}\boldsymbol{d})\right\|_\infty + \left\|N^{i+1}(D^{-1}\boldsymbol{d})\right\|_\infty + \left\|N^{i+3}(D^{-1}\boldsymbol{d})\right\|_\infty + \cdots \\
&\leq \|N\|_\infty^{i-1}\left\|D^{-1}\boldsymbol{d}\right\|_\infty + \|N\|_\infty^{i+1}\left\|D^{-1}\boldsymbol{d}\right\|_\infty + \|N\|_\infty^{i+3}\left\|D^{-1}\boldsymbol{d}\right\|_\infty + \cdots \\
&= \left[\left(\delta^{-1}\right)^{i-1} + \left(\delta^{-1}\right)^{i+1} + \left(\delta^{-1}\right)^{i+3} + \cdots\right]\left\|D^{-1}\boldsymbol{d}\right\|_\infty \\
&= \delta^{-i+1}\left(1+\delta^{-2}+\delta^{-4}+\cdots\right)\left\|D^{-1}\boldsymbol{d}\right\|_\infty \\
&= \frac{\delta^{-i+1}}{1-\delta^{-2}}\left\|D^{-1}\boldsymbol{d}\right\|_\infty .
\end{aligned}$$

(ii) En este segundo caso, los productos tienen la siguiente forma

$$D^{-1}\boldsymbol{d} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ * \end{bmatrix}$$

$$N(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & \\ \ddots & \ddots & \ddots & & & \\ & * & 0 & * & & \\ & & * & 0 & * & \\ & & & * & 0 & * \\ & & & & * & 0 \end{bmatrix}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ * \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ * \\ 0 \end{bmatrix}$$

$$N^2(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & & \\ \ddots & \ddots & \ddots & & & & \\ & * & 0 & * & & & \\ & & * & 0 & * & & \\ & & & * & 0 & * & \\ & & & & * & 0 & \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ * \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ * \\ 0 \\ * \end{bmatrix}$$

$$N^3(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & & \\ \ddots & \ddots & \ddots & & & & \\ & * & 0 & * & & & \\ & & * & 0 & * & & \\ & & & * & 0 & * & \\ & & & & * & 0 & \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ * \\ 0 \\ * \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ * \\ 0 \\ * \\ 0 \end{bmatrix}$$

El primer término de la serie (2.3) que afecta a la componente x_i está en $N^{n-i}(D^{-1}\boldsymbol{d})$. Los términos $N^{n-i+1}(D^{-1}\boldsymbol{d})$, $N^{n-i+3}(D^{-1}\boldsymbol{d}), \ldots$ no influyen en el cálculo de x_i, en cambio los términos $N^{n-i+2}(D^{-1}\boldsymbol{d})$, $N^{n-i+4}(D^{-1}\boldsymbol{d}), \ldots$ sí lo hacen. En consecuencia

$$\begin{aligned} |x_i| &\leq \left\|N^{n-i}(D^{-1}\boldsymbol{d})\right\|_\infty + \left\|N^{n-i+2}(D^{-1}\boldsymbol{d})\right\|_\infty + \left\|N^{n-i+4}(D^{-1}\boldsymbol{d})\right\|_\infty + \cdots \\ &\leq \|N\|_\infty^{n-i} \left\|D^{-1}\boldsymbol{d}\right\|_\infty + \|N\|_\infty^{n-i+2} \left\|D^{-1}\boldsymbol{d}\right\|_\infty + \|N\|_\infty^{n-i+4} \left\|D^{-1}\boldsymbol{d}\right\|_\infty + \cdots \\ &= \left[\left(\delta^{-1}\right)^{n-i} + \left(\delta^{-1}\right)^{n-i+2} + \left(\delta^{-1}\right)^{n-i+4} + \cdots\right] \left\|D^{-1}\boldsymbol{d}\right\|_\infty \\ &= \delta^{i-n}\left(1 + \delta^{-2} + \delta^{-4} + \cdots\right) \left\|D^{-1}\boldsymbol{d}\right\|_\infty \\ &= \frac{\delta^{i-n}}{1-\delta^{-2}} \left\|D^{-1}\boldsymbol{d}\right\|_\infty \end{aligned}$$

(iii) Ahora, los productos tienen la forma

$$D^{-1}\boldsymbol{d} = \begin{bmatrix} * \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ * \end{bmatrix}$$

$$N(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & & & & \\ * & 0 & * & & & & & & \\ & * & 0 & * & & & & & \\ & & * & 0 & * & & & & \\ & & & \ddots & \ddots & \ddots & & & \\ & & & & * & 0 & * & & \\ & & & & & * & 0 & * & \\ & & & & & & * & 0 & * \\ & & & & & & & * & 0 \end{bmatrix} \begin{bmatrix} * \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ * \end{bmatrix} = \begin{bmatrix} 0 \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ * \\ 0 \end{bmatrix}$$

$$N^2(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & & & & \\ * & 0 & * & & & & & & \\ & * & 0 & * & & & & & \\ & & * & 0 & * & & & & \\ & & & \ddots & \ddots & \ddots & & & \\ & & & & * & 0 & * & & \\ & & & & & * & 0 & * & \\ & & & & & & * & 0 & * \\ & & & & & & & * & 0 \end{bmatrix} \begin{bmatrix} 0 \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ * \\ 0 \end{bmatrix} = \begin{bmatrix} * \\ 0 \\ * \\ 0 \\ \vdots \\ 0 \\ * \\ 0 \\ * \end{bmatrix}$$

$$N^3(D^{-1}\boldsymbol{d}) = \begin{bmatrix} 0 & * & & & & & & & & \\ * & 0 & * & & & & & & & \\ & * & 0 & * & & & & & & \\ & & * & 0 & * & & & & & \\ & & & \ddots & \ddots & \ddots & & & & \\ & & & & * & 0 & * & & & \\ & & & & & * & 0 & * & & \\ & & & & & & * & 0 & * & \\ & & & & & & & * & 0 & \end{bmatrix} \begin{bmatrix} * \\ 0 \\ * \\ 0 \\ \vdots \\ 0 \\ * \\ 0 \\ * \end{bmatrix} = \begin{bmatrix} 0 \\ * \\ 0 \\ * \\ \vdots \\ * \\ 0 \\ * \\ 0 \end{bmatrix}$$

Si $i \leq n-i+1$, el primer término de la serie (2.3) que afecta a la componente x_i está en $N^{i-1}(D^{-1}\boldsymbol{d})$. Razonando de forma análoga a la del apartado (i) se tiene que

$$|x_i| \leq \frac{\delta^{-i+1}}{1-\delta^{-2}} \left\| D^{-1}\boldsymbol{d} \right\|_\infty .$$

Por el contrario, si $i > n-i+1$ el primer término de la serie (2.3) que afecta a la componente x_i está en $N^{n-i}(D^{-1}\boldsymbol{d})$. Actuando como en el apartado (ii) se obtiene

$$|x_i| \leq \frac{\delta^{i-n}}{1-\delta^{-2}} \left\| D^{-1}\boldsymbol{d} \right\|_\infty .$$

■

Teorema 2.2 *Sea $\epsilon > 0$ un número real, m el número de ecuaciones que se solapan al resolver el sistema (2.4) mediante el método de las particiones superpuestas y μ una cota superior de la norma infinito de la solución del mismo. Si*

$$m \geq \left\lceil \frac{1}{\log \delta^{-1}} \log \frac{\epsilon(1-\delta^{-2})}{\mu} \right\rceil , \tag{2.14}$$

entonces el error cometido al aplicar el método es menor que ϵ.

Demostración: Se considera en el sistema (2.4) la partición en bloques dada en la expresión (2.7). A partir de ella se obtiene una nueva partición añadiendo $2m$ filas (respectivamente, componentes) a cada uno de los bloques intermedios de la matriz de coeficientes

(respectivamente, vector de términos independientes) y m filas (respectivamente, componentes) al primer y último bloque de la matriz de coeficientes (respectivamente, vector de términos independientes). Como consecuencia, cada uno de los nuevos bloques está solapado con sus adyacentes y se obtiene el nuevo conjunto de subsistemas dado en la expresión (2.9).

Para simplificar, se redefine cualquier subsistema $\hat{A}_i\hat{\boldsymbol{x}}_i = \hat{\boldsymbol{d}}_i$ como $\hat{A}\hat{\boldsymbol{x}} = \hat{\boldsymbol{d}}$ y los índices del mismo desde 1 hasta r. La situación del subsistema $\hat{A}\hat{\boldsymbol{x}} = \hat{\boldsymbol{d}}$ en el sistema general tiene la siguiente forma

$$\begin{array}{ccccccccccccc}
\tilde{c}_1\tilde{x}_0 & + & \hat{a}_1\hat{x}_1 & + & \hat{b}_1\hat{x}_2 & & & & & & & = & \hat{d}_1 \\
 & & \hat{c}_2\hat{x}_1 & + & \hat{a}_2\hat{x}_2 & + & \hat{b}_2\hat{x}_3 & & & & & = & \hat{d}_2 \\
 & & \ddots & & \ddots & & \ddots & & & & & \vdots & \\
 & & & & \hat{c}_{r-1}\hat{x}_{r-2} & + & \hat{a}_{r-1}\hat{x}_{r-1} & + & \hat{b}_{r-1}\hat{x}_r & & & = & \hat{d}_{r-1} \\
 & & & & & & \hat{c}_r\hat{x}_{r-1} & + & \hat{a}_r\hat{x}_r & + & \tilde{b}_r\tilde{x}_{r+1} & = & \hat{d}_r
\end{array}$$

donde $\tilde{c}_1$ y $\tilde{b}_r$ son los elementos omitidos del sistema general al resolver el subsistema $\hat{A}\hat{\boldsymbol{x}} = \hat{\boldsymbol{d}}$. Nótese que $\tilde{c}_1 = 0$ para el primer subsistema y $\tilde{b}_r = 0$ para el último.

Si $\boldsymbol{x}$ es la solución del sistema general, se representa por $\boldsymbol{x}'$ a la restricción de $\boldsymbol{x}$ a las variables del subsistema $\hat{A}\hat{\boldsymbol{x}} = \hat{\boldsymbol{d}}$. Se puede escribir entonces

$$\boldsymbol{x} = \left[\begin{array}{c} \vdots \\ \hline \boldsymbol{x}' \\ \hline \vdots \end{array}\right] = \left[\begin{array}{c} \vdots \\ \hline x'_0 \\ \hline x'_1 \\ \vdots \\ x'_r \\ \hline x'_{r+1} \\ \vdots \end{array}\right],$$

verificándose

$$\begin{aligned}
\tilde{c}_1 x'_0 + \hat{a}_1 x'_1 + \hat{b}_1 x'_2 &= \hat{d}_1 \\
\hat{c}_r x'_{r-1} + \hat{a}_r x'_r + \tilde{b}_r x'_{r+1} &= \hat{d}_r.
\end{aligned}$$

El error que se comete al tomar $\boldsymbol{x}'$ como solución del subsistema $\hat{A}\hat{\boldsymbol{x}} = \hat{\boldsymbol{d}}$ viene determinado por

$$\boldsymbol{e} = \hat{\boldsymbol{x}} - \boldsymbol{x}',$$

siendo $\hat{\boldsymbol{x}}$ el vector solución del mismo. Como

$$\hat{A}\hat{\boldsymbol{x}} = \begin{bmatrix} \hat{d}_1 \\ \hat{d}_2 \\ \vdots \\ \hat{d}_{r-1} \\ \hat{d}_r \end{bmatrix}$$

y

$$\hat{A}\boldsymbol{x}' = \begin{bmatrix} \hat{a}_1 x'_1 + \hat{b}_1 x'_2 \\ \hat{d}_2 \\ \vdots \\ \hat{d}_{r-1} \\ \hat{c}_r x'_{r-1} + \hat{a}_r x'_r \end{bmatrix} = \begin{bmatrix} \hat{d}_1 - \tilde{c}_1 x'_0 \\ \hat{d}_2 \\ \vdots \\ \hat{d}_{r-1} \\ \hat{d}_r - \tilde{b}_r x'_{r+1} \end{bmatrix},$$

se tiene

$$\hat{A}\hat{\boldsymbol{x}} - \hat{A}\boldsymbol{x}' = \begin{bmatrix} \tilde{c}_1 x'_0 \\ 0 \\ \vdots \\ 0 \\ \tilde{b}_r x'_{r+1} \end{bmatrix} = \boldsymbol{c},$$

por lo que

$$\hat{A}\boldsymbol{e} = \boldsymbol{c}.$$

Aplicando el lema 2.1 al sistema anterior, se tiene la siguiente acotación para la i-ésima componente de $\boldsymbol{e}$

$$|e_i| \leq \frac{\delta^{-h+1}}{1-\delta^{-2}} \left\| D^{-1}\boldsymbol{c} \right\|_\infty, \text{ para } i = 1, 2, \ldots, n;$$

donde

$$h = \begin{cases} i, & \text{en el último subsistema,} \\ r - i + 1, & \text{en el primer subsistema,} \\ \min\{i, r - i + 1\}, & \text{en los subsistemas intermedios} \end{cases}$$

y $D = \operatorname{diag}(\hat{A})$.

Ahora bien, para el primer subsistema

$$\frac{|\hat{a}_r|}{\left|\tilde{b}_r\right|} \geq \min_{1 \leq i \leq n} \left\{ \frac{|a_i|}{|b_i| + |c_i|} \right\} = \delta$$

con lo que

$$\left\| D^{-1} \boldsymbol{c} \right\|_\infty = \frac{\left|\tilde{b}_r\right| \left|x'_{r+1}\right|}{|\hat{a}_r|} \leq \delta^{-1} \left|x'_{r+1}\right|.$$

Para el último subsistema

$$\frac{|\hat{a}_1|}{|\tilde{c}_1|} \geq \delta$$

con lo que

$$\left\| D^{-1} \boldsymbol{c} \right\|_\infty = \frac{|\tilde{c}_1| \, |x'_0|}{|\hat{a}_1|} \leq \delta^{-1} |x'_0|.$$

Finalmente, para los subsistemas intermedios

$$\frac{|\hat{a}_1|}{|\tilde{c}_1|} \geq \delta \quad \text{y} \quad \frac{|\hat{a}_r|}{\left|\tilde{b}_r\right|} \geq \delta$$

con lo que

$$\left\| D^{-1} \boldsymbol{c} \right\|_\infty = \max \left\{ \frac{|\tilde{c}_1| \, |x'_0|}{|\hat{a}_1|}, \frac{\left|\tilde{b}_r\right| \left|x'_{r+1}\right|}{|\hat{a}_r|} \right\} \leq \max \left\{ \delta^{-1} |x'_0|, \delta^{-1} \left|x'_{r+1}\right| \right\}.$$

En cualquier caso

$$\left\| D^{-1} \boldsymbol{c} \right\|_\infty \leq \delta^{-1} \mu$$

donde μ es una cota superior de $\|\boldsymbol{x}\|_\infty$ y así

$$|e_i| \leq \frac{\delta^{-h}}{1-\delta^{-2}}\mu, \quad \text{para } i = 1, 2, \ldots, r. \tag{2.15}$$

Por otra parte, al ser $\delta > 1$, se cumple que

$$1 > \delta^{-1} > \delta^{-2} > \cdots > \delta^{-r}.$$

En el último subsistema, las m primeras componentes de $\boldsymbol{e}$ provienen de ecuaciones de solapamiento, en las demás se tiene la siguiente acotación, para $j = m+1, m+2, \ldots, r$,

$$|e_j| \leq \frac{\delta^{-j}}{1-\delta^{-2}}\,\mu < \frac{\delta^{-m}}{1-\delta^{-2}}\,\mu, \tag{2.16}$$

En el primer subsistema las m últimas ecuaciones son de solapamiento. Las $r-m$ primeras componentes de $\boldsymbol{e}$ cumplen

$$|e_j| \leq \frac{\delta^{-(r-j+1)}}{1-\delta^{-2}}\,\mu < \frac{\delta^{-m}}{1-\delta^{-2}}\,\mu, \tag{2.17}$$

con $j = 1, 2, \ldots, r-m$.

En los subsistemas intermedios las m primeras y últimas ecuaciones son de solapamiento. Las $r-2m$ componentes intermedias del vector de error están acotadas como sigue

$$|e_j| \leq \frac{\delta^{-\min\{j,n-j+1\}}}{1-\delta^{-2}}\,\mu < \frac{\delta^{-m}}{1-\delta^{-2}}\,\mu, \tag{2.18}$$

con $j = m+1, m+2, \ldots, r-m$.

Si se toma m de manera que cumpla la desigualdad 2.14 entonces

$$m \log \delta^{-1} \leq \log \frac{\epsilon\,(1-\delta^{-2})}{\mu},$$

esto es

$$\delta^{-m} \leq \frac{\epsilon\,(1-\delta^{-2})}{\mu},$$

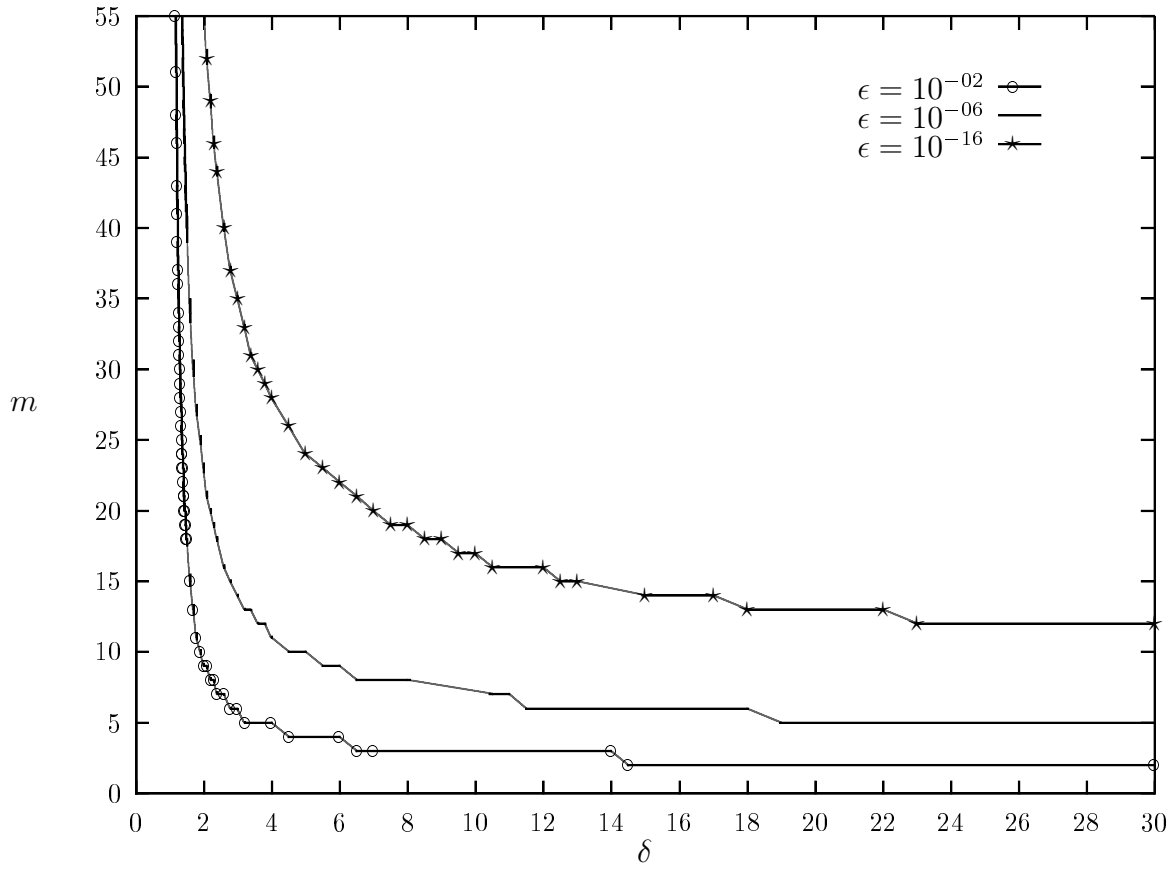


Figura 2.3: *Valor de m en función de δ para distintos valores de ϵ.*

o sea

$$\frac{\delta^{-m}}{1-\delta^{-2}} < \epsilon,$$

lo que implica, por las expresiones (2.16), (2.17) y (2.18), que el error cometido es menor que ϵ. ■

Para cada sistema en particular, de las expresiones (2.14) y (2.10) se puede determinar m y obtener, aplicando el método, la solución $\boldsymbol{x}$.

En la figura 2.3 se muestra una gráfica de los diferentes valores de m en función de la diagonal dominanza δ y el máximo error permitido ϵ, donde μ se ha calculado utilizando

		m				
		5	10	100	500	1000
	10^{-02}	3.14899	1.88411	1.09877	1.02438	1.01332
ϵ	10^{-06}	18.23523	4.37983	1.19132	1.04131	1.02172
	10^{-16}	1802.18329	42.55320	1.48022	1.08736	1.04409

Tabla 2.1: *Rango de la diagonal dominanza δ dependiendo del máximo error tolerado ϵ y del número de ecuaciones solapadas m.*

la expresión (2.10) para el caso concreto

$$\max_{1\leq i\leq n}\left\{\left|\frac{d_i}{a_i}\right|\right\} = 1.9.$$

Para otros valores de μ se obtienen gráficas con formas semejantes a la de la figura 2.3.

Se observa que para valores de la diagonal dominanza no muy próximos a 1, se obtienen valores de m pequeños (aumentando este valor cuanto menor sea el error ϵ tolerado) y que para valores de δ próximos a 1 el valor de m crece con tendencia a $+\infty$ (para $\delta = 1.01$ y $\epsilon = 10^{-16}$ se obtiene $m = 4626$). Además, diferentes pares (δ, ϵ) producen el mismo valor de m, debido a que este parámetro está determinado por una función entera de variable real.

En la tabla 2.1 se muestra el mínimo valor de δ necesario para no superar el error ϵ, dado un valor de solapamiento m fijo.

2.4 Paralelización del método.

Este método es fácilmente paralelizable pues basta con considerar que p es el número de procesadores (generalmente, los procesadores se representarán por P_i, para $i = 0, 1, \ldots, p-1$) y asignar a cada procesador la solución de uno de los subsistemas dados en la expresión (2.9). El siguiente algoritmo describe esa tarea.

Algoritmo 2.1 Método de las particiones superpuestas en paralelo

Paso 1

Se calcula δ conforme a la expresión (2.3), se obtiene μ mediante la expresión (2.10) y utilizando la expresión (2.14) se determina m.

Paso 2

A partir de m, se asignan a cada procesador las ecuaciones necesarias para obtener la partición superpuesta.

Paso 3

Cada procesador resuelve el subsistema correspondiente dado por la expresión (2.9).

Paso 4

El vector solución del sistema original se obtiene tomando las k primeras componentes del vector solución de $\hat{A}_0\hat{\boldsymbol{x}}_0 = \hat{\boldsymbol{d}}_0$, las k componentes centrales de los vectores solución de $\hat{A}_i\hat{\boldsymbol{x}}_i = \hat{\boldsymbol{d}}_i$, para $i = 1, 2, \ldots, p-2$, y las k últimas componentes del vector solución de $\hat{A}_{p-1}\hat{\boldsymbol{x}}_{p-1} = \hat{\boldsymbol{d}}_{p-1}$.

Ejemplo 2.2 *En el sistema (2.4), se considera* $n = 24$, $a_i = 30$, *para* $i = 1, 2, \ldots, n$, $b_i = -c_{i+1} = 1$, *para* $i = 1, 2, \ldots, n-1$ *y* $d_i = c_i + a_i + b_i$, *para* $i = 1, 2, \ldots, n$ *(con* $b_n = c_1 = 0$*). Por la forma en que se ha construido, resulta inmediato que la solución del sistema tiene todas sus componentes iguales a* 1.

La diagonal dominanza δ *está dada por*

$$\begin{aligned}
\delta &= \min_{1 \le i \le 24} \left\{ \frac{|a_i|}{|b_i| + |c_i|} \right\} \\
&= \min \left\{ \frac{30}{1}, \frac{30}{1+1}, \ldots, \frac{30}{1+1}, \frac{30}{1} \right\} \\
&= 15.
\end{aligned}$$

El parámetro μ vale

$$\begin{aligned}
\mu &= \frac{\max\limits_{1\le i\le 24}\left\{\left|\frac{d_i}{a_i}\right|\right\}}{1-\delta^{-1}} \\
&= \frac{\max\left\{\frac{31}{30},\frac{30}{30},\ldots,\frac{29}{30},\frac{29}{30}\right\}}{1-15^{-1}} \\
&= \frac{31}{28} \\
&= 1.1071.
\end{aligned}$$

Si se quiere que el error cometido ϵ sea inferior a 10^{-3}, se debe tomar m de manera que

$$\begin{aligned}
m &\ge \left\lceil \frac{1}{\log \delta^{-1}} \log \frac{\epsilon(1-\delta^{-2})}{\mu} \right\rceil \\
&= \left\lceil \frac{1}{\log 15^{-1}} \log \frac{10^{-3}(1-15^{-2})}{\frac{31}{28}} \right\rceil \\
&= \left\lceil -\frac{1}{\log 15}\left(\log 10^{-3} + \log \frac{1026}{1141}\right) \right\rceil \\
&= \left\lceil -\frac{1}{\log 15}\left(-3 + \log \frac{1026}{1141}\right) \right\rceil \\
&= \lceil 2.5901 \rceil,
\end{aligned}$$

por ejemplo $m = 3$.

El procesador P_0 resuelve el subsistema $\hat{A}_0\hat{\boldsymbol{x}}_0 = \hat{\boldsymbol{d}}_0$, esto es

$$\left[\begin{array}{ccccccccc} 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \hline x_7 \\ x_8 \\ x_9 \end{array}\right] = \left[\begin{array}{c} 31 \\ 30 \\ 30 \\ 30 \\ 30 \\ 30 \\ \hline 30 \\ 30 \\ 30 \end{array}\right],$$

cuya solución es

$$\hat{\boldsymbol{x}}_0 = \left[\begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \hline 1 \\ 0.9989 \\ 1.0333 \end{array}\right].$$

Cada uno de los procesadores P_1 y P_2 deben resolver, respectivamente, el subsistema

$\hat{A}_1\hat{\boldsymbol{x}}_1 = \hat{\boldsymbol{d}}_1$ *y* $\hat{A}_2\hat{\boldsymbol{x}}_2 = \hat{\boldsymbol{d}}_2$ *, esto es*

$$\left[\begin{array}{cccccccccccc}
30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30
\end{array}\right]
\left[\begin{array}{c} x_4 \\ x_5 \\ x_6 \\ \hline x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ \hline x_{13} \\ x_{14} \\ x_{15} \end{array}\right]
=
\left[\begin{array}{c} 30 \\ 30 \\ 30 \\ \hline 30 \\ 30 \\ 30 \\ 30 \\ 30 \\ 30 \\ \hline 30 \\ 30 \\ 30 \end{array}\right]$$

y

$$\left[\begin{array}{cccccccccccc}
30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30
\end{array}\right]
\left[\begin{array}{c} x_{10} \\ x_{11} \\ x_{12} \\ \hline x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \\ x_{17} \\ x_{18} \\ \hline x_{19} \\ x_{20} \\ x_{21} \end{array}\right]
=
\left[\begin{array}{c} 30 \\ 30 \\ 30 \\ \hline 30 \\ 30 \\ 30 \\ 30 \\ 30 \\ 30 \\ \hline 30 \\ 30 \\ 30 \end{array}\right],$$

respectivamente, cuyas soluciones son

$$\hat{\boldsymbol{x}}_1 = \hat{\boldsymbol{x}}_2 = \left[\begin{array}{c} 0.9667 \\ 0.9989 \\ 1 \\ \hline 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \hline 1 \\ 0.9989 \\ 1.0333 \end{array}\right].$$

Por último, el procesador P_3 debe resolver el subsistema $\hat{A}_3\hat{\boldsymbol{x}}_3 = \hat{\boldsymbol{d}}_3$, esto es

$$\left[\begin{array}{ccccccccc} 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 30 \end{array}\right] \left[\begin{array}{c} x_{16} \\ x_{17} \\ x_{18} \\ \hline x_{19} \\ x_{20} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{array}\right] = \left[\begin{array}{c} 30 \\ 30 \\ 30 \\ \hline 30 \\ 30 \\ 30 \\ 30 \\ 30 \\ 29 \end{array}\right],$$

cuya solución viene dada por

$$\hat{\boldsymbol{x}}_3 = \left[\begin{array}{c} 0.9667 \\ 0.99891 \\ 1 \\ \hline 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{array}\right].$$

La solución del sistema original se obtiene tomando las seis primeras componentes de $\hat{\boldsymbol{x}}_0$*, las seis componentes centrales de* $\hat{\boldsymbol{x}}_1$ *y* $\hat{\boldsymbol{x}}_2$ *y las seis últimas componentes de* $\hat{\boldsymbol{x}}_3$*.* ■

2.5 Algoritmos BSP

La implementación del algoritmo 2.1 según el modelo BSP se describe en el algoritmo 2.2. Se supone que tanto la matriz de coeficientes como el vector de términos independientes se encuentran en el procesador principal P_0.

Algoritmo 2.2 Algoritmo BSP del método OPM para un sistema tridiagonal

Superpaso 1

- El procesador principal P_0 calcula $k = \frac{n}{p}$ y los parámetros δ, μ y m usando las expresiones (2.3), (2.10) y (2.14).
- El procesador P_0 envía al procesador P_i, para $i = 0, 1, \ldots, p-1$, el bloque $\hat{A}_i$ y las componentes del vector $\hat{\boldsymbol{d}}_{\boldsymbol{i}}$, correspondientes a la partición superpuesta.
- El procesador P_0 envía el valor de m al procesador P_i, para $i = 1, 2, \ldots, p-1$.

Superpaso 2

- Para $i = 0, 1, \ldots, p-1$, cada procesador P_i resuelve el subsistema $\hat{A}_i \hat{\boldsymbol{x}}_i = \hat{\boldsymbol{d}}_i$ por eliminación Gaussiana.
- Para $i = 1, 2, \ldots, p-1$, el procesador P_i comunica al procesador P_0 las componentes $m+1, m+2, \ldots, m+k$ de $\hat{\boldsymbol{x}}_i$.
- El procesador principal forma la solución $\boldsymbol{x}$ del sistema (2.4), haciendo:
  - Para $j = 1, 2, \ldots, k$, la componente j-ésima de $\boldsymbol{x}$ igual a la componente j-ésima de $\hat{\boldsymbol{x}}_0$.
  - Para $i = 1, 2, \ldots, p-1$ y $j = 1, 2, \ldots, k$, la componente $(ki+j)$-ésima de $\boldsymbol{x}$ igual a la componente $(m+j)$-ésima de $\hat{\boldsymbol{x}}_i$.

Para obtener el coste computacional del algoritmo, se calcula el coste de cada uno de los superpasos individuales.

Coste del superpaso 1. El coste aritmético de este superpaso viene determinado por el cálculo en el procesador principal de los parámetros k, δ, μ y m. La obtención de k requiere una operación, para calcular la diagonal dominanza se necesita realizar $2n$ operaciones, para el cálculo de μ —de acuerdo con la expresión (2.10)— se necesitan $n+3$ operaciones y para calcular m —mediante (2.14)— 9 operaciones aritméticas. Por tanto el coste aritmético de este superpaso es $3n + 13$.

En cuanto al coste de comunicación, el procesador que más comunica es el principal que envía a P_i, para $i = 1, 2, \ldots, p-1$, el valor de m, el bloque $\hat{A}_i$ y las componentes del vector $\hat{\boldsymbol{d}}_i$; comunica por tanto $p - 1 + [4(k+2m) - 2](p-2) + 4(k+m) - 1$ elementos, por lo que el coste total del superpaso es

$$3n + 13 + (4n - 4k - 12m + 8mp - p + 2)g + l. \tag{2.19}$$

Coste del superpaso 2. Resolver un sistema tridiagonal de r ecuaciones mediante el método de eliminación de Gauss para sistemas tridiagonales requiere un total de $8r - 7$ operaciones, por lo que el coste aritmético del superpaso es $8(k+2m) - 7$, que corresponde al número de operaciones realizadas por cualquiera de los procesadores intermedios.

Los procesadores P_i, para $i = 1, \ldots, p-1$, comunican k componentes de su solución parcial al procesador principal, lo que supone un coste de comunicación $k(p-1)$. En consecuencia, el coste del superpaso 2 es

$$(8k + 16m - 7) + (n - k)g + l. \tag{2.20}$$

Sumando las expresiones (2.19) y (2.20), se obtiene que el coste total del algoritmo es

$$(3n + 8k + 16m + 6) + [5n - 5k - 12m + 8mp - p + 2]g + 2l.$$

El cálculo de los parámetros δ y μ requiere $3n + 3$ operaciones, tarea que puede realizarse en paralelo. Aunque esto conlleva un aumento del número de superpasos y un ligero incremento en el numero de comunicaciones, en muchos casos supone una reducción en el coste total del algoritmo.

Para $i = 0, 1, \ldots, p-1$, se definen

$$\delta_i = \min_{ik+1 \le j \le (i+1)k} \left\{ \frac{|a_j|}{|c_j| + |b_j|} \right\}, \tag{2.21}$$

$$\mu_i = \max_{ik+1 \le j \le (i+1)k} \left\{ \frac{|d_j|}{|a_j|} \right\}, \tag{2.22}$$

entonces

$$\delta = \min_{0 \le i \le p-1} \{\delta_i\}, \tag{2.23}$$

$$\mu = \frac{\max_{0 \le i \le p-1} \{\mu_i\}}{1 - \delta^{-1}}, \tag{2.24}$$

El siguiente algoritmo BSP modifica el algoritmo 2.3 para realizar el cálculo de δ y μ en paralelo.

Algoritmo 2.3 Algoritmo BSP del método OPM para un sistema tridiagonal, con cálculo de δ y μ en paralelo

Superpaso 1

- El procesador principal P_0 calcula $k = \frac{n}{p}$.
- Para $i = 1, 2, \ldots, p-1$, el procesador P_0 envía a P_i los elementos c_{ik+j}, a_{ik+j}, b_{ik+j} y d_{ik+j}, con $j = 1, 2, \ldots, k$, excepto $b_n = 0$.

Superpaso 2

- Para $i = 0, 1, \ldots, p-1$, P_i calcula δ_i y μ_i de acuerdo con las expresiones (2.21) y (2.22).
- Para $i = 1, 2, \ldots, p-1$, P_i envía a P_0 los valores de los parámetros δ_i y μ_i.

Superpaso 3

- El procesador P_0 calcula los parámetros δ, μ y m haciendo uso de las expresiones (2.23), (2.24) y (2.14).
- Para $i = 1, 2, \ldots, p-2$, el procesador principal P_0 envía a P_i las filas j-ésima y $(m+k+j)$-ésima de $\hat{A}_i$ y $\hat{\boldsymbol{d}}_i$, con $j = 1, 2, \ldots, m$.
- El procesador principal envía a P_{p-1} la fila j-ésima de $\hat{A}_{p-1}$ y $\hat{\boldsymbol{d}}_{p-1}$, con $j = 1, 2, \ldots, m$.
- Para $i = 1, 2, \ldots, p-1$, el procesador P_0 envía a P_i el valor de m.

Superpaso 4

Como el superpaso 2 del algoritmo 2.2.

Al repartir el cálculo de δ y μ entre p procesadores, el número de operaciones baja de $3n+3$ a $3k+3$ y el coste aritmético de $3n+8k+16m+6$ a $11k+16m+6$; se han introducido dos superpasos más, por lo que el coste de sincronización aumenta de $2l$ a $4l$ y

se ha incrementado la comunicación en $2(p-1)$ elementos (ya que desde los procesadores remotos se envía a P_0 los parámetros δ_i y μ_i), en consecuencia, el coste de comunicación vale

$$[5n - 5k - 12m + 8mp - p + 2 + 2(p-1)]g = (5n - 5k - 12m + 8mp + p)g.$$

A continucación se obtendrá con detalle el coste del algoritmo 2.3.

Coste del superpaso 1. La obtención de k requiere una operación. Cada procesador recibe del principal $4k$ elementos, por lo que el coste de comunicación es $[4k(p-1)-1]g$ y el coste del superpaso

$$1 + (4n - 4k - 1)g + l. \tag{2.25}$$

Coste del superpaso 2. El cálculo de δ_i en cualquier procesador requiere $2k$ operaciones mientras que el de μ_i necesita k operaciones. Cada procesador (excepto P_0) envía al procesador principal los valores de δ_i y μ_i que en total recibe $2(p-1)$ elementos; en consecuencia el coste del superpaso es

$$3k + (2p - 2)g + l. \tag{2.26}$$

Coste del superpaso 3. El procesador principal debe realizar 3 operaciones para calcular μ y 9 para obtener el valor de m por lo que el coste aritmético es 12. En este superpaso el procesador que más comunica es el principal que transmite m a todos los procesadores, envía a P_{p-1} la cantidad de $4m-1$ elementos y a P_i, para $i = 1, 2, \ldots, p-2$, un total de $8m-2$ elementos; o sea, comunica $p-1+4m-1+(8m-2)(p-2)$ elementos. Se tiene por tanto el siguiente coste para el superpaso 3

$$12 + (-12m + 8mp - p + 2)g + l. \tag{2.27}$$

Coste del superpaso 4. Puesto que este superpaso coincide con el superpaso 2 del algoritmo 2.2, su coste viene dado por la expresión (2.20)

La suma de las expresiones (2.25), (2.26), (2.27) y (2.20) da como resultado el coste global del algoritmo 2.3

$$(11k + 16m + 6) + (5n - 5k - 12m + 8mp + p)g + 4l. \tag{2.28}$$

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	26
	2	3294	9.5	25
	4	5366	12.4	25
	6	8164	12.5	25

(a) *IBM SP2 switch*

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	8
	2	20235	709.7	3
	4	54163	1362.6	9
	6	121958	3211.2	9

(b) *IBM SP2 ethernet*

s	p	l	g	$n_{\frac{1}{2}}$
16.4	1	23	0.2	22
	2	2556	6.9	5
	4	5152	7.4	4
	6	7538	6.8	4

(c) *Cluster de PC's*

Tabla 2.2: *Valores de parámetros BSP.*

2.6 Resultados numéricos

En esta sección se analizan los tiempos previstos teóricamente y los tiempos experimentales de los algoritmos 2.2 y 2.3 (a los que se referenciará en las tablas y figuras como OPM1 y OPM2 respectivamente). Las pruebas experimentales se han realizado en el IBM SP2 y el *cluster* de PC's cuyas características se han descrito en la subsección 1.5.3. Por comodidad, en la tabla 2.2 se repiten los parámetros obtenidos para esas máquinas que se muestran en la tabla 1.1.

El tiempo teórico se ha obtenido considerando que el tamaño de bloque de los mensajes que se comunican entre los procesadores es $k = \frac{n}{p}$ y que el coste de comunicación de una

palabra de 32 bits es

$$g(k) = \left(\frac{n_{\frac{1}{2}}}{k} + 1 \right) g_\infty,$$

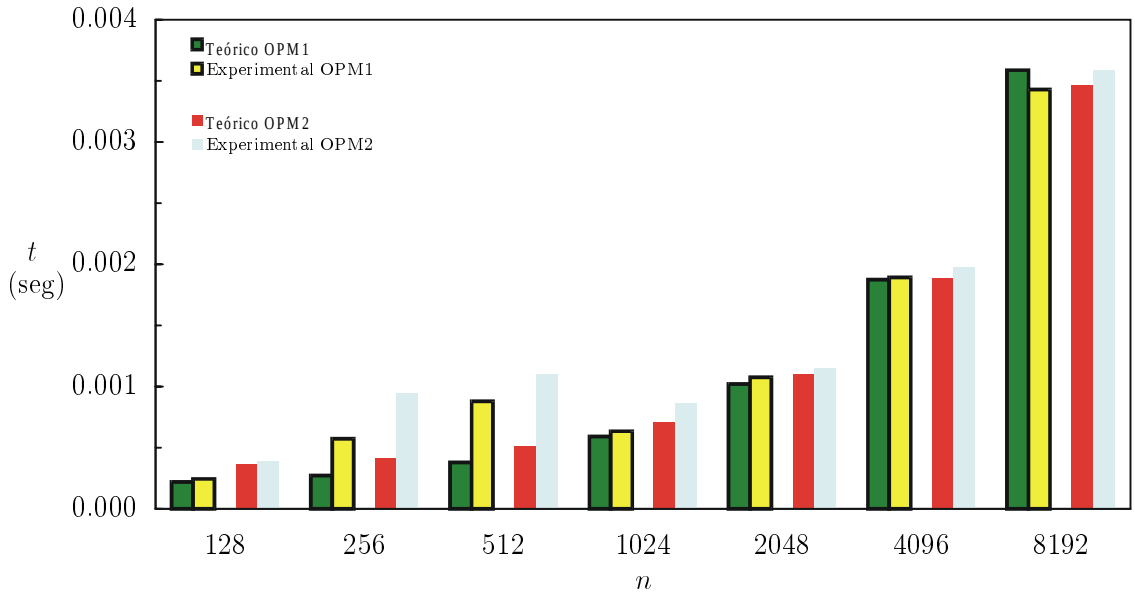
además se ha tomado $m = 5$. Los algoritmos 2.2 (OPM1) y 2.3 (OPM2) han sido implementados en Fortran usando la versión v1.3 de la librería BSPLib, se ha generando el sistema (2.4) obteniendo aleatoriamente los elementos de la matriz de coeficientes A y, para simplificar, eligiendo el vector de términos independientes $\boldsymbol{d}$ de manera que la solución sea $\boldsymbol{x} = [1, 1, \ldots, 1]^T$. A la elección aleatoria de los coeficientes se le ha añadido la restricción de que $m = 5$ (véase la tabla 2.1). Si en el ejemplo 2.2 se toma $\epsilon = 10^{-5}$, se obtiene de manera sencilla un sistema tridiagonal en el que $m = 5$.

En general, para medir cualquier tiempo se ha adoptado la estrategia que consiste en realizar un cierto número de veces el mismo experimento y quedarse con el valor mínimo, repitiéndose esta rutina en diversos días y horas, siempre con uso exclusivo de las máquinas (o los nodos necesarios en el caso del IBM SP2) y tomando como valor definitivo el mínimo de todos ellos. Esta estrategia, si es exhaustiva, permite medir el mejor rendimiento posible.

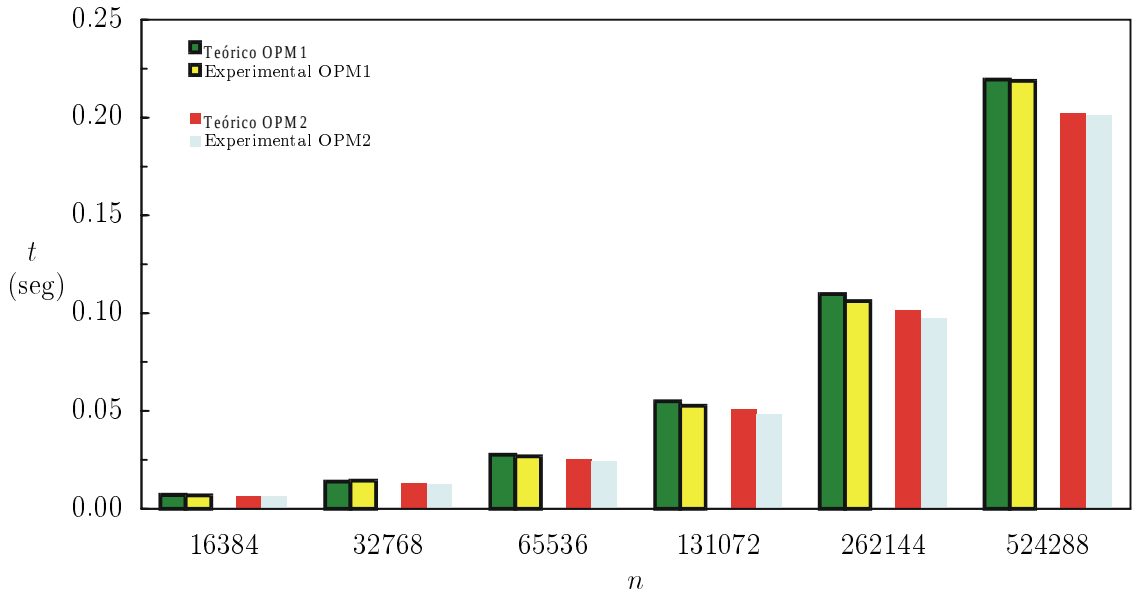
En las tablas 2.3, 2.4 y 2.5 y figuras 2.4, 2.5 y 2.6 se muestran los tiempos teóricos y experimentales, medidos en segundos, de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en el IBM SP2 para 2, 4 y 6 procesadores utilizando *switch*, mientras que en las tablas 2.6, 2.7 y 2.8 y figuras 2.7, 2.8 y 2.9 se muestran los tiempos teóricos y experimentales, medidos en segundos, de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en el IBM SP2 para 2, 4 y 6 procesadores utilizando *ethernet*. Estos tiempos han sido obtenidos para diferentes tamaños de la matriz de coeficientes en un rango que varía desde 128 a 524288 para 2 y 4 procesadores y desde 126 a 516096 para 6 procesadores. En las tablas 2.9, 2.10 y 2.11 y figuras 2.10, 2.11 y 2.12 se muestran los tiempos teóricos y experimentales, medidos en segundos, de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en el *cluster* de PC's para 2, 4 y 6 procesadores, en este caso el tamaño de la matriz de coeficientes varía desde 128 a 65536 para 2 y 4 procesadores y desde 126 a 64512 para 6 procesadores.

IBM SP2 2 procesadores *switch*				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0002	0.0002	0.0004	0.0004
256	0.0003	0.0006	0.0004	0.0009
512	0.0004	0.0009	0.0005	0.0011
1024	0.0006	0.0006	0.0007	0.0009
2048	0.0010	0.0011	0.0011	0.0011
4096	0.0019	0.0019	0.0019	0.0020
8192	0.0036	0.0034	0.0035	0.0036
16384	0.0070	0.0068	0.0066	0.0063
32768	0.0139	0.0144	0.0129	0.0123
65536	0.0276	0.0268	0.0255	0.0243
131072	0.0550	0.0527	0.0507	0.0484
262144	0.1098	0.1062	0.1012	0.0974
524288	0.2193	0.2187	0.2020	0.2012

Tabla 2.3: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 2 procesadores interconectados mediante switch, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

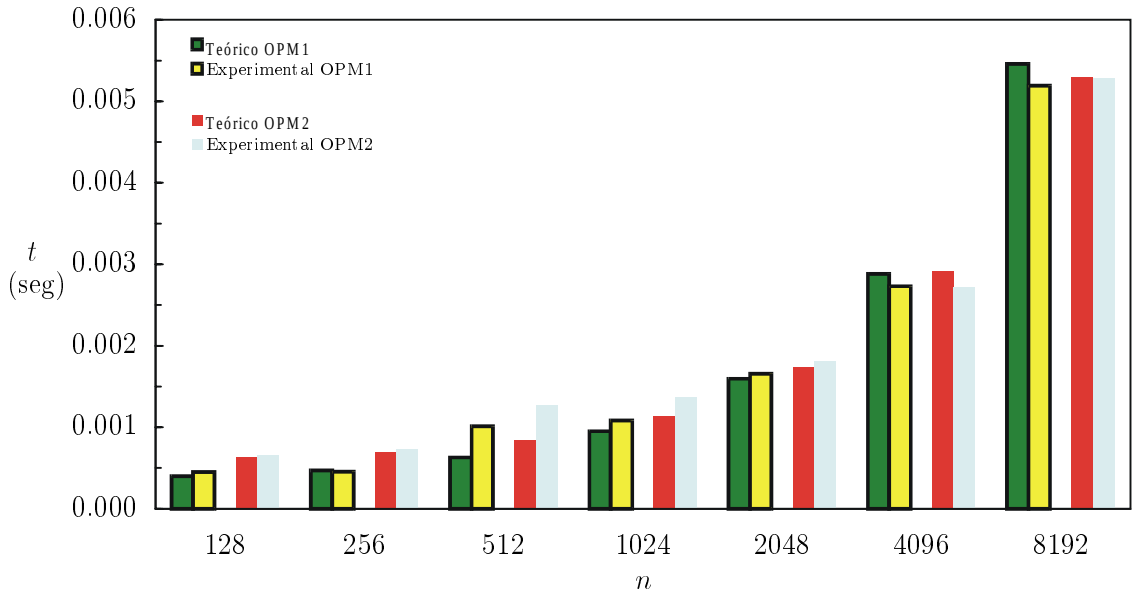


(b) $16384 \leq n \leq 524288$

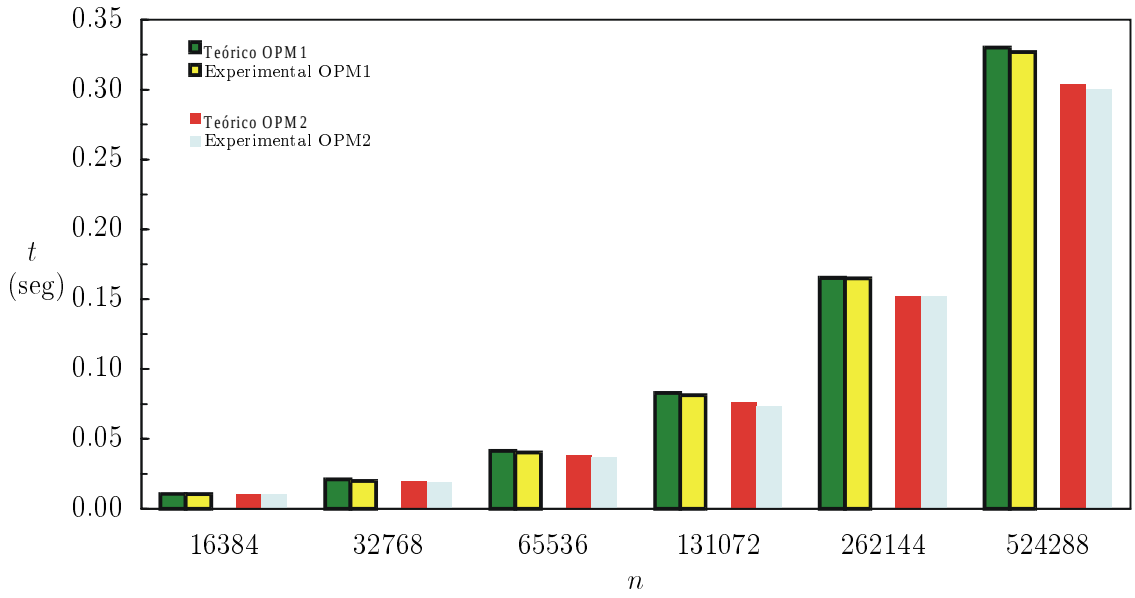
Figura 2.4: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 2 procesadores interconectados mediante switch.*

IBM SP2 4 procesadores *switch*				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0004	0.0005	0.0006	0.0007
256	0.0005	0.0005	0.0007	0.0007
512	0.0006	0.0010	0.0008	0.0013
1024	0.0010	0.0011	0.0011	0.0014
2048	0.0016	0.0017	0.0017	0.0018
4096	0.0029	0.0027	0.0029	0.0027
8192	0.0055	0.0052	0.0053	0.0053
16384	0.0106	0.0105	0.0100	0.0099
32768	0.0209	0.0200	0.0195	0.0188
65536	0.0415	0.0403	0.0385	0.0370
131072	0.0828	0.0812	0.0764	0.0734
262144	0.1652	0.1649	0.1523	0.1517
524288	0.3301	0.3269	0.3041	0.3006

Tabla 2.4: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 4 procesadores interconectados mediante switch, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

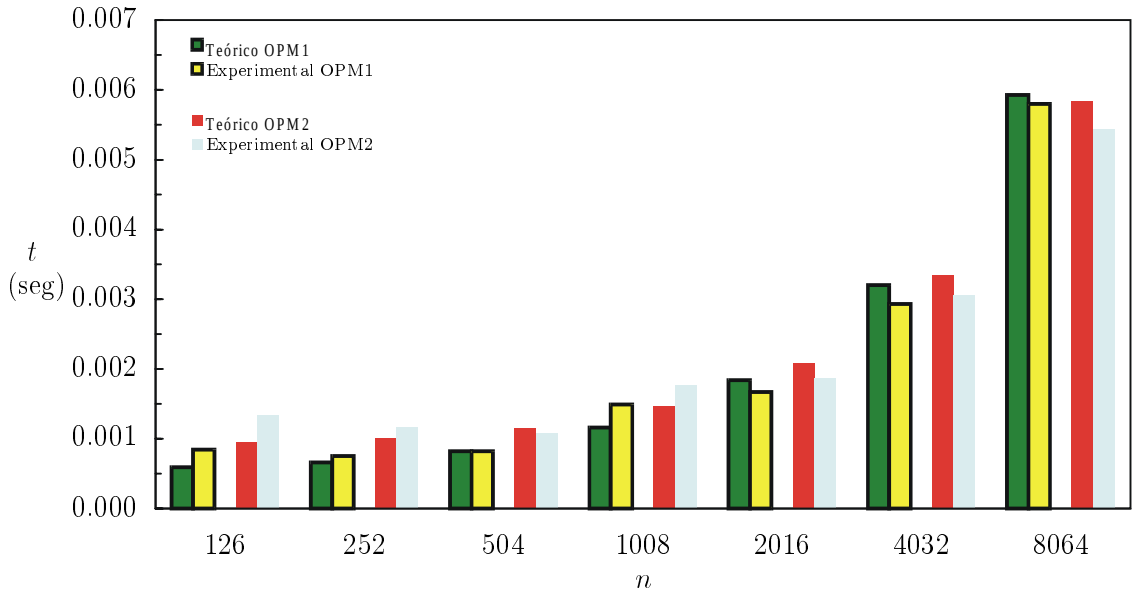


(b) $16384 \leq n \leq 524288$

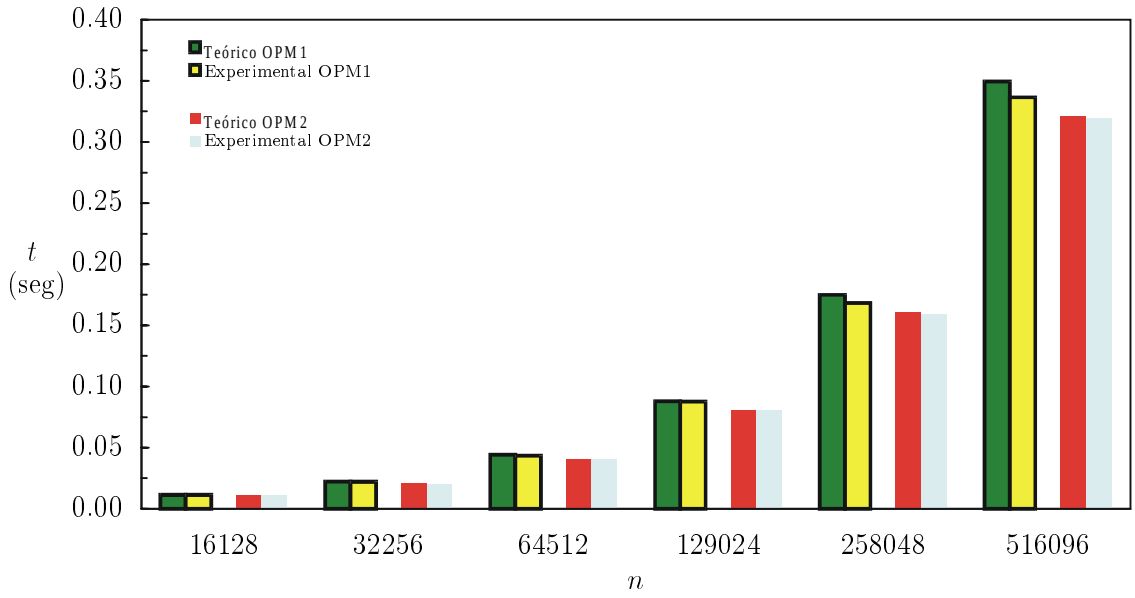
Figura 2.5: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 4 procesadores interconectados mediante switch.*

IBM SP2 6 procesadores *switch*				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0006	0.0008	0.0009	0.0013
252	0.0007	0.0008	0.0010	0.0012
504	0.0008	0.0008	0.0012	0.0011
1008	0.0012	0.0015	0.0015	0.0018
2016	0.0018	0.0017	0.0021	0.0019
4032	0.0032	0.0029	0.0033	0.0031
8064	0.0059	0.0058	0.0058	0.0054
16128	0.0114	0.0113	0.0108	0.0106
32256	0.0223	0.0221	0.0209	0.0200
64512	0.0441	0.0432	0.0409	0.0401
129024	0.0877	0.0877	0.0809	0.0808
258048	0.1750	0.1682	0.1610	0.1591
516096	0.3494	0.3366	0.3211	0.3196

Tabla 2.5: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 6 procesadores interconectados mediante switch, para* $126 \leq n \leq 516096$.



(a) $126 \leq n \leq 8064$

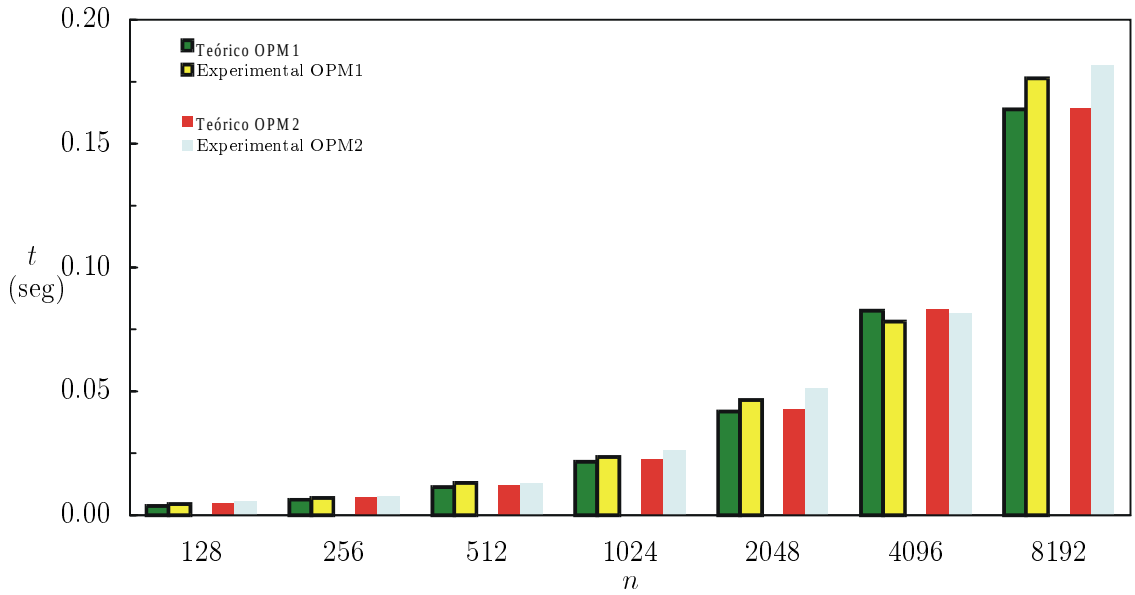


(b) $16128 \leq n \leq 516096$

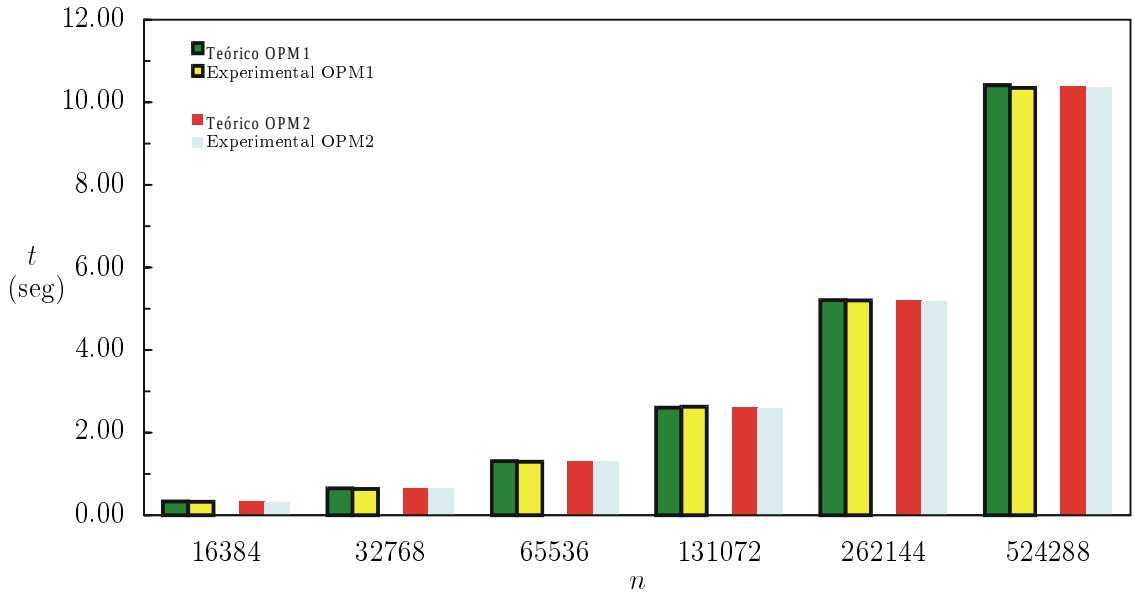
Figura 2.6: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 6 procesadores interconectados mediante switch.*

IBM SP2 2 procesadores *ethernet*				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0037	0.0046	0.0046	0.0056
256	0.0063	0.0069	0.0072	0.0076
512	0.0114	0.0130	0.0122	0.0130
1024	0.0215	0.0235	0.0224	0.0264
2048	0.0419	0.0465	0.0427	0.0511
4096	0.0826	0.0782	0.0833	0.0815
8192	0.1639	0.1764	0.1646	0.1817
16384	0.3267	0.3213	0.3271	0.3125
32768	0.6522	0.6367	0.6520	0.6581
65536	1.3033	1.2936	1.3020	1.3145
131072	2.6053	2.6271	2.6019	2.5932
262144	5.2095	5.2019	5.2016	5.1770
524288	10.4178	10.3532	10.4012	10.3583

Tabla 2.6: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 2 procesadores interconectados mediante ethernet, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

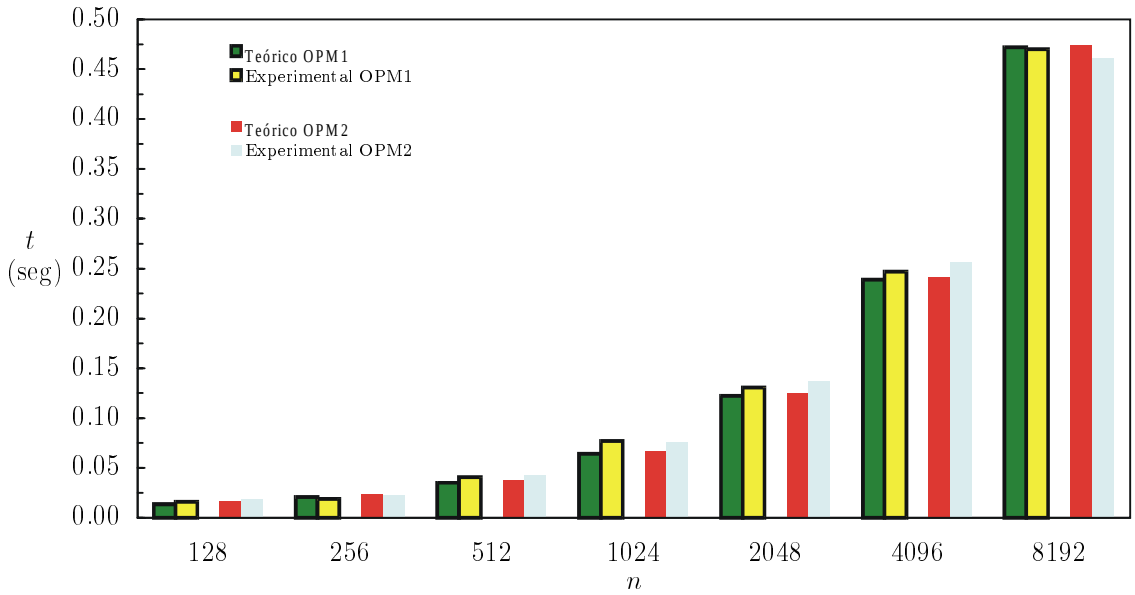


(b) $16384 \leq n \leq 524288$

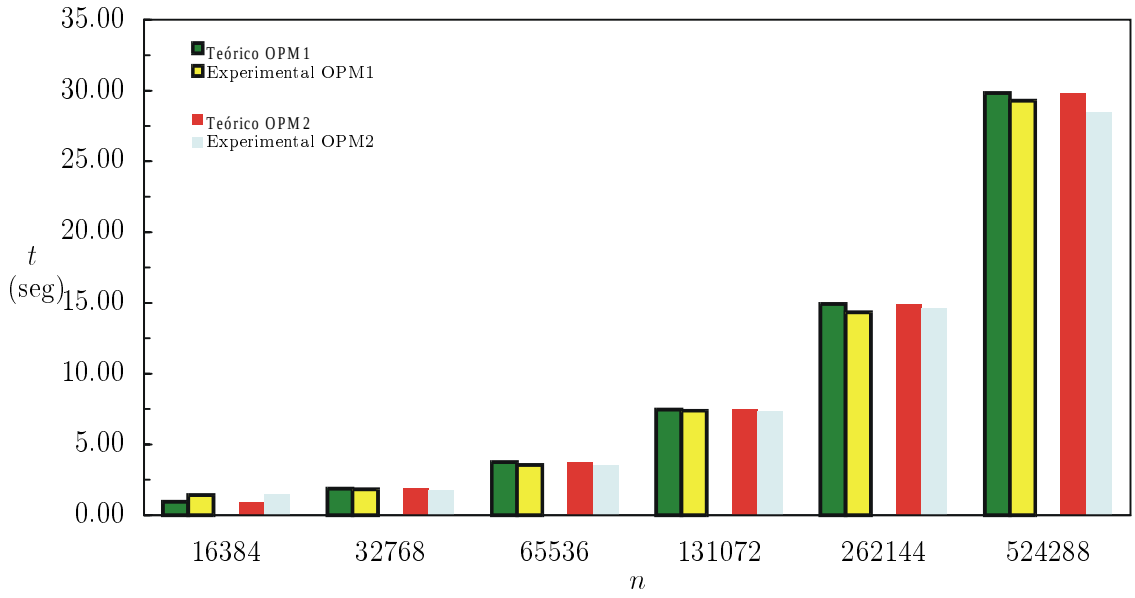
Figura 2.7: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 2 procesadores interconectados mediante ethernet.*

IBM SP2 4 procesadores *ethernet*				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0136	0.0159	0.0162	0.0181
256	0.0207	0.0189	0.0232	0.0219
512	0.0352	0.0408	0.0376	0.0425
1024	0.0642	0.0770	0.0667	0.0757
2048	0.1225	0.1309	0.1249	0.1369
4096	0.2390	0.2467	0.2413	0.2562
8192	0.4720	0.4700	0.4741	0.4612
16384	0.9380	1.4150	0.9397	1.4889
32768	1.8700	1.8280	1.8709	1.7308
65536	3.7341	3.5331	3.7333	3.5249
131072	7.4623	7.3731	7.4582	7.3865
262144	14.9186	14.3351	14.9080	14.6099
524288	29.8313	29.2817	29.8075	28.4591

Tabla 2.7: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 4 procesadores interconectados mediante ethernet, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

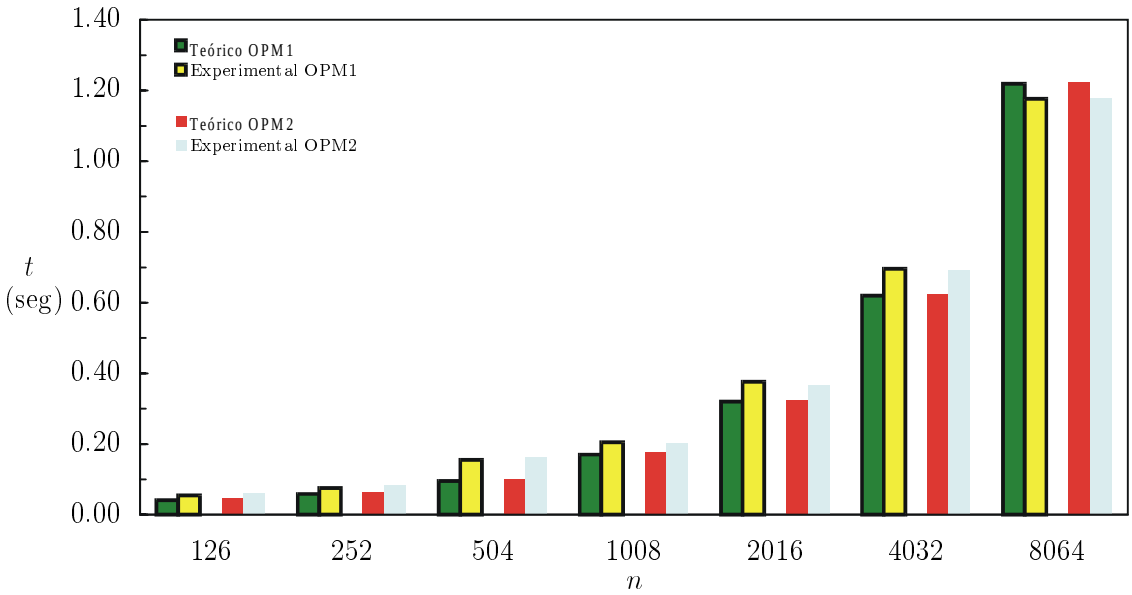


(b) $16384 \leq n \leq 524288$

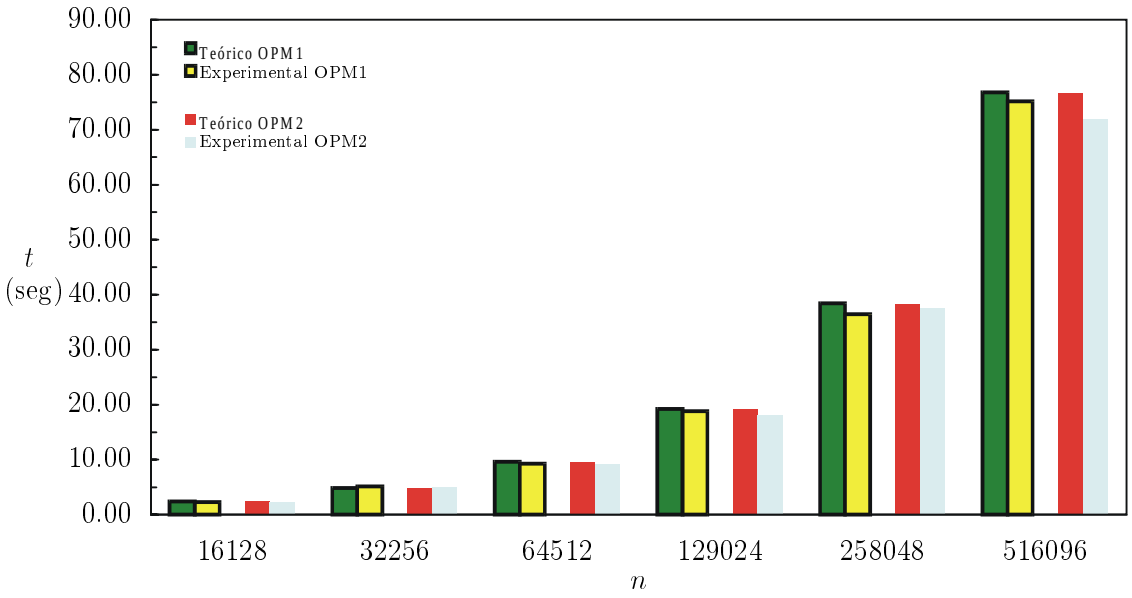
Figura 2.8: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 4 procesadores interconectados mediante ethernet.*

IBM SP2 6 procesadores *ethernet*				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0412	0.0546	0.0471	0.0622
252	0.0586	0.0758	0.0644	0.0824
504	0.0954	0.1552	0.1012	0.1627
1008	0.1700	0.2054	0.1758	0.2019
2016	0.3198	0.3761	0.3255	0.3665
4032	0.6196	0.6954	0.6252	0.6916
8064	1.2194	1.1769	1.2247	1.1771
16128	2.4190	2.3189	2.4239	2.3334
32256	4.8182	5.1487	4.8222	5.0714
64512	9.6167	9.3015	9.6189	9.2169
129024	19.2137	18.8224	19.2123	18.1330
258048	38.4077	36.4179	38.3992	37.6399
516096	76.7957	75.1663	76.7728	71.9154

Tabla 2.8: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un IBM SP2 con 6 procesadores interconectados mediante ethernet, para* $126 \leq n \leq 516096$.



(a) $126 \leq n \leq 8064$

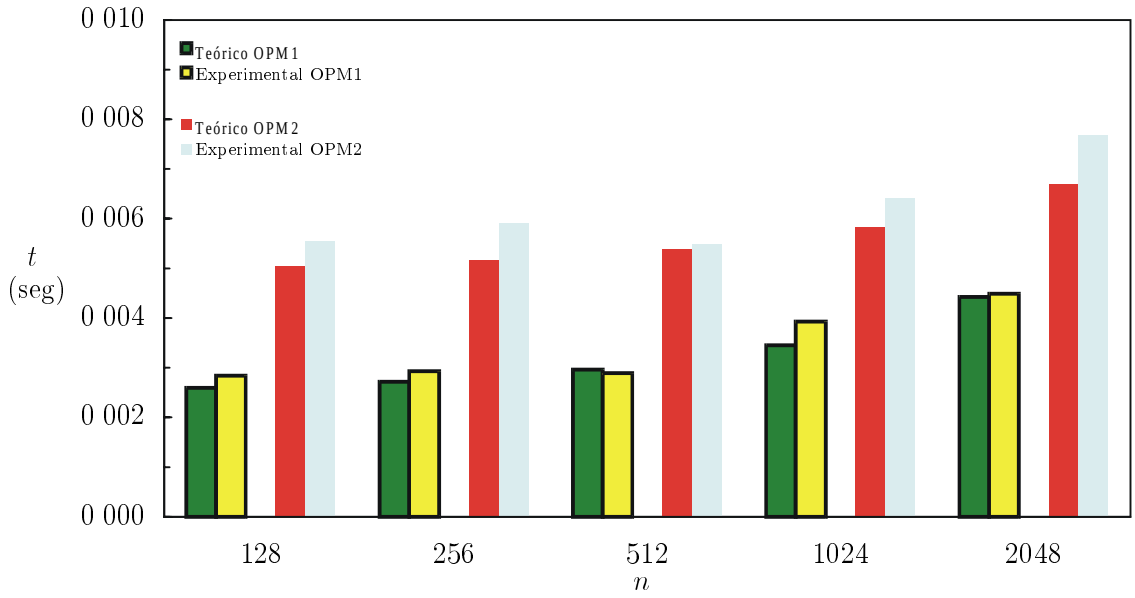


(b) $16128 \leq n \leq 516096$

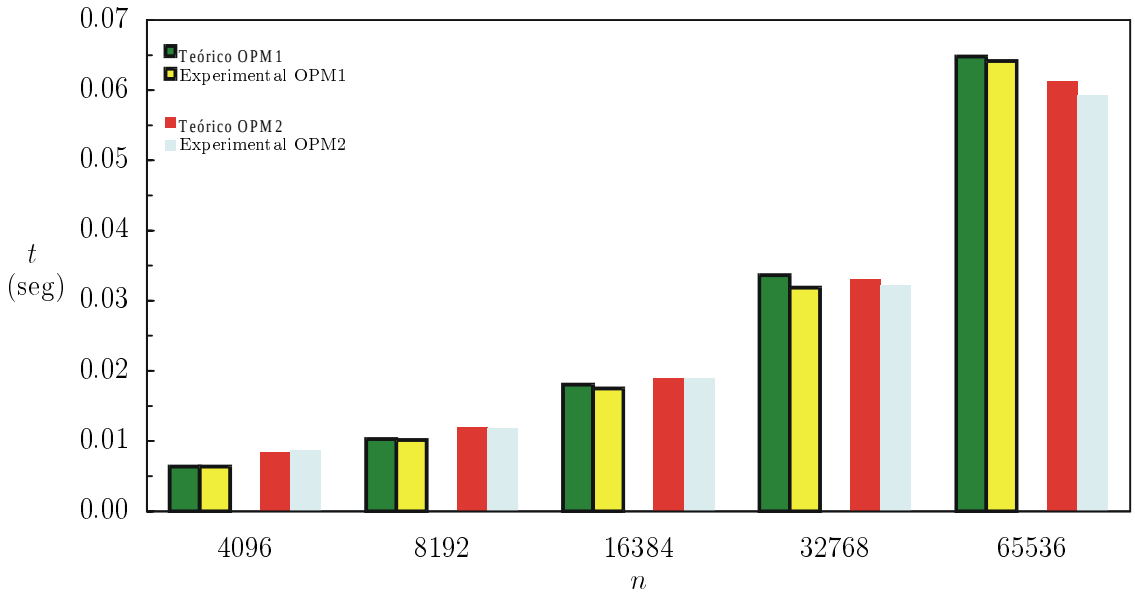
Figura 2.9: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con 6 procesadores interconectados mediante ethernet.*

Cluster de PC's 2 procesadores				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0026	0.0028	0.0050	0.0055
256	0.0027	0.0029	0.0052	0.0059
512	0.0030	0.0029	0.0054	0.0055
1024	0.0035	0.0039	0.0058	0.0064
2048	0.0044	0.0045	0.0067	0.0077
4096	0.0064	0.0064	0.0085	0.0086
8192	0.0103	0.0101	0.0120	0.0119
16384	0.0181	0.0175	0.0190	0.0190
32768	0.0336	0.0319	0.0331	0.0323
65536	0.0648	0.0641	0.0613	0.0592

Tabla 2.9: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un cluster de PC's, para* 2 *procesadores y* $128 \leq n \leq 65536$.



(a) $128 \leq n \leq 2048$

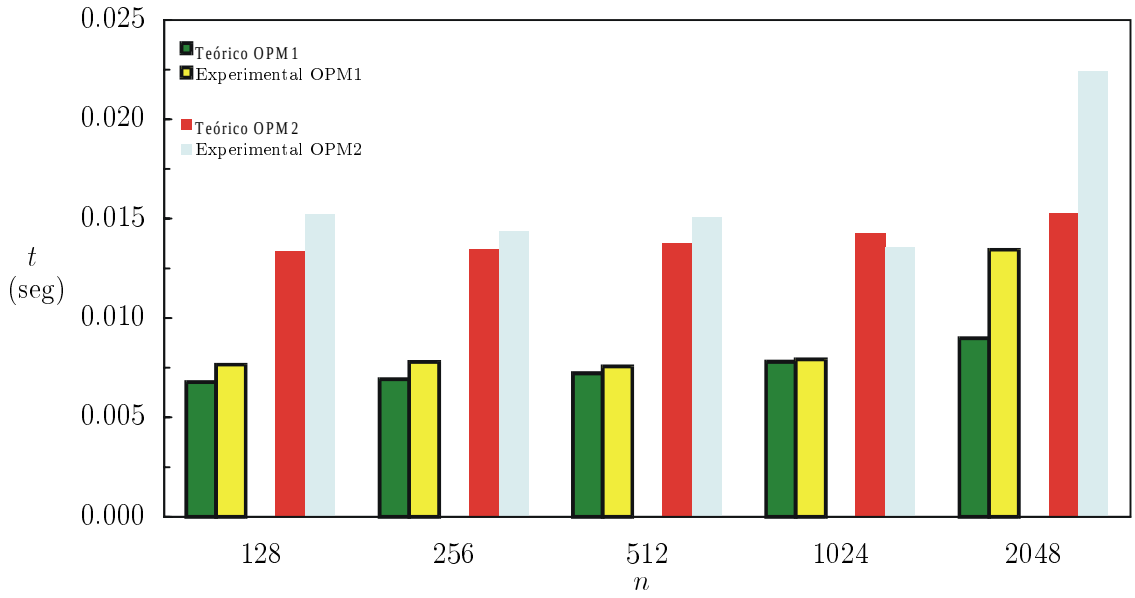


(b) $4096 \leq n \leq 65536$

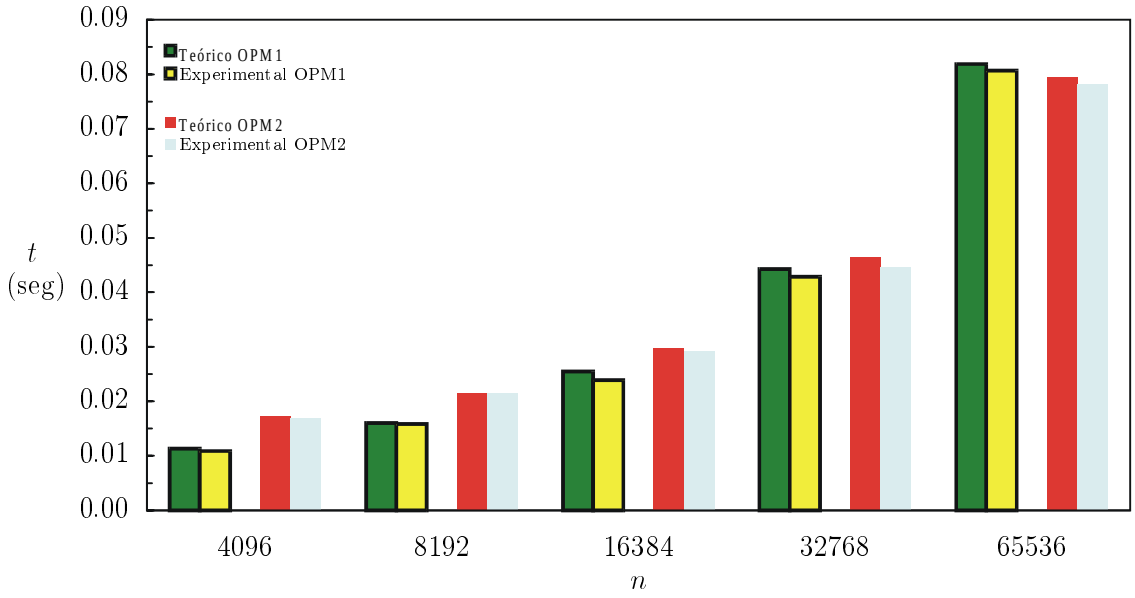
Figura 2.10: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un cluster de PC's con 2 procesadores.*

Cluster de PC's 4 procesadores				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0068	0.0077	0.0134	0.0152
256	0.0069	0.0078	0.0135	0.0144
512	0.0072	0.0076	0.0137	0.0151
1024	0.0078	0.0079	0.0143	0.0136
2048	0.0090	0.0134	0.0153	0.0224
4096	0.0113	0.0109	0.0174	0.0169
8192	0.0160	0.0158	0.0215	0.0215
16384	0.0254	0.0239	0.0298	0.0292
32768	0.0442	0.0429	0.0463	0.0446
65536	0.0819	0.0807	0.0795	0.0782

Tabla 2.10: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un cluster de PC's, para 4 procesadores y* $128 \leq n \leq 65536$.



(a) $128 \leq n \leq 2048$

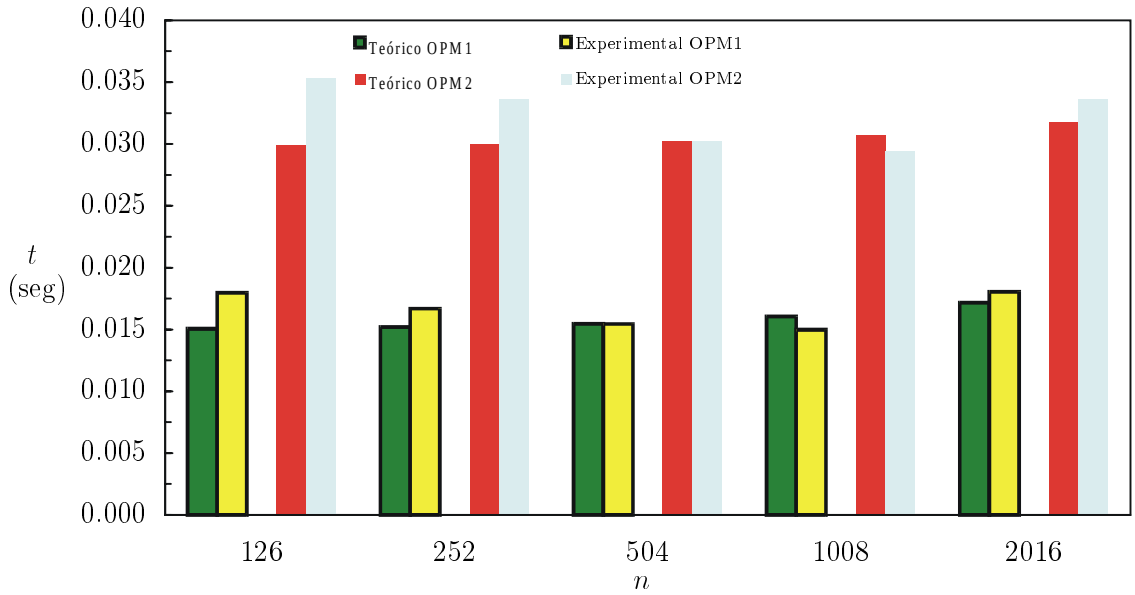


(b) $4096 \leq n \leq 65536$

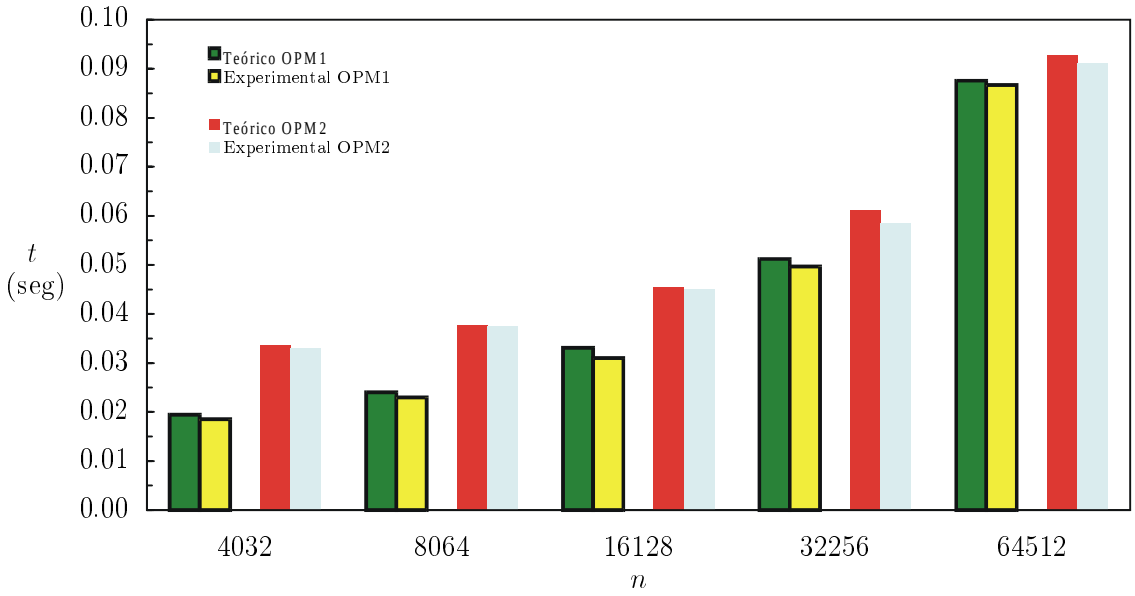
Figura 2.11: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un cluster de PC's con 4 procesadores.*

Cluster de PC's 6 procesadores				
	OPM1		OPM2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0150	0.0180	0.0299	0.0353
252	0.0152	0.0167	0.0300	0.0336
504	0.0155	0.0154	0.0302	0.0302
1008	0.0160	0.0150	0.0307	0.0294
2016	0.0172	0.0180	0.0317	0.0336
4032	0.0194	0.0186	0.0337	0.0330
8064	0.0240	0.0230	0.0376	0.0374
16128	0.0331	0.0310	0.0455	0.0451
32256	0.0512	0.0496	0.0612	0.0585
64512	0.0876	0.0867	0.0926	0.0912

Tabla 2.11: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2), medidos en un cluster de PC's, para* 6 *procesadores y* $126 \leq n \leq 64512$.



(a) $126 \leq n \leq 2016$



(b) $4032 \leq n \leq 64512$

Figura 2.12: *Tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un cluster de PC's con 6 procesadores.*

En las figuras 2.13, 2.14 y 2.15 se muestra el porcentaje de desviación del tiempo experimental con respecto al teórico en función del tamaño del sistema[1].

Las mayores diferencias entre el tiempo previsto y el experimental se obtienen en los sistemas de tamaño pequeño en los que, por lo general, suele ser mayor el tiempo experimental que el teórico. A medida que aumenta el tamaño de sistema el tiempo obtenido experimentalmente se ajusta mejor al esperado, para tamaños grandes la desviación (salvo alguna excepción) oscila entre el 0% y el 5%.

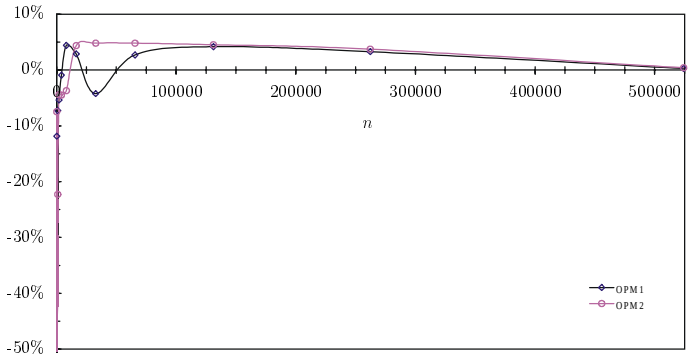
Este comportamiento obedece al ajuste teórico realizado, mediante la curva de Miller dada por la expresión (1.2), sobre los valores experimentales del coste de comunicación, $g(x)$, de una palabra de 32 bits para tamaños de mensaje x y patrón de comunicación trid. Por ejemplo, para el IBM SP2 *switch* y 2 procesadores los valores experimentales de $g(x)$ para tamaños de mensaje x comprendidos entre 64 y 512 palabras está por encima del coste teórico, como puede observarse en las figuras 1.15 y 1.15, en la práctica los tiempos obtenidos para valores de n comprendidos entre 128 y 1024 son[2] mayores que los previstos, como puede observarse en la figura 2.4(a). En esta figura se aprecia, además, que para valores de n comprendidos entre 2048 y 8192 el tiempo obtenido experimentalmente supera al predicho (aunque en menor proporción que para los tamaños comprendidos entre 128 y 1024). En este hecho influyen dos factores:

- el primero depende del ajuste realizado mediante la curva de Miller, que puede variar dependiendo de la estimación del valor de $n_{\frac{1}{2}}$;
- el segundo está relacionado con la propia estructura del algoritmo ya que se comunican algunos mensajes de tamaño 1 palabra de 32 bits y en el cómputo teórico se considera que su coste es el mismo que el de enviar una palabra para un bloque de tamaño k. El coste para tamaño 1 es mayor que para tamaño k y aunque tiene peso en el tiempo obtenido para tamaños de sistema pequeños es irrelevante para tamaños de sistema grandes.

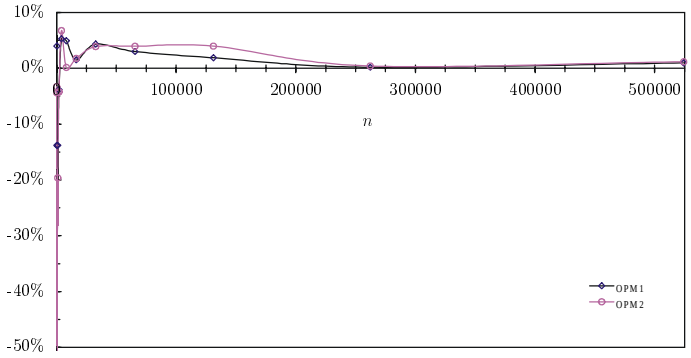
En las tablas 2.12, 2.13 y 2.14 se muestra el algoritmo más rápido de los dos, teórica y

[1] Se considera que si el tiempo teórico es 100 y el tiempo experimental es 90, se ha producido un 10% de desviación y en cambio si el tiempo experimental es 110 se ha producido un -10% de desviación.

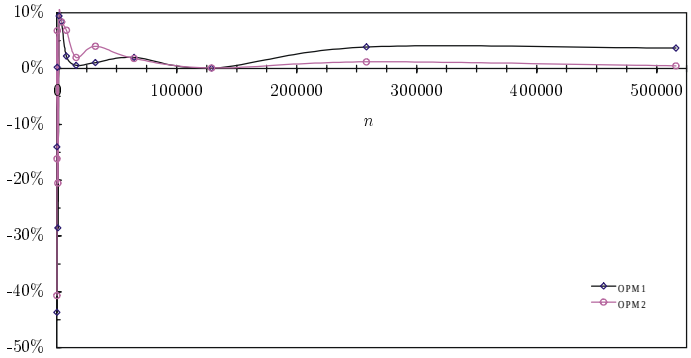
[2] Los tamaños de bloque enviados para un sistema de tamaño n y 2 procesadores son iguales a $\frac{n}{2}$.



(a) 2 *procesadores*

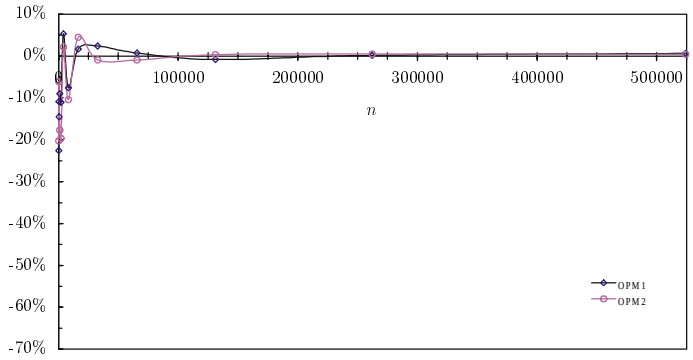


(b) 4 *procesadores*

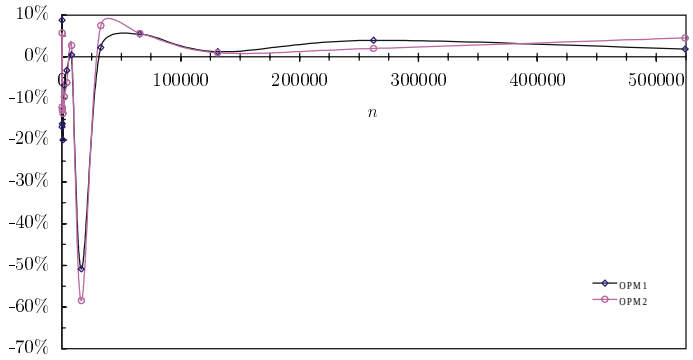


(c) 6 *procesadores*

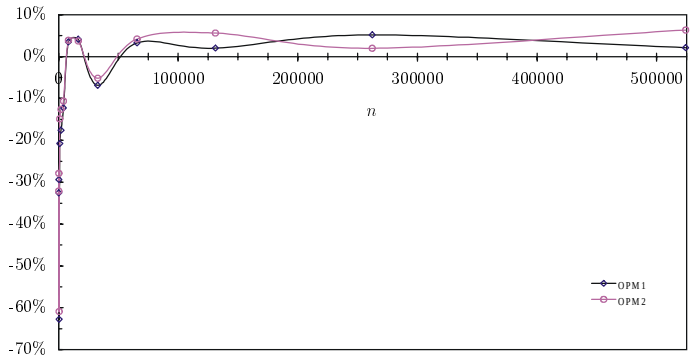
Figura 2.13: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con switch.*



(a) 2 *procesadores*

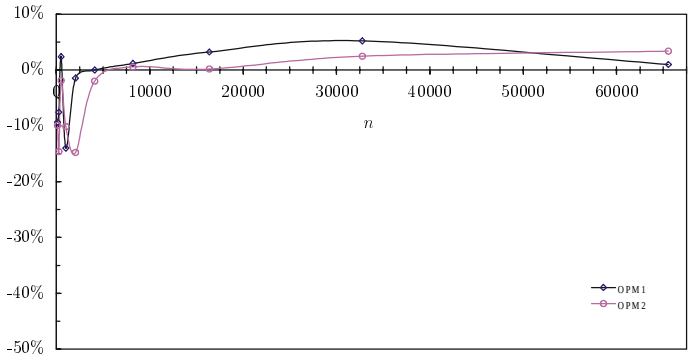


(b) 4 *procesadores*

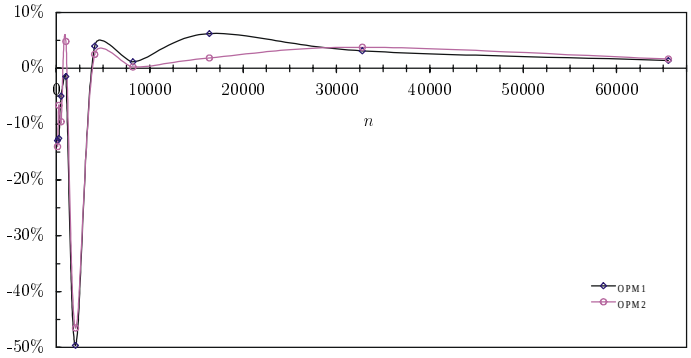


(c) 6 *procesadores*

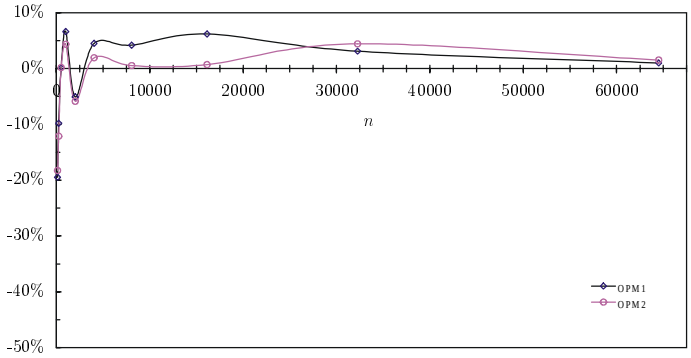
Figura 2.14: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 con ethernet.*



(a) 2 *procesadores*



(b) 4 *procesadores*



(c) 6 *procesadores*

Figura 2.15: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un cluster de PC's.*

IBM SP2 *switch*							
	2 procesadores		4 procesadores			6 procesadores	
n	Teór.	Exper.	Teór.	Exper.	n	Teór.	Exper.
128	OPM1	OPM1	OPM1	OPM1	126	OPM1	OPM1
256	OPM1	OPM1	OPM1	OPM1	252	OPM1	OPM1
512	OPM1	OPM1	OPM1	OPM1	504	OPM1	OPM1
1024	OPM1	OPM1	OPM1	OPM1	1008	OPM1	OPM1
2048	OPM1	OPM1	OPM1	OPM1	2016	OPM1	OPM1
4096	OPM1	OPM1	OPM1	OPM2	4032	OPM1	OPM1
8192	OPM2	OPM1	OPM2	OPM1	8064	OPM2	OPM2
16384	OPM2	OPM2	OPM2	OPM2	16128	OPM2	OPM2
32768	OPM2	OPM2	OPM2	OPM2	32256	OPM2	OPM2
65536	OPM2	OPM2	OPM2	OPM2	64512	OPM2	OPM2
131072	OPM2	OPM2	OPM2	OPM2	129024	OPM2	OPM2
262144	OPM2	OPM2	OPM2	OPM2	258048	OPM2	OPM2
524288	OPM2	OPM2	OPM2	OPM2	516096	OPM2	OPM2

Tabla 2.12: *Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 switch.*

experimentalmente, en cada una de las máquinas para los distintos tamaños de sistema. Se observa que, por lo general, a partir de 17000 ecuaciones (excepto en el IBM SP2 *switch*) es más rápido el algoritmo 2.3 (OPM2), que realiza el cálculo de m en paralelo.

No tiene mucho sentido estudiar el *speed-up* (incremento de velocidad) con respecto al tiempo de ejecución del propio algoritmo en un procesador por que se conoce el mejor algoritmo secuencial: el método de eliminación de Gauss para sistemas tridiagonales, cuyo coste para el sistema dado por la expresión 2.4 es $8n - 7$. Con respecto al método de eliminación de Gauss para sistemas tridiagonales se obtienen los resultados de la tabla 2.15, considerando los tiempos teóricos de los sistemas de tamaño máximo de los que se han obtenido resultados experimentales en cada una de las máquinas.

IBM SP2 *ethernet*							
	2 procesadores		4 procesadores			6 procesadores	
n	Teór.	Exper.	Teór.	Exper.	n	Teór.	Exper.
128	OPM1	OPM1	OPM1	OPM1	126	OPM1	OPM1
256	OPM1	OPM1	OPM1	OPM1	252	OPM1	OPM1
512	OPM1	OPM2	OPM1	OPM1	504	OPM1	OPM1
1024	OPM1	OPM1	OPM1	OPM2	1008	OPM1	OPM2
2048	OPM1	OPM1	OPM1	OPM1	2016	OPM1	OPM2
4096	OPM1	OPM1	OPM1	OPM1	4032	OPM1	OPM2
8192	OPM1	OPM1	OPM1	OPM2	8064	OPM1	OPM1
16384	OPM1	OPM2	OPM1	OPM1	16128	OPM1	OPM1
32768	OPM2	OPM1	OPM1	OPM2	32256	OPM1	OPM2
65536	OPM2	OPM1	OPM2	OPM2	64512	OPM1	OPM2
131072	OPM2	OPM2	OPM2	OPM1	129024	OPM2	OPM2
262144	OPM2	OPM2	OPM2	OPM1	258048	OPM2	OPM1
524288	OPM2	OPM1	OPM2	OPM2	516096	OPM2	OPM2

Tabla 2.13: *Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 ethernet.*

Cluster de PC's							
	2 procesadores		4 procesadores			6 procesadores	
n	Teór.	Exper.	Teór.	Exper.	n	Teór.	Exper.
128	OPM1	OPM1	OPM1	OPM1	126	OPM1	OPM1
256	OPM1	OPM1	OPM1	OPM1	252	OPM1	OPM1
512	OPM1	OPM1	OPM1	OPM1	504	OPM1	OPM1
1024	OPM1	OPM1	OPM1	OPM1	1008	OPM1	OPM1
2048	OPM1	OPM1	OPM1	OPM1	2016	OPM1	OPM1
4096	OPM1	OPM1	OPM1	OPM1	4032	OPM1	OPM1
8192	OPM1	OPM1	OPM1	OPM1	8064	OPM1	OPM1
16384	OPM1	OPM1	OPM1	OPM1	16128	OPM1	OPM1
32768	OPM2	OPM1	OPM1	OPM1	32256	OPM1	OPM1
65536	OPM2	OPM2	OPM2	OPM2	64512	OPM1	OPM1

Tabla 2.14: *Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un cluster de PC's.*

IBM SP2 *switch*				
	OPM1		OPM2	
p	S_p	E_p	S_p	E_p
2	0.42	21.25%	0.46	23.07%
4	0.28	7.06%	0.31	7.66%
6	0.26	4.38%	0.29	4.76%

(a) *IBM SP2 switch*

IBM SP2 *ethernet*				
	OPM1		OPM2	
p	S_p	E_p	S_p	E_p
2	0.01	0.45%	0.01	0.45%
4	0.00	0.08%	0.00	0.08%
6	0.00	0.02%	0.00	0.02%

(b) *IBM SP2 ethernet*

Cluster de PC's				
	OPM1		OPM2	
p	S_p	E_p	S_p	E_p
2	0.49	24.61%	0.52	26.02%
4	0.39	9.74%	0.40	10.03%
6	0.36	5.97%	0.34	5.65%

(c) *Cluster de PC's*

Tabla 2.15: *Speed-up (S_p) y eficiencia (E_p) de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un IBM SP2 y un cluster de PC's.*

s	p	l	g	$n_{\frac{1}{2}}$
12	1	68	0.3	94
	2	164	0.7	71
	4	168	0.7	66
	8	175	0.8	59
	16	181	0.9	61
	32	201	1.1	28
	64	148	1	27
	128	301	1.1	20
	256	387	1.2	15

(a) *CRAY T3D*

s	p	l	g	$n_{\frac{1}{2}}$
46.7	1	86	2.12	9
	2	269	0.87	33
	4	357	0.87	40
	8	506	0.81	40
	16	751	1.04	38
	32	1252	1.31	45

(b) *CRAY T3E*

Tabla 2.16: *Valores de parámetros BSP.*

La eficiencia que se obtiene no es muy buena, y en caso del IBM SP2 con *ethernet* es pésima; sin duda un factor determinante es el valor de g. Para otras máquinas con menor valor de g se obtiene mejor eficiencia, como es el caso de un CRAY T3D o un CRAY T3E. Los valores de los parámetros de estas máquinas se muestran en la tabla 2.16; para las mismas se obtiene el *speed-up* y la eficiencia que figura en la tabla 2.17, considerando el mismo tamaño de sistema que para la tabla 2.15.

CRAY T3D				
	OPM1		OPM2	
p	S_p	E_p	S_p	E_p
2	1.13	56.43%	1.43	71.57%
4	1.27	31.68%	1.97	49.20%
8	1.39	17.38%	2.56	31.97%
16	1.42	8.91%	2.85	17.84%
32	1.35	4.22%	2.65	8.29%
64	1.43	2.23%	3.02	4.73%
128	1.38	1.08%	2.82	2.20%
256	1.32	0.52%	2.61	1.02%

(a) *CRAY T3D*

CRAY T3E				
	OPM1		OPM2	
p	S_p	E_p	S_p	E_p
2	0.99	49.45%	1.21	60.70%
4	1.13	28.15%	1.65	41.19%
8	1.39	17.32%	2.54	31.72%
16	1.35	8.41%	2.55	15.95%
32	1.22	7.60%	2.12	13.25%

(b) *CRAY T3E*

Tabla 2.17: *Speed-up (S_p) y eficiencia (E_p) de los algoritmos 2.2 (OPM1) y 2.3 (OPM2) en un CRAY T3D y un CRAY T3E.*

Capítulo 3

Métodos bidireccionales

Se considera de nuevo el problema de resolver el sistema

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

donde A es la matriz tridiagonal dada por (3.2), estrictamente diagonal dominante e irreducible y

$$\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

es el vector de términos independientes.

3.1 Factorización LDU de matrices tridiagonales

Una matriz tridiagonal A de tamaño $n \times n$,

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix}, \tag{3.2}$$

se puede descomponer como producto de tres matrices

$$A = LDU \tag{3.3}$$

donde D es una matriz diagonal y las matrices L y U son bidiagonales con la característica particular de que su diagonal principal está formada por unos; más explícitamente

$$A = \begin{bmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & l_{n-1} & 1 & \\ & & & l_n & 1 \end{bmatrix} \begin{bmatrix} e_1 & & & & \\ & e_2 & & & \\ & & \ddots & & \\ & & & e_{n-1} & \\ & & & & e_n \end{bmatrix} \begin{bmatrix} 1 & u_1 & & & \\ & 1 & u_2 & & \\ & & \ddots & \ddots & \\ & & & 1 & u_{n-1} \\ & & & & 1 \end{bmatrix},$$

véase Golub y Van Loan [50].

Una vez obtenida la factorización (3.3) se puede resolver el sistema de ecuaciones lineales (3.1) en dos pasos: en el primero, se resuelve el sistema bidiagonal $LD\boldsymbol{z} = \boldsymbol{d}$, obteniéndose así el vector auxiliar $\boldsymbol{z}$; en el segundo, se obtiene la solución final del sistema (3.1) resolviendo el sistema bidiagonal $U\boldsymbol{x} = \boldsymbol{z}$.

El producto LDU viene dado por

$$LDU = \begin{bmatrix} e_1 & u_1e_1 & & & & \\ l_2e_1 & u_1l_2e_1+e_2 & u_2e_2 & & & \\ & l_3e_2 & u_2l_3e_2+e_3 & u_3e_3 & & \\ & \ddots & \ddots & \ddots & & \\ & & l_{n-1}e_{n-2} & u_{n-2}l_{n-1}e_{n-2}+e_{n-1} & u_{n-1}e_{n-1} & \\ & & & l_ne_{n-1} & u_{n-1}l_ne_{n-1}+e_n \end{bmatrix},$$

relacionando los elementos de A con los del producto LDU se tiene

$$a_1 = e_1,$$

y para $i = 2, 3, \ldots, n$,

$$c_i = l_ie_{i-1}, \tag{3.4}$$

$$a_i = u_{i-1}l_ie_{i-1} + e_i, \tag{3.5}$$

$$b_{i-1} = u_{i-1}e_{i-1}. \tag{3.6}$$

De las expresiones (3.4) y (3.6) se tiene

$$l_i = \frac{c_i}{e_{i-1}}, \qquad u_{i-1} = \frac{b_{i-1}}{e_{i-1}}, \quad \text{para} \quad i = 2, 3, \ldots, n, \tag{3.7}$$

y sustituyendo la expresión (3.7) en la expresión (3.5) se obtiene

$$e_i = a_i - \frac{b_{i-1}}{e_{i-1}}\frac{c_i}{e_{i-1}}e_{i-1}, \quad \text{para} \quad i = 2, 3, \ldots, n.$$

Por tanto, los elementos de la diagonal principal de D pueden obtenerse de forma sencilla mediante la siguiente relación de recurrencia

$$\begin{aligned} e_1 &= a_1, \\ e_i &= a_i - c_i\frac{b_{i-1}}{e_{i-1}}, \quad \text{para} \quad i = 2, 3, \ldots, n, \end{aligned}$$

y los de las matrices L y U mediante las recurrencias (3.7).

3.2 Método bidireccional para dos procesadores

3.2.1 Descripción del método

Aunque a primera vista la factorización (3.3) parece de naturaleza secuencial, es posible encontrar un paralelismo de grado dos en los cálculos (véase Ortega [91]), es decir, se pueden utilizar dos procesadores para obtener una factorización no exactamente igual que la anterior, pero sí equivalente en cuanto a complejidad. La idea original aparece en la eliminación Gaussiana bidireccional (*two-sided Gaussian elimination*) introducida por Babuska [4].

Para paralelizar la factorización (3.3), se supone además que $n = 2q$ para algún $q \geq 1$. Se considera la matriz de coeficientes particionada en bloques del siguiente modo

$$A = \left[\begin{array}{ccccc|ccccc} a_1 & b_1 & & & & & & & & \\ c_2 & a_2 & b_2 & & & & & & & \\ & \ddots & \ddots & \ddots & & & & & & \\ & & c_q & a_q & & b_q & & & & \\ \hline & & & c_{q+1} & & a_{q+1} & b_{q+1} & & & \\ & & & & & \ddots & \ddots & \ddots & & \\ & & & & & & c_{n-1} & a_{n-1} & b_{n-1} & \\ & & & & & & & c_n & a_n & \end{array}\right].$$

Sean las matrices

$$M = \left[\begin{array}{ccccc|ccccc} 1 & & & & & & & & & \\ l_2 & 1 & & & & & & & & \\ & l_3 & 1 & & & & & & & \\ & & \ddots & \ddots & & & & & & \\ & & & l_q & 1 & & & & & \\ \hline & & & & l_{q+1} & 1 & u_{q+1} & & & \\ & & & & & & 1 & u_{q+2} & & \\ & & & & & & & 1 & \ddots & \\ & & & & & & & & \ddots & u_{n-1} \\ & & & & & & & & & 1 \end{array}\right],$$

$$D = \left[\begin{array}{ccccc|ccccc} e_1 & & & & & & & & & \\ & e_2 & & & & & & & & \\ & & e_3 & & & & & & & \\ & & & \ddots & & & & & & \\ & & & & e_q & & & & & \\ \hline & & & & & e_{q+1} & & & & \\ & & & & & & e_{q+2} & & & \\ & & & & & & & \ddots & & \\ & & & & & & & & e_{n-1} & \\ & & & & & & & & & e_n \end{array}\right].$$

y

$$V = \left[\begin{array}{ccccc|ccccc} 1 & u_1 & & & & & & & & \\ & 1 & u_2 & & & & & & & \\ & & 1 & \ddots & & & & & & \\ & & & \ddots & u_{q-1} & & & & & \\ & & & & 1 & u_q & & & & \\ \hline & & & & & 1 & & & & \\ & & & & & l_{q+2} & 1 & & & \\ & & & & & & l_{q+3} & \ddots & & \\ & & & & & & & & \ddots & \\ & & & & & & & & l_n & 1 \end{array}\right],$$

tales que

$$A = MDV. \tag{3.8}$$

Si se procede como en la factorización LDU de la matriz A, los elementos de M, D y V pueden calcularse mediante las siguientes relaciones

- Sea $e_1 = a_1$.
- Para $i = 2, 3, \ldots, q$, sea

$$u_{i-1} = \frac{b_{i-1}}{e_{i-1}}; \quad l_i = \frac{c_i}{e_{i-1}} \tag{3.9}$$

  y calcúlese

$$\begin{aligned} e_i &= a_i - l_i e_{i-1} u_{i-1} \\ &= a_i - c_i u_{i-1}. \end{aligned} \tag{3.10}$$

- Sea

$$u_q = \frac{b_q}{e_q}, \quad l_{q+1} = \frac{c_{q+1}}{e_q}.$$

- Sea $e_n = a_n$ y calcúlese

$$l_n = \frac{c_n}{e_n}, \quad u_{n-1} = \frac{b_{n-1}}{e_n}. \tag{3.11}$$

- Para $i = n-1, n-2, \ldots, q+2$, calcúlese

$$\begin{aligned} e_i &= a_i - l_{i+1} e_{i+1} u_i \\ &= a_i - l_{i+1} b_i. \end{aligned} \tag{3.12}$$

  y sea

$$l_i = \frac{c_i}{e_i}; \quad u_{i-1} = \frac{b_{i-1}}{e_i}. \tag{3.13}$$

- Finalmente, calcúlese

$$\begin{aligned} e_{q+1} &= a_{q+1} - l_{q+1} e_q u_q - l_{q+2} e_{q+2} u_{q+2} \\ &= a_{q+1} - c_{q+1} u_q - l_{q+2} b_{q+1}. \end{aligned} \tag{3.14}$$

Para realizar la implementación de los cálculos anteriores en un ordenador con dos procesadores, se utiliza una técnica similar a la desarrollada por Van der Vorst [103] para matrices simétricas. Obsérvese que mediante las relaciones de recurrencia (3.10) y (3.12) se calculan todos los elementos de la diagonal principal de D excepto e_{q+1}, estos cálculos pueden realizarse perfectamente en paralelo (sin necesidad de comunicación de datos) ya que cada procesador contiene los elementos necesarios. Sin embargo, para calcular e_{q+1} (en el procesador P_1) mediante la relación (3.14) se necesita conocer el valor de e_q, calculado en el procesador P_0. Por lo tanto, es necesario un paso de comunicación para transmitir los datos necesarios desde el procesador P_0 al procesador P_1. En la figura 3.1 se muestran los cálculos en cada uno de los procesadores y las comunicaciones necesarias cuando $n = 8$.

Para resolver el sistema (3.1) usando la factorización (3.8), se necesita resolver en primer lugar el sistema $MD\boldsymbol{z} = \boldsymbol{d}$, para $\boldsymbol{z}$, y a continuación el sistema $V\boldsymbol{x} = \boldsymbol{z}$, para $\boldsymbol{x}$. Como consecuencia de la estructura especial de las matrices M y D, el vector

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

$\boxed{P_0}$	$\boxed{P_1}$
$e_1 = a_1; \quad u_1 = \dfrac{b_1}{e_1}$	$e_8 = a_8; \quad l_8 = \dfrac{c_8}{e_8}$
$e_2 = a_2 - c_2u_1; \quad u_2 = \dfrac{b_2}{e_2}$	$e_7 = a_7 - l_8b_7; \quad l_7 = \dfrac{c_7}{e_7}$
$e_3 = a_3 - c_3u_2; \quad u_3 = \dfrac{b_3}{e_3}$	$e_6 = a_6 - l_7b_6; \quad l_6 = \dfrac{c_6}{e_6}$
$e_4 = a_4 - c_4u_3; \quad u_4 = \dfrac{b_4}{e_4}$	
$u_4 \quad \longrightarrow\longrightarrow\longrightarrow$	$\boxed{u_4}$
	$e_5 = a_5 - c_5\boxed{u_4} - l_6b_5$

Figura 3.1: *Cálculo en paralelo de los elementos de D para una matriz con $n = 8$.*

se puede obtener directamente de las expresiones

$$z_1 = \frac{d_1}{e_1}, \tag{3.15}$$

$$z_i = \frac{d_i - c_i z_{i-1}}{e_i}, \quad \text{para } i = 2, 3, \ldots, q, \tag{3.16}$$

$$z_n = \frac{d_n}{e_n}, \tag{3.17}$$

$$z_i = \frac{d_i - b_i z_{i+1}}{e_i}, \quad \text{para } i = n-1, n-2, \ldots, q+2, \tag{3.18}$$

$$z_{q+1} = \frac{d_{q+1} - c_{q+1}z_q - b_{q+1}z_{q+2}}{e_{q+1}}. \tag{3.19}$$

Como consecuencia de la estructura de la matriz V, al resolver $V\boldsymbol{x} = \boldsymbol{z}$ se obtiene que

$$x_{q+1} = z_{q+1}$$

y por tanto

$$x_i = z_i - u_i x_{i+1}, \quad \text{para } i = q, q-1, \ldots, 1, \tag{3.20}$$

$$x_i = z_i - l_i x_{i-1}, \quad \text{para } i = q+2, \ldots, n. \tag{3.21}$$

$\boxed{P_0}$	$\boxed{P_1}$
$e_1 = a_1; \quad u_1 = \dfrac{b_1}{e_1}; \quad z_1 = \dfrac{d_1}{e_1}$	$e_8 = a_8; \quad l_8 = \dfrac{c_8}{e_8}; \quad z_8 = \dfrac{d_8}{e_8}$
$e_2 = a_2 - c_2u_1; \quad u_2 = \dfrac{b_2}{e_2}; \quad z_2 = \dfrac{d_2 - c_2z_1}{e_2}$	$e_7 = a_7 - l_8b_7; \quad l_7 = \dfrac{c_7}{e_7}; \quad z_7 = \dfrac{d_7 - b_7z_8}{e_7}$
$e_3 = a_3 - c_3u_2; \quad u_3 = \dfrac{b_3}{e_3}; \quad z_3 = \dfrac{d_3 - c_3z_2}{e_3}$	$e_6 = a_6 - l_7b_6; \quad l_6 = \dfrac{c_6}{e_6}; \quad z_6 = \dfrac{d_6 - b_6z_7}{e_6}$
$e_4 = a_4 - c_4u_3; \quad u_4 = \dfrac{b_4}{e_4}; \quad z_4 = \dfrac{d_4 - c_4z_3}{e_4}$	
$u_4, \quad z_4 \longrightarrow\longrightarrow\longrightarrow$	$\boxed{u_4}, \boxed{z_4}$
$\boxed{z_6}, \boxed{l_6}$	$\longleftarrow\longleftarrow\longleftarrow \quad z_6, \quad l_6$
$e_5 = a_5 - c_5u_4 - \boxed{l_6}b_5$	$e_5 = a_5 - c_5\boxed{u_4} - l_6b_5$
$x_5 = z_5 = \dfrac{d_5 - c_5z_4 - b_5\boxed{z_6}}{e_5}$	$z_5 = \dfrac{d_5 - c_5\boxed{z_4} - b_5z_6}{e_5}$
$x_4 = z_4 - u_4x_5$	
$x_3 = z_3 - u_3x_4$	$x_6 = z_6 - l_6x_5$
$x_2 = z_2 - u_2x_3$	$x_7 = z_7 - l_7x_6$
$x_1 = z_1 - u_1x_2$	$x_8 = z_8 - l_8x_7$

Figura 3.2: *Cálculo en paralelo de la solución del sistema (2.4) para una matriz con* $n = 8$.

Debe observarse que todos los cálculos expresados en las relaciones (3.15)–(3.21) pueden desarrollarse en paralelo en dos procesadores excepto el del elemento centrales z_{q+1}. Todas las comunicaciones que deben realizarse están orientadas precisamente al cálculo de este elemento.

En la figura 3.2 se muestran los cálculos realizados en cada procesador para obtener la solución del sistema (3.1) cuando $n = 8$.

Ejemplo 3.1 *Sea el sistema* $A\boldsymbol{x} = \boldsymbol{d}$*, donde*

$$A = \begin{bmatrix} 66 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 66 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 66 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 66 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 66 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 66 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 66 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 66 \end{bmatrix} \quad y \quad \boldsymbol{d} = \begin{bmatrix} 67 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 65 \end{bmatrix}.$$

Puesto que cada término independiente se ha obtenido como suma de los coeficientes de su ecuación, la solución del sistema tiene todas sus componentes iguales a 1.

Si se aplican las relaciones (3.10)–(3.14) se obtiene

$$M = \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{66} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{66}{4357} & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{66}{4357} & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & -\frac{66}{4357} & 1 & \frac{66}{4357} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \frac{66}{4357} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{1}{66} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right],$$

$$D = \left[\begin{array}{cccc|cccc} 66 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{4357}{66} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{4357}{66} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{4357}{66} & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \frac{2179}{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{4357}{66} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{4357}{66} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 66 \end{array}\right],$$

$$V = \left[\begin{array}{cccc|cccc} 1 & \frac{1}{66} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \frac{66}{4357} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \frac{66}{4357} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \frac{66}{4357} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{66}{4357} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{66}{4357} & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{66} & 1 \end{array}\right].$$

Utilizando las expresiones (3.15)-(3.19) se obtiene la solución del sistema $MD\boldsymbol{z} = \boldsymbol{d}$,

$$\boldsymbol{z} = \begin{bmatrix} \frac{67}{66} \\ \frac{4423}{4357} \\ \frac{4423}{4357} \\ \frac{4423}{4357} \\ 1 \\ \frac{4291}{4357} \\ \frac{4291}{4357} \\ \frac{65}{66} \end{bmatrix}.$$

Finalmente, mediante las relaciones (3.20) y (3.21) se obtiene la solución de $V\boldsymbol{x} = \boldsymbol{z}$ *que coincide con la del sistema* $A\boldsymbol{x} = \boldsymbol{d}$;

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

■

3.2.2 Algoritmo BSP para dos procesadores

Todo lo expuesto en la subsección 3.2.1 sugiere el siguiente algoritmo BSP para dos procesadores. Se supone que la matriz A y el vector de términos independientes $\boldsymbol{d}$ están

inicialmente almacenados en el procesador principal P_0.

Algoritmo 3.1 Algoritmo BSP para dos procesadores

Superpaso 1

El procesador P_0 envía a P_1 los elementos a_i, b_i, c_i, d_i, para $i = q+1, q+2, \ldots, n$, excepto $b_n = 0$.

Superpaso 2

Cálculo de los elementos e_i y z_i.

- En el procesador P_0,
  - Sea $e_1 = a_1$ y calcúlese z_1 mediante la expresión (3.15).
  - Para $i = 2, 3, \ldots, q$, calcúlese u_{i-1}, e_i y z_i usando las expresiones (3.9), (3.10) y (3.16).
  - El procesador P_0 envía al procesador P_1 los elementos u_q y z_q.
- En el procesador P_1,
  - Sea $e_n = a_n$ y calcúlense l_n, z_n utilizando las expresiones (3.11) y (3.17).
  - Para $i = n-1, n-2, \ldots, q+2$, calcúlense e_i, l_i y z_i usando las expresiones (3.12) y (3.13) y (3.18).
  - El procesador P_1 envía a P_0 los elementos l_{q+2} y z_{q+2}.

Superpaso 3

Cálculo de los elementos e_{q+1}, z_{q+1} y de la solución.

- Ambos procesadores calculan e_{q+1} y z_{q+1} de acuerdo con (3.14) y (3.19), respectivamente. Sea $x_{q+1} = z_{q+1}$, entonces
  - En el procesador P_0 se calcula x_i, para $i = q, q-1, \ldots, 1$, usando la expresión (3.20).

  - En el procesador P_1 se calcula x_i, para $i = q+2, q+3, \ldots, n$, usando la expresión (3.21).

- El procesador P_1 envía al procesador P_0 las componentes x_i, para $i = q+2, q+3, \ldots, n$, del vector solución $\boldsymbol{x}$.

El coste computacional del algoritmo 3.1 viene dado por la suma de los costes individuales de cada uno de los superpasos que lo integran.

Coste del superpaso 1. En este primer superpaso sólo hay comunicación, por lo que el coste aritmético es cero. El coste de comunicación viene determinado por el envío desde el procesador principal a P_1 de $4q - 1$ elementos, correspondientes a la matriz de coeficientes y el vector de términos independientes. Por tanto, el coste total de este superpaso es

$$(2n - 1)g + l.$$

Coste del superpaso 2. El procesador P_0 realiza una división para calcular z_1, una división para calcular u_i, con $i = 1, 2, \ldots, q$, dos operaciones para el cálculo de d_i y tres operaciones para el cálculo de z_i, con $i = 2, 3, \ldots, q$; en total $6q-4$ operaciones. Mientras tanto, el procesador P_1 realiza $6q-10$ (seis menos) operaciones, que corresponden a las $q-1$ divisiones necesarias para calcular l_i, con $i = n, n-1, \ldots, q+2$, una división para el cálculo de z_n, 2 operaciones para calcular d_i y 3 para calcular z_i, con $i = n-1, n-2, \ldots, q+2$.

El procesador principal envía dos elementos a P_1 y éste a su vez comunica dos a P_0. Luego el coste de comunicación es $2g$. En consecuencia, el coste del superpaso es

$$3n - 4 + 2g + l.$$

Coste del superpaso 3. En este superpaso se deben contabilizar las operaciones realizadas por el procesador principal ya que P_1 realiza dos operaciones menos. Para calcular los elementos e_{q+1} y z_{q+1} se requieren 9 operaciones y para el cálculo de x_i, con $i = 1, 2, \ldots, q$, se precisan $2q$ operaciones, luego el coste aritmético es $2q + 9$.

En cuanto al coste de comunicación, el procesador P_1 envía al procesador principal las componentes $x_{q+2}, x_{q+3}, \ldots, x_n$ de la solución $\boldsymbol{x}$, lo que supone un coste de $(q-1)g$. Se

tiene por tanto que el coste de este superpaso es

$$n + 9 + \left(\frac{n}{2} - 1\right) g + l.$$

Sumando los costes se obtiene que el coste del algoritmo 3.1 es

$$4n + 5 + \frac{5}{2}ng + 3l. \tag{3.22}$$

3.3 Método bidireccional para un número par de procesadores

En esta sección se presenta una generalización para p procesadores del método estudiado en la sección 3.2, basada en algunos de los resultados obtenidos en el método de las particiones superpuestas que se ha estudiado en el capítulo 2. Se supone que tanto el número de procesadores, p, como el orden, n, de la matriz de coeficientes es par; asimismo se supone que existe un número natural k tal que $k = \dfrac{n}{p}$.

3.3.1 Descripción del método

Sea m el valor obtenido al aplicar la expresión (2.14), esto es

$$m = \left\lceil \frac{1}{\log \delta^{-1}} \log \frac{\epsilon(1 - \delta^{-2})}{\mu} \right\rceil, \tag{3.23}$$

donde ϵ es el máximo error permitido al resolver el sistema (3.1) mediante el método de las particiones superpuestas y

$$\mu = \frac{\max\limits_{1 \le i \le n} \left\{ \left| \dfrac{d_i}{a_i} \right| \right\}}{1 - \delta^{-1}},$$

es una cota superior de la norma infinito de la solución del mismo. Se define

$$m' = 2 \left\lceil \frac{m}{2} \right\rceil. \tag{3.24}$$

Se considera el sistema (3.1) particionado en $\frac{p}{2}$ bloques de tamaño $2k \times 2k$ como sigue

$$\begin{bmatrix} A_1 & B_1 & & & \\ C_2 & A_2 & B_2 & & \\ & \ddots & \ddots & \ddots & \\ & & C_{\frac{p}{2}-1} & A_{\frac{p}{2}-1} & B_{\frac{p}{2}-1} \\ & & & C_{\frac{p}{2}} & A_{\frac{p}{2}} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_{\frac{p}{2}-1} \\ \boldsymbol{x}_{\frac{p}{2}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ \vdots \\ \boldsymbol{d}_{\frac{p}{2}-1} \\ \boldsymbol{d}_{\frac{p}{2}} \end{bmatrix} \tag{3.25}$$

y una nueva partición en bloques (basada en la anterior) añadiendo $2m$ filas (respectivamente, componentes) a cada uno de los bloques intermedios de la matriz de coeficientes (respectivamente, vector de términos independientes) y m' filas (respectivamente, componentes) a los bloques extremos de A (respectivamente, $\boldsymbol{d}$). Los nuevos bloques están solapados y contienen un número par de filas, de este modo se obtiene un conjunto de subsistemas (los centrales con $2k+2m$ ecuaciones e incógnitas y los extremos con $2k+m'$ ecuaciones e incógnitas)

$$\hat{A}_i \hat{\boldsymbol{x}}_i = \hat{\boldsymbol{d}}_i, \quad i = 1, 2, \ldots, \frac{p}{2}, \tag{3.26}$$

donde

- $\hat{A}_1$ es la submatriz de A de tamaño $(2k+m') \times (2k+m')$ formada por las filas y columnas $1, 2, \ldots, 2k+m'$ y $\hat{\boldsymbol{d}}_1$ es el vector formado por las componentes $1, 2, \ldots, 2k+m'$ de $\boldsymbol{d}$;
- $\hat{A}_i$, para $i = 2, 3, \ldots, \frac{p}{2}-1$, es la submatriz de A de tamaño $(2k+2m) \times (2k+2m)$ formada por las filas y columnas $2(i-1)k-m+1, 2(i-1)k-m+2, \ldots, 2ik+m$ y $\hat{\boldsymbol{d}}_i$ es el vector formado por las componentes $2(i-1)k-m+1, 2(i-1)k-m+2, \ldots, 2ik+m$ de $\boldsymbol{d}$;
- $\hat{A}_{\frac{p}{2}}$ es la submatriz de A de tamaño $(2k+m') \times (2k+m')$ formada por las filas y columnas $n-2k-m'+1, n-2k-m'+2, \ldots, n$ y, finalmente, $\hat{\boldsymbol{d}}_{p-1}$ es el vector formado por las componentes $n-2k-m'+1, n-2k-m'+2, \ldots, n$, de $\boldsymbol{d}$.

Estos subsistemas son resueltos aplicando el algoritmo 3.1 en cada par de procesadores (P_{2i-2}, P_{2i-1}), para $i = 1, 2, \ldots, \frac{p}{2}$.

El método consta de tres fases: en la primera, se comunican datos desde el procesador principal al resto; en la segunda, se aplica el algoritmo 3.1 para resolver el sistema (3.26) en cada par de procesadores (P_{2i-2}, P_{2i-1}), con $i = 1, 2, \ldots, \dfrac{p}{2}$; finalmente, en la tercera fase, cada procesador comunica sus soluciones parciales al procesador principal. A continuación se describen cada una de esas fases.

Fase 1

Como se ha comentado anteriormente, cada par de procesadores recibe los bloques $\hat{A}_i$ y $\hat{\boldsymbol{d}}_i$ de los subsistemas (3.26). En un único superpaso, cada procesador recibe los datos necesarios para ejecutar el algoritmo 3.1 en la siguiente fase, el número total de datos recibidos por el procesador j, para $j = 1, 2, \ldots, p-1$, es $4t(j) - 1$, con

$$t(j) = \begin{cases} k + \frac{m'}{2}, & \text{si } j = 1, p-2, p-1, \\ k + m, & \text{en otro caso.} \end{cases} \tag{3.27}$$

La primera fila de A y $\boldsymbol{d}$ que el procesador j, para $j = 1, 2, \ldots, p-1$, debe recibir del principal es $\phi_1(j)$, con

$$\phi_1(j) = \begin{cases} t(j) + 1, & \text{si } j = 1, \\ 2ik - m + 1, & \text{si } j = 2i, \quad \text{con } i = 1, 2, \ldots, \dfrac{p}{2} - 2, \\ (2i+1)k + 1, & \text{si } j = 2i+1, \text{ con } i = 1, 2, \ldots, \dfrac{p}{2} - 2, \\ n - 2t(j) + 1, & \text{si } j = p-2, \\ n - t(j) + 1, & \text{si } j = p-1; \end{cases} \tag{3.28}$$

nótese que si $p = 4$ la expresión anterior se limita a

$$\phi_1(j) = \begin{cases} t(j) + 1, & \text{si } j = 1, \\ n - 2t(j) + 1, & \text{si } j = p-2, \\ n - t(j) + 1, & \text{si } j = p-1. \end{cases}$$

Ejemplo 3.2 *Para $p = 8$, $n = 40$ y $m = 3$, la partición (3.25) tiene la forma*

$$\begin{bmatrix} A_1 & B_1 & & \\ C_2 & A_2 & B_2 & \\ & C_3 & A_3 & B_3 \\ & & C_4 & A_4 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ \boldsymbol{d}_3 \\ \boldsymbol{d}_4, \end{bmatrix} \tag{3.29}$$

$m' = 4$ *y*

$$t(j) = \begin{cases} 7, & \textit{si } j = 1, 6, 7 \\ 8, & \textit{en otro caso.} \end{cases}$$

Este último valor representa el número total de filas que cada procesador debe recibir de la matriz de coeficientes y del vector de términos independientes.

Al calcular la función ϕ_1, usando la expresión (3.28), se obtiene

$$\begin{array}{lllllllll} \phi_1(1) & = & 8, & \phi_1(2) & = & 8, & \phi_1(3) & = & 16, \\ \phi_1(4) & = & 18, & \phi_1(5) & = & 26, & \phi_1(6) & = & 27, \\ \phi_1(7) & = & 34, & & & & & & \end{array}$$

lo que significa que el procesador P_1 recibe los elementos de A y $\boldsymbol{d}$ situados desde la fila 8 a la 14 excepto b_{14}, P_2 recibe los elementos situados desde la fila 8 a la 15 excepto c_8, P_3 recibe los elementos situados desde la fila 16 a la 23 excepto b_{23}, P_4 recibe los elementos situados desde la fila 18 a la 25 excepto c_{18}, P_5 recibe los elementos situados desde la fila 26 a la 33 excepto b_{33}, P_6 recibe los elementos situados desde la fila 27 a la 33 excepto c_{27} y, finalmente, P_7 recibe los elementos situados desde la fila 34 a la 40. ■

Fase 2

En esta fase cada par de procesadores (P_{2i-2}, P_{2i-1}), para $i = 1, 2, \ldots, \dfrac{p}{2}$, ejecutan el algoritmo 3.1 (excepto la primera y última comunicación) con los elementos que han recibido en la fase 1. Obsérvese que, en este caso, los procesadores pares ejecutan las mismas operaciones y comunicaciones que el procesador P_0 en el algoritmo 3.1, mientras

que los procesadores impares realizan las mismas operaciones y comunicaciones que el procesador P_1 en el algoritmo 3.1.

Fase 3

Tras la ejecución del algoritmo 3.1 en la fase 2, cada procesador j, para $j = 1, 2, \ldots, p-1$, ha obtenido un vector solución parcial de $t(j)$ componentes (la solución parcial de P_0 tiene $k + \frac{m'}{2}$ componentes). El objetivo de esta fase es determinar qué componentes de su solución parcial debe enviar cada procesador al principal a fin de obtener la solución general del sistema (3.1).

Ejemplo 3.3 *Continuando con el ejemplo 3.2, cuando se aplica la fase 2 del algoritmo 3.1 los procesadores P_2 a P_6 calculan un vector solución parcial de 8 componentes mientras que la solución parcial de los procesadores P_0, P_1, P_6 y P_7 tiene 7 componentes. De cada uno de los cuatro bloques de la partición (3.29) se eligen 10 componentes como sigue: de $\boldsymbol{x}_1$ las 10 primeras componentes, de $\boldsymbol{x}_4$ las 10 últimas y para el resto las 10 componentes centrales. Obsérvese que parte de las componentes de $\boldsymbol{x}_i$, para $i = 1, 2, 3, 4$, están en el procesador P_{2i-2} y parte en el procesador P_{2i-1}, se necesita pues determinar qué componentes se eligen de cada procesador.* ■

Todas las componentes del vector solución parcial del primer y último procesador forman parte de la solución general. Se deben tomar las k últimas componentes en los procesadores pares, excepto en el último procesador par en el que se eligen las $k - \frac{m'}{2}$ últimas componentes, y las k primeras componentes en los procesadores impares, excepto en el primer procesador impar en el que se toman las primeras $k - \frac{m'}{2}$ componentes. En la figura 3.3 se muestra, para $p = 8$, la formación de la solución del sistema (3.1) a partir de las soluciones parciales en cada procesador.

3.3.2 Algoritmos BSP para un número par de procesadores

El siguiente algoritmo BSP implementa el nuevo método paralelo para resolver sistemas tridiagonales, basado en las tres fases que se han descrito anteriormente.

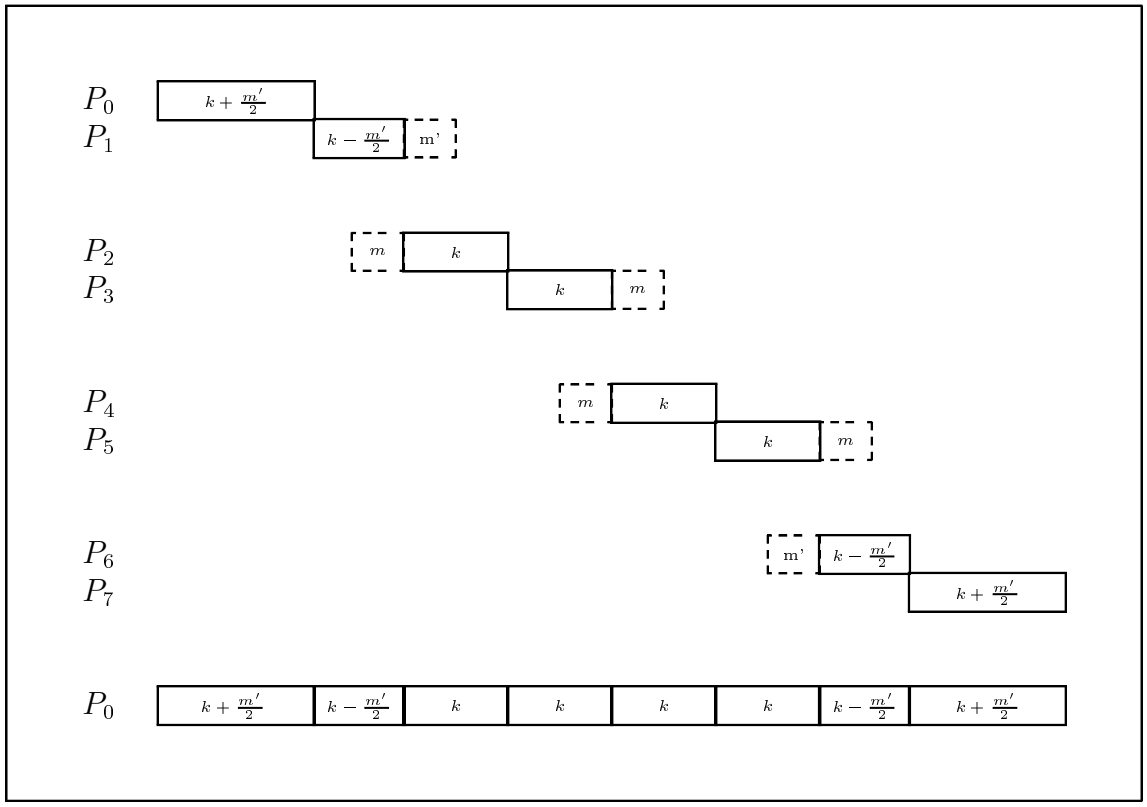


Figura 3.3: *Obtención de la solución general a partir de las soluciones parciales para* $p = 8$.

Algoritmo 3.2 Algoritmo BSP para resolver sistemas tridiagonales para p procesadores.

Superpaso 1

Inicio.

- El procesador principal calcula $k = \dfrac{n}{p}$, m de acuerdo con (3.23) y m' mediante (3.24).
- Cada procesador recibe desde el procesador principal los correspondientes elementos de A y $\boldsymbol{d}$ de acuerdo con la función ϕ_1 definida en (3.28), esto es
  - P_0 envía a P_1 la fila j-ésima de $\hat{A}_1$ y $\hat{\boldsymbol{d}}_1$, para $j = k+\dfrac{m'}{2}+1, k+\dfrac{m'}{2}+2, \ldots, 2k+m'$.
  - Para $i = 1, 2, \ldots, \dfrac{p}{2} - 2$.

- $\diamond$ P_0 envía a P_{2i} la fila j-ésima de $\hat{A}_i$ y $\hat{\boldsymbol{d}}_i$, para $j = 1, 2, \ldots, k+m$.
- $\diamond$ P_0 envía a P_{2i+1} la fila j-ésima de $\hat{A}_i$ y $\hat{\boldsymbol{d}}_i$, para $j = k+m+1, k+m+2, \ldots, 2k+2m$.

- $\circ$ P_0 envía a P_{p-2} y P_{p-1} la fila j-ésima de $\hat{A}_{\frac{p}{2}}$ y $\hat{\boldsymbol{d}}_{\frac{p}{2}}$, para $j = 1, 2, \ldots, k + \dfrac{m'}{2}$ y $j = k + \dfrac{m'}{2} + 1, k + \dfrac{m'}{2} + 2, \ldots, 2k + m'$, respectivamente.

- El procesador principal envía m al resto de procesadores.

Superpaso 2

- Para $i = 1, p-2, p-1$, el procesador P_i calcula m'.
- Para $i = 1, 2, \ldots, \dfrac{p}{2}$, los procesadores (P_{2i-2}, P_{2i-1}) resuelven el subsistema $\hat{A}_i \hat{\boldsymbol{x}}_i = \hat{d}_i$ ejecutando los superpasos 2 y 3 del algoritmo 3.1, excepto la comunicación final de soluciones del final del superpaso 3.
- Comunicación de soluciones parciales.
  - $\circ$ P_1 comunica a P_0 las componentes $k + \dfrac{m'}{2} + 1, k + \dfrac{m'}{2} + 2, \ldots, 2k$ de $\hat{\boldsymbol{x}}_1$.
  - $\circ$ Para $i = 1, 2, \ldots, \dfrac{p}{2} - 2$
    - $\diamond$ P_{2i} comunica a P_0 las componentes $m+1, m+2, \ldots, m+k$ de $\hat{\boldsymbol{x}}_{i+1}$.
    - $\diamond$ P_{2i+1} comunica a P_0 las componentes $m+k+1, m+k+2, \ldots, m+2k$ de $\hat{\boldsymbol{x}}_{i+1}$.
  - $\circ$ P_{p-2} comunica a P_0 las componentes $m'+1, m'+2, \ldots, k + \dfrac{m'}{2}$ de $\hat{\boldsymbol{x}}_{\frac{p}{2}}$.
  - $\circ$ P_{p-1} comunica a P_0 las componentes $k + \dfrac{m'}{2} + 1, k + \dfrac{m'}{2} + 2, \ldots, 2k + m'$ de $\hat{\boldsymbol{x}}_{\frac{p}{2}}$.
- El procesador principal forma la solución $\boldsymbol{x}$ del sistema (3.1) haciendo:
  - $\circ$ Para $j = 1, 2, \ldots, 2k$, las componentes j-ésima y $(n-2k+j)$-ésima de $\boldsymbol{x}$ igual a las componentes j-ésima y $(m'+j)$-ésima de $\hat{\boldsymbol{x}}_1$ y $\hat{\boldsymbol{x}}_{\frac{p}{2}}$, respectivamente.

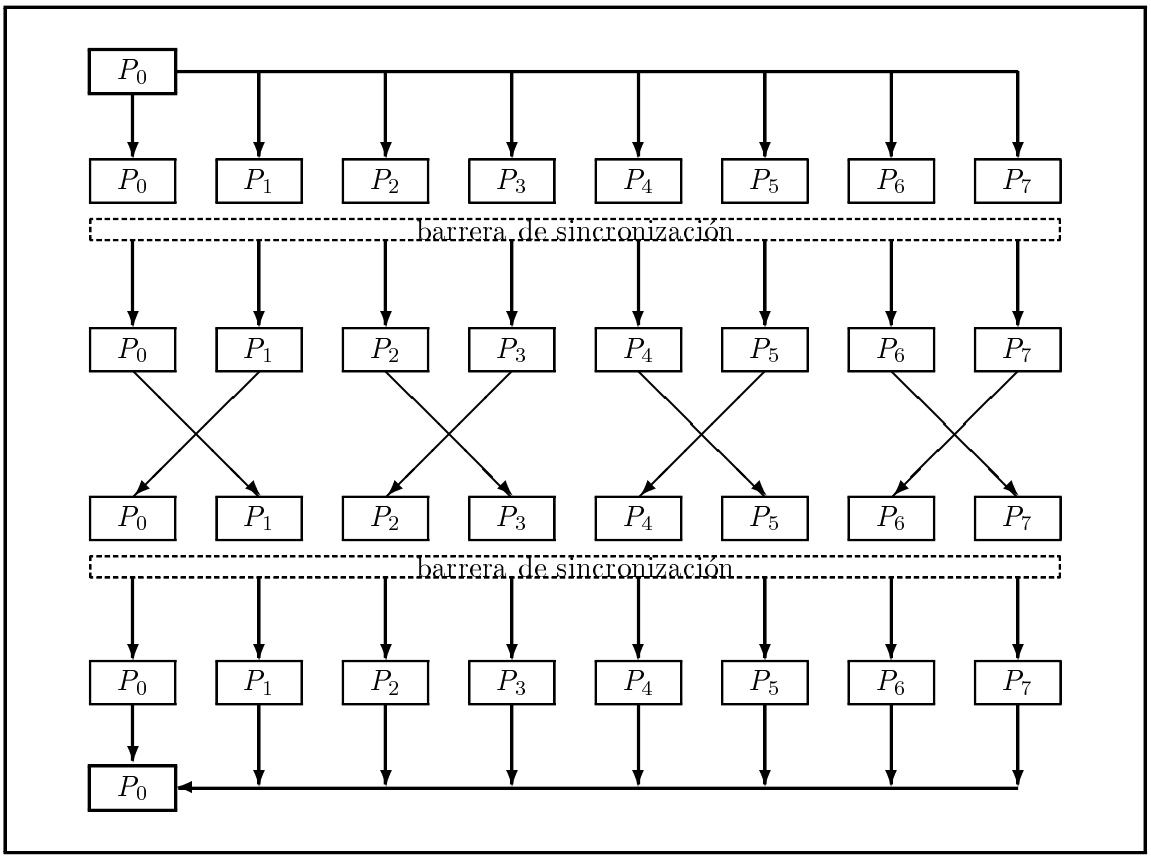


Figura 3.4: *Esquema de comunicación para* 8 *procesadores del método descrito en el algoritmo 3.2.*

- Para $i = 1, 2, \ldots, \dfrac{p}{2} - 2$ y $j = 1, 2, \ldots, 2k$, la componente $(2ik + j)$-ésima de $\boldsymbol{x}$ igual a la componente $(m + j)$-ésima de $\hat{\boldsymbol{x}}_{i+1}$.

Obsérvese que el algoritmo 3.2 contiene tres barreras de sincronización ya que en el superpaso 2 se ejecutan los superpasos 2 y 3 del algoritmo 3.1. En la figura 3.4 se muestra un ejemplo de ejecución del algoritmo para $p = 8$; obsérvese el especial patrón de comunicaciones necesario: cada par de procesadores comunican entre sí en el superpaso central.

A continuación se obtiene el coste computacional del algoritmo 3.2.

Coste del superpaso 1. Para calcular m, a partir de los parámetros δ y μ, se necesitan $3n+12$ operaciones (véase la expresión (2.19) en la página 77). El cálculo de k requiere una operación y el de m' dos. En consecuencia, el coste aritmético del superpaso es $3n+15$.

Para obtener el coste de comunicación de este superpaso, debe tenerse en cuenta que el procesador principal envía $4(k+m)-1$ elementos al procesador P_i, para $i=2,3,\ldots,p-3$, y envía $4\left(k+\frac{m'}{2}\right)-1$ elementos al procesador P_j, para $j=1,p-2,p-3$, además envía m a todos los procesadores. Como en el peor de los casos $m'=m+1$, en total envía $(p-4)(4k+4m-1)+3(4k+2m+1)+p-1$ elementos, luego el coste de comunicación es $(4n-4k-10m+4mp+6)g$.

El coste computacional del superpaso es, por tanto,

$$3n+15+(4n-4k-10m+4mp+6)g+l. \tag{3.30}$$

Coste del superpaso 2. Los procesadores que más operaciones realizan son P_i con $i=2,3,\ldots,p-3$. Cada pareja de procesadores resuelve un sistema de tamaño $2(k+m)\times 2(k+m)$, en consecuencia el coste aritmético del superpaso 2 del algoritmo 3.1 es $3[2(k+m)]-4$ y el del superpaso 3 es $2(k+m)+9$ (véase la expresión (3.22)). El coste aritmético de este superpaso es, por tanto, $8k+8m+5$.

En este superpaso se pueden considerar dos etapas de comunicación. La primera contiene la comunicación del superpaso 2 del algoritmo 3.1, cada pareja de procesadores comunican entre sí 2 elementos. La segunda está relacionada con la obtención del vector solución final en el procesador principal, este recibe en total $n-\left(k+\frac{m'}{2}\right)$ elementos del resto de procesadores. El coste total de comunicación de este superpaso es $\left(n-k-\frac{m}{2}+\frac{3}{2}\right)g$, considerando que en el peor de los casos $m'=m+1$.

El coste global de este superpaso es

$$8k+8m+5+\left(n-k-\frac{m}{2}+\frac{3}{2}\right)g+2l. \tag{3.31}$$

Sumando las expresiones (3.30) y (3.31) se obtiene el coste computacional del método descrito por el algoritmo 3.2

$$3n+8k+8m+20+\left(5n-5k-\frac{21}{2}m+4mp+\frac{15}{2}\right)g+3l.$$

Ejemplo 3.4 *Sea el sistema* $A\boldsymbol{x} = \boldsymbol{d}$*, de* 12 *ecuaciones lineales, donde*

$$A = \begin{bmatrix} 66 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 66 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 66 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 66 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 66 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 66 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 66 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 66 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 66 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 66 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 66 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 66 \end{bmatrix} \quad y \quad \boldsymbol{d} = \begin{bmatrix} 67 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 65 \end{bmatrix}.$$

Puesto que cada término independiente se ha obtenido como suma de los coeficientes de su ecuación, la solución del sistema tiene sus doce componentes iguales a 1.

A continuación se va a resolver el sistema aplicando el algoritmo 3.2 para 6 *procesadores.*

La diagonal dominanza de A *vale*

$$\delta = \min\left\{\frac{67}{1}, \frac{66}{1+1}, \cdots, \frac{66}{1+1}, \frac{65}{1}\right\} = 33.$$

Una cota superior de la norma infinito de la solución del sistema viene dada por

$$\mu = \frac{\max\left\{\frac{67}{66}, \frac{66}{66}, \cdots, \frac{66}{66}, \frac{65}{66}\right\}}{1 - 33^{-1}} = \frac{67}{64} = 1.0469.$$

Si se toma un error $\epsilon = 10^{-3}$*, se obtiene para* m *el siguiente valor*

$$m = \left\lceil \frac{1}{\log 33^{-1}} \log \frac{10^{-3}(1 - 33^{-2})}{\frac{67}{64}} \right\rceil = \lceil 1.989 \rceil = 2,$$

en consecuencia $m' = 2$.

De las expresiones (3.27) y (3.28) se obtiene

$$\begin{aligned} t(1) &= 3, & \phi_1(1) &= 4, \\ t(2) &= 4, & \phi_1(2) &= 3, \\ t(3) &= 4, & \phi_1(3) &= 7, \\ t(4) &= 3, & \phi_1(4) &= 7, \\ t(5) &= 3, & \phi_1(5) &= 10, \end{aligned}$$

esto significa que el procesador P_1 *recibe de* P_0 *los elementos de* A *y* $\boldsymbol{d}$ *situados desde la fila* 4 *hasta la fila* 6 *excepto el elemento* b_6, P_2 *desde la fila* 3 *hasta la fila* 6 *excepto el elemento* c_3, P_3 *desde la fila* 7 *hasta la fila* 10 *excepto el elemento* b_{10}, P_4 *desde la fila* 7 *hasta la fila* 9 *excepto el elemento* c_7 *y* P_5 *desde la fila* 10 *hasta la fila* 12 *excepto el elemento* b_{12}.

Los procesadores P_0 *y* P_1 *aplican los superpasos 2 y 3 del algoritmo 3.1 al subsistema*

$$\begin{bmatrix} 66 & 1 & 0 & 0 & 0 & 0 \\ -1 & 66 & 1 & 0 & 0 & 0 \\ 0 & -1 & 66 & 1 & 0 & 0 \\ 0 & 0 & -1 & 66 & 1 & 0 \\ 0 & 0 & 0 & -1 & 66 & 1 \\ 0 & 0 & 0 & 0 & -1 & 66 \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \hat{x}_6 \end{bmatrix} = \begin{bmatrix} 67 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \end{bmatrix},$$

obteniendo como solución del mismo

$$\hat{\boldsymbol{x}}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0.9998 \\ 1.0151 \end{bmatrix}.$$

Los procesadores P_2 y P_3 aplican los superpasos 2 y 3 del algoritmo 3.1 al subsistema

$$\begin{bmatrix} 66 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 66 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 66 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 66 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 66 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 66 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 66 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 66 \end{bmatrix} \begin{bmatrix} \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \hat{x}_6 \\ \hat{x}_7 \\ \hat{x}_8 \\ \hat{x}_9 \\ \hat{x}_{10} \end{bmatrix} = \begin{bmatrix} 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 66 \end{bmatrix},$$

obteniendo como solución del mismo

$$\hat{\boldsymbol{x}}_2 = \begin{bmatrix} 0.9849 \\ 0.9998 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0.9998 \\ 1.0151 \end{bmatrix}.$$

Los procesadores P_4 y P_5 aplican los superpasos 2 y 3 del algoritmo 3.1 al subsistema

$$\begin{bmatrix} 66 & 1 & 0 & 0 & 0 & 0 \\ -1 & 66 & 1 & 0 & 0 & 0 \\ 0 & -1 & 66 & 1 & 0 & 0 \\ 0 & 0 & -1 & 66 & 1 & 0 \\ 0 & 0 & 0 & -1 & 66 & 1 \\ 0 & 0 & 0 & 0 & -1 & 66 \end{bmatrix} \begin{bmatrix} \hat{x}_7 \\ \hat{x}_8 \\ \hat{x}_9 \\ \hat{x}_{10} \\ \hat{x}_{11} \\ \hat{x}_{12} \end{bmatrix} = \begin{bmatrix} 66 \\ 66 \\ 66 \\ 66 \\ 66 \\ 65 \end{bmatrix},$$

obteniendo como solución del mismo

$$\hat{\boldsymbol{x}}_3 = \begin{bmatrix} 0.9849 \\ 0.9998 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

En la siguiente tabla se muestra la solución parcial de cada procesador.

Procesador	*Solución parcial*			
P_0	1	1	1	—
P_1	1	0.9998	1.0151	—
P_2	0.9849	0.9998	1	1
P_3	1	1	0.9998	1.0151
P_4	0.9849	0.9998	1	—
P_5	1	1	1	—

Finalmente, el procesador principal obtiene la solución general a partir de las soluciones parciales, tomando las tres componentes del vector solución parcial en P_0, la primera en P_1, la tercera y cuarta componentes de la solución parcial en P_2, las dos primeras en P_3, la tercera en P_4 y las tres componentes de la solución parcial en P_5. ■

En el capítulo 2 se definieron, para $i = 0, 1, \ldots, p-1$,

$$\delta_i = \min_{ik+1 \le j \le (i+1)k} \left\{ \frac{|a_j|}{|c_j| + |b_j|} \right\}, \tag{3.32}$$

$$\mu_i = \max_{ik+1 \le j \le (i+1)k} \left\{ \frac{|d_j|}{|a_j|} \right\}, \tag{3.33}$$

entonces

$$\delta = \min_{0\le i\le p-1}\{\delta_i\}, \tag{3.34}$$

$$\mu = \frac{\max\limits_{0\le i\le p-1}\{\mu_i\}}{1-\delta^{-1}}. \tag{3.35}$$

Al igual que se hizo en el capítulo 2, el cálculo de los parámetros δ y μ se puede realizar en paralelo. En este caso, el procesador principal envía a cada procesador P_i, para $i = 1, 2, \ldots, p-1$, las filas $(ik+j)$-ésismas, con $j = 1, 2, \ldots, k$, del sistema de manera que éste pueda calcular los parámetros δ_i y μ_i de acuerdo con las expresiones (3.32) y (3.33).

Una vez calculado por P_0 el valor de m, debe comunicar a los otros procesadores el resto de elementos del sistema (3.1) que se necesitan para que cada pareja de procesadores pueda resolver el subsistema de (3.26) que le corresponda. Todos los procesadores deben recibir m filas excepto P_1 y P_{p-2} que deben recibir m' filas y P_{p-1} que debe recibir $\dfrac{m'}{2}$ filas. La primera fila de A y $\boldsymbol{d}$ que el procesador P_j, para $j = 1, 2, \ldots, p-1$, debe recibir del principal es $\phi_2(j)$, con

$$\phi_2(j) = \begin{cases} 2k+1, & \text{si } j = 1, \\ 2ik - m + 1, & \text{si } j = 2i \quad \text{con } i = 1, 2, \ldots, \frac{p}{2}-2, \\ 2(i+1)k+1, & \text{si } j = 2i+1 \text{ con } i = 1, 2, \ldots, \frac{p}{2}-2, \\ n - 2k - m' + 1, & \text{si } j = p-2, \\ n - k - \dfrac{m'}{2} + 1, & \text{si } j = p-1. \end{cases} \tag{3.36}$$

El siguiente algoritmo BSP modifica el algoritmo 3.2 para realizar el cálculo de δ y μ en paralelo.

Algoritmo 3.3 Algoritmo BSP para p procesadores con cálculo de δ y μ en paralelo

Superpaso 1

- El procesador principal P_0 calcula $k = \dfrac{n}{p}$.

- Para $i = 1, 2, \ldots, p-1$, el procesador P_0 envía a P_i los elementos c_{ik+j}, a_{ik+j}, b_{ik+j} y d_{ik+j}, con $j = 1, 2, \ldots, k$ excepto $b_n = 0$.

Superpaso 2

- Para $i = 0, 1, \ldots, p-1$, P_i calcula δ_i y μ_i de acuerdo con las expresiones (3.32) y (3.33).
- Para $i = 1, 2, \ldots, p-1$, P_i envía a P_0 los valores de los parámetros δ_i y μ_i.

Superpaso 3

- El procesador P_0 calcula los parámetros δ, μ, m y m' haciendo uso de las expresiones (3.34), (3.35), (3.23) y (3.24).
- Para $i = 1, 2, \ldots, p-1$, el procesador principal P_0 envía a P_i los correspondientes elementos de A y d, de acuerdo con la función ϕ_2 definida en (3.36), esto es
  - P_0 envía a P_1 los elementos c_j, a_j, b_j y d_j, con $j = 2k+1, 2k+2, \ldots, 2k+m'$, excepto $b_{2k+m'}$.
  - Para $i = 1, 2, \ldots, \dfrac{p}{2} - 2$
    - P_0 envía a P_{2i} los elementos c_j, a_j, b_j y d_j, con $j = 2ik - m + 1$, $2ik - m + 2, \ldots, 2ik$, excepto $c_{2ik-m+1}$.
    - P_0 envía a P_{2i+1} los elementos c_j, a_j, b_j y d_j, con $j = 2(i+1)k + 1$, $2(i+1)k + 2, \ldots, 2(i+1)k + m$, excepto $b_{2(i+1)k+m}$.
  - P_0 envía a P_{p-2} los elementos c_j, a_j, b_j y d_j, con $j = n - 2k - m' + 1$, $n - 2k - m' + 2, \ldots, n - 2k$, excepto $c_{n-2k-m'+1}$.
  - P_0 envía a P_{p-1} los elementos c_j, a_j, b_j y d_j, con $j = n - k - \dfrac{m'}{2} + 1$, $n - k - \dfrac{m'}{2} + 2, \ldots, n - k$.
- El procesador principal envía m al resto de procesadores.

Superpaso 4

Como el superpaso 2 del algoritmo 3.2.

Al repartir el cálculo de δ y μ entre p procesadores, el número de operaciones baja de $3n+3$ a $3k+3$, luego el coste aritmético pasa de $3n+8k+8m+20$ a $11k+8m+20$; se han introducido dos superpasos más, por lo que el coste de sincronización aumenta de $3l$ a $5l$. La comunicación se ha incrementado en $2(p-1)$ elementos (que corresponden al envío desde los procesadores remotos a P_0 de los parámetros δ_i y μ_i) por un lado y $4m'$ elementos (correspondientes a las $\dfrac{m'}{2}$ filas enviadas a P_1 que no serán necesarias para resolver el subsistema $\hat{A}_1\hat{x}_1 = \hat{d}_1$ y las $\dfrac{m'}{2}$ filas enviadas a P_{p-2} no necesarias para la resolución del subsistema $\hat{A}_{\frac{p}{2}}\hat{x}_{\frac{p}{2}} = \hat{d}_{\frac{p}{2}}$) por otro; en consecuencia, el coste de comunicación vale

$$\begin{aligned}&\left[\left(5n-5k-\frac{21}{2}m+4mp+\frac{15}{2}\right)+(4m+4+2p-2)\right]g\\ =\ &\left(5n-5k-\frac{13}{2}m+4mp+2p+\frac{19}{2}\right)g,\end{aligned}$$

donde se ha considerado que en el peor de los casos $m' = m+1$.

A continuación se obtiene con detalle el coste del algoritmo 3.3.

Coste del superpaso 1. Como se ha visto en el capítulo 2 (véase la expresión (2.25) de la página 80), el coste de este superpaso es

$$1+(4n-4k-1)g+l. \tag{3.37}$$

Coste del superpaso 2. En el capítulo 2 se obtuvo el coste de este superpaso (véase (2.26), página 80), que es

$$3k+(2p-2)g+l. \tag{3.38}$$

Coste del superpaso 3. El procesador principal debe realizar tres operaciones para calcular μ, nueve operaciones para obtener el valor de m y dos operaciones para calcular

m'; en total 14 operaciones. En este superpaso, envía el valor de m a todos los procesadores, envía $4m'-1$ elementos a cada uno de los procesadores P_1 y P_{p-2}, envía $2m'$ elementos al procesador P_{p-1} y envía $4m-1$ elementos al procesador P_i, con $i = 2, 3, \ldots, p-3$, por tanto el coste de comunicación es $[p-1+2(4m'-1)+2m'+(4m-1)(p-4)]\,g$. Como en el peor de los casos $m' = m+1$, se tiene que el coste del superpaso 3 es

$$14 + (-6m + 4mp + 11)g + l. \tag{3.39}$$

Coste del superpaso 4. Puesto que este superpaso coincide con el superpaso 2 del algoritmo 3.2, su coste viene dado por la expresión (3.31).

Sumando las expresiones (3.37), (3.38),(3.39) y (3.31) se obtiene el coste del algoritmo 3.3

$$11k + 8m + 20 + \left(5n - 5k - \frac{13}{2}m + 4mp + 2p + \frac{19}{2}\right) g + 5l. \tag{3.40}$$

3.4 Resultados numéricos

En esta sección se analizan los tiempos previstos teóricamente y los tiempos experimentales de los algoritmos 3.1, 3.2 y 3.3 (a los que se referenciará en las tablas y figuras como TW, TW1 y TW2 respectivamente). Las pruebas experimentales se han realizado en el IBM SP2 y el *cluster* de PC's cuyas características se han descrito en la subsección 1.5.3. Por comodidad, en la tabla 3.1 se repiten los parámetros obtenidos para esas máquinas que se muestran en la tabla 1.1.

El tiempo teórico se ha obtenido considerando que el tamaño de bloque de los mensajes que se comunican entre los procesadores es $k = \dfrac{n}{p}$ y que el coste de comunicación de una palabra de 32 bits es

$$g(k) = \left(\frac{n_{\frac{1}{2}}}{k} + 1\right) g_\infty,$$

además se ha tomado $m = 5$. Los algoritmos 3.1 (TW), 3.2(TW1) y 3.3(TW2) han sido implementados en Fortran usando la versión v1.3 de la librería BSPLib, se ha generando el sistema (3.1) obteniendo aleatoriamente los elementos de la matriz de coeficientes A

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	26
	2	3294	9.5	25
	4	5366	12.4	25
	6	8164	12.5	25

(a) *IBM SP2 switch*

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	8
	2	20235	709.7	3
	4	54163	1362.6	9
	6	121958	3211.2	9

(b) *IBM SP2 ethernet*

s	p	l	g	$n_{\frac{1}{2}}$
16.4	1	23	0.2	22
	2	2556	6.9	5
	4	5152	7.4	4
	6	7538	6.8	4

(c) *Cluster de PC's*

Tabla 3.1: *Valores de parámetros BSP.*

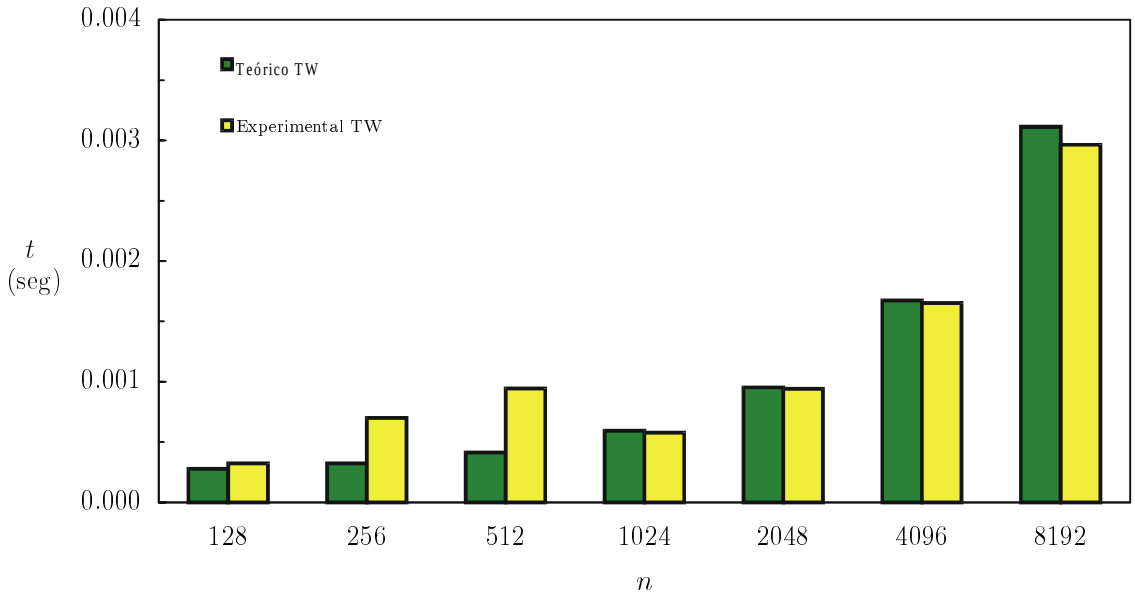
y, para simplificar, eligiendo el vector de términos independientes $\boldsymbol{d}$ de manera que la solución sea $\boldsymbol{x} = [1, 1, \ldots, 1]^T$. A la elección aleatoria de los coeficientes se le ha añadido la restricción de que $m = 5$ (véase la tabla 2.1 en la página 70).

En la tabla 3.2 y la figura 3.5 se muestran los tiempos teóricos y experimentales, medidos en segundos, del algoritmo 3.1 (TW) en el IBM SP2 con 2 procesadores utilizando *switch* y en las tablas 3.3, 3.4 y figuras 3.6, 3.7 se muestran los tiempos teóricos y experimentales, medidos en segundos, de las algoritmos 3.2(TW1) y 3.3(TW2) en el IBM SP2 para 4 y 6 procesadores utilizando *switch*.

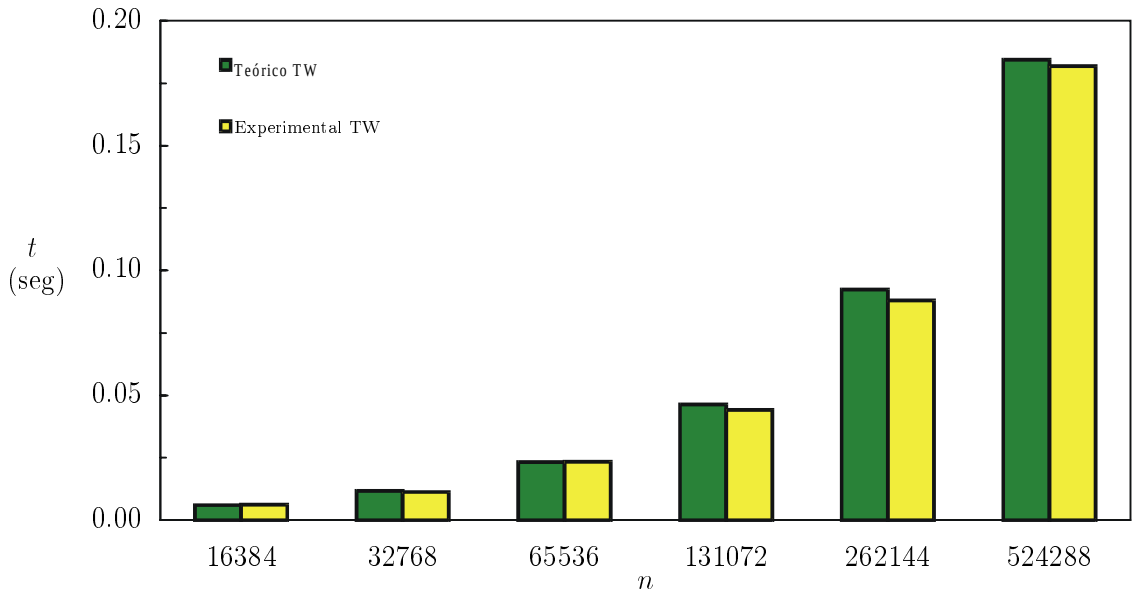
En la tabla 3.5 y la figura 3.8 se muestran los tiempos teóricos y experimentales, medidos en segundos, del algoritmo 3.1 (TW) en el IBM SP2 con 2 procesadores utilizando *ethernet* y en las tablas 3.6, 3.7 y figuras 3.9, 3.10 se muestran los tiempos teóricos y experimentales, medidos en segundos, de las algoritmos 3.2(TW1) y 3.3(TW2) en el IBM SP2 para 4 y 6 procesadores utilizando *ethernet*.

IBM SP2 2 procesadores *switch*		
	TW	
n	Teórico	Experimental
128	0.0003	0.0003
256	0.0003	0.0007
512	0.0004	0.0009
1024	0.0006	0.0006
2048	0.0010	0.0009
4096	0.0017	0.0017
8192	0.0031	0.0030
16384	0.0060	0.0062
32768	0.0117	0.0113
65536	0.0233	0.0234
131072	0.0463	0.0441
262144	0.0923	0.0880
524288	0.1845	0.1818

Tabla 3.2: *Tiempos teóricos y experimentales del algoritmo 3.1 (TW), medidos en un IBM SP2 con 2 procesadores interconectados mediante switch, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

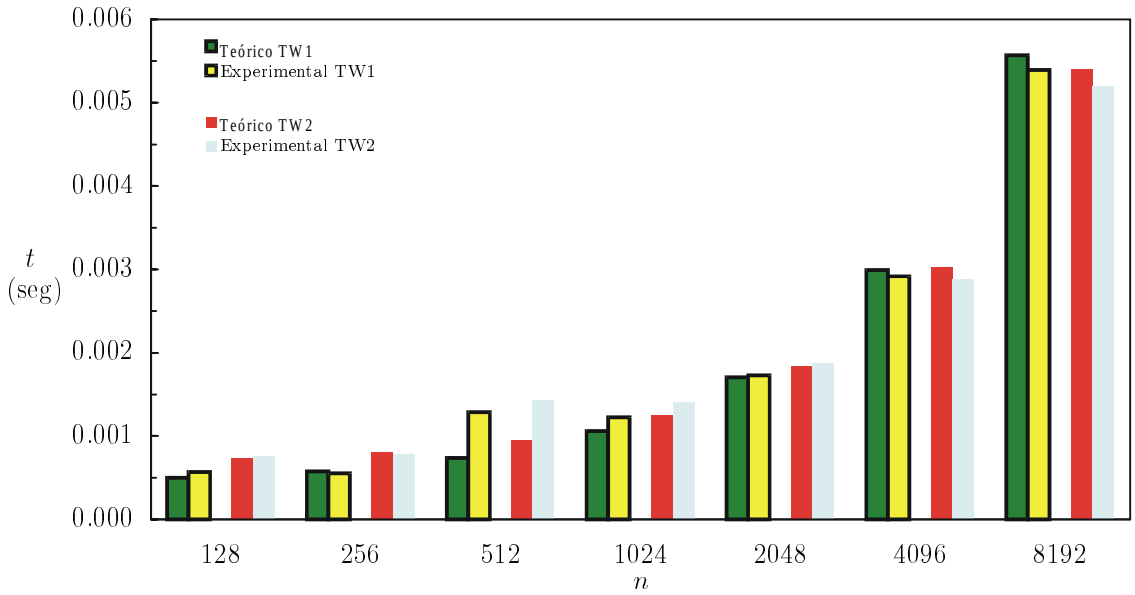


(b) $16384 \leq n \leq 524288$

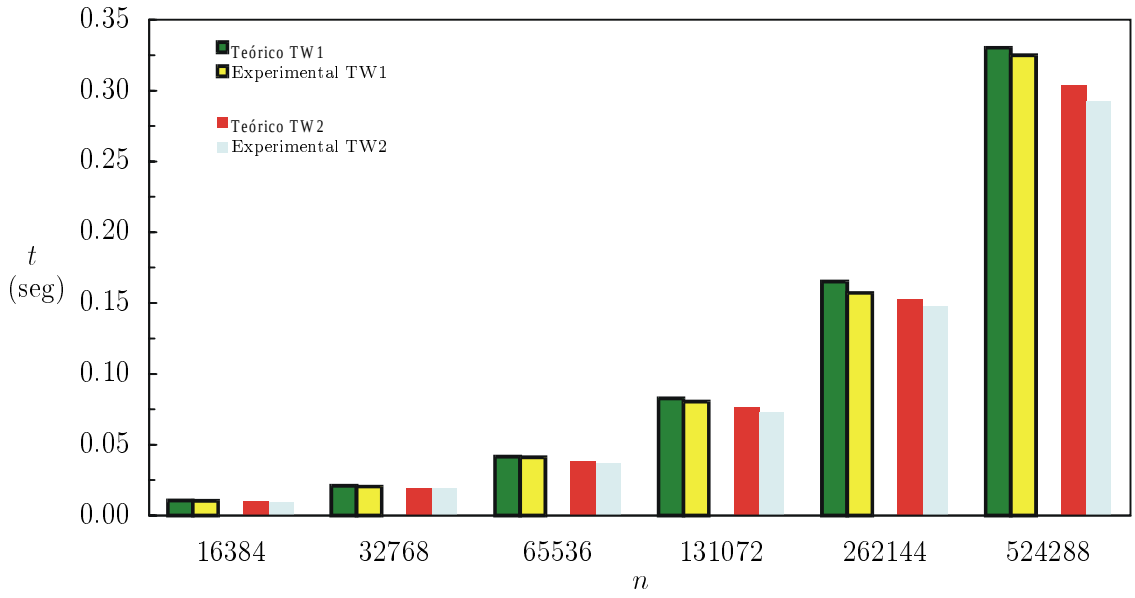
Figura 3.5: *Tiempos teóricos y experimentales del algoritmo 3.1 (TW) en un IBM SP2 con 2 procesadores interconectados mediante switch.*

IBM SP2 4 procesadores *switch*				
	TW1		TW2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0005	0.0006	0.0007	0.0008
256	0.0006	0.0006	0.0008	0.0008
512	0.0007	0.0013	0.0010	0.0014
1024	0.0011	0.0012	0.0013	0.0014
2048	0.0017	0.0017	0.0018	0.0019
4096	0.0030	0.0029	0.0030	0.0029
8192	0.0056	0.0054	0.0054	0.0052
16384	0.0107	0.0105	0.0101	0.0098
32768	0.0210	0.0206	0.0196	0.0193
65536	0.0416	0.0413	0.0386	0.0372
131072	0.0829	0.0805	0.0766	0.0735
262144	0.1653	0.1572	0.1525	0.1478
524288	0.3302	0.3249	0.3043	0.2928

Tabla 3.3: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2), medidos en un IBM SP2 con 4 procesadores interconectados mediante switch, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

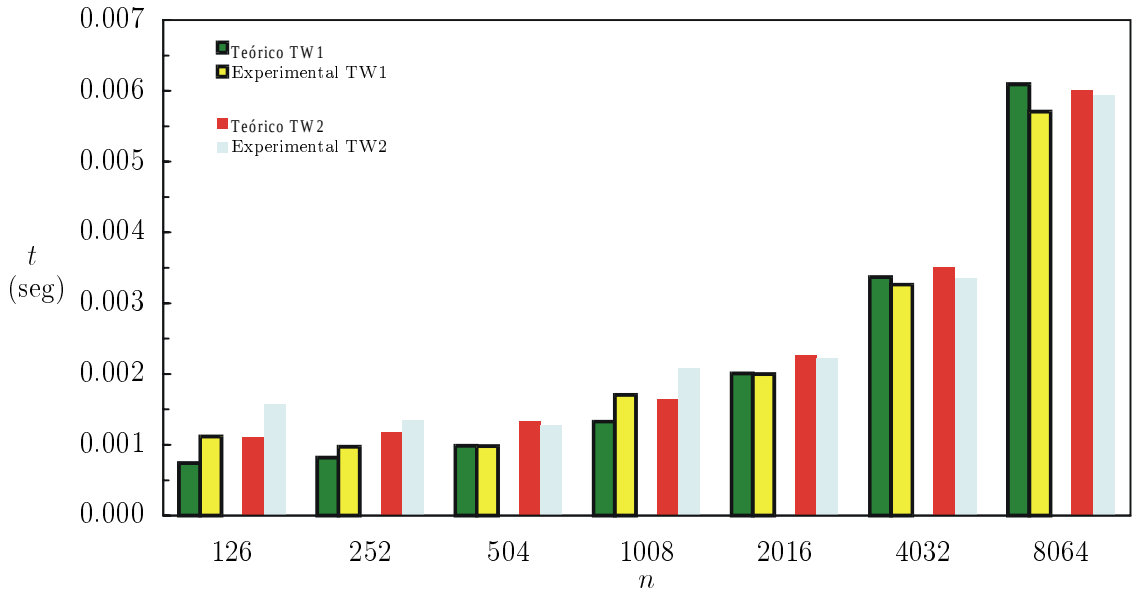


(b) $16384 \leq n \leq 524288$

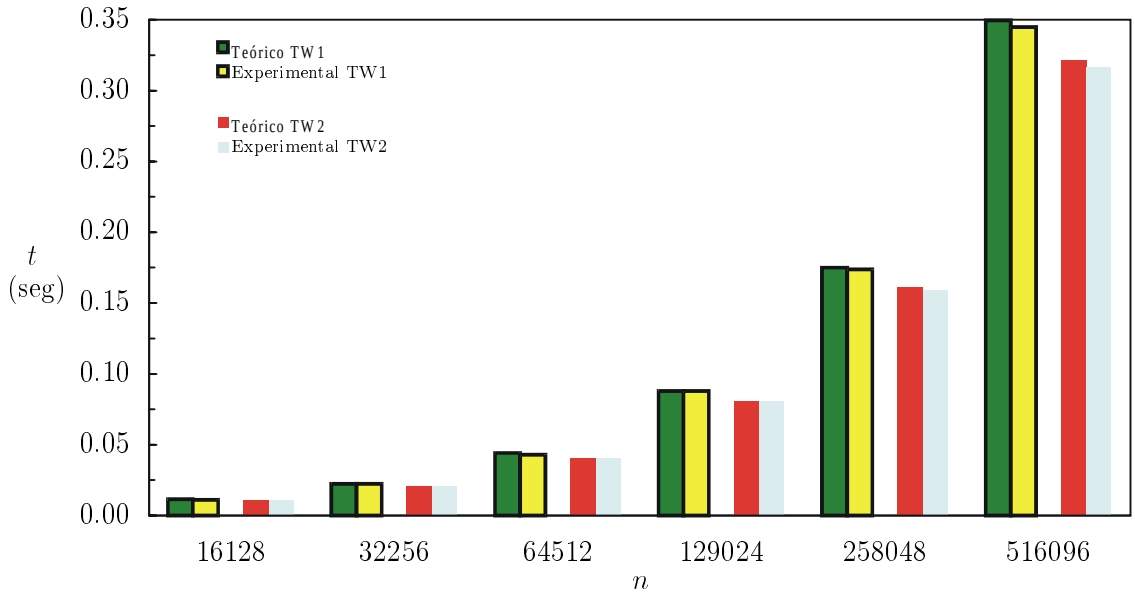
Figura 3.6: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con 4 procesadores interconectados mediante switch.*

IBM SP2 6 procesadores *switch*				
	TW1		TW2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0007	0.0011	0.0011	0.0016
252	0.0008	0.0010	0.0012	0.0013
504	0.0010	0.0010	0.0013	0.0013
1008	0.0013	0.0017	0.0016	0.0021
2016	0.0020	0.0020	0.0023	0.0022
4032	0.0034	0.0033	0.0035	0.0034
8064	0.0061	0.0057	0.0060	0.0059
16128	0.0115	0.0112	0.0110	0.0107
32256	0.0225	0.0224	0.0210	0.0207
64512	0.0443	0.0429	0.0410	0.0409
129024	0.0879	0.0878	0.0811	0.0810
258048	0.1751	0.1739	0.1612	0.1594
516096	0.3496	0.3450	0.3213	0.3168

Tabla 3.4: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2), medidos en un IBM SP2 con* 6 *procesadores interconectados mediante switch, para* $126 \leq n \leq 516096$.



(a) $126 \leq n \leq 8064$

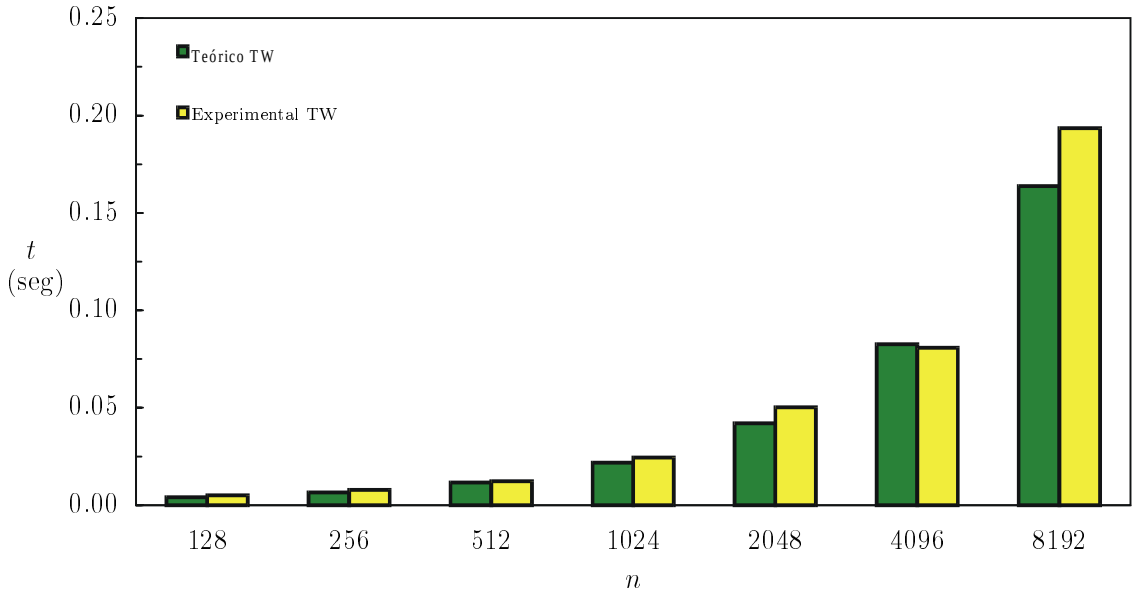


(b) $16128 \leq n \leq 516096$

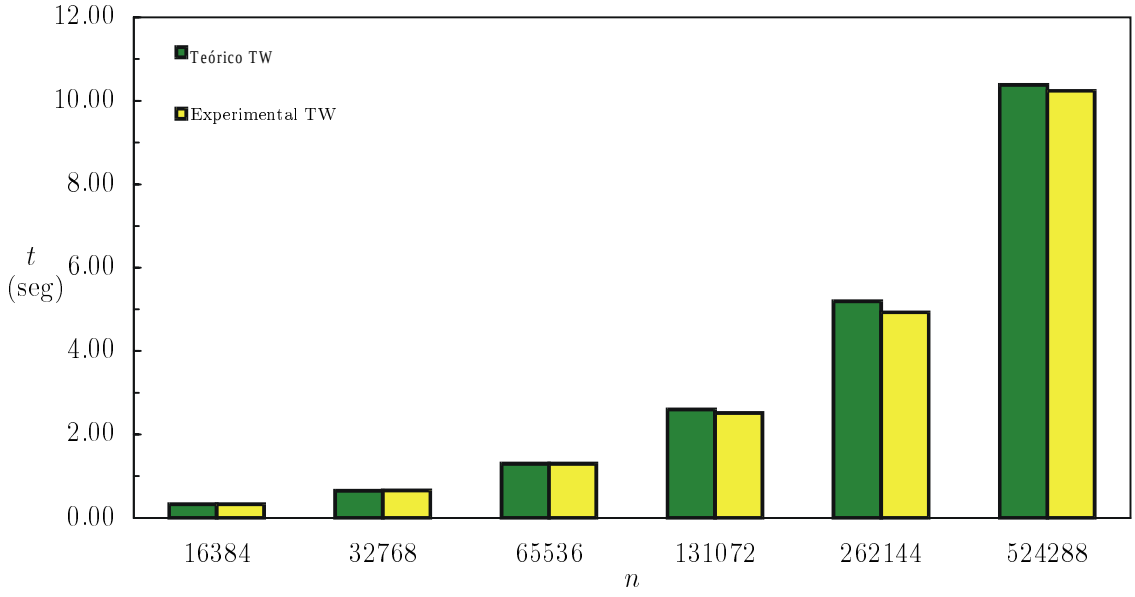
Figura 3.7: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con 6 procesadores interconectados mediante switch.*

IBM SP2 2 procesadores *ethernet*		
	TW	
n	Teórico	Experimental
128	0.0040	0.0051
256	0.0065	0.0078
512	0.0116	0.0122
1024	0.0217	0.0245
2048	0.0420	0.0503
4096	0.0826	0.0807
8192	0.1637	0.1935
16384	0.3259	0.3267
32768	0.6503	0.6570
65536	1.2992	1.2962
131072	2.5969	2.5170
262144	5.1923	4.9264
524288	10.3831	10.2384

Tabla 3.5: *Tiempos teóricos y experimentales del algoritmo 3.1 (TW), medidos en un IBM SP2 con 2 procesadores interconectados mediante ethernet, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

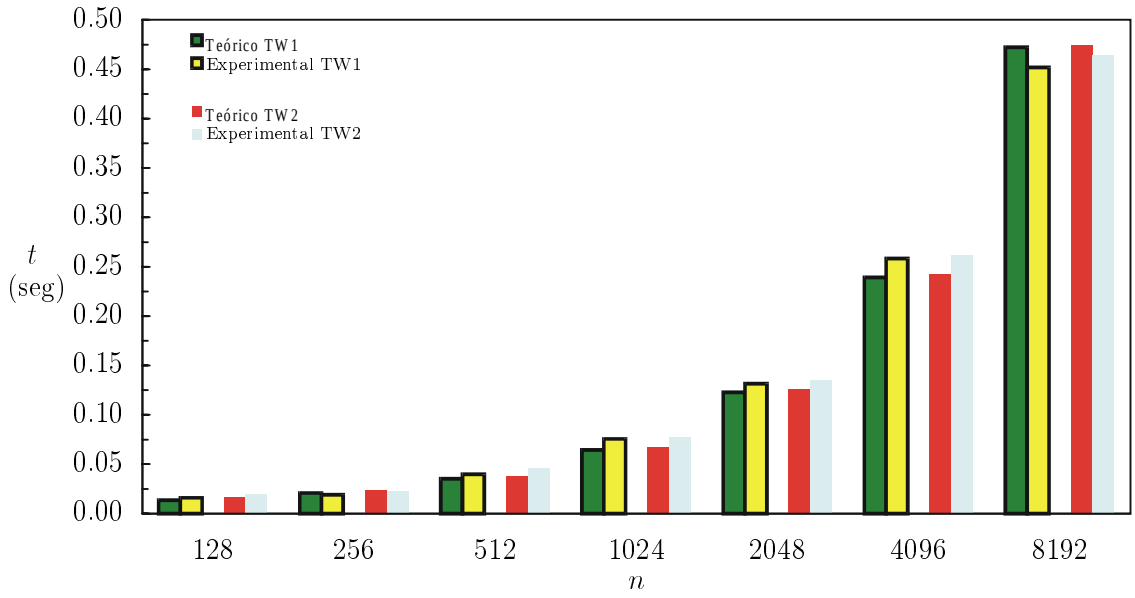


(b) $16384 \leq n \leq 524288$

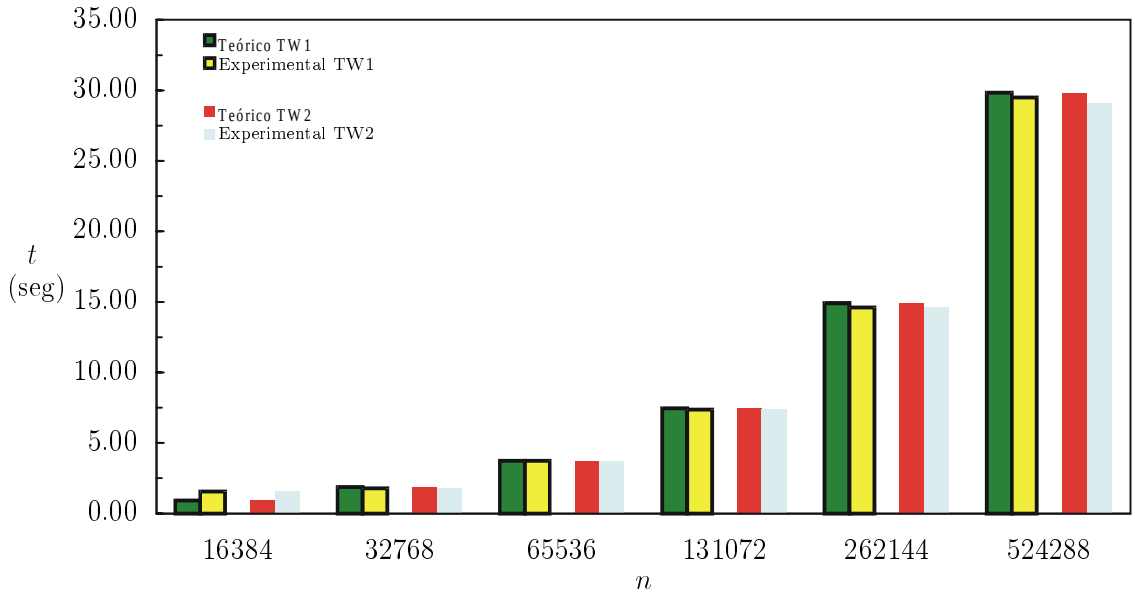
Figura 3.8: *Tiempos teóricos y experimentales del algoritmo 3.1 (TW) en un IBM SP2 con 2 procesadores interconectados mediante ethernet.*

IBM SP2 4 procesadores *ethernet*				
	TW1		TW2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0136	0.0159	0.0167	0.0199
256	0.0208	0.0192	0.0238	0.0223
512	0.0353	0.0397	0.0383	0.0457
1024	0.0645	0.0756	0.0673	0.0776
2048	0.1227	0.1317	0.1255	0.1353
4096	0.2392	0.2581	0.2419	0.2614
8192	0.4722	0.4516	0.4747	0.4645
16384	0.9382	1.5570	0.9403	1.5768
32768	1.8703	1.7849	1.8716	1.8026
65536	3.7344	3.7286	3.7340	3.7290
131072	7.4625	7.3648	7.4589	7.4057
262144	14.9188	14.6156	14.9087	14.6577
524288	29.8315	29.5015	29.8082	29.0884

Tabla 3.6: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2), medidos en un IBM SP2 con 4 procesadores interconectados mediante ethernet, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

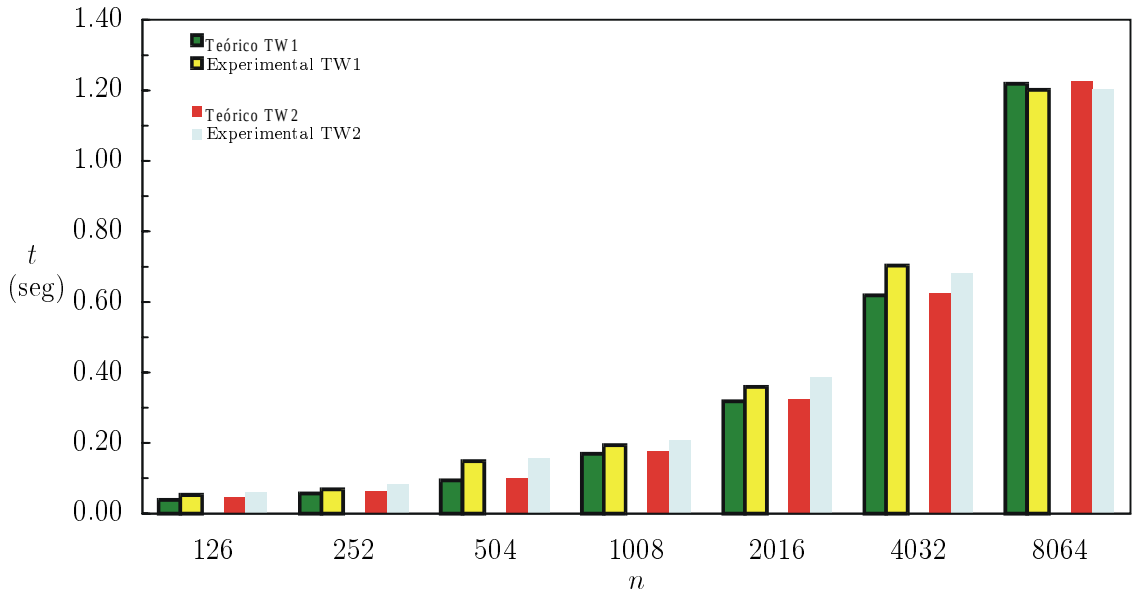


(b) $16384 \leq n \leq 524288$

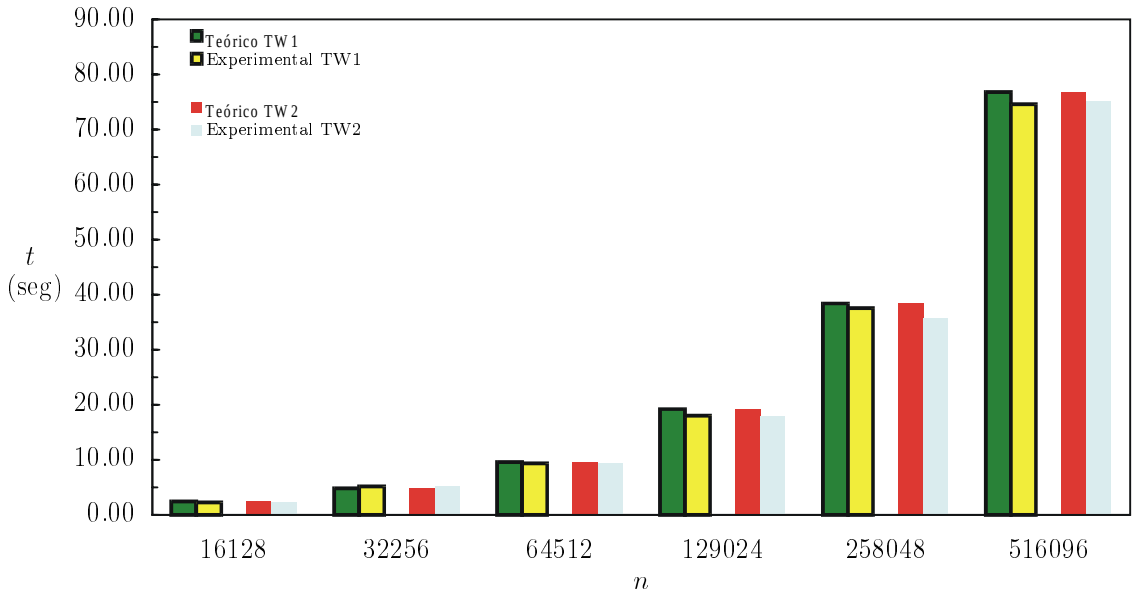
Figura 3.9: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con 4 procesadores interconectados mediante ethernet.*

IBM SP2 6 procesadores *switch*				
	TW1		TW2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0387	0.0530	0.0462	0.0613
252	0.0569	0.0693	0.0640	0.0826
504	0.0941	0.1485	0.1011	0.1578
1008	0.1689	0.1939	0.1758	0.2076
2016	0.3188	0.3595	0.3256	0.3873
4032	0.6187	0.7034	0.6253	0.6810
8064	1.2185	1.2018	1.2249	1.2027
16128	2.4181	2.2668	2.4241	2.2816
32256	4.8173	5.1613	4.8224	5.2240
64512	9.6158	9.3562	9.6191	9.4004
129024	19.2128	17.9982	19.2125	17.9316
258048	38.4068	37.5757	38.3993	35.7255
516096	76.7948	74.5992	76.7730	75.1931

Tabla 3.7: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2), medidos en un IBM SP2 con 6 procesadores interconectados mediante ethernet, para* $126 \leq n \leq 516096$.



(a) $126 \leq n \leq 8064$



(b) $16128 \leq n \leq 516096$

Figura 3.10: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con 6 procesadores interconectados mediante ethernet.*

En la tabla 3.8 y la figura 3.11 se muestran los tiempos teóricos y experimentales, medidos en segundos, del algoritmo 3.1 (TW) en el *cluster* de PC's para 2 procesadores y en las tablas 3.9, 3.10 y figuras 3.12, 3.13 se muestran los tiempos teóricos y experimentales, medidos en segundos, de las algoritmos 3.2(TW1) y 3.3(TW2) en el *cluster* de PC's para 4 y 6 procesadores.

Los tiempos han sido obtenidos para diferentes tamaños de la matriz de coeficientes, en el IBM SP2 el tamaño varía desde 128 a 524288 para 2 y 4 procesadores y desde 126 a 516096 para 6 procesadores; en el *cluster* de PC's el tamaño de la matriz de coeficientes varía desde 128 a 65536 para 2 y 4 procesadores y desde 126 a 64512 para 6 procesadores.

En las figuras 3.14, 3.15 y 3.16 se muestra el porcentaje de desviación del tiempo experimental con respecto al teórico en función del tamaño n del sistema.

Las mayores diferencias entre el tiempo previsto y el experimental se obtienen en los sistemas de tamaño pequeño en los que, por lo general, suele ser mayor el tiempo experimental que el teórico. A medida que aumenta el tamaño de sistema el tiempo obtenido experimentalmente se ajusta mejor al esperado, para tamaños grandes la desviación (salvo alguna excepción) oscila entre el 0% y el 5%.

En las tablas 3.11, 3.12 y 3.13 se muestra el algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 3.2 (TW1) y 3.3 (TW2), en cada una de las máquinas para los distintos tamaños de sistema. Se observa que en el IBM SP2 utilizando *switch* es mejor calcular m en paralelo (esto es, el algoritmo 3.3) para valores de $n > 4100$. En el IBM SP2 utilizando *ethernet* y en el *cluster* de PC's es mejor el algoritmo 3.3 para tamaños de sistema mayores, por encima de 50000 ecuaciones.

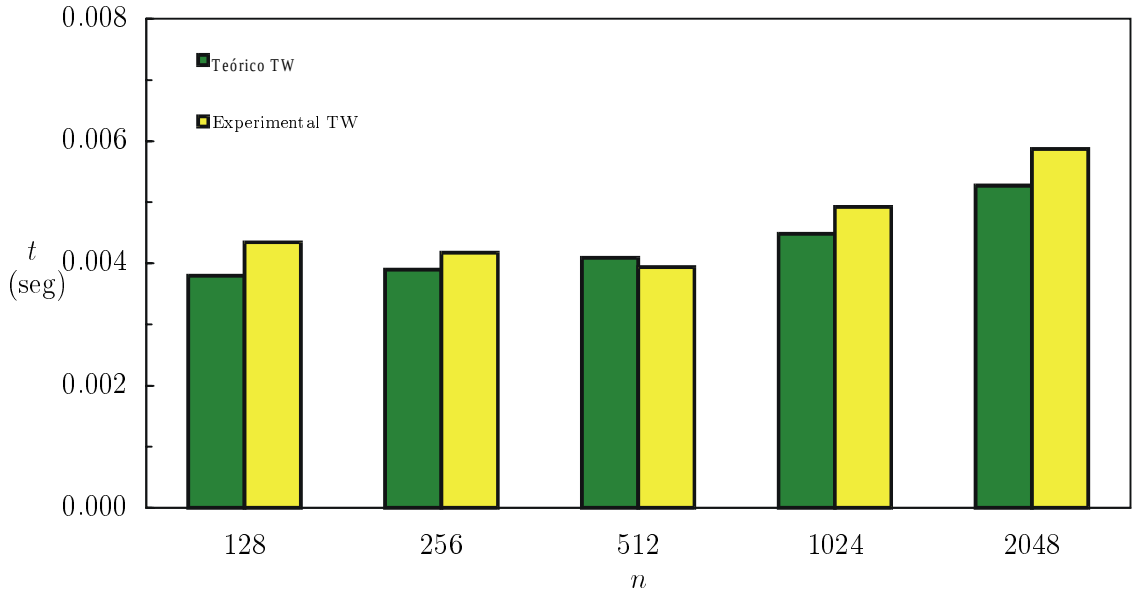
En la tabla 3.14 se muestra el *speed-up* y la eficiencia de los algoritmos 3.1 (TW), 3.2(TW1) y 3.3(TW2) con respecto al método de eliminación de Gauss para sistemas tridiagonales, que es el mejor algoritmo secuencial para la resolución de sistemas tridiagonales, considerando los tiempos teóricos de los sistemas de tamaño máximo de los que se han obtenido resultados experimentales en cada una de las máquinas.

La eficiencia que se obtiene no es muy buena, y en caso del IBM SP2 con *ethernet* es pésima; sin duda un factor determinante es el valor de g. Para otras máquinas con menor valor de g se obtiene mejor eficiencia, como es el caso de un CRAY T3D o un CRAY T3E. Por comodidad, en la tabla 3.15 se repiten los valores de los parámetros de estas

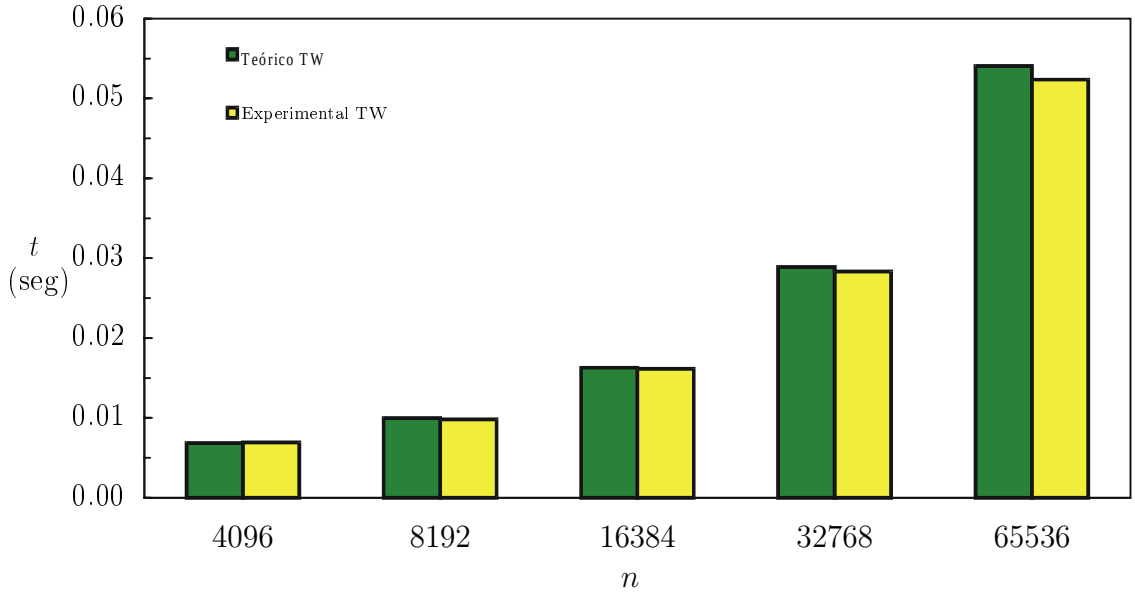
Cluster de PC's 2 procesadores		
	TW	
n	Teórico	Experimental
128	0.0038	0.0043
256	0.0039	0.0042
512	0.0041	0.0039
1024	0.0045	0.0049
2048	0.0053	0.0059
4096	0.0068	0.0069
8192	0.0100	0.0098
16384	0.0163	0.0161
32768	0.0289	0.0283
65536	0.0541	0.0523

Tabla 3.8: *Tiempos teóricos y experimentales del algoritmo 3.1 (TW) medidos en un cluster de PC's, para 2 procesadores y* $128 \leq n \leq 65536$.

máquinas que se muestran en la tabla 2.16, para las mismas se obtiene el *speed-up* y la eficiencia que figura en la tabla 3.16 considerando el mismo tamaño de sistema que para la tabla 3.14.



(a) $128 \leq n \leq 2048$

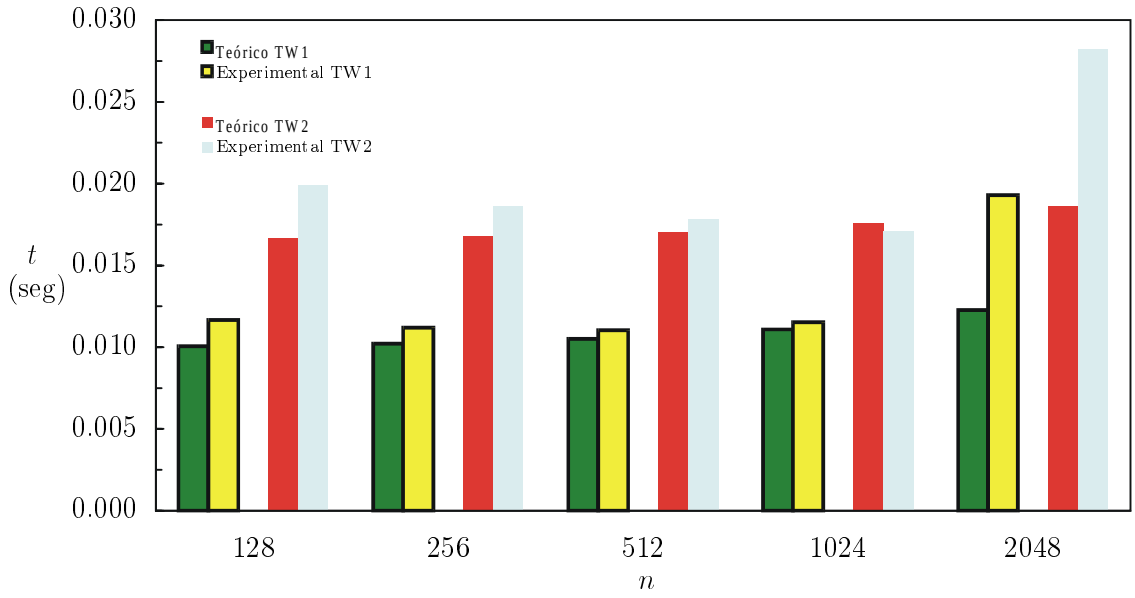


(b) $4096 \leq n \leq 65536$

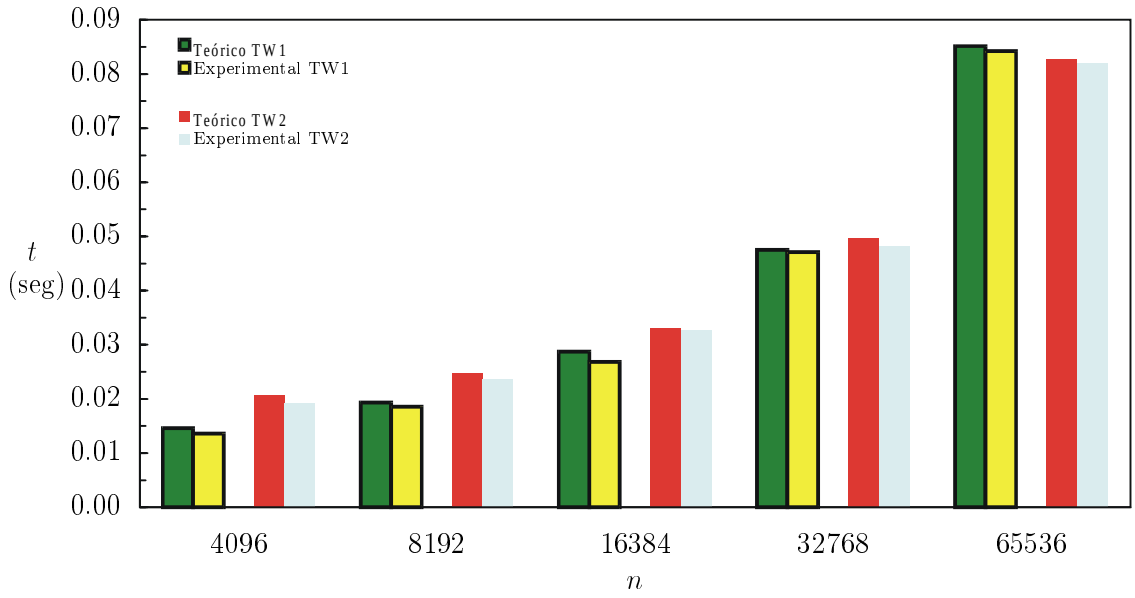
Figura 3.11: *Tiempos teóricos y experimentales del algoritmo 3.1 (TW) en un cluster de PC's para 2 procesadores.*

Cluster de PC's 4 procesadores				
	TW1		TW2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0101	0.0116	0.0166	0.0199
256	0.0102	0.0112	0.0168	0.0186
512	0.0105	0.0110	0.0170	0.0178
1024	0.0111	0.0115	0.0175	0.0171
2048	0.0123	0.0193	0.0186	0.0282
4096	0.0146	0.0136	0.0206	0.0192
8192	0.0193	0.0186	0.0248	0.0237
16384	0.0287	0.0268	0.0331	0.0326
32768	0.0475	0.0471	0.0496	0.0481
65536	0.0851	0.0842	0.0828	0.0820

Tabla 3.9: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) medidos en un cluster de PC's, para* 4 *procesadores y* $128 \leq n \leq 65536$.



(a) $128 \leq n \leq 2048$

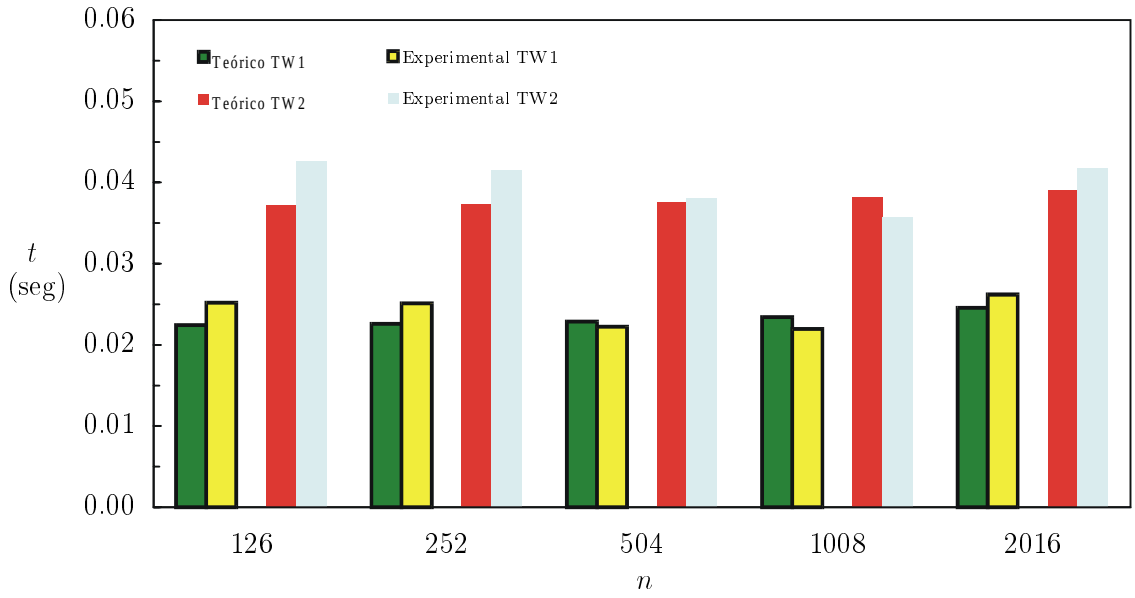


(b) $4096 \leq n \leq 65536$

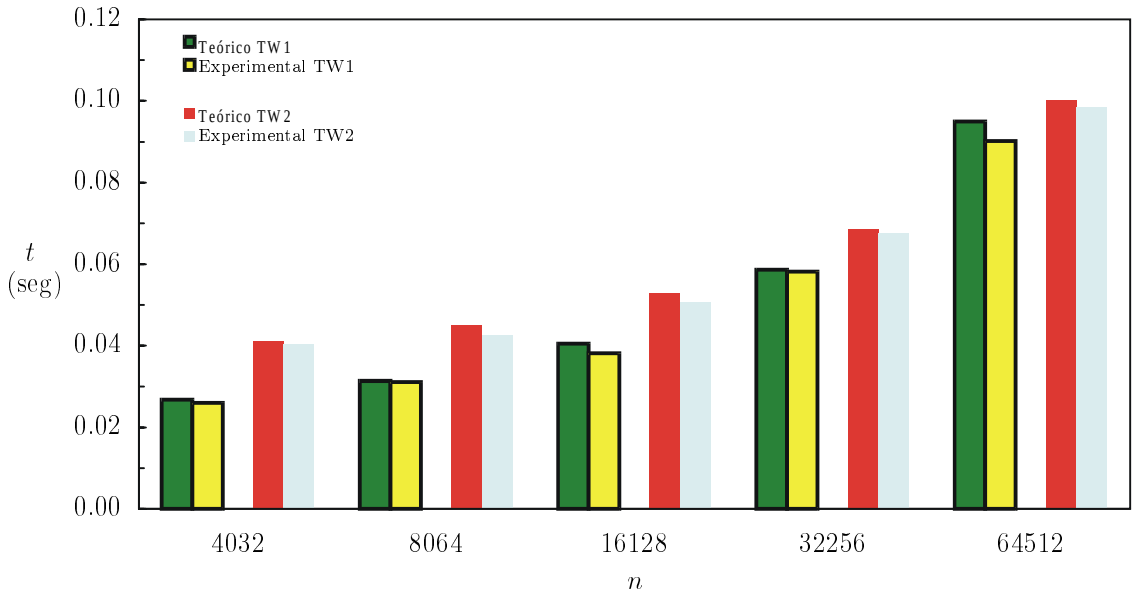
Figura 3.12: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un cluster de PC's para 4 procesadores.*

Cluster de PC's 6 procesadores				
	TW1		TW2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0224	0.0252	0.0373	0.0426
252	0.0226	0.0251	0.0374	0.0416
504	0.0229	0.0223	0.0376	0.0381
1008	0.0234	0.0220	0.0381	0.0357
2016	0.0246	0.0262	0.0391	0.0417
4032	0.0268	0.0260	0.0411	0.0403
8064	0.0314	0.0311	0.0450	0.0425
16128	0.0405	0.0382	0.0529	0.0506
32256	0.0586	0.0582	0.0686	0.0675
64512	0.0950	0.0902	0.1000	0.0984

Tabla 3.10: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) medidos en un cluster de PC's, para* 6 *procesadores y* $126 \leq n \leq 64512$.

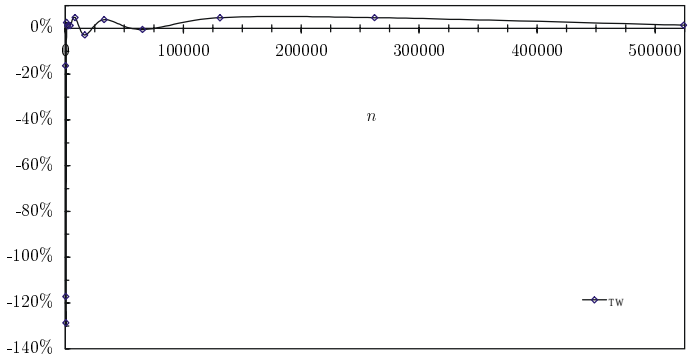


(a) $126 \leq n \leq 2016$

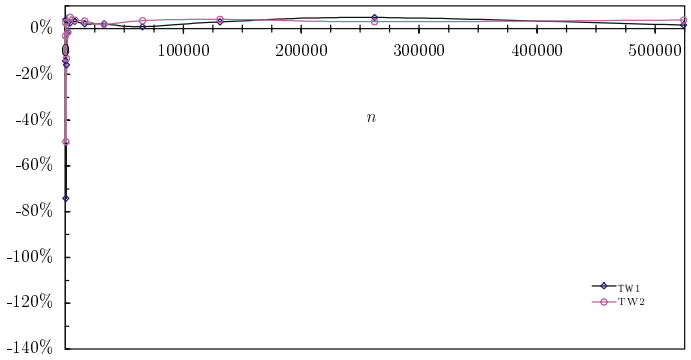


(b) $4032 \leq n \leq 64512$

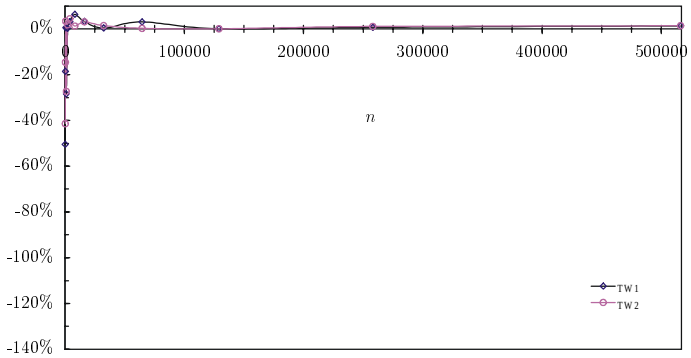
Figura 3.13: *Tiempos teóricos y experimentales de los algoritmos 3.2 (TW1) y 3.3 (TW2) en un cluster de PC's para* 6 *procesadores.*



(a) 2 *procesadores*

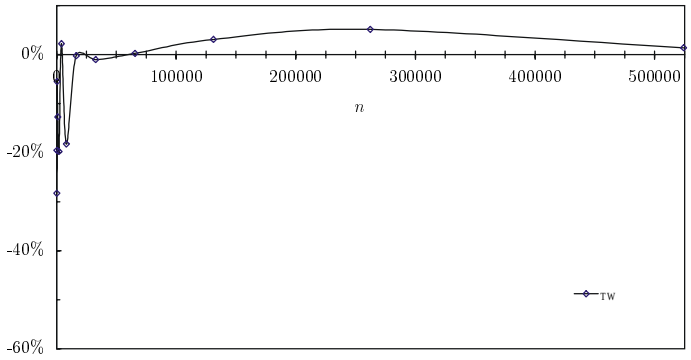


(b) 4 *procesadores*

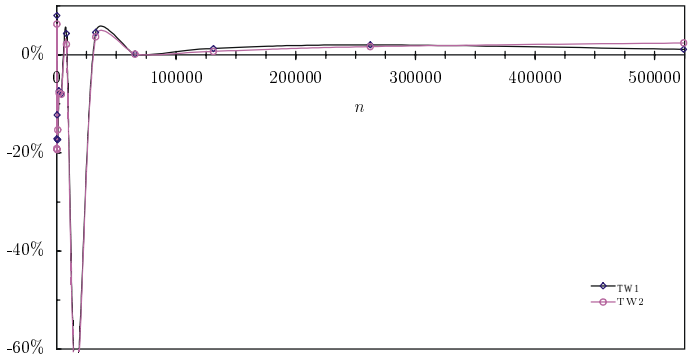


(c) 6 *procesadores*

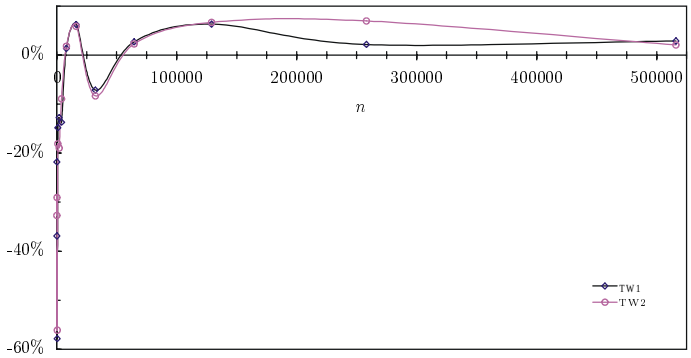
Figura 3.14: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con switch.*



(a) 2 *procesadores*

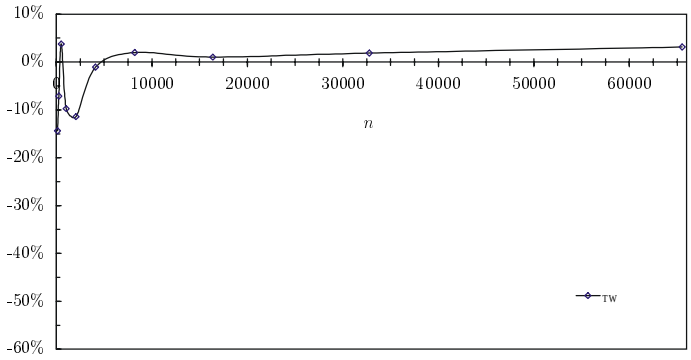


(b) 4 *procesadores*

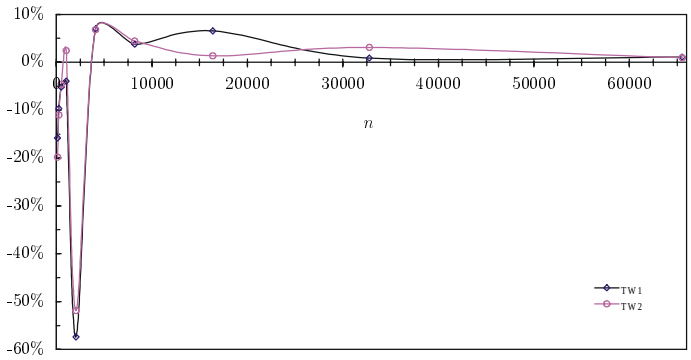


(c) 6 *procesadores*

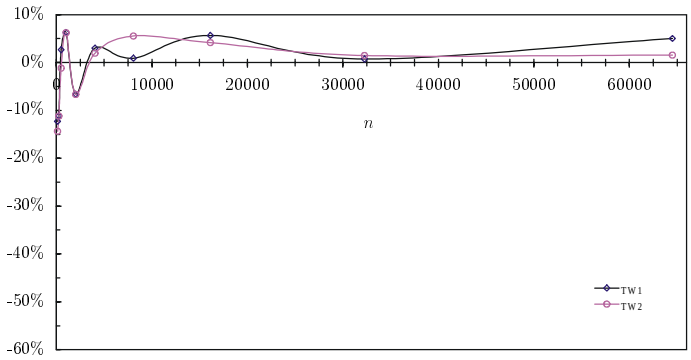
Figura 3.15: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 con ethernet.*



(a) 2 *procesadores*



(b) 4 *procesadores*



(c) 6 *procesadores*

Figura 3.16: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un cluster de PC's.*

IBM SP2 *switch*					
	4 procesadores			6 procesadores	
n	Teór.	Exper.	n	Teór.	Exper.
128	TW1	TW1	126	TW1	TW1
256	TW1	TW1	252	TW1	TW1
512	TW1	TW1	504	TW1	TW1
1024	TW1	TW1	1008	TW1	TW1
2048	TW1	TW1	2016	TW1	TW1
4096	TW1	TW2	4032	TW1	TW1
8192	TW2	TW2	8064	TW2	TW1
16384	TW2	TW2	16128	TW2	TW2
32768	TW2	TW2	32256	TW2	TW2
65536	TW2	TW2	64512	TW2	TW2
131072	TW2	TW2	129024	TW2	TW2
262144	TW2	TW2	258048	TW2	TW2
524288	TW2	TW2	516096	TW2	TW2

Tabla 3.11: *Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 switch.*

IBM SP2 *switch*					
	4 procesadores			6 procesadores	
n	Teór.	Exper.	n	Teór.	Exper.
128	TW1	TW1	126	TW1	TW1
256	TW1	TW1	252	TW1	TW1
512	TW1	TW1	504	TW1	TW1
1024	TW1	TW1	1008	TW1	TW1
2048	TW1	TW1	2016	TW1	TW1
4096	TW1	TW1	4032	TW1	TW2
8192	TW1	TW1	8064	TW1	TW1
16384	TW1	TW1	16128	TW1	TW1
32768	TW1	TW1	32256	TW1	TW1
65536	TW2	TW1	64512	TW1	TW1
131072	TW2	TW1	129024	TW2	TW2
262144	TW2	TW1	258048	TW2	TW2
524288	TW2	TW2	516096	TW2	TW1

Tabla 3.12: *Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 ethernet.*

Cluster de PC's					
	4 procesadores			6 procesadores	
n	Teór.	Exper.	n	Teór.	Exper.
128	TW1	TW1	126	TW1	TW1
256	TW1	TW1	252	TW1	TW1
512	TW1	TW1	504	TW1	TW1
1024	TW1	TW1	1008	TW1	TW1
2048	TW1	TW1	2016	TW1	TW1
4096	TW1	TW1	4032	TW1	TW1
8192	TW1	TW1	8064	TW1	TW1
16384	TW1	TW1	16128	TW1	TW1
32768	TW1	TW1	32256	TW1	TW1
65536	TW2	TW2	64512	TW1	TW1

Tabla 3.13: *Algoritmo más rápido (teórica y experimentalmente) entre los algoritmos 3.2 (TW1) y 3.3 (TW2) en un cluster de PC's.*

IBM SP2 *switch*				
	TW o TW1		TW2	
p	S_p	E_p	S_p	E_p
2	0.51	25.26%		
4	0.28	7.06%	0.31	7.66%
6	0.26	4.37%	0.29	4.76%

(a) *IBM SP2 switch*

IBM SP2 *ethernet*				
	TW o TW1		TW2	
p	S_p	E_p	S_p	E_p
2	0.01	0.45%		
4	0.00	0.08%	0.00	0.08%
6	0.00	0.02%	0.00	0.02%

(b) *IBM SP2 ethernet*

Cluster de PC's				
	TW o TW1		TW2	
p	S_p	E_p	S_p	E_p
2	0.59	29.50%		
4	0.37	9.36%	0.39	9.63%
6	0.33	5.51%	0.31	5.23%

(c) *Cluster de PC's*

Tabla 3.14: *Speed-up (S_p) y eficiencia (E_p) de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un IBM SP2 y un cluster de PC's.*

s	p	l	g	$n_{\frac{1}{2}}$
12	1	68	0.3	94
	2	164	0.7	71
	4	168	0.7	66
	8	175	0.8	59
	16	181	0.9	61
	32	201	1.1	28
	64	148	1	27
	128	301	1.1	20
	256	387	1.2	15

(a) *CRAY T3D*

s	p	l	g	$n_{\frac{1}{2}}$
46.7	1	86	2.12	9
	2	269	0.87	33
	4	357	0.87	40
	8	506	0.81	40
	16	751	1.04	38
	32	1252	1.31	45

(b) *CRAY T3E*

Tabla 3.15: *Valores de parámetros BSP.*

CRAY T3D				
	TW o TW1		TW2	
p	S_p	E_p	S_p	E_p
2	1.96	97.84%		
4	1.27	31.67%	1.97	49.20%
8	1.39	17.38%	2.56	31.96%
16	1.42	8.91%	2.85	17.84%
32	1.35	4.22%	2.65	8.29%
64	1.43	2.23%	3.03	4.73%
128	1.38	1.08%	2.82	2.21%
256	1.32	0.52%	2.61	1.02%

(a) *CRAY T3D*

CRAY T3E				
	TW o TW1		TW2	
p	S_p	E_p	S_p	E_p
2	1.57	78.60%		
4	1.17	29.27%	1.73	43.32%
8	1.39	17.31%	2.54	31.72%
16	1.35	8.41%	2.55	15.95%
32	1.22	7.60%	2.12	13.25%

(b) *CRAY T3E*

Tabla 3.16: *Speed-up (S_p) y eficiencia (E_p) de los algoritmos 3.1 (TW), 3.2 (TW1) y 3.3 (TW2) en un CRAY T3D y un CRAY T3E.*

Capítulo 4

Método de Wang

4.1 Descripción del método

Dada una matriz tridiagonal A de tamaño $n \times n$

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix}, \tag{4.1}$$

estrictamente diagonal dominante e irreducible, de nuevo se considera el problema general de obtener la solución del sistema

$$A\boldsymbol{x} = \boldsymbol{d}, \tag{4.2}$$

donde

$$\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

es el vector de términos independientes.

Se supone que existen dos números naturales k y p tales que $k = \dfrac{n}{p}$ y, de nuevo, se considera en A la siguiente partición por bloques

$$A = \begin{bmatrix} A_0 & B_0 & & & \\ C_1 & A_1 & B_1 & & \\ & \ddots & \ddots & \ddots & \\ & & C_{p-2} & A_{p-2} & B_{p-2} \\ & & & C_{p-1} & A_{p-1} \end{bmatrix} \tag{4.3}$$

donde cada uno de los bloques diagonales

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

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

$$C_{i+1} = \left[\begin{array}{cccc|c} 0 & 0 & \cdots & 0 & c_{(i+1)k+1} \\ \hline & & & & 0 \\ & & O & & \vdots \\ & & & & 0 \\ & & & & 0 \end{array}\right]$$

y superdiagonal

$$B_i = \left[\begin{array}{c|ccc} 0 & & & \\ 0 & & O & \\ \vdots & & & \\ 0 & & & \\ \hline b_{(i+1)k} & 0 & \cdots & 0 \quad 0 \end{array}\right]$$

es una matriz de tamaño $k \times k$ con un sólo elemento no nulo.

En los vectores $\boldsymbol{x}$ y $\boldsymbol{d}$ se considera una partición por bloques acorde con la realizada en la matriz de coeficientes A, es decir

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{p-2} \\ \boldsymbol{x}_{p-1} \end{bmatrix} \quad \text{y} \quad \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_{p-2} \\ \boldsymbol{d}_{p-1} \end{bmatrix}, \tag{4.4}$$

con

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

En consecuencia, el sistema (4.2) se puede escribir como

$$\begin{bmatrix} A_0 & B_0 & & & \\ C_1 & A_1 & B_1 & & \\ & \ddots & \ddots & \ddots & \\ & & C_{p-2} & A_{p-2} & B_{p-2} \\ & & & C_{p-1} & A_{p-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{p-2} \\ \boldsymbol{x}_{p-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_{p-2} \\ \boldsymbol{d}_{p-1} \end{bmatrix}. \tag{4.5}$$

La idea central en que se basa este método consiste en proceder de forma simultánea en los distintos bloques de la partición (4.3) a la eliminación de los elementos situados por encima y por debajo de la diagonal principal de la matriz de coeficientes, realizando las oportunas actualizaciones en los elementos de A afectados en cada paso, hasta transformar A en una matriz diagonal. Al mismo tiempo se realiza una actualización sobre los elementos del vector de términos independientes que resulten afectados por las operaciones sobre la matriz de coeficientes.

El método se puede resumir en los siguientes pasos:

1 Se realiza sobre la matriz A y los vectores $\boldsymbol{x}, \boldsymbol{d}$ la partición (4.5).

2 Se anulan los elementos subdiagonales de los bloques diagonales; este proceso hace que en los bloques subdiagonales aparezcan nuevos elementos no nulos en las columnas ik-ésimas, para $i = 1, 2, \ldots, p-1$, elementos que se denotan por f_j, con $j = k+1, k+2, \ldots, n$.

3 Se anulan los elementos superdiagonales de los bloques diagonales salvo los situados en las columnas ik–ésimas, para $i = 1, 2, \ldots, p-1$, y los elementos no nulos de los bloques superdiagonales; esto hace que en las columnas ik–ésimas, para $i = 1, 2, \ldots, p$, aparezcan nuevos elementos no nulos que se denotan por g_j, con $j = 1, 2, \ldots, n-1$.

4 En cada bloque subdiagonal se anulan los elementos f_j, situados por debajo de la diagonal principal, obteniendo así una matriz triangular superior.

5 La matriz A se diagonaliza anulando los elementos por encima de la diagonal principal y se obtiene la solución de forma inmediata.

A continuación se dará una descripción más detallada del método en general y se irá aplicando al caso particular en que $n = 16$ y $p = 4$, véase la figura 4.1.

Inicialmente en cada bloque A_i, para $i = 0, 1, \ldots, p-1$, se eliminan los elementos situados por debajo de la diagonal principal. Se realizan las siguientes operaciones,

- para $i = 1, 2, \ldots, p-1$,

$$f_{ik+1} \leftarrow c_{ik+1};$$

- para $i = 0, 1, \ldots, p-1$ y $j = 2, 3, \ldots, k$,

$$\left.\begin{array}{rcl} c_{ik+j} & \leftarrow & \frac{c_{ik+j}}{a_{ik+j-1}}, \\ a_{ik+j} & \leftarrow & a_{ik+j} - c_{ik+j} b_{ik+j-1}, \\ d_{ik+j} & \leftarrow & d_{ik+j} - c_{ik+j} d_{ik+j-1}, \end{array}\right\} \tag{4.6}$$

- para $i = 1, 2, \ldots, p-1$ y $j = 2, 3, \ldots, k$,

$$f_{ik+j} \leftarrow -c_{ik+j} f_{ik+j-1}.$$

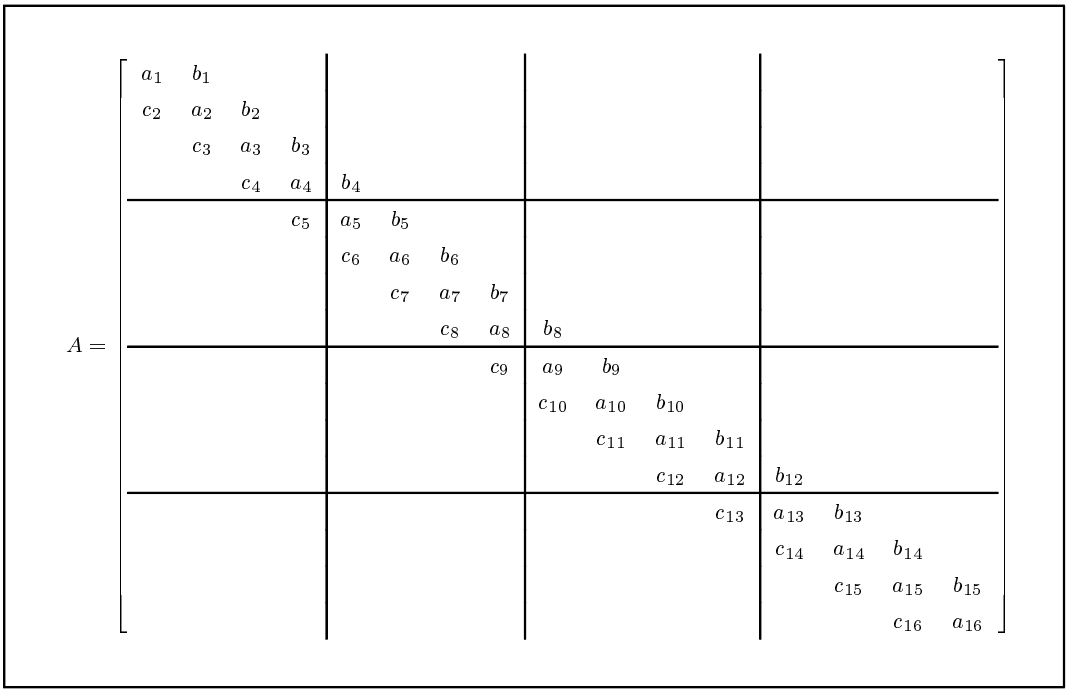


Figura 4.1: *Matriz de coeficientes para el caso* $n = 16$.

Después de este paso inicial, la matriz de coeficientes del caso particular tiene la forma que se muestra en la figura 4.2.

Nótese que en este primer paso aparecen nuevos elementos no nulos f_{ik+j}, para $i = 1, 2, \ldots, p-1$ y $j = 1, 2, \ldots, k$, en las columnas ik-ésimas.

En segundo lugar, en cada bloque diagonal, se anulan los elementos superdiagonales excepto los que se encuentran en las columnas ik-ésimas, para $i = 1, 2, \ldots, p-1$. Se realizan las siguientes operaciones,

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

$$g_{(i+1)k-1} \leftarrow b_{(i+1)k-1};$$

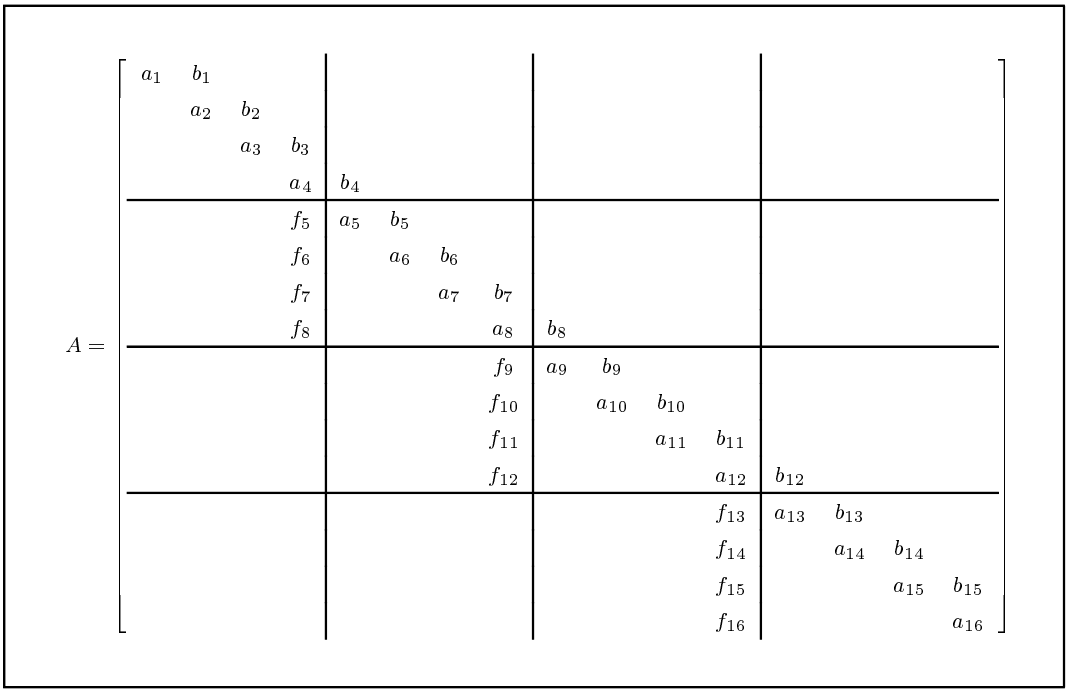


Figura 4.2: *Transformación de la matriz de coeficientes después de la eliminación de los elementos situados por debajo de la diagonal principal, para el caso* $n = 16$.

- para $i = 0, 1, \ldots, p-1$ y $j = 2, 3, \ldots, k-1$,

$$\left.\begin{array}{rcl} b_{(i+1)k-j} & \leftarrow & \frac{b_{(i+1)k-j}}{a_{(i+1)k+1-j}}, \\ g_{(i+1)k-j} & \leftarrow & -b_{(i+1)k-j} g_{(i+1)k+1-j}, \\ d_{(i+1)k-j} & \leftarrow & d_{(i+1)k-j} - b_{(i+1)k-j} d_{(i+1)k+1-j}, \end{array}\right\} \tag{4.7}$$

- para $i = 1, 2, \ldots, p-1$ y $j = 2, 3, \ldots, k-1$,

$$f_{(i+1)k-j} \leftarrow f_{(i+1)k-j} - b_{(i+1)k-j} f_{(i+1)k+1-j}.$$

La figura 4.3 muestra la forma en que queda la matriz tras la realización de estas operaciones.

A continuación, para $i = 1, 2, \ldots, p-1$, se anulan los elementos b_{ik} que se encuentran en las columnas $(ik+1)$-ésimas (en la figura 4.3 son los elementos remarcados). Nótese que para anular estos elementos deben realizarse operaciones elementales en las que intervienen elementos que no se encuentran en la misma fila de la partición (en el sentido de que los

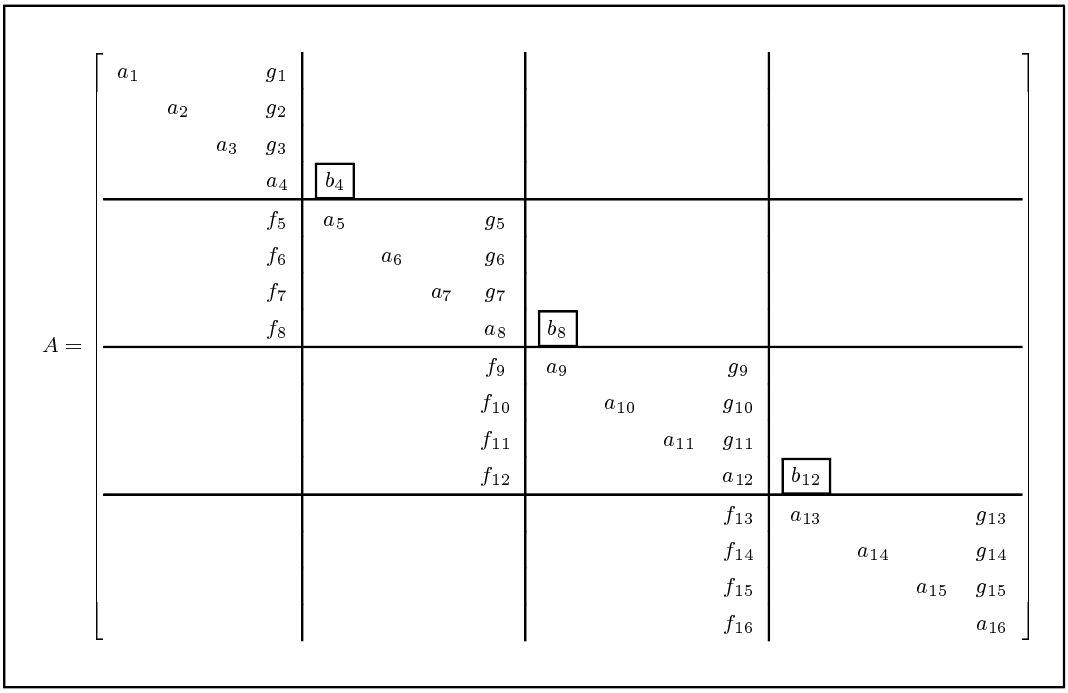


Figura 4.3: *Transformación de la matriz de coeficientes después de la eliminación los elementos situados por encima de la diagonal principal, para el caso* $n = 16$.

subíndices de los bloques C_i, A_i, y B_i coinciden) ya que para eliminar el elemento b_{ik}, que se encuentra en los bloques C_{i-1}, A_{i-1}, B_{i-1}, se efectúan operaciones en las que intervienen elementos de la fila $(ik+1)$-ésima y esta fila forma parte de los bloques C_i, A_i, B_i.

Las operaciones que se llevan a cabo son, para $i = 1, 2, \ldots, p-1$,

$$\left.\begin{array}{rcl} b_{ik} & \leftarrow & \frac{b_{ik}}{a_{ik+1}}, \\ a_{ik} & \leftarrow & a_{ik} - b_{ik} f_{ik+1}, \\ g_{ik} & \leftarrow & -b_{ik} g_{ik+j}, \\ d_{ik} & \leftarrow & d_{ik} - b_{ik} d_{ik+1}. \end{array}\right\} \tag{4.8}$$

Aparecen así nuevos elementos g_{ik} en las columnas $(i+1)k$-ésimas, para $i = 1, 2, \ldots, p-1$.

La figura 4.4 muestra la forma en que queda la matriz del ejemplo una vez anulados los elementos b_4, b_8 y b_{12}; los nuevos elementos g_j han sido remarcados.

Para anular b_{ik}, para $i = 1, 2, \ldots, p-1$, utilizando las expresiones (4.8), se necesi-

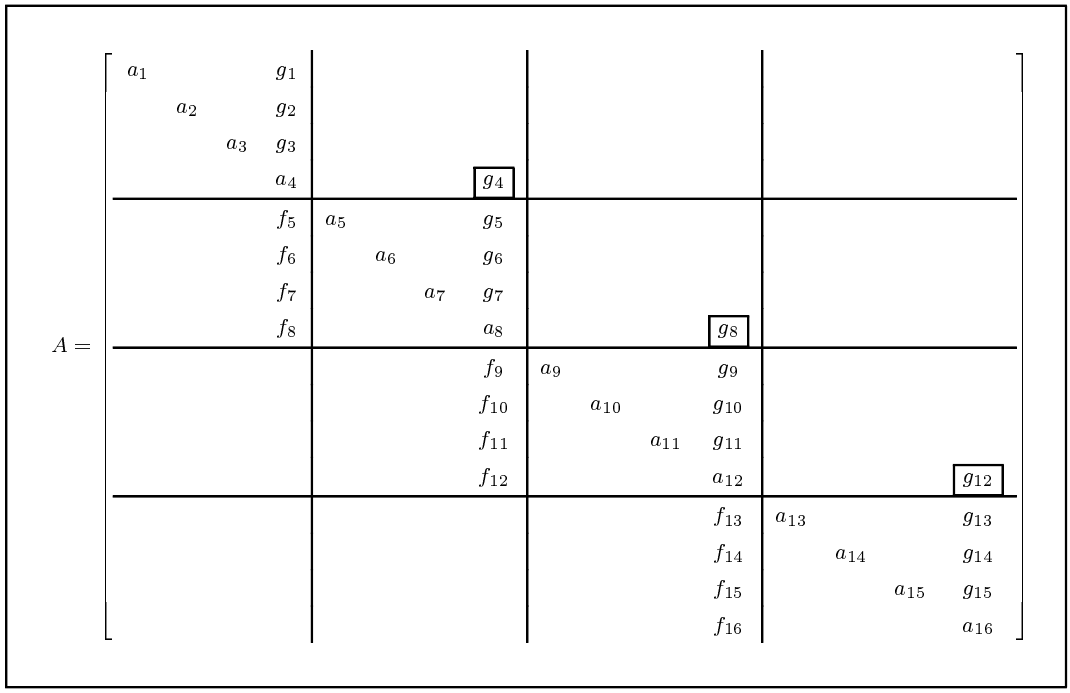


Figura 4.4: *Transformación de la matriz de coeficientes después de anular los elementos b_4, b_8 y b_{12}, para el caso $n = 16$.*

tan elementos que se encuentran en la fila $(ik+1)$-ésima; esto supone una dificultad al paralelizar el método ya que los elementos implicados se encuentran en procesadores distintos, lo que obliga a efectuar una comunicación entre los procesadores para actualizar los elementos en uno de ellos y, una vez actualizados, volverlos a enviar al procesador origen.

El siguiente paso consiste en la eliminación (en cada uno de los bloques diagonales correspondientes) de los elementos f_{ik+j}, para $i = 1, 2, \ldots, p-1$ y $j = 1, 2, \ldots, k$, consiguiendo de este modo que la matriz de coeficientes sea triangular superior. Para eliminar estos elementos se realizan las siguientes operaciones

$$\left.\begin{array}{rcl} f_{ik+j} & \leftarrow & \frac{f_{ik+j}}{a_{ik}}, \\ g_{ik+j} & \leftarrow & g_{ik+j} - f_{ik+j} g_{ik}, \\ d_{ik+j} & \leftarrow & d_{ik+j} - f_{ik+j} d_{ik}. \end{array}\right\}$$

Después de este paso, la matriz del ejemplo queda en la forma que se muestra en la

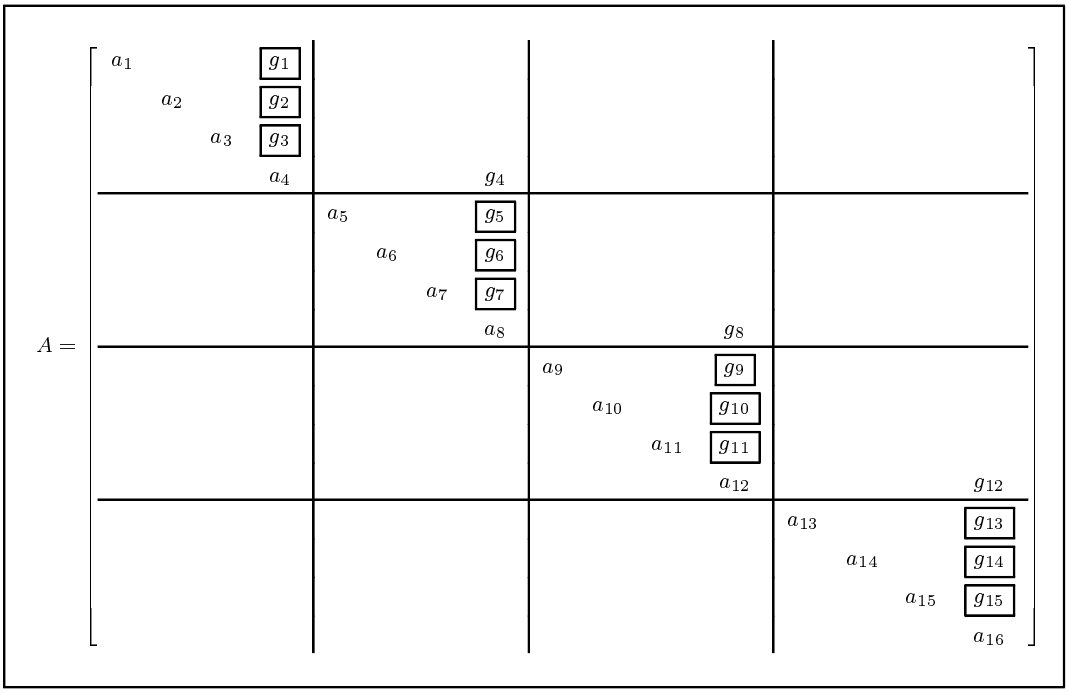


Figura 4.5: *Transformación de la matriz de coeficientes después de anular los elementos* f_{4i+j}*, con* $i = 1, 2, 3$ *y* $j = 1, 2, 3, 4$*, para el caso* $n = 16$*.*

figura 4.5.

Ahora únicamente quedan elementos no nulos fuera de la diagonal principal en las columnas ik-ésimas, con $i = 1, 2, \ldots, p$. Para anularlos se realizan las siguientes operaciones,

- para $i = p-1, p-2, \ldots, 1$ y $j = 1, 2, \ldots, k$,

$$\left.\begin{array}{rcl} g_{ik+j-1} & \leftarrow & \frac{g_{ik+j-1}}{a_{(i+1)\,k}}, \\ d_{ik+j-1} & \leftarrow & d_{ik+j-1} - g_{ik+j-1} d_{(i+1)k}, \end{array}\right\}$$

- para $j = 1, 2, \ldots, k-1$,

$$\left.\begin{array}{rcl} g_j & \leftarrow & \frac{g_j}{a_k}, \\ d_j & \leftarrow & d_j - g_j d_k. \end{array}\right\}$$

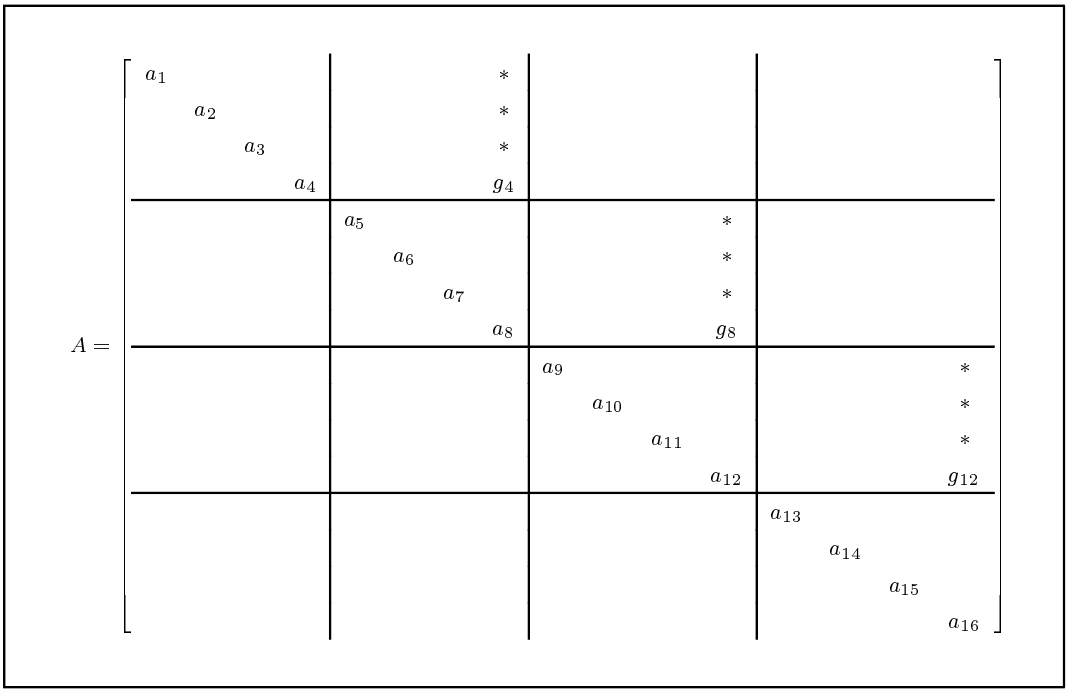


Figura 4.6: *Transformación de la matriz de coeficientes después de anular los elementos* g_{4i+j}*, con* $i = 0, 1, 2, 3$ *y* $j = 1, 2, 3$*, para el caso* $n = 16$*.*

Conviene aclarar que (en principio) el proceso de eliminación de estos elementos no puede efectuarse simultáneamente en todos los bloques diagonales (y por ende en paralelo) ya que al realizar las operaciones elementales necesarias aparecerían nuevos elementos no nulos, por ello el proceso de eliminación comienza con g_{n-1} y finaliza con g_1. La matriz final es diagonal y la solución se obtiene de forma inmediata mediante

$$x_i = \frac{d_i}{a_i}, \quad i = 1, 2, \ldots, n.$$

En las figura 4.6 se muestra cómo quedaría la matriz A del ejemplo si se anulan simultáneamente los elementos g_{4i+j}, con $i = 0, 1, 2, 3$ y $j = 1, 2, 3$ (remarcados en la figura 4.5) y en la figura 4.7 cómo quedaría la matriz A si, a continuación, se eliminan los nuevos elementos no nulos ($*$) que aparecen en la figura 4.6.

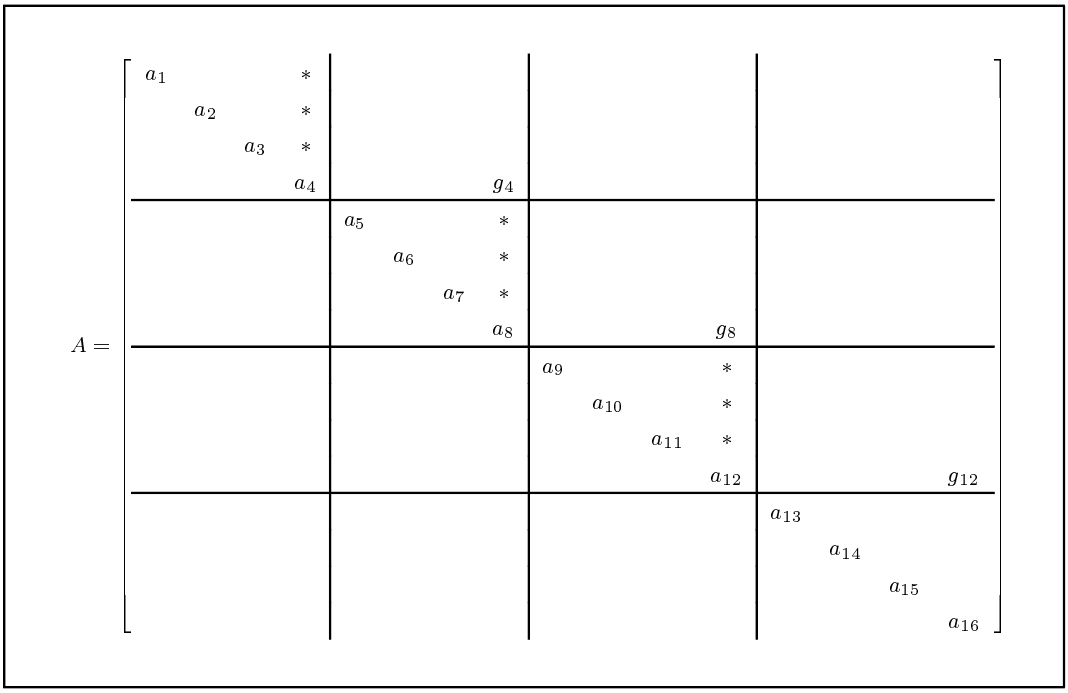


Figura 4.7: *Transformación de la matriz de coeficientes después de eliminar los nuevos elementos no nulos, para el caso* $n = 16$.

4.2 Paralelización del método

Para paralelizar el método se considera que p es el número de procesadores y se asigna a cada uno de los procesadores P_i, para $i = 0, 1, \ldots, p-1$, los bloques C_i, A_i, B_i y $\boldsymbol{d}_i$ de la partición del sistema (4.2) dada en la expresión (4.5). Como ya se ha comentado en la sección 4.1 el método presenta dificultades para su paralelización a la hora de eliminar los elementos interprocesadores b_{ik} que se encuentran en las filas ik-ésimas, para $i = 1, 2, \ldots, p-1$; su anulación obliga a efectuar un paso de comunicación y actualización de los elementos que resultan modificados. Asimismo, la eliminación en paralelo de los elementos g_{ik} (también situados en las filas ik-ésimas), para $i = 1, 2, \ldots, p-1$, presenta una dificultad que será estudiada en esta sección.

El siguiente algoritmo describe la forma de paralelizar el método.

Algoritmo 4.1 Método de las particiones de Wang en paralelo.

Paso 1

Se considera en el sistema (4.2) la partición dada en la expresión (4.5) y se asigna al procesador P_0 los bloques A_0, B_0 y $\boldsymbol{d}_0$, al procesador P_i, para $i = 1, 2, \ldots, p-2$, los bloques C_i, A_i, B_i y $\boldsymbol{d}_i$ y, finalmente, al procesador P_{p-1} los bloques C_{p-1}, A_{p-1}, y $\boldsymbol{d}_{p-1}$.

Paso 2

Cada procesador P_i, para $i = 0, 1, \ldots, p-1$, anula los elementos subdiagonales y superdiagonales (excepto $b_{(i+1)k-1}$) del bloque A_i y realiza las oportunas actualizaciones en el resto de bloques.

Paso 3

Cada procesador P_i, para $i = p-1, p-2, \ldots, 1$, comunica al procesador P_{i-1} los elementos necesarios para que este último anule el elemento b_{ik} y actualice los elementos afectados.

Paso 4

Cada procesador P_{i-1}, para $i = 1, 2, \ldots, p-1$, comunica al procesador P_i los elementos necesarios para que este último anule los elementos f_j, con $j = ik+1, ik+2, \ldots, (i+1)k$, que han aparecido en los bloques C_i y actualice los elementos afectados.

Paso 5

Cada procesador P_i, para $i = 1, 2, \ldots, p-2$ comunica al procesador P_0 los valores actualizados de los bloques A_i, B_i y $\boldsymbol{d}_i$ y el procesador P_{p-1} comunica a P_0 los valores actualizados de los bloques A_{p-1} y $\boldsymbol{d}_{p-1}$. El procesador P_0 anula los elementos superdiagonales g_j, para $j = n-1, n-2, \ldots, 1$, de la matriz A y obtiene la solución del sistema de forma inmediata ya que A es diagonal.

El paso 5 del algoritmo 4.1 puede ser paralelizado tomando los elementos que se encuentran en las filas interprocesadores y formando un sistema auxiliar

$$H\bar{\boldsymbol{x}} = \bar{\boldsymbol{d}}, \tag{4.9}$$

cuya solución, en cada procesador, permite obtener a cada procesador el resto de compo-

nentes de su solución parcial.

En el siguiente ejemplo se va a obtener el sistema auxiliar (4.9) y a describir la forma de obtener la solución parcial, en el caso particular en que $n = 16$ y $p = 4$. A continuación se estudiará el caso general.

Ejemplo 4.1 *Se parte inicialmente de la matriz de la figura (4.4), en la que los únicos elementos que se encuentran fuera de la diagonal principal están en las columnas ik-ésimas, para $i = 1, 2, \ldots, 4$. El sistema auxiliar (4.9) tiene la forma*

$$\begin{bmatrix} a_1 & g_1 & & & \\ f_4 & a_4 & g_4 & & \\ & f_8 & a_8 & g_8 & \\ & & f_{12} & a_{12} & g_{12} \\ & & & f_{16} & a_{16} \end{bmatrix} \begin{bmatrix} x_1 \\ x_4 \\ x_8 \\ x_{12} \\ x_{16} \end{bmatrix} \begin{bmatrix} d_1 \\ d_4 \\ d_8 \\ d_{12} \\ d_{16} \end{bmatrix}. \tag{4.10}$$

Nótese que H es una matriz cuadrada tridiagonal de tamaño $p+1 = 5$ con la característica de que $f_4 = 0$.

La idea de la paralelización del método consiste en que cada procesador reciba los elementos necesarios para formar y resolver el sistema auxiliar (4.9) y así poder seguir trabajando en paralelo para obtener el resto de componentes de su solución parcial.

En el ejemplo, una vez resuelto en cada procesador el sistema auxiliar (4.10), cada procesador conoce las componentes x_1, x_4, x_8, x_{12} y x_{16} y debe proceder a obtener el resto de componentes de su solución parcial. Nótese que cada procesador debe calcular $k - 1$ componentes de su solución parcial a excepción del primero que sólo necesita calcular $k - 2$ componentes, ya que x_1 ha sido obtenida al resolver el sistema auxiliar (4.10). Por comodidad en la notación y puesto que no se aumenta el coste computacional del método, se considera que el procesador P_0 calcula el mismo número de componentes de su solución parcial que el resto.

El procesador P_0 tiene el subsistema

$$\left.\begin{array}{rcrcrcrcl} a_1x_1 & & & + & g_1x_4 & & & = & d_1 \\ & a_2x_2 & & + & g_2x_4 & & & = & d_2 \\ & & a_3x_3 & + & g_3x_4 & & & = & d_3 \\ & & & & a_4x_4 & + & g_4x_8 & = & d_4 \end{array}\right\}$$

y conoce las valores de x_4 y x_8 por lo que únicamente debe calcular x_1, x_2 y x_3, de manera que debe resolver el sistema

$$\left.\begin{array}{rcrcrcl} a_1x_1 & & & + & g_1x_4 & = & d_1 \\ & a_2x_2 & & + & g_2x_4 & = & d_2 \\ & & a_3x_3 & + & g_3x_4 & = & d_3 \end{array}\right\}.$$

Despejando las incógnitas x_1, x_2 y x_3 del sistema anterior se tiene

$$x_i = \frac{1}{a_i}\left(d_i - g_i x_4\right), \quad i = 1, 2, 3.$$

El procesador P_1 tiene el subsistema

$$\left.\begin{array}{rcrcrcrcrcl} f_5x_4 & + & a_5x_5 & & & & + & g_5x_8 & & = & d_5 \\ f_6x_4 & + & & a_6x_6 & & & + & g_6x_8 & & = & d_6 \\ f_7x_4 & + & & & a_7x_7 & + & g_7x_8 & & & = & d_7 \\ f_8x_4 & + & & & & & a_8x_8 & + & g_8x_{12} & = & d_8 \end{array}\right\}$$

y conoce los valores de x_4, x_8 y x_{12} por lo que, únicamente, debe calcular x_5, x_6 y x_7, de manera que debe resolver el sistema

$$\left.\begin{array}{rcrcrcrcl} f_5x_4 & + & a_5x_5 & & & + & g_5x_8 & = & d_5 \\ f_6x_4 & + & & a_6x_6 & & + & g_6x_8 & = & d_6 \\ f_7x_4 & + & & & a_7x_7 & + & g_7x_8 & = & d_7 \end{array}\right\}.$$

Despejando las incógnitas x_5, x_6 y x_7 se obtiene

$$x_i = \frac{1}{a_i}\left(d_i - g_i x_8 - f_i x_4\right), \quad i = 5, 6, 7.$$

Razonando de igual manera, se obtiene en P_2

$$x_i = \frac{1}{a_i}\left(d_i - g_i x_{12} - f_i x_8\right), \quad i = 9, 10, 11,$$

y en P_3

$$x_i = \frac{1}{a_i}\left(d_i - g_i x_{16} - f_i x_{12}\right), \quad i = 13, 14, 15.$$

Mediante esta técnica se consigue que el proceso de cálculo de las soluciones se realice en paralelo hasta el final del mismo. ■

Se puede generalizar el resultado obtenido en el ejemplo 4.1 al caso en que la matriz A es de orden n y se trabaja con p procesadores. Se suponen realizadas en A las operaciones elementales necesarias para que los únicos elementos de A que se encuentren fuera de la diagonal principal estén en las columnas ik-ésimas, para $i = 1, 2, \ldots, p$ (esto es, la matriz A tiene una forma similar a la de la figura (4.4). Ahora, el sistema (4.9) tiene la forma

$$\begin{bmatrix} a_1 & g_1 & & & & \\ f_k & a_k & g_k & & & \\ & f_{2k} & a_{2k} & g_{2k} & & \\ & & \ddots & \ddots & \ddots & \\ & & & f_{(p-1)k} & a_{(p-1)k} & g_{(p-1)k} \\ & & & & f_{pk} & a_{pk} \end{bmatrix} \begin{bmatrix} x_1 \\ x_k \\ x_{2k} \\ \vdots \\ x_{(p-1)k} \\ x_{pk} \end{bmatrix} \begin{bmatrix} d_1 \\ d_k \\ d_{2k} \\ \vdots \\ d_{(p-1)k} \\ d_{pk} \end{bmatrix}. \tag{4.11}$$

El procesador P_0 tiene el subsistema

$$\left.\begin{array}{ccccccccc} a_1x_1 & & & & + & g_1x_k & & = & d_1 \\ & a_2x_2 & & & + & g_2x_k & & = & d_2 \\ & & \ddots & & \vdots & & & \vdots & \\ & & & a_{k-1}x_{k-1} & + & g_{k-1}x_k & & = & d_{k-1} \\ & & & & & a_kx_k & + \; g_kx_{2k} & = & d_k \end{array}\right\},$$

los valores de x_k y x_{2k} son obtenidos al resolver el sistema auxiliar (4.9) por lo que únicamente debe calcular x_j, para $j = 1, 2, \ldots, k-1$, de manera que debe resolver el

sistema

$$\left.\begin{array}{rrrrrrrrr} a_1x_1 & & & & & + & g_1x_k & = & d_1 \\ & a_2x_2 & & & & + & g_2x_k & = & d_2 \\ & & \ddots & & & \vdots & & \vdots & \\ & & & a_{k-1}x_{k-1} & + & g_{k-1}x_k & = & d_{k-1} \end{array}\right\}.$$

Despejando se tiene

$$x_j = \frac{1}{a_j}\left(d_j - g_j x_k\right), \quad \text{para} \quad j = 1, 2, \ldots, k-1.$$

El procesador P_i, para $i = 1, 2, \ldots, p-2$, tiene el subsistema

$$\left.\begin{array}{rcrcrcrcrcrcl} f_{ik+1}x_{ik} & + & a_{ik+1}x_{ik+1} & & & & & & & + & g_{ik+1}x_{(i+1)k} & = & d_{ik+1} \\ f_{ik+2}x_{ik} & + & & & a_{ik+2}x_{ik+2} & & & & & + & g_{ik+2}x_{(i+1)k} & = & d_{ik+2} \\ & \vdots & & & & & \ddots & & & \vdots & & \vdots & \\ f_{(i+1)k-1}x_{ik} & + & & & & & & & a_{(i+1)k-1}x_{(i+1)k-1} & + & g_{(i+1)k-1}x_{(i+1)k} & = & d_{(i+1)k-1} \\ f_{(i+1)k}x_{ik} & + & & & & & & & a_{(i+1)k}x_{(i+1)k} & + & g_{(i+1)k}x_{(i+2)k} & = & d_{(i+1)k} \end{array}\right\},$$

los valores de x_{ik}, $x_{(i+1)k}$ y $x_{(i+2)k}$ son obtenidos al resolver el sistema auxiliar (4.9) por lo que únicamente debe calcular x_{ik+j}, para $j = 1, 2, \ldots, k-1$, de manera que debe resolver el sistema

$$\left.\begin{array}{rcrcrcrcrcrcl} f_{ik+1}x_{ik} & + & a_{ik+1}x_{ik+1} & & & & & & & + & g_{ik+1}x_{(i+1)k} & = & d_{ik+1} \\ f_{ik+2}x_{ik} & + & & & a_{ik+2}x_{ik+2} & & & & & + & g_{ik+2}x_{(i+1)k} & = & d_{ik+2} \\ & \vdots & & & & & \ddots & & & \vdots & & \vdots & \\ f_{(i+1)k-1}x_{ik} & + & & & & & & & a_{(i+1)k-1}x_{(i+1)k-1} & + & g_{(i+1)k-1}x_{(i+1)k} & = & d_{(i+1)k-1} \end{array}\right\}.$$

Despejando se tiene

$$x_{ik+j} = \frac{1}{a_{ik+j}}\left(d_{ik+j} - g_{ik+j}x_{(i+1)k} - f_{ik+j}x_{ik}\right), \quad \text{para} \quad j = 1, 2, \ldots, k-1.$$

El procesador P_{p-1} tiene el subsistema

$$\left.\begin{array}{rcrcrcrcrcrcl} f_{n-k+1}x_{n-k} & + & a_{n-k+1}x_{n-k+1} & & & & & & & + & g_{n-k+1}x_n & = & d_{n-k+1} \\ f_{n-k+2}x_{n-k} & + & & & a_{n-k+2}x_{n-k+2} & & & & & + & g_{n-k+2}x_n & = & d_{n-k+2} \\ & \vdots & & & & & \ddots & & & \vdots & & \vdots & \\ f_{n-1}x_{n-k} & + & & & & & & & a_{n-1}x_{n-1} & + & g_{n-1}x_n & = & d_{n-1} \\ f_nx_{n-k} & + & & & & & & & & & a_nx_n & = & d_n \end{array}\right\},$$

los valores de x_{n-k} y x_n son obtenidos al resolver el sistema auxiliar (4.9) por lo que únicamente debe calcular x_{n-k+j}, para $j = 1, 2, \ldots, k-1$, de manera que debe resolver el sistema

$$\left.\begin{array}{lcccccccccc}
f_{n-k+1}x_{n-k} & + & a_{n-k+1}x_{n-k+1} & & & & & & + & g_{n-k+1}x_n & = & d_{n-k+1} \\
f_{n-k+2}x_{n-k} & + & & a_{n-k+2}x_{n-k+2} & & & & & + & g_{n-k+2}x_n & = & d_{n-k+2} \\
 & \vdots & & & \ddots & & & & \vdots & & \vdots & \\
f_{n-1}x_{n-k} & + & & & & & a_{n-1}x_{n-1} & + & g_{n-1}x_n & = & d_{n-1}
\end{array}\right\},$$

Despejando se tiene

$$x_{n-k+j} = \frac{1}{a_{n-k+j}}\left(d_{n-k+j} - g_{n-k+j}x_n - f_{n-k+j}x_{n-k}\right), \quad \text{para} \quad j = 1, 2, \ldots, k-1.$$

Si se supone $f_1 = f_2 = \cdots = f_k = 0$, se puede escribir, para $i = 0, 1, \ldots, p-1$,

$$x_{ik+j} = \frac{1}{a_{ik+j}}\left(d_{ik+j} - g_{ik+j}x_{(i+1)k} - f_{ik+j}x_{ik}\right), \quad \text{con} \quad j = 1, 2, \ldots, k-1. \tag{4.12}$$

Nótese que, utilizando (4.12) se puede pasar de la matriz en la forma de la figura (4.4) a la obtención de la solución en cada procesador. Para ello, es necesario un paso de comunicación en el que cada procesador envía a los demás los elementos necesarios para que todos puedan construir el sistema auxiliar (4.9), esta comunicación debe realizarse justo después de la eliminación de los elementos b_{ik}, para $i = 1, 2, \ldots, p-1$, lo que conlleva la actualización de los elementos a_{ik}, d_{ik} y, por otra parte, que se generen los elementos g_{ik}.

4.3 Algoritmos BSP

El siguiente algoritmo BSP recoge las características del método expuestas en la sección 4.2.

Algoritmo 4.2 Un algoritmo BSP para resolución de sistemas tridiagonales basado en el método de Wang.

Superpaso 1

El procesador P_0 envía al procesador P_i, para $i = 1, 2, \ldots, p-1$, los elementos c_{ik+j}, a_{ik+j}, b_{ik+j} y d_{ik+j}, con $j = 1, 2, \ldots, k$, excepto $b_n = 0$.

Superpaso 2

- Para $i = 0, 1, \ldots, p-1$,
  - El procesador P_i anula los elementos c_{ik+j}, para $j = 2, 3, \ldots, k$, utilizando la expresión (4.6).
  - El procesador P_i anula los elementos $b_{(i+1)k-j}$, $j = 2, 3, \ldots, k-1$, utilizando la expresión (4.7).

- Para $i = 1, 2, \ldots, p-1$, el procesador P_{i-1} envía al procesador P_i los elementos a_{ik}, b_{ik} y d_{ik}.

Superpaso 3

- Para $i = 1, 2, \ldots, p-1$,
  - El procesador P_{i-1} anula el elemento b_{ik}.
  - El procesador P_i actualiza los elementos a_{ik}, d_{ik} y calcula el elemento g_{ik} por medio de (4.8).

- El procesador P_0 comunica al resto de procesadores los elementos a_1, g_1 y d_1.
- Para $i = 1, 2, \ldots, p-2$, el procesador P_i comunica al resto de procesadores los elementos a_{ik}, g_{ik}, d_{ik} y $f_{(i+1)k}$.
- El procesador P_{p-1}, comunica al resto de procesadores los elementos $a_{(p-1)k}$, $g_{(p-1)k}$, $d_{(p-1)k}$, f_{pk} y a_{pk}.

Superpaso 4

- Para $i = 0, 1, \ldots, p-1$,
  - P_i construye el sistema auxiliar (4.9), utilizando la expresión (4.11).
  - P_i resuelve el sistema auxiliar (4.9) por el método de Gauss.
  - P_i calcula las componentes de la solución que le corresponden utilizando la expresión (4.12).
- Para $i = 1, 2, \ldots, p-1$, P_i envía a P_0 su solución parcial.

La resolución del sistema auxiliar en cada procesador es una fase fundamental del algoritmo 4.2 ya que permite la obtención de la solución del sistema (4.2) en paralelo. Sin embargo, presenta el inconveniente de que para un número de procesadores alto el orden del sistema auxiliar (4.9) es grande y, además, debe resolverse el mismo sistema en todos los procesadores al mismo tiempo, lo que supone una pérdida de eficiencia en el método. Por esta razón, es interesante encontrar una alternativa a la resolución del sistema (4.9) que permita la eliminación en paralelo de todos los elementos exteriores a la diagonal principal, especialmente en los casos en que el número de procesadores es alto.

Se supone que se han realizado las anulaciones de elementos de la matriz de coeficientes oportunas hasta obtener una matriz triangular superior con forma similar a la de la figura 4.5, esto es, falta aplicar el paso 5 del algoritmo 4.1. Para paralelizar el paso 5, se realiza una modificación sobre el algoritmo 4.1 consistente en la comunicación de cada procesador P_i, para $i = 2, 3, \ldots, p$, a los procesadores P_j, con $j = 1, 2, \ldots, i-1$, de los elementos g_{ik}, $a_{(i+1)k}$ y $d_{(i+1)k}$ y así poder proceder a anular simultáneamente en todos los procesadores los elementos superdiagonales.

El procesador P_i, para $i = 0, 1, \ldots, p-2$, actualiza el elemento $d_{(i+1)k}$ a partir de la relación de recurrencia

$$\left.\begin{array}{rcl} r_p & = & d_n, \\ r_t & = & d_{tk} - \dfrac{g_{tk}}{a_{(t+1)k}} r_{t+1}, \end{array}\right\} \quad \text{para } t = p-1, p-2, \ldots, 2, \tag{4.13}$$

y de los elementos recibidos de los procesadores P_j, para $j = i+1, i+2, \ldots, p-1$, mediante la siguiente operación elemental

$$d_{(i+1)k} \leftarrow d_{(i+1)k} - \frac{g_{(i+1)k}}{a_{(i+2)k}} r_{i+2}. \tag{4.14}$$

Una vez actualizado este valor, elimina los elementos no nulos de cada bloque diagonal y actualiza los correspondientes valores d_{ik+j}, para $j = 1, 2, \ldots, p-1$.

Esta estrategia evita la pérdida de paralelismo en los cálculos finales, conducentes a la diagonalización de la matriz, y ahorra un paso de comunicación al procesador principal.

El algoritmo 4.3 implementa, según el modelo BSP, el método de las particiones de Wang introduciendo la modificación que sugieren las expresiones (4.13) y (4.14).

Algoritmo 4.3 Algoritmo de Wang modificado.

Superpaso 1

Al igual que en el superpaso 1 del algoritmo 4.2, el procesador P_0 envía al procesador P_i, para $i = 1, 2, \ldots, p-1$, los elementos c_{ik+j}, a_{ik+j}, b_{ik+j} y d_{ik+j}, con $j = 1, 2, \ldots, k$, excepto $b_n = 0$.

Superpaso 2

Como en el superpaso 2 del algoritmo 4.2:

- Para $i = 0, 1, \ldots, p-1$,
  - El procesador P_i anula los elementos c_{ik+j}, para $j = 2, 3, \ldots, k$, utilizando (4.6).
  - El procesador P_i anula los elementos $b_{(i+1)k-j}$, para $j = 2, 3, \ldots, k-1$, utilizando (4.7).

- Para $i = 1, 2, \ldots, p-1$, el procesador P_{i-1} envía al procesador P_i los elementos a_{ik}, b_{ik} y d_{ik}.

Superpaso 3

Para $i = 1, 2, \ldots, p-1$,

- El procesador P_{i-1} anula el elemento b_{ik}.
- El procesador P_i actualiza los elementos a_{ik}, d_{ik} y calcula los elementos g_{ik}, por medio de la expresión (4.8).
- El procesador P_i comunica al procesador P_{i-1} los elementos a_{ik}, d_{ik} ya actualizados.

Superpaso 4

Para $i = 1, 2, \ldots, p-1$,

- El procesador P_i anula los elementos f_{ik+j} para $j = 1, 2, \ldots, k$.
- El procesador P_i comunica al procesador P_j, siendo $j < i$, los elementos g_{ik}, $a_{(i+1)k}$ y $d_{(i+1)k}$.

Superpaso 5

- Para $i = 0, 1, \ldots, p-2$,
  - El procesador P_i anula el elemento $g_{(i+1)k}$ y actualiza el elemento $d_{(i+1)k}$ de acuerdo con (4.13) y (4.14).
  - El procesador P_i anula los elementos g_{ik+j} y actualiza los elementos d_{ik+j}, para $j = 1, 2, \ldots, k-1$.
- Para $i = 0, 1, \ldots, p-1$, el procesador P_i calcula

$$x_{ik+j} = \frac{d_{ik+j}}{a_{ik+j}}, \quad \text{con} \quad j = 1, 2, \ldots, k.$$

- Para $i = 1, 2, \ldots, p-1$, el procesador P_i envía su solución parcial al procesador principal P_0.

4.4 Coste computacional de los algoritmos basados en el método de Wang

El coste computacional del algoritmo 4.2 se obtiene sumando los costes individuales de cada uno de sus superpasos.

Coste del superpaso 1. En este superpaso sólo existe comunicación de elementos desde el procesador principal al resto, cada procesador recibe $4k$ elementos de P_0 por lo que el número de elementos comunicados es $4k(p-1)-1$. En consecuencia, el coste del superpaso es

$$(4n-4k-1)g+l. \tag{4.15}$$

Coste del superpaso 2. En cada procesador P_i, para $i=0,1,\ldots,p-1$, el número de operaciones necesarias para anular los elementos c_{ik+j} y actualizar los elementos d_{ik+j}, a_{ik+j} y f_{ik+j}, con $j=2,3,\ldots,k$, es $6(k-1)$; para anular los elementos $b_{(i+1)k-j}$ y actualizar los elementos $g_{(i+1)k-j}$, $d_{(i+1)k-j}$ y $f_{(i+1)k-j}$, con $j=2,3,\ldots,k-1$, se requieren $6(k-2)$ operaciones (véanse las expresiones (4.6) y (4.7)). En consecuencia, el coste aritmético del superpaso es $12k-18$.

Cada procesador P_i, para $i=1,2,\ldots,p-1$, recibe tres elementos del procesador P_{i-1}, lo que supone un coste de comunicación $3g$. El coste total del superpaso es por tanto

$$12k-18+3g+l. \tag{4.16}$$

Coste del superpaso 3. La eliminación de los elementos b_{ik}, para $i=1,2,\ldots,p-1$, supone un total de 6 operaciones en cada uno de los procesadores activos, que corresponden a la actualización de los elementos a_{ik}, d_{ik} y g_{ik}.

En cuanto al coste de comunicación, el procesador principal comunica 3 elementos a los demás, los procesadores centrales comunican al resto 4 elementos y el último comunica a los otros procesadores un total de 5 elementos. Por tanto, el procesador P_{p-1} es el que más elementos envía o recibe ya que envía $5(p-1)$ elementos y recibe $3+4(p-2)$ elementos. En consecuencia, el coste del superpaso 3 es

$$6+5(p-1)g+l. \tag{4.17}$$

Coste del superpaso 4. La formación del sistema auxiliar (4.9) en cada procesador no precisa ninguna operación, su resolución por el método de Gauss necesita un total de $8(p+1)-7$ operaciones, ya que el número de ecuaciones es $p+1$, y el cálculo de cada una de las componentes restantes de la solución parcial, utilizando la expresión (4.12), requiere un total de 6 operaciones por cada una de las $k-1$ componentes. De esta forma, el coste aritmético del superpaso es $8(p+1)-7+6(k-1)=8p+6k-5$.

En este superpaso, cada procesador comunica su solución parcial a P_0, por lo que el procesador principal recibe un total de $n-k$ elementos. Se tiene que el coste del superpaso 4 es

$$6k+8p-5+(n-k)g+l. \tag{4.18}$$

Sumando las expresiones (4.15), (4.16), (4.17) y (4.18) se obtiene que el coste computacional del algoritmo 4.2 es

$$18k+8p-17+(5n-5k+5p-3)g+4l.$$

A continuación se obtiene el coste computacional del algoritmo 4.3.

Coste del superpaso 1. El coste computacional de este superpaso es el mismo que el del superpaso 1 del algoritmo 4.2, que está dado por la expresión (4.15).

Coste del superpaso 2. El coste es el mismo que el del superpaso 2 del algoritmo 4.2, que está dado por la expresión (4.16).

Coste del superpaso 3. En este superpaso, para $i=1,2,\ldots,p-1$, el procesador P_{i-1} anula el elemento b_{ik} y el procesador P_i actualiza los elementos a_{ik}, d_{ik}, g_{ik} para lo que necesita realizar 6 operaciones. Además, P_i comunica los elementos a_{ik}, d_{ik}, al procesador P_{i-1}, lo que supone un coste de comunicación $2g$

El coste total del superpaso 3 es por tanto

$$6+2g+l. \tag{4.19}$$

Coste del superpaso 4. Para eliminar en cada procesador P_i, para $i=1,2,\ldots,p-1$, los elementos f_{ik+j}, con $j=1,2,\ldots,k$, se necesitan $5k$ operaciones aritméticas. El

coste de comunicación de los elementos g_{ik}, $a_{(i+1)k}$ y $d_{(i+1)k}$ desde el procesador P_i, para $i = 1, 2, \ldots, p-1$, a los procesadores P_j, siendo $j < i$, es $3(p-1)g$ ya que el procesador que más elementos recibe es P_0 (recibe $3(p-1)$ elementos) y el procesador que más envía es P_{p-1} (envía $3(p-1)$ elementos). En resumen, el coste del superpaso 4 es

$$5k + (3p-3)g + l. \tag{4.20}$$

Coste del superpaso 5. En este superpaso se distinguen varias fases en el cálculo aritmético. En primer lugar, y en todas las fases para $i = 0, 1, \ldots, p-2$, se anulan los elementos $g_{(i+1)k}$ lo que conlleva la actualización de los elementos $d_{(i+1)k}$; el procesador que más operaciones realiza es P_0 pues debe efectuar $p-1$ divisiones, $p-1$ productos y $p-1$ sumas para actualizar el elemento d_k. En segundo lugar, la anulación de los elementos g_{ik+j}, con $j = 1, 2, \ldots, k-1$, supone $3(k-1)$ operaciones. Finalmente, el cálculo de las componentes x_{ik+j}, con $j = 1, 2, \ldots, k$, de la solución parcial en cada procesador requiere k divisiones. En consecuencia, el coste aritmético total es $3(p-1)+3(k-1)+k = 4k+3p-6$ operaciones.

Con respecto al coste de comunicación, cada procesador envía al procesador principal su vector de soluciones parciales, esto es $k(p-1)$ elementos.

El coste total del superpaso 5 es por tanto

$$(4k + 3p - 6) + (n-k)g + l. \tag{4.21}$$

Sumando las expresiones (4.15), (4.16), (4.19), (4.20) y (4.21) se obtiene que el coste total del algoritmo es

$$21k + 3p - 18 + (5n - 5k + 3p + 1)g + 5l.$$

Nótese que la implementación del algoritmo 4.2 requiere un superpaso menos que el del algoritmo 4.3, que el coste de comunicación del algoritmo 4.2 es mayor que el del algoritmo 4.3 para $p > 2$ y que para p alto y k no muy grande, el coste aritmético del primer algoritmo es mayor que en el algoritmo 4.3 (por ejemplo, para $p = 256$ y $n = 65536$ el algoritmo 4.2 realiza 513 operaciones más que el segundo algoritmo).

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	26
	2	3294	9.5	25
	4	5366	12.4	25
	6	8164	12.5	25

(a) *IBM SP2 switch*

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	8
	2	20235	709.7	3
	4	54163	1362.6	9
	6	121958	3211.2	9

(b) *IBM SP2 ethernet*

s	p	l	g	$n_{\frac{1}{2}}$
16.4	1	23	0.2	22
	2	2556	6.9	5
	4	5152	7.4	4
	6	7538	6.8	4

(c) *Cluster de PC's*

Tabla 4.1: *Valores de parámetros BSP.*

4.5 Resultados numéricos

En esta sección se analizan los tiempos previstos teóricamente y los tiempos experimentales de los algoritmos 4.2 y 4.3 (a los que se referenciará en las tablas y figuras como WANG1 y WANG2 respectivamente). Las pruebas experimentales se han realizado en el IBM SP2 y el *cluster* de PC's cuyas características se han descrito en la subsección 1.5.3. Por comodidad, en la tabla 4.1 se repiten los parámetros obtenidos para esas máquinas que se muestran en la tabla 1.1.

El tiempo teórico se ha obtenido considerando que el tamaño de bloque de los mensajes que se comunican entre los procesadores es $k = \frac{n}{p}$ y que el coste de comunicación de una

palabra de 32 bits es

$$g(k) = \left(\frac{n_{\frac{1}{2}}}{k} + 1 \right) g_\infty.$$

Los algoritmos 4.2 (WANG1) y 4.3 (WANG2) han sido implementados en Fortran usando la versión v1.3 de la librería BSPLib, se ha generando el sistema (4.2) obteniendo aleatoriamente los elementos de la matriz de coeficientes A y, para simplificar, eligiendo el vector de términos independientes $\boldsymbol{d}$ de manera que la solución sea $\boldsymbol{x} = [1, 1, \ldots, 1]^T$.

En las tablas 4.2, 4.3, 4.4 y figuras 4.8, 4.9, 4.10 se muestran los tiempos teóricos y experimentales, medidos en segundos, de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en el IBM SP2 utilizando *switch*.

En las tablas 4.5, 4.6, 4.7 y figuras 4.11, 4.12, 4.13 se muestran los tiempos teóricos y experimentales, medidos en segundos, de los algoritmos 4.2 (WANG1) y 4.3 (WANG1) en el IBM SP2 utilizando *ethernet*.

En las tablas 4.8, 4.9, 4.10 y figuras 4.14, 4.15, 4.16 se muestran los tiempos teóricos y experimentales, medidos en segundos, de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en el *cluster* de PC's.

Los tiempos han sido obtenidos para diferentes tamaños de la matriz de coeficientes, en el IBM SP2 el tamaño varía desde 128 a 524288 para 2 y 4 procesadores y desde 126 a 516096 para 6 procesadores; en el *cluster* de PC's el tamaño de la matriz de coeficientes varía desde 128 a 65536 para 2 y 4 procesadores y desde 126 a 64512 para 6 procesadores.

En las figuras 4.17, 4.18 y 4.19 se muestra el porcentaje de desviación del tiempo experimental con respecto al teórico en función del tamaño n del sistema [1].

Las mayores diferencias entre el tiempo previsto y el experimental se obtienen en los sistemas de tamaño pequeño en los que, por lo general, suele ser mayor el tiempo experimental que el teórico. A medida que aumenta el tamaño de sistema el tiempo obtenido experimentalmente se ajusta mejor al esperado, para tamaños grandes la desviación (salvo alguna excepción) oscila entre el 0% y el 5%.

[1] Se considera que si el tiempo teórico es 100 y el tiempo experimental es 90, se ha producido un 10% de desviación y en cambio si el tiempo experimental es 110 se ha producido un -10% de desviación.

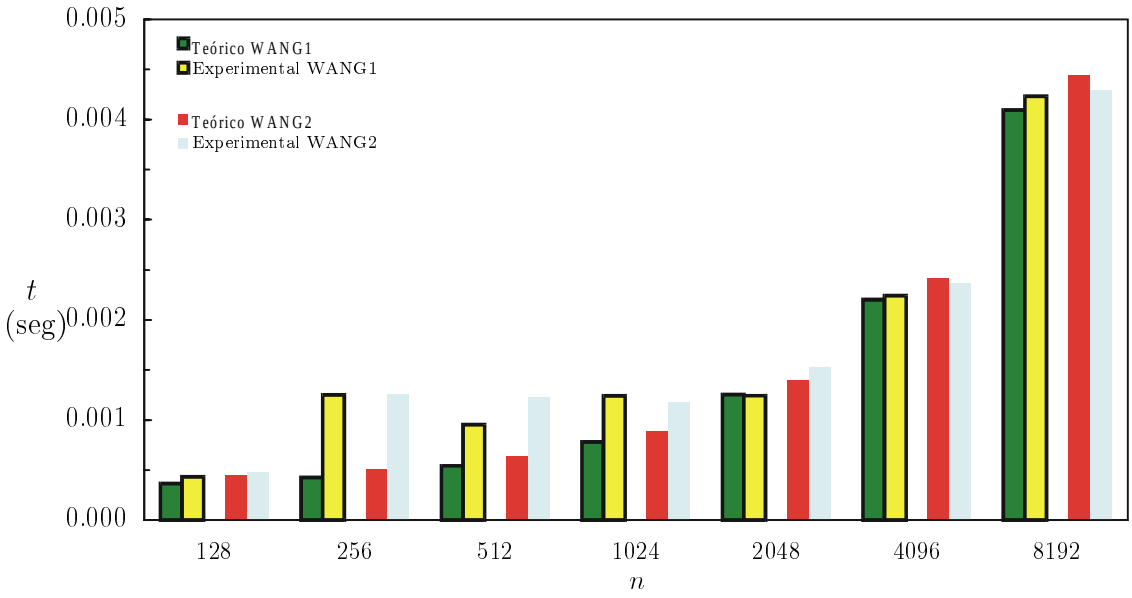
Para las máquinas en las que se han obtenido resultados experimentales, el algoritmo 4.2 (WANG1) es siempre más rápido que el algoritmo 4.3 (WANG2). Esto no sucede cuando el número de procesadores es grande, por ejemplo en un CRAY T3D para $p = 256$ es más rápido el algoritmo 4.3 (WANG2) hasta tamaños de sistema de unas 65000 ecuaciones, como se muestra en la tabla 4.11. Los tiempos se han obtenido en base a los parámeros que se muestran en la tabla 4.13(a)

En la tabla 4.12 se muestra el *speed-up* y la eficiencia de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) con respecto al método de eliminación de Gauss para sistemas tridiagonales, que es el mejor algoritmo secuencial para la resolución de sistemas tridiagonales, considerando los tiempos teóricos de los sistemas de tamaño máximo de los que se han obtenido resultados experimentales en cada una de las máquinas.

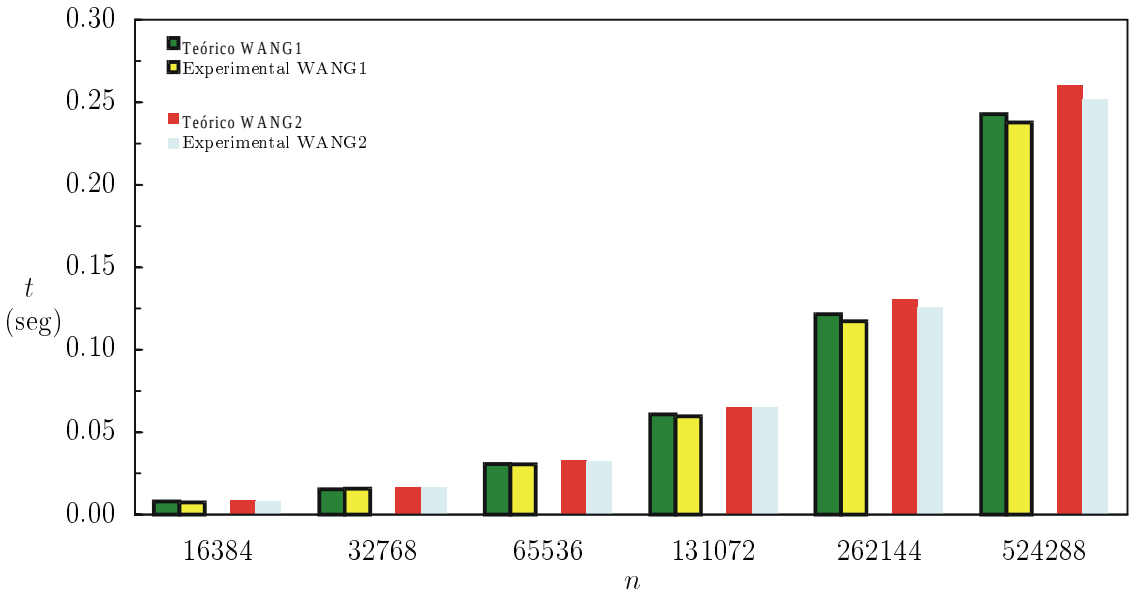
La eficiencia que se obtiene no es muy buena, y en caso del IBM SP2 con *ethernet* es pésima; sin duda un factor determinante es el valor de g. Para otras máquinas con menor valor de g se obtiene mejor eficiencia, como es el caso de un CRAY T3D o un CRAY T3E. Por comodidad, en la tabla 4.13 se repiten los valores de los parámetros de estas máquinas que se muestran en la tabla 2.16, para las mismas se obtiene el *speed-up* y la eficiencia que figura en la tabla 4.14 considerando el mismo tamaño de sistema que para la tabla 4.12.

IBM SP2 2 procesadores *switch*				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0004	0.0004	0.0004	0.0005
256	0.0004	0.0013	0.0005	0.0013
512	0.0005	0.0010	0.0006	0.0012
1024	0.0008	0.0012	0.0009	0.0012
2048	0.0013	0.0012	0.0014	0.0015
4096	0.0022	0.0022	0.0024	0.0024
8192	0.0041	0.0042	0.0044	0.0043
16384	0.0079	0.0076	0.0085	0.0083
32768	0.0155	0.0157	0.0166	0.0165
65536	0.0306	0.0305	0.0329	0.0323
131072	0.0609	0.0598	0.0654	0.0654
262144	0.1215	0.1172	0.1304	0.1257
524288	0.2428	0.2379	0.2603	0.2520

Tabla 4.2: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con 2 procesadores interconectados mediante switch, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

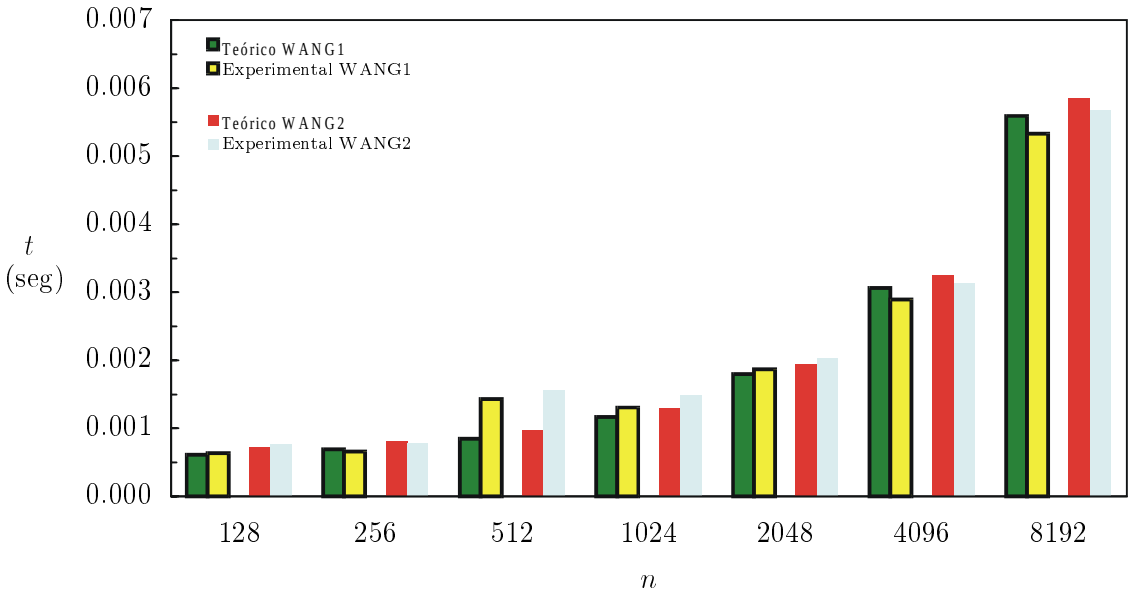


(b) $16384 \leq n \leq 524288$

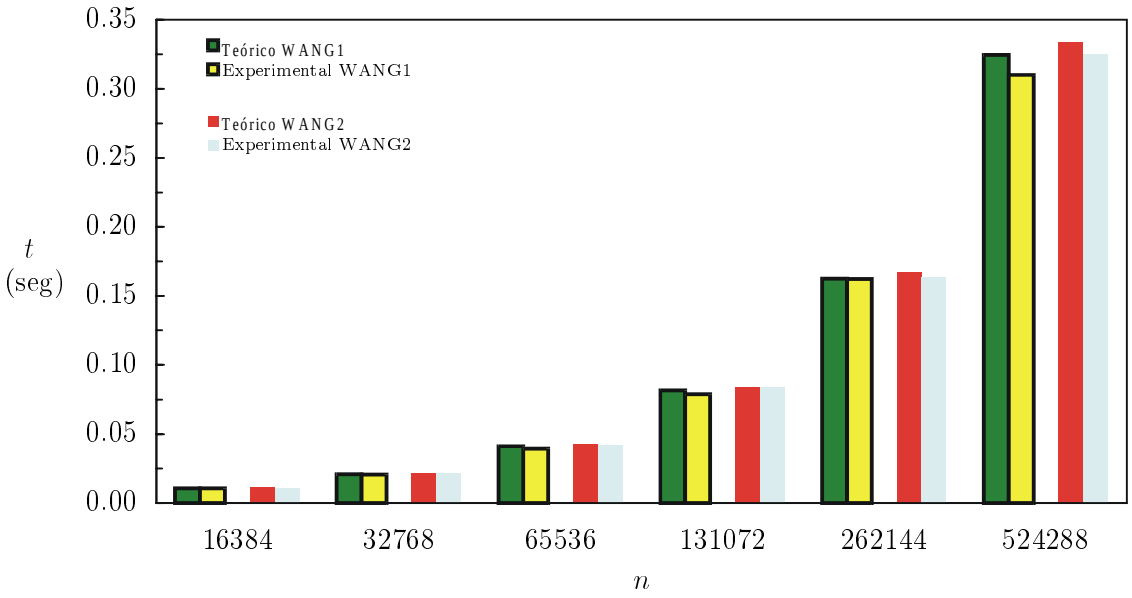
Figura 4.8: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 2 procesadores interconectados mediante switch.*

IBM SP2 4 procesadores *switch*				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0006	0.0006	0.0007	0.0008
256	0.0007	0.0007	0.0008	0.0008
512	0.0008	0.0014	0.0010	0.0016
1024	0.0012	0.0013	0.0013	0.0015
2048	0.0018	0.0019	0.0019	0.0020
4096	0.0031	0.0029	0.0032	0.0031
8192	0.0056	0.0053	0.0058	0.0057
16384	0.0107	0.0105	0.0110	0.0107
32768	0.0208	0.0205	0.0214	0.0213
65536	0.0410	0.0394	0.0422	0.0419
131072	0.0815	0.0787	0.0838	0.0837
262144	0.1625	0.1623	0.1670	0.1636
524288	0.3245	0.3102	0.3334	0.3250

Tabla 4.3: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con* 4 *procesadores interconectados mediante switch, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

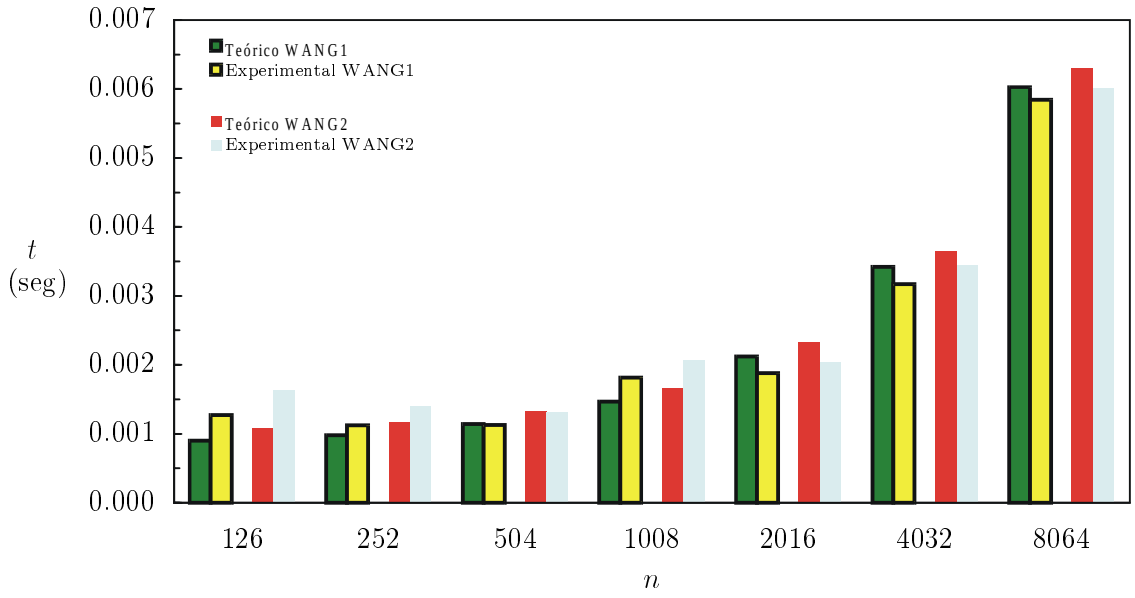


(b) $16384 \leq n \leq 524288$

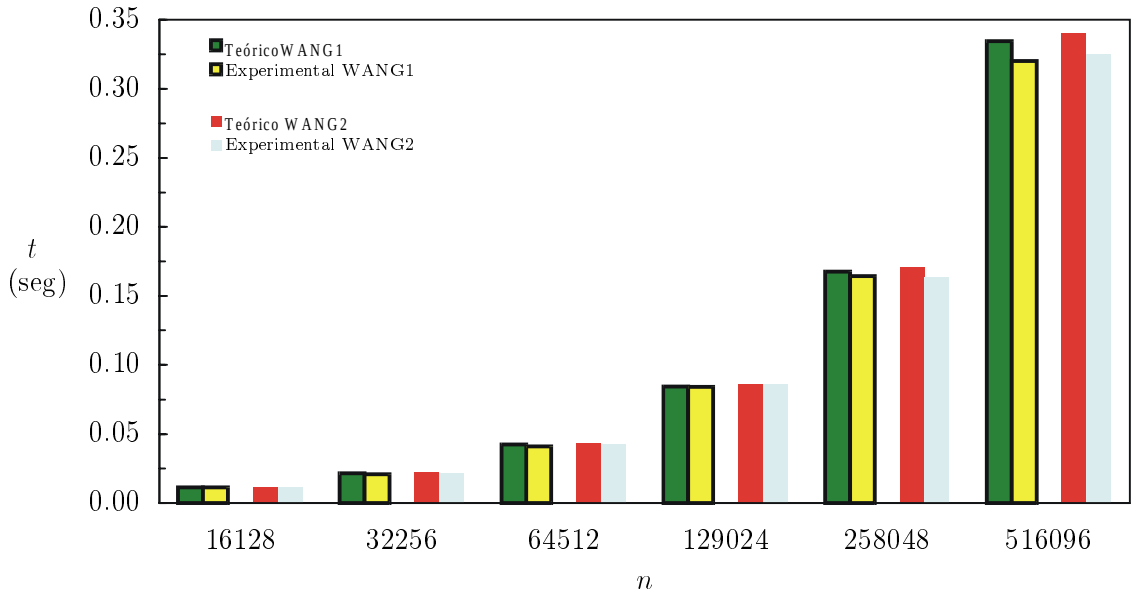
Figura 4.9: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 4 procesadores interconectados mediante switch.*

IBM SP2 6 procesadores *switch*				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0009	0.0013	0.0011	0.0016
252	0.0010	0.0011	0.0012	0.0014
504	0.0011	0.0011	0.0013	0.0013
1008	0.0015	0.0018	0.0017	0.0021
2016	0.0021	0.0019	0.0023	0.0020
4032	0.0034	0.0032	0.0036	0.0035
8064	0.0060	0.0058	0.0063	0.0060
16128	0.0112	0.0112	0.0116	0.0114
32256	0.0217	0.0210	0.0222	0.0213
64512	0.0425	0.0410	0.0434	0.0425
129024	0.0842	0.0842	0.0858	0.0858
258048	0.1677	0.1644	0.1707	0.1635
516096	0.3345	0.3203	0.3404	0.3252

Tabla 4.4: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con* 6 *procesadores interconectados mediante switch, para* $126 \leq n \leq 516096$.



(a) $126 \leq n \leq 8064$

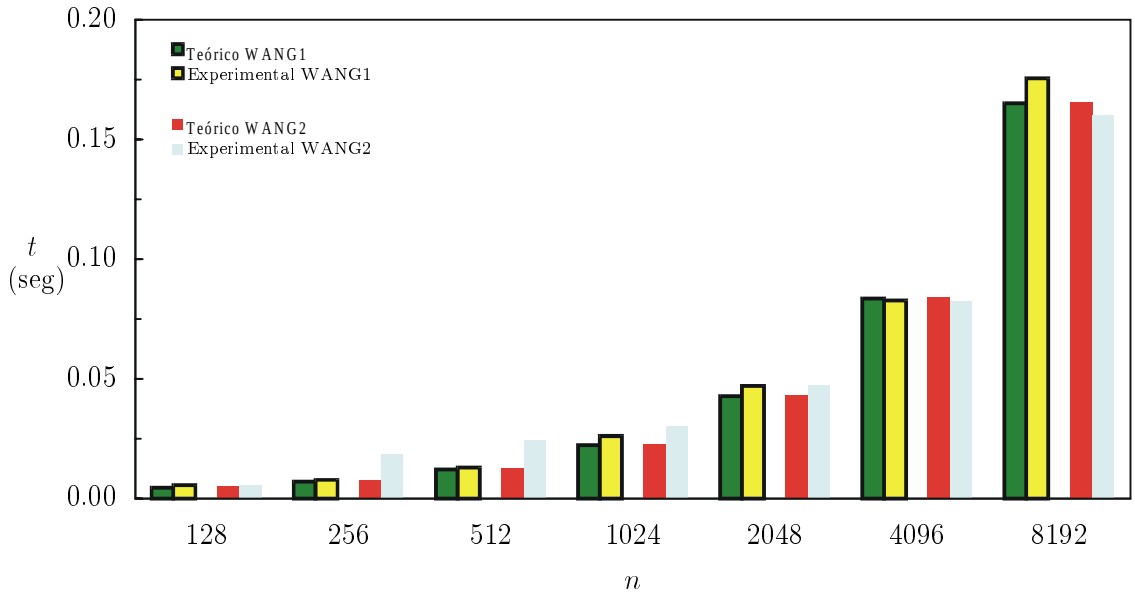


(b) $16128 \leq n \leq 516096$

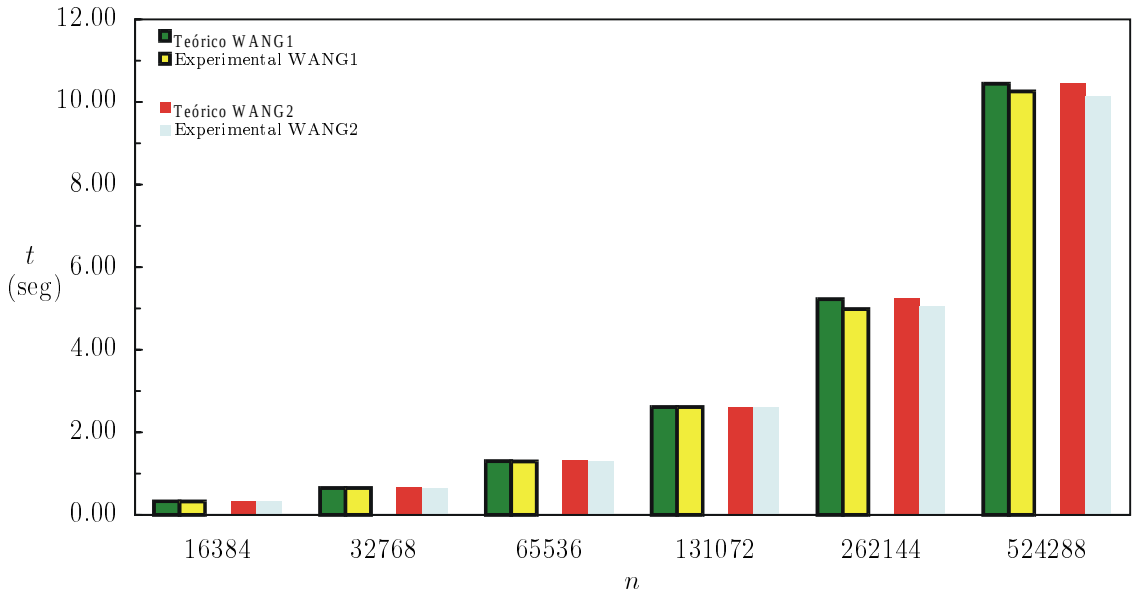
Figura 4.10: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 6 procesadores interconectados mediante switch.*

IBM SP2 2 procesadores *ethernet*				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0045	0.0056	0.0050	0.0054
256	0.0071	0.0078	0.0075	0.0186
512	0.0122	0.0130	0.0126	0.0245
1024	0.0224	0.0261	0.0228	0.0302
2048	0.0428	0.0470	0.0433	0.0474
4096	0.0835	0.0827	0.0841	0.0823
8192	0.1651	0.1756	0.1658	0.1601
16384	0.3282	0.3238	0.3292	0.3224
32768	0.6545	0.6506	0.6560	0.6520
65536	1.3070	1.2939	1.3096	1.2850
131072	2.6119	2.6161	2.6168	2.6161
262144	5.2219	4.9811	5.2311	5.0445
524288	10.4419	10.2511	10.4598	10.1243

Tabla 4.5: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con 2 procesadores interconectados mediante ethernet, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

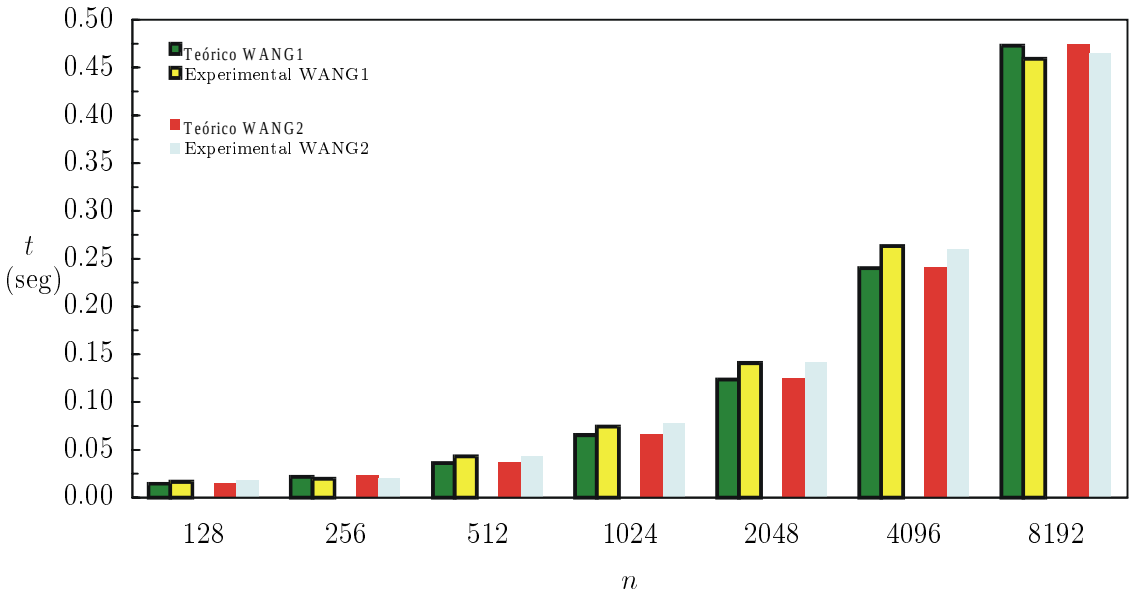


(b) $16384 \leq n \leq 524288$

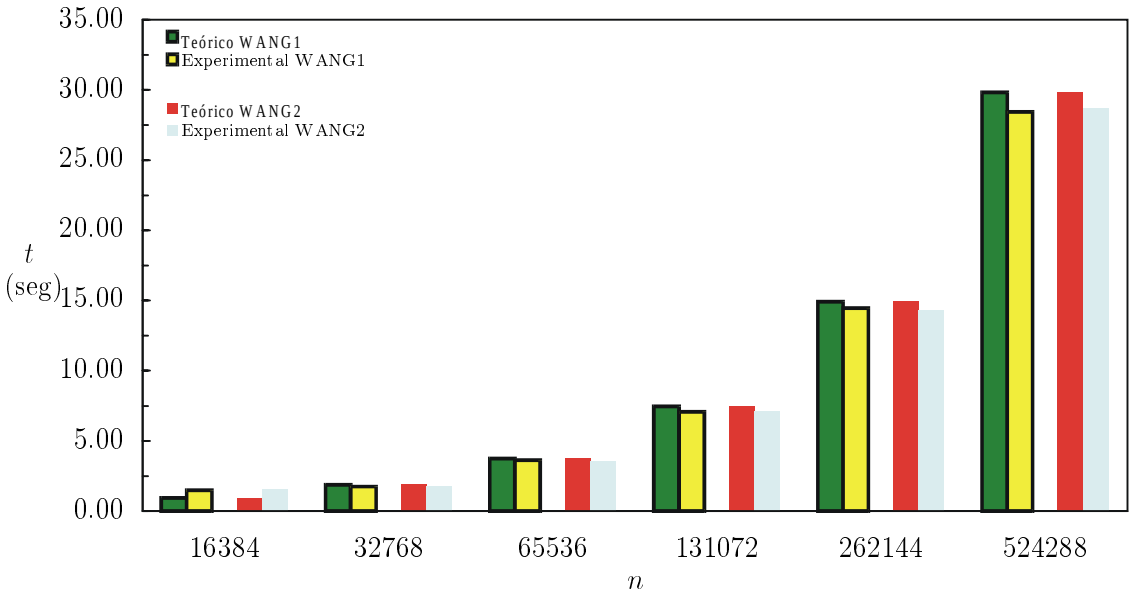
Figura 4.11: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3(WANG2) en un IBM SP2 con 2 procesadores interconectados mediante ethernet.*

IBM SP2 4 procesadores *ethernet*				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0145	0.0165	0.0156	0.0183
256	0.0217	0.0197	0.0229	0.0204
512	0.0363	0.0433	0.0374	0.0435
1024	0.0654	0.0744	0.0665	0.0782
2048	0.1236	0.1407	0.1248	0.1418
4096	0.2401	0.2634	0.2413	0.2598
8192	0.4730	0.4594	0.4743	0.4647
16384	0.9390	1.5034	0.9404	1.5297
32768	1.8708	1.7549	1.8725	1.7755
65536	3.7346	3.6297	3.7368	3.5466
131072	7.4620	7.0647	7.4653	7.1182
262144	14.9169	14.4628	14.9224	14.3265
524288	29.8266	28.4446	29.8365	28.7219

Tabla 4.6: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con* 4 *procesadores interconectados mediante ethernet, para* $128 \leq n \leq 524288$.



(a) $128 \leq n \leq 8192$

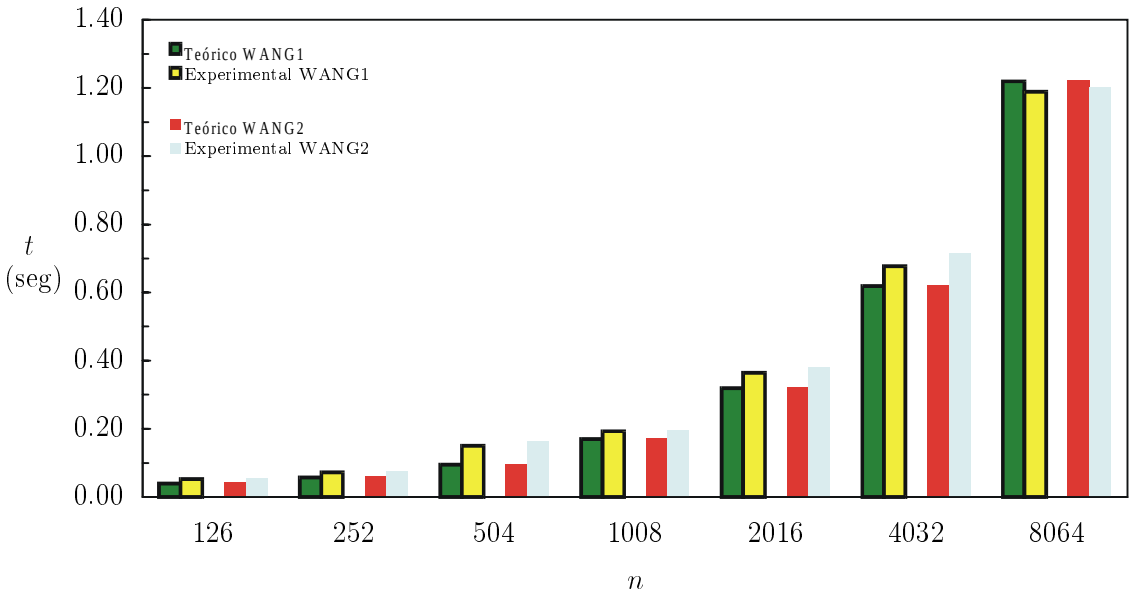


(b) $16384 \leq n \leq 524288$

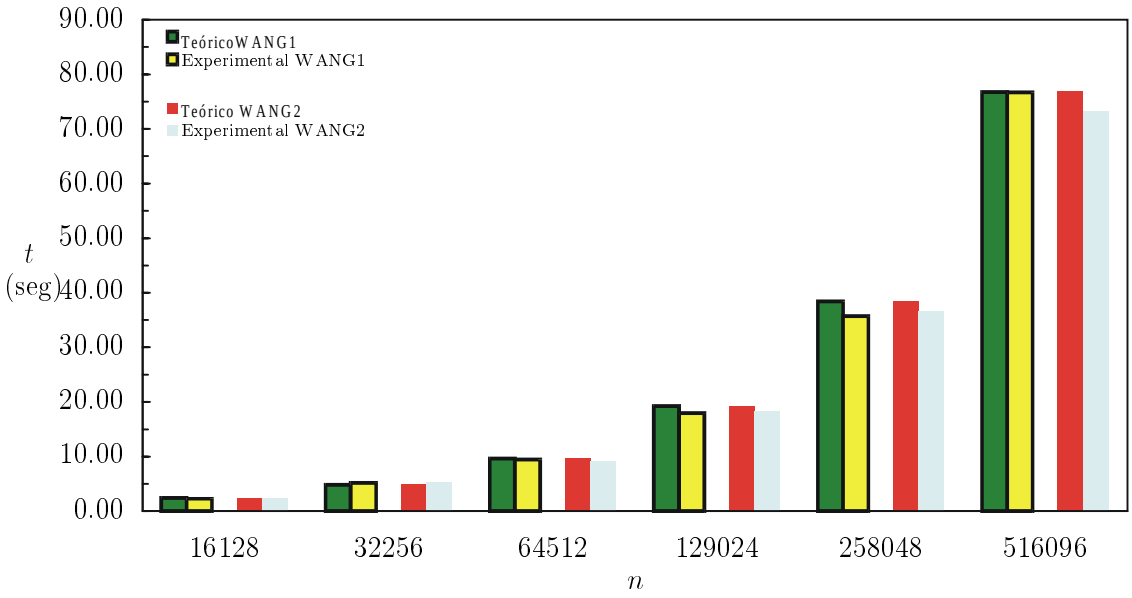
Figura 4.12: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 4 procesadores interconectados mediante ethernet.*

IBM SP2 6 procesadores *ethernet*				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0390	0.0523	0.0413	0.0541
252	0.0575	0.0729	0.0599	0.0752
504	0.0949	0.1504	0.0973	0.1631
1008	0.1698	0.1932	0.1722	0.1951
2016	0.3197	0.3647	0.3221	0.3812
4032	0.6195	0.6776	0.6220	0.7137
8064	1.2192	1.1891	1.2217	1.2026
16128	2.4186	2.2895	2.4212	2.3126
32256	4.8174	5.1402	4.8202	5.2438
64512	9.6149	9.4244	9.6181	9.1213
129024	19.2100	17.9577	19.2139	18.1939
258048	38.4002	35.7446	38.4055	36.5398
516096	76.7805	76.7245	76.7887	73.1317

Tabla 4.7: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2), medidos en un IBM SP2 con* 6 *procesadores interconectados mediante ethernet, para* $126 \leq n \leq 516096$.



(a) $126 \leq n \leq 8064$

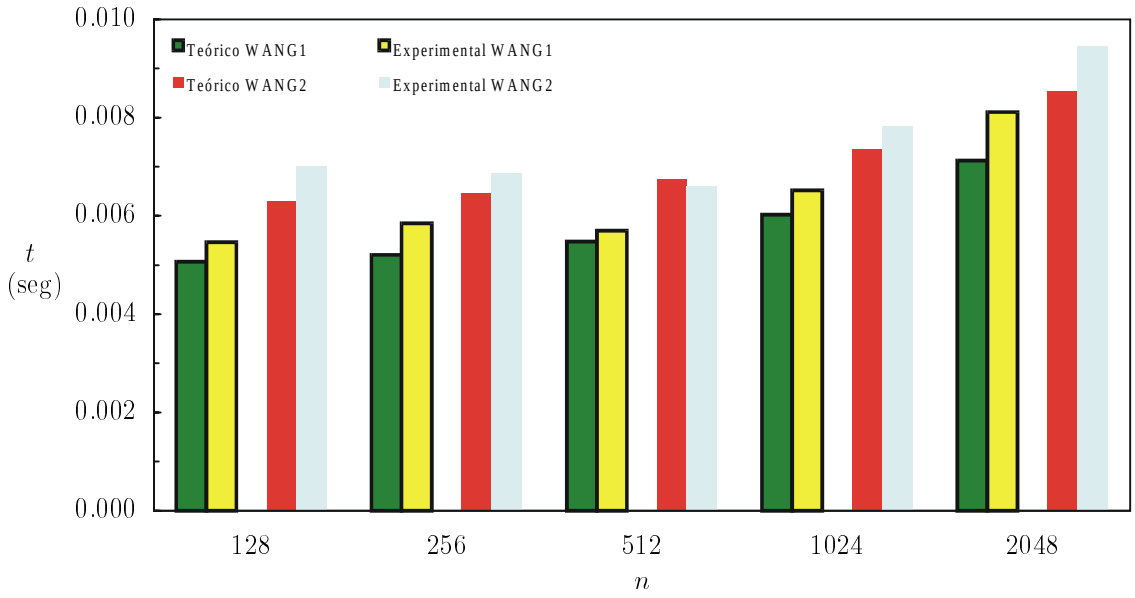


(b) $16128 \leq n \leq 516096$

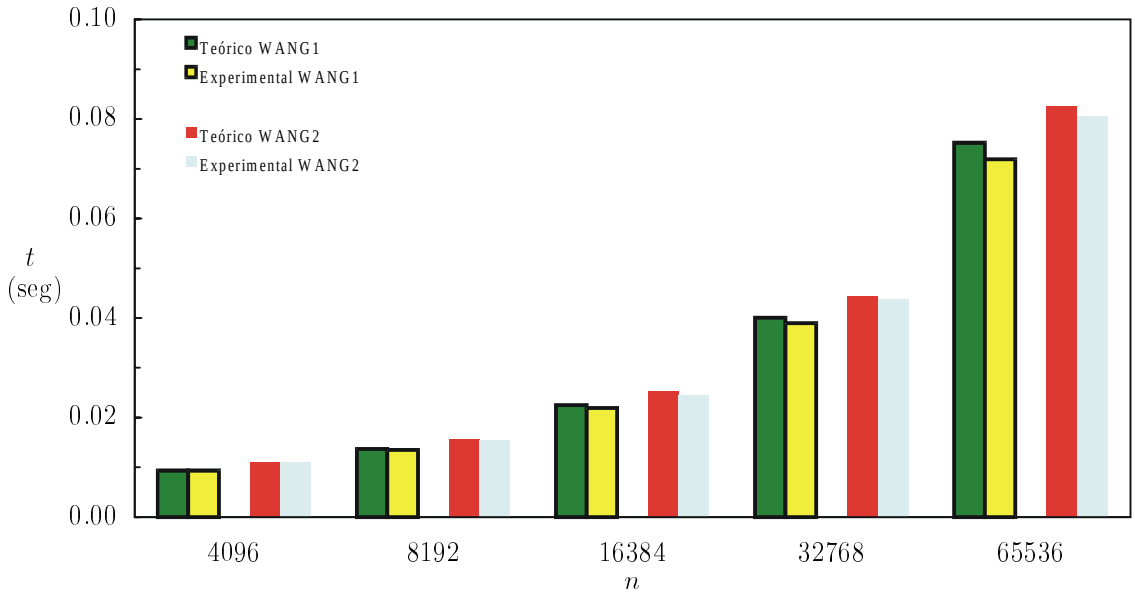
Figura 4.13: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 6 procesadores interconectados mediante ethernet.*

Cluster de PC's 2 procesadores				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0051	0.0055	0.0063	0.0070
256	0.0052	0.0058	0.0065	0.0069
512	0.0055	0.0057	0.0068	0.0066
1024	0.0060	0.0065	0.0074	0.0078
2048	0.0071	0.0081	0.0085	0.0095
4096	0.0093	0.0094	0.0109	0.0110
8192	0.0137	0.0135	0.0157	0.0154
16384	0.0225	0.0219	0.0252	0.0244
32768	0.0401	0.0390	0.0443	0.0438
65536	0.0752	0.0719	0.0824	0.0804

Tabla 4.8: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) medidos en un cluster de PC's, para 2 procesadores y* $128 \leq n \leq 65536$.



(a) $128 \leq n \leq 2048$

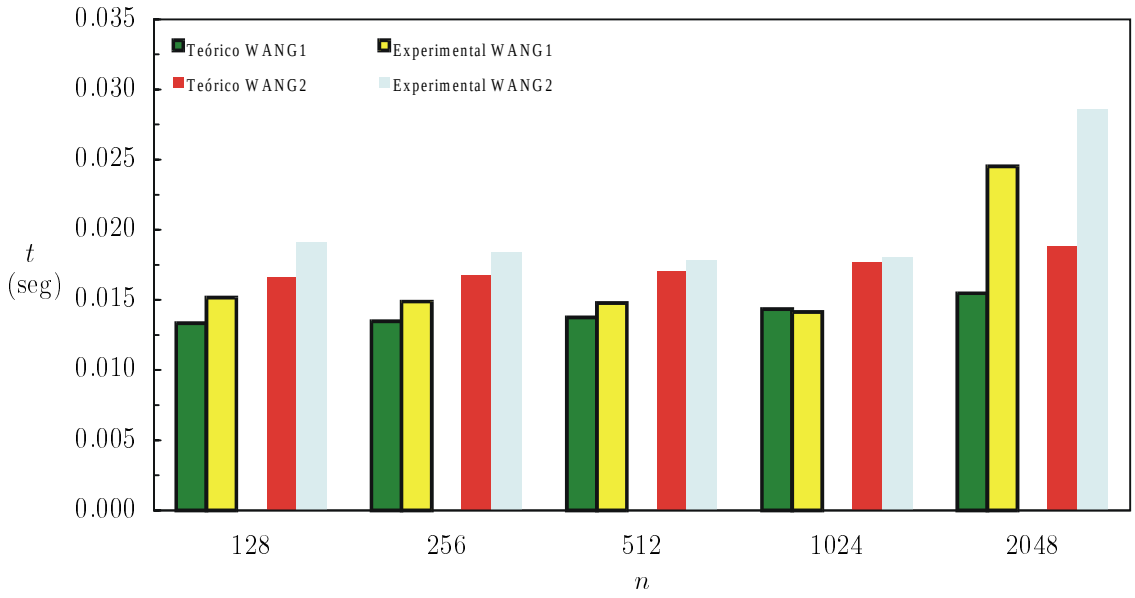


(b) $4096 \leq n \leq 65536$

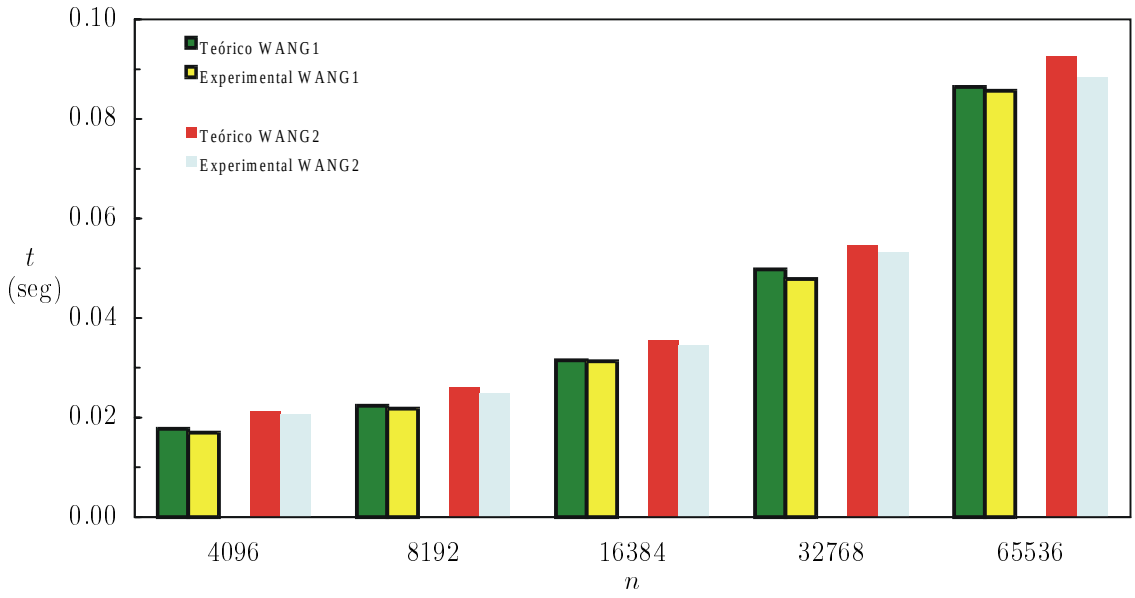
Figura 4.14: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un cluster de PC's para 2 procesadores.*

Cluster de PC's 4 procesadores				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
128	0.0133	0.0152	0.0166	0.0191
256	0.0135	0.0149	0.0168	0.0184
512	0.0138	0.0148	0.0171	0.0179
1024	0.0143	0.0141	0.0177	0.0180
2048	0.0155	0.0245	0.0189	0.0286
4096	0.0178	0.0170	0.0213	0.0207
8192	0.0224	0.0218	0.0260	0.0250
16384	0.0315	0.0313	0.0355	0.0346
32768	0.0498	0.0478	0.0546	0.0532
65536	0.0864	0.0857	0.0927	0.0884

Tabla 4.9: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) medidos en un cluster de PC's, para 4 procesadores y* $128 \leq n \leq 65536$.



(a) $128 \leq n \leq 2048$

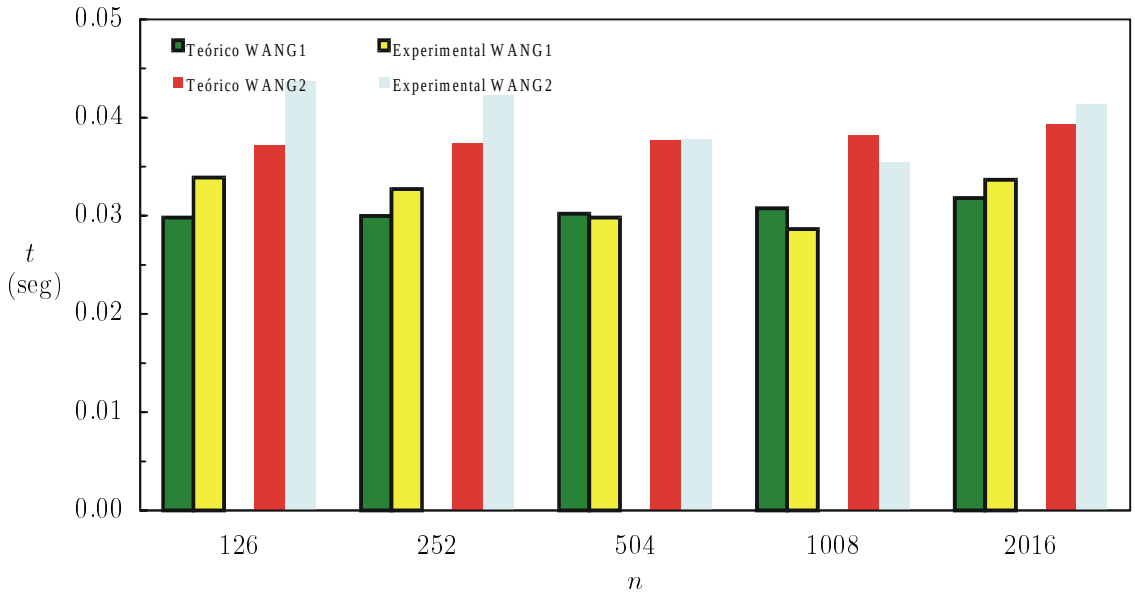


(b) $4096 \leq n \leq 65536$

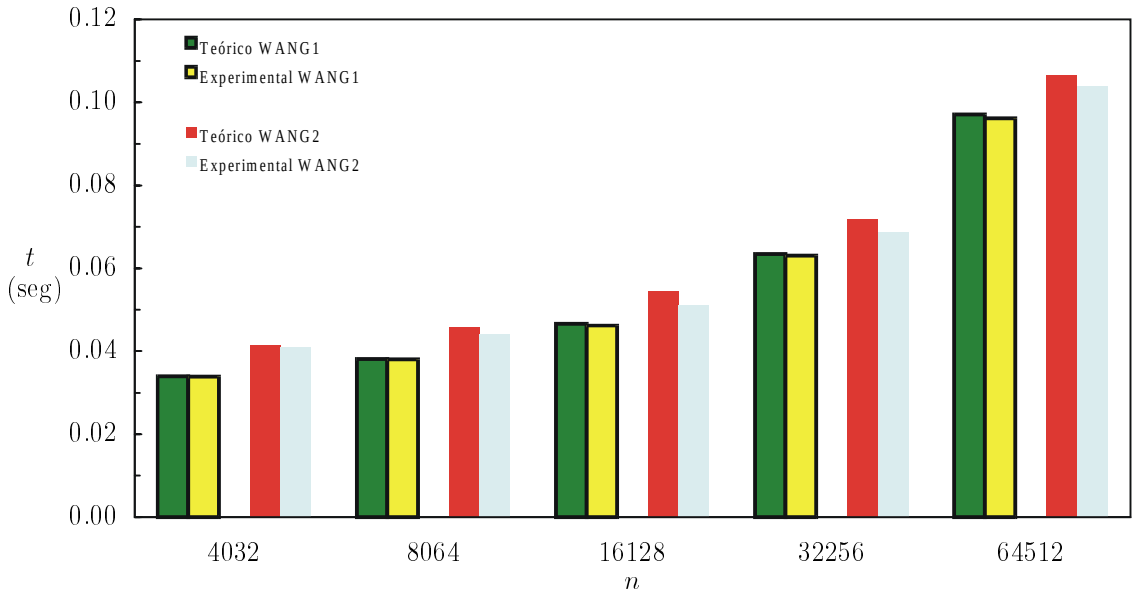
Figura 4.15: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un cluster de PC's para 4 procesadores.*

Cluster de PC's 6 procesadores				
	WANG1		WANG2	
n	Teórico	Experimental	Teórico	Experimental
126	0.0298	0.0339	0.0373	0.0438
252	0.0300	0.0327	0.0374	0.0423
504	0.0302	0.0298	0.0377	0.0378
1008	0.0308	0.0286	0.0382	0.0355
2016	0.0318	0.0337	0.0393	0.0414
4032	0.0339	0.0339	0.0415	0.0408
8064	0.0381	0.0381	0.0458	0.0441
16128	0.0466	0.0462	0.0545	0.0511
32256	0.0634	0.0631	0.0718	0.0686
64512	0.0971	0.0962	0.1065	0.1038

Tabla 4.10: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) medidos en un cluster de PC's, para* 6 *procesadores y* $126 \leq n \leq 64512$.

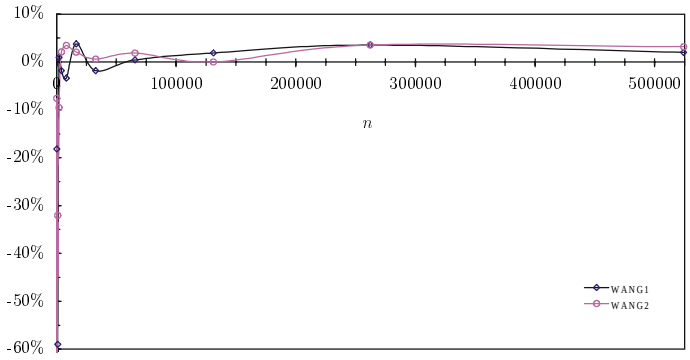


(a) $126 \leq n \leq 2016$

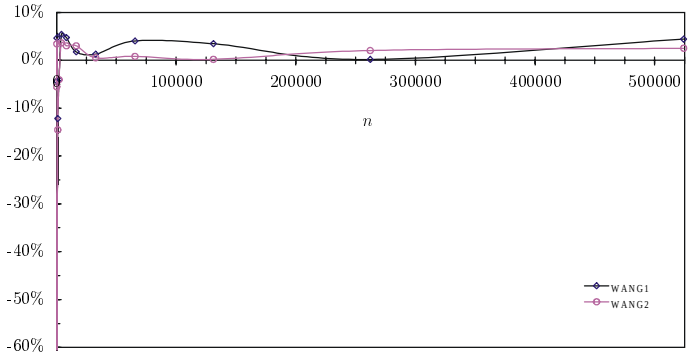


(b) $4032 \leq n \leq 64512$

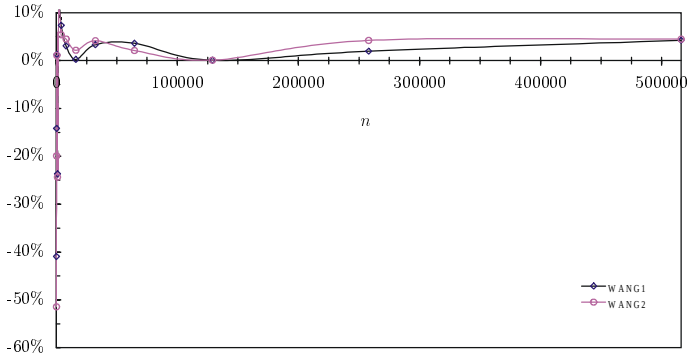
Figura 4.16: *Tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un cluster de PC's para* 6 *procesadores.*



(a) 2 *procesadores*

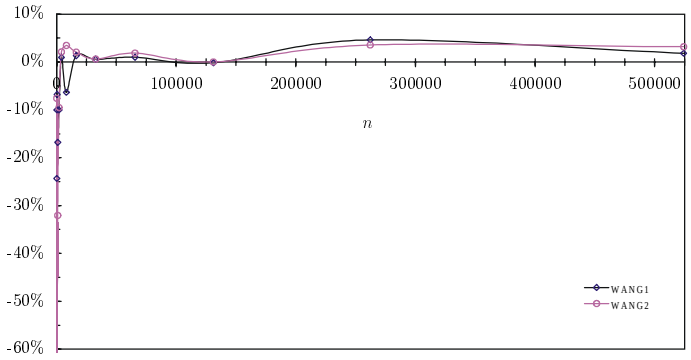


(b) 4 *procesadores*

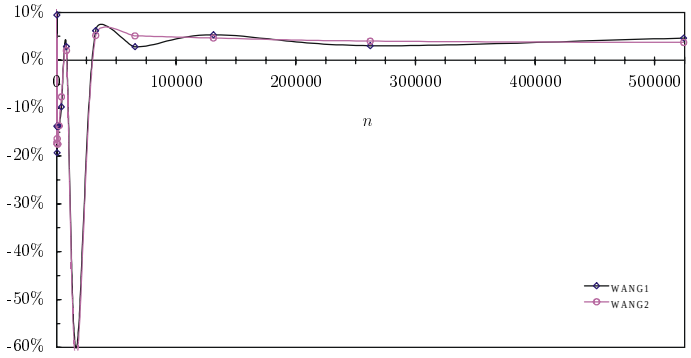


(c) 6 *procesadores*

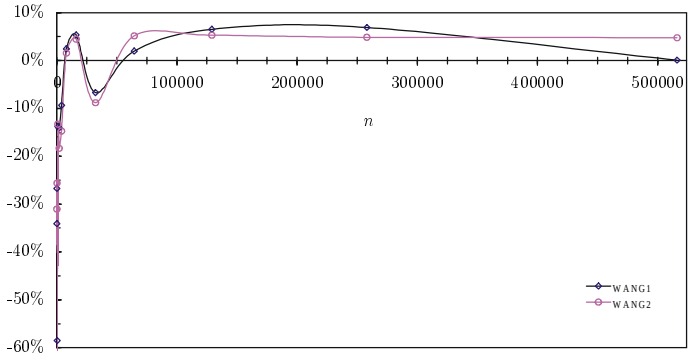
Figura 4.17: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con switch.*



(a) 2 *procesadores*

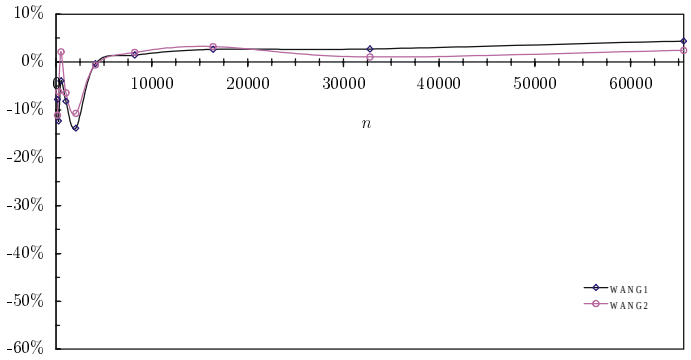


(b) 4 *procesadores*

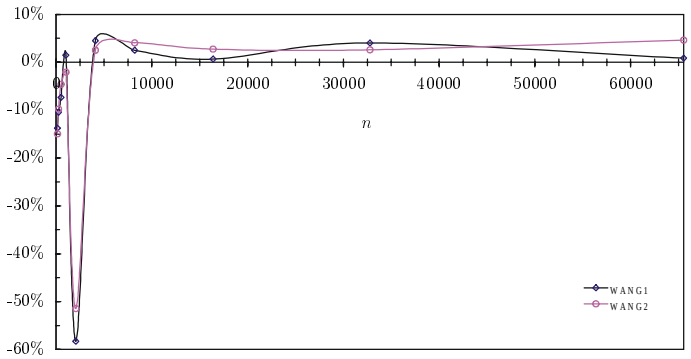


(c) 6 *procesadores*

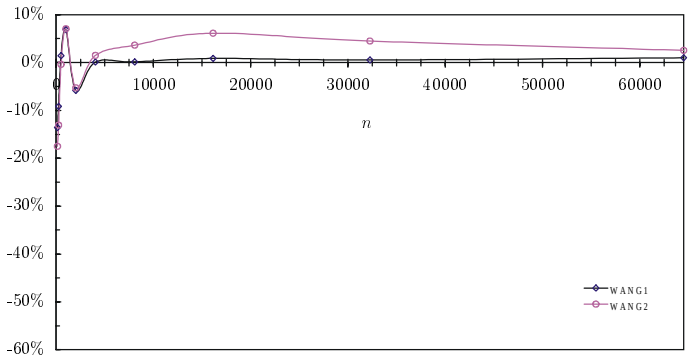
Figura 4.18: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con ethernet.*



(a) 2 *procesadores*



(b) 4 *procesadores*



(c) 6 *procesadores*

Figura 4.19: *Diferencias porcentuales entre los tiempos teóricos y experimentales de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un cluster de PC's.*

CRAY T3D 256 procesadores			
n	WANG1	WANG2	Mejor
512	0.0019	0.0016	WANG2
1024	0.0018	0.0016	WANG2
2048	0.0020	0.0018	WANG2
4096	0.0024	0.0023	WANG2
8192	0.0034	0.0033	WANG2
16384	0.0055	0.0054	WANG2
32768	0.0097	0.0096	WANG2
65536	0.0180	0.0180	WANG2
131072	0.0347	0.0348	WANG1
262144	0.0681	0.0683	WANG1
524288	0.1350	0.1354	WANG1

Tabla 4.11: *Tiempos teóricos de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) medidos en un CRAY T3D, para* 256 *procesadores y* $512 \leq n \leq 524288$.

IBM SP2 *switch*				
	WANG1		WANG2	
p	S_p	E_p	S_p	E_p
2	0.38	19.19%	0.36	17.90%
4	0.29	7.18%	0.28	6.99%
6	0.27	4.57%	0.27	4.49%

(a) *IBM SP2 switch*

IBM SP2 *ethernet*				
	WANG1		WANG2	
p	S_p	E_p	S_p	E_p
2	0.01	0.45%	0.01	0.45%
4	0.00	0.08%	0.00	0.08%
6	0.00	0.02%	0.00	0.02%

(b) *IBM SP2 ethernet*

Cluster de PC's				
	WANG1		WANG2	
p	S_p	E_p	S_p	E_p
2	0.42	21.20%	0.39	19.35%
4	0.37	9.22%	0.34	8.60%
6	0.32	5.39%	0.29	4.91%

(c) *Cluster de PC's*

Tabla 4.12: *Speed-up (S_p) y eficiencia (E_p) de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 y un cluster de PC's.*

s	p	l	g	$n_{\frac{1}{2}}$
12	1	68	0.3	94
	2	164	0.7	71
	4	168	0.7	66
	8	175	0.8	59
	16	181	0.9	61
	32	201	1.1	28
	64	148	1	27
	128	301	1.1	20
	256	387	1.2	15

(a) *CRAY T3D*

s	p	l	g	$n_{\frac{1}{2}}$
46.7	1	86	2.12	9
	2	269	0.87	33
	4	357	0.87	40
	8	506	0.81	40
	16	751	1.04	38
	32	1252	1.31	45

(b) *CRAY T3E*

Tabla 4.13: *Valores de parámetros BSP.*

CRAY T3D				
	WANG1		WANG2	
p	S_p	E_p	S_p	E_p
2	0.88	44.01%	0.76	37.77%
4	1.38	34.40%	1.22	30.47%
8	2.00	24.98%	1.83	22.84%
16	2.47	15.43%	2.33	14.59%
32	2.47	7.73%	2.40	7.51%
64	2.91	4.54%	2.86	4.47%
128	2.77	2.16%	2.75	2.15%
256	2.59	1.01%	2.58	1.01%

(a) *CRAY T3D*

CRAY T3E				
	WANG1		WANG2	
p	S_p	E_p	S_p	E_p
2	0.79	39.64%	0.69	34.51%
4	1.30	32.60%	1.16	29.05%
8	1.99	24.83%	1.82	22.71%
16	2.24	14.00%	2.13	13.30%
32	1.90	11.88%	1.82	11.37%

(b) *CRAY T3E*

Tabla 4.14: *Speed-up (S_p) y eficiencia (E_p) de los algoritmos 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D y un CRAY T3E.*

Capítulo 5

Comparación entre métodos

En este capítulo se comparan entre sí los distintos métodos y algoritmos descritos y analizados en los capítulos 2, 3 y 4. Más en concreto, se compara el tiempo teórico de todos los algoritmos BSP en las distintas máquinas sobre las que se han obtenido resultados numéricos y se busca el algoritmo óptimo para cada una de las situaciones. Como se ha comprobado en los capítulos mencionados, el tiempo experimental se ajusta mejor al tiempo teórico esperado para tamaños de sistema grandes, por ello se realiza la comparación para tamaños de sistema entre 4096 y 65536 ecuaciones para el *cluster* de PC's y entre 16384 y 524288 ecuaciones para el resto de máquinas. Por comodidad, en la tabla 5.1 se vuelven a mostrar los valores de los parámetros BSP de todas las máquinas para las que se han obtenido resultados numéricos.

En las figuras 5.1–5.22 se muestran los tiempos teóricos, medidos en segundos, de todos los algoritmos BSP en las distintas máquinas. Cada una de las figuras contiene una gráfica, que para los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1) y 3.3 (TW2) representa el tiempo esperado tomando $m = 5$, y una tabla que muestra el algoritmo más rápido considerando diversos valores de m. Conviene recordar que m es el número de ecuaciones superpuestas y viene dado por la expresión 2.14 de la página 64.

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	26
	2	3294	9.5	25
	4	5366	12.4	25
	6	8164	12.5	25

(a) *IBM SP2 switch*

s	p	l	g	$n_{\frac{1}{2}}$
45	1	423	2.3	8
	2	20235	709.7	3
	4	54163	1362.6	9
	6	121958	3211.2	9

(b) *IBM SP2 ethernet*

s	p	l	g	$n_{\frac{1}{2}}$
16.4	1	23	0.2	22
	2	2556	6.9	5
	4	5152	7.4	4
	6	7538	6.8	4

(c) *Cluster de PC's*

s	p	l	g	$n_{\frac{1}{2}}$
12	1	68	0.3	94
	2	164	0.7	71
	4	168	0.7	66
	8	175	0.8	59
	16	181	0.9	61
	32	201	1.1	28
	64	148	1	27
	128	301	1.1	20
	256	387	1.2	15

(d) *CRAY T3D*

s	p	l	g	$n_{\frac{1}{2}}$
46.7	1	86	2.12	9
	2	269	0.87	33
	4	357	0.87	40
	8	506	0.81	40
	16	751	1.04	38
	32	1252	1.31	45

(e) *CRAY T3E*

Tabla 5.1: *Valores de parámetros BSP.*

5.1 IBM SP2

5.1.1 *Switch*

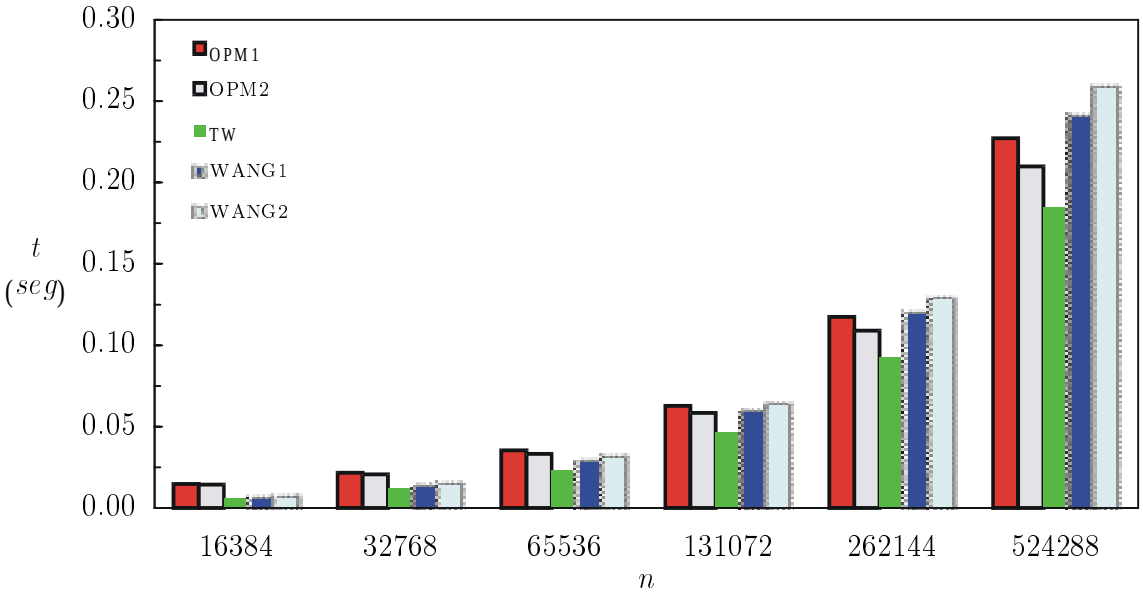
En las figuras 5.1, 5.2 y 5.3, se muestran los tiempos teóricos, medidos en segundos, en un IBM SP2 utilizando *switch*.

Para 2 procesadores el algoritmo 3.1 (TW) es el más rápido, con una diferencia sobre los demás algoritmos que va desde el 9.52% hasta el 41.13% para $n = 524288$ y $m = 5$. Los porcentajes representan la cantidad de tiempo adicional sobre el que necesita el algoritmo más rápido.

Para 4 procesadores el algoritmo más rápido es el 2.3 (OPM2), con muy poca diferencia sobre el algoritmo 3.3 (TW2) (el 0.04% para el tamaño máximo de sistema). Si $m = 100$, el algoritmo más rápido es ahora 3.3 (TW2), con poca diferencia sobre el algoritmo 2.3 (OPM2); si $m = 1000$ esta diferencia es mayor y el algoritmo 4.2 (WANG1) es más rápido para $n \in \{16384, 32768\}$.

Para 6 procesadores se tiene una situación parecida a la existente para 4 procesadores, la diferencia está en que el algoritmo 4.2 (WANG1) es óptimo en más ocasiones (para $m = 100$, $n = 16128$ y $m = 1000$, $n = 64512$).

En cualquiera de los casos, 2, 4, ó 6 procesadores, es más rápido resolver el sistema en secuencial mediante el método de eliminación de Gauss para sistemas tridiagonales. Para $n = 524288$ y $m = 5$, el incremento de tiempo del algoritmo paralelo más rápido sobre el tiempo necesario en secuencial varía desde el 97.91% para 2 procesadores hasta el $250,01\%$ para 6 procesadores. Es evidente por tanto que para el IBM SP2 utilizando *switch*, los algoritmos paralelos que se han analizado no son útiles si el objetivo es ahorrar tiempo en la resolución de los sistemas (ese objetivo se cumple mejor en máquinas con valores de g menores), ahora bien resultan de utilidad si el objetivo es resolver sistemas de tamaño superior al máximo admitido por un sólo procesador. Se pueden resolver, por ejemplo, sistemas de tamaño 3145728 ($6 \cdot 524288$) sólo en paralelo.

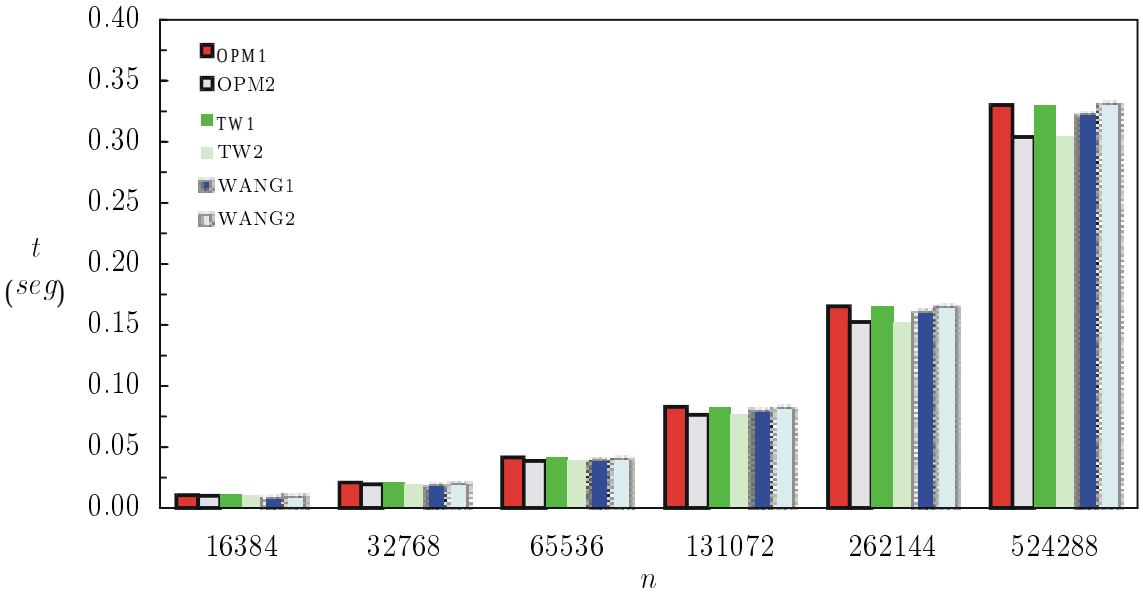


(a) $16384 \leq n \leq 524288$

IBM SP2 2 procesadores *switch*				
	m			
n	5	10	100	1000
16384	TW	TW	TW	TW
32768	TW	TW	TW	TW
65536	TW	TW	TW	TW
131072	TW	TW	TW	TW
262144	TW	TW	TW	TW
524288	TW	TW	TW	TW

(b) *Algoritmo óptimo*

Figura 5.1: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 2 procesadores interconectados mediante switch, para* $16384 \leq n \leq 524288$.

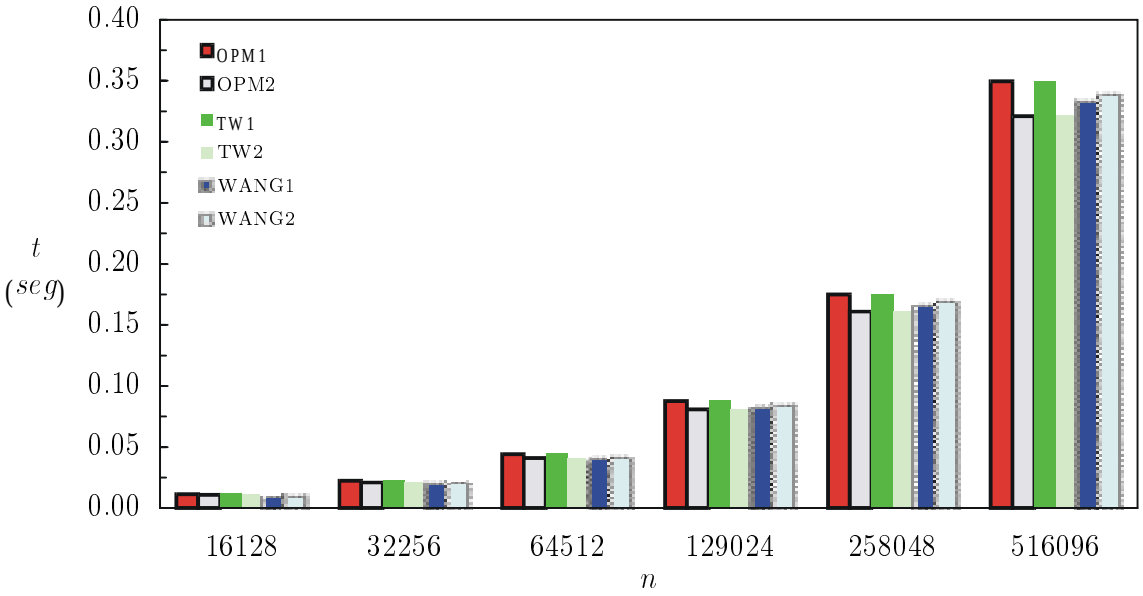


(a) $16384 \leq n \leq 524288$

IBM SP2 4 procesadores				
	m			
n	5	10	100	1000
16384	OPM2	OPM2	TW2	WANG1
32768	OPM2	OPM2	TW2	WANG1
65536	OPM2	OPM2	TW2	TW2
131072	OPM2	OPM2	TW2	TW2
262144	OPM2	OPM2	TW2	TW2
524288	OPM2	OPM2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.2: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 4 procesadores interconectados mediante switch, para* $16384 \leq n \leq 524288$.



(a) $16128 \leq n \leq 516096$

IBM SP2 6 procesadores *switch*				
	m			
n	5	10	100	1000
16128	OPM2	OPM2	WANG1	WANG1
32256	OPM2	OPM2	TW2	WANG1
64512	OPM2	OPM2	TW2	WANG1
129024	OPM2	OPM2	TW2	TW2
258048	OPM2	OPM2	TW2	TW2
516096	OPM2	OPM2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.3: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 6 procesadores interconectados mediante switch, para* $16128 \leq n \leq 516096$.

5.1.2 *Ethernet*

En las figuras 5.4, 5.5 y 5.6, se muestran los tiempos teóricos, medidos en segundos, en un IBM SP2 utilizando *ethernet*.

Para 2 procesadores, de nuevo, el algoritmo 3.1 (TW) es el más rápido, pero con poca diferencia sobre todos los demás. Para 4 procesadores el algoritmo más rápido para $m \geq 100$ es (casi siempre) el 4.2 (WANG1) y para 6 procesadores lo es el algoritmo 4.2 (WANG1) para $m \geq 10$; véanse las figuras 5.5(b) y 5.6(b). En todos los casos no existe gran diferencia en los tiempos.

Utilizando *ethernet* los tiempos que se obtienen son mucho mayores que utilizando *switch*, por lo que resulta siempre mejor utilizar este último dispositivo de comunicación entre los procesadores.

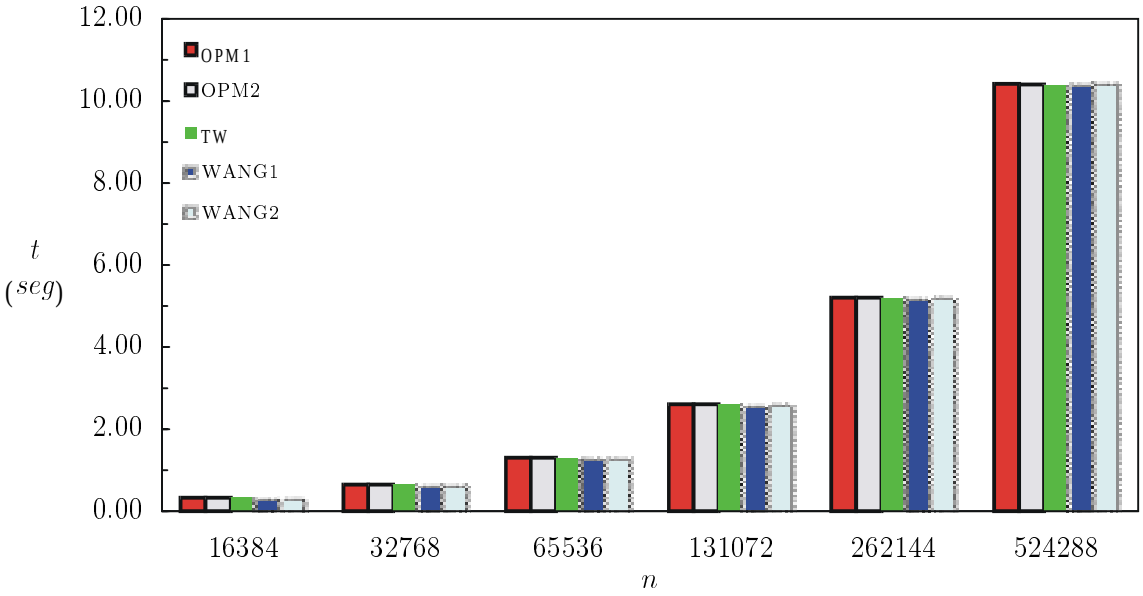
5.2 *Cluster* de PC's

En las figuras 5.7, 5.8 y 5.9, se muestran los tiempos teóricos, medidos en segundos, en un *cluster* de PC's.

Para 2 procesadores el algoritmo más rápido es, otra vez, el 3.1 (TW) (excepto para $n = 4096$ y $m \in \{5, 10, 100\}$), la diferencia sobre los demás algoritmos, para $n = 65536$ y $m = 5$, varía desde el 13.36% hasta el 52.50%.

Para 4 procesadores el algoritmo más rápido, en la mayoría de situaciones, es el 2.2 (OPM1). Para el máximo tamaño de sistema es más rápido el algoritmo 2.3 (OPM2), la diferencia sobre los demás algoritmos varía desde el 2.99% hasta el 16.65%. Para $m = 1000$ y $n \in \{4096, 8192, 16384, 32768\}$ es mejor el algoritmo 3.2 (TW1).

En cualquiera de los casos, 2, 4 ó 6 procesadores, es más rápido resolver el sistema en secuencial mediante el método de eliminación de Gauss para sistemas tridiagonales. Para el máximo tamaño de sistema y $m = 5$, el incremento de tiempo del algoritmo paralelo más rápido sobre el tiempo necesario en secuencial varía desde el 69.49% para 2 procesadores hasta el 205.67% para 6 procesadores. En el *cluster* de PC's ocurre lo mismo que en el IBM SP2, los algoritmos paralelos que se han analizado son más útiles

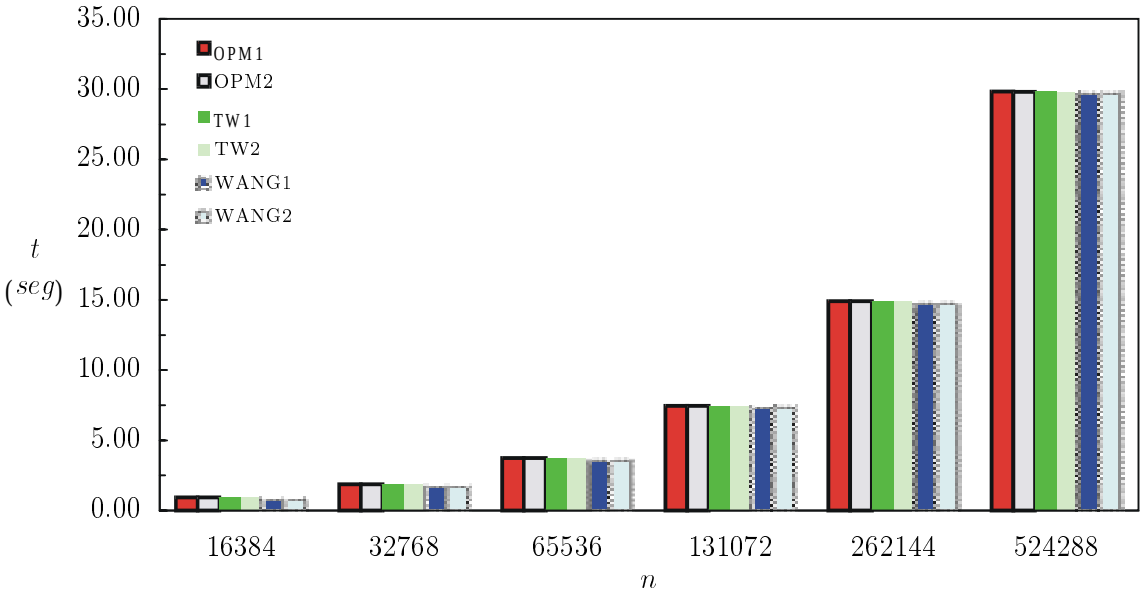


(a) $16384 \leq n \leq 524288$

IBM SP2 2 procesadores *ethernet*				
	m			
n	5	10	100	1000
16384	TW	TW	TW	TW
32768	TW	TW	TW	TW
65536	TW	TW	TW	TW
131072	TW	TW	TW	TW
262144	TW	TW	TW	TW
524288	TW	TW	TW	TW

(b) *Algoritmo óptimo*

Figura 5.4: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 2 procesadores interconectados mediante ethernet, para* $16384 \leq n \leq 524288$.

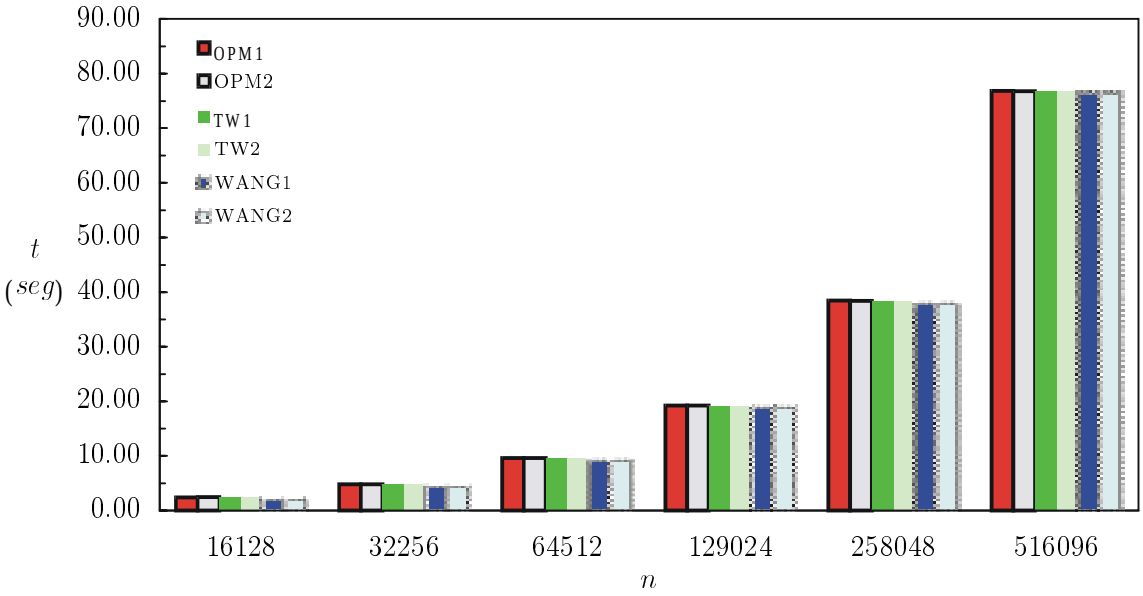


(a) $16384 \leq n \leq 524288$

IBM SP2 4 procesadores *ethernet*				
	m			
n	5	10	100	1000
16384	OPM1	TW1	WANG1	WANG1
32768	OPM1	TW1	WANG1	WANG1
65536	OPM2	WANG1	WANG1	WANG1
131072	OPM2	TW2	WANG1	WANG1
262144	OPM2	TW2	WANG1	WANG1
524288	OPM2	TW2	TW2	WANG1

(b) *Algoritmo óptimo*

Figura 5.5: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 4 procesadores interconectados mediante ethernet, para* $16384 \leq n \leq 524288$.

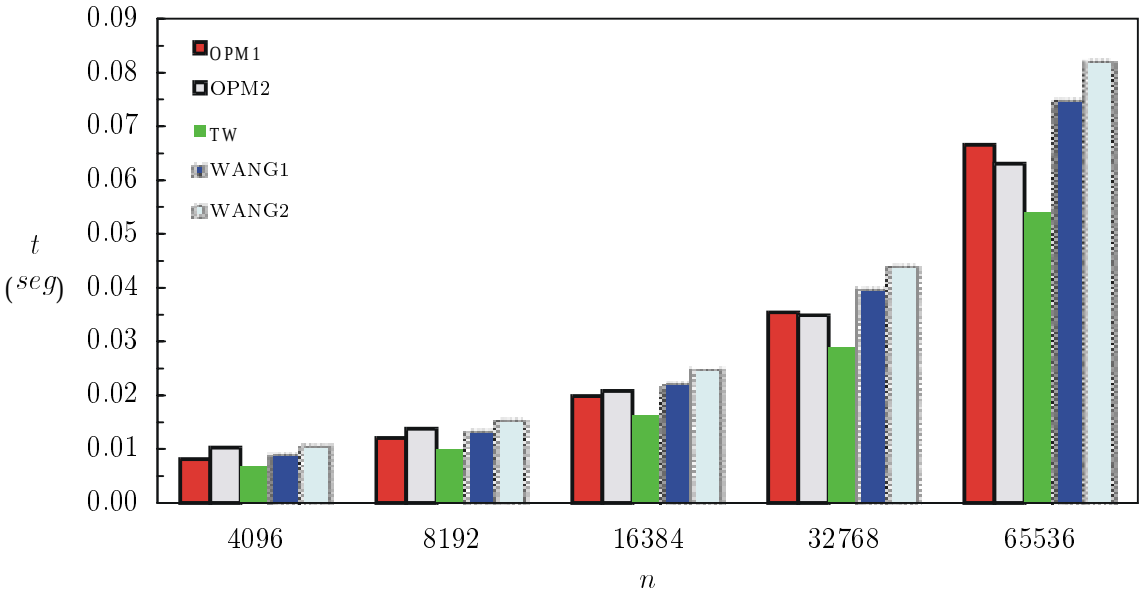


(a) $16128 \leq n \leq 516096$

IBM SP2 6 procesadores *ethernet*				
	m			
n	5	10	100	1000
16128	TW1	WANG1	WANG1	WANG1
32256	TW1	WANG1	WANG1	WANG1
64512	WANG1	WANG1	WANG1	WANG1
129024	WANG1	WANG1	WANG1	WANG1
258048	OPM2	WANG1	WANG1	WANG1
516096	OPM2	TW2	WANG1	WANG1

(b) *Algoritmo óptimo*

Figura 5.6: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un IBM SP2 con 6 procesadores interconectados mediante ethernet, para* $16128 \leq n \leq 516096$.

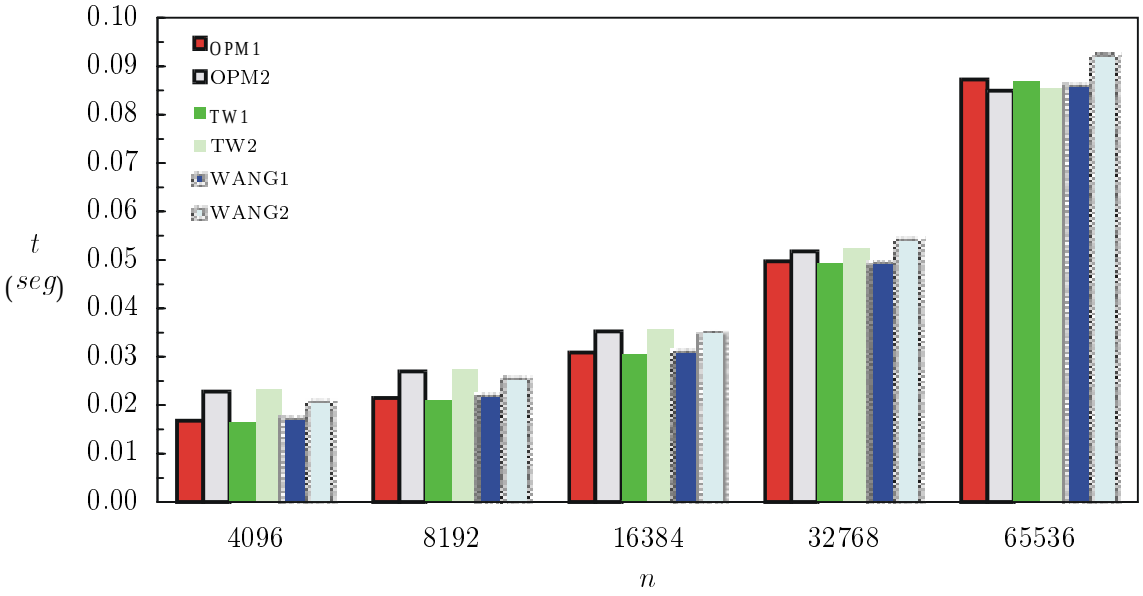


(a) $4096 \leq n \leq 65536$

Cluster de PC's 2 procesadores				
	m			
n	5	10	100	1000
4096	OPM1	OPM1	OPM1	TW
8192	TW	TW	TW	TW
16384	TW	TW	TW	TW
32768	TW	TW	TW	TW
65536	TW	TW	TW	TW

(b) *Algoritmo óptimo*

Figura 5.7: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un cluster de PC's, para 2 procesadores y* $4096 \leq n \leq 65536$.

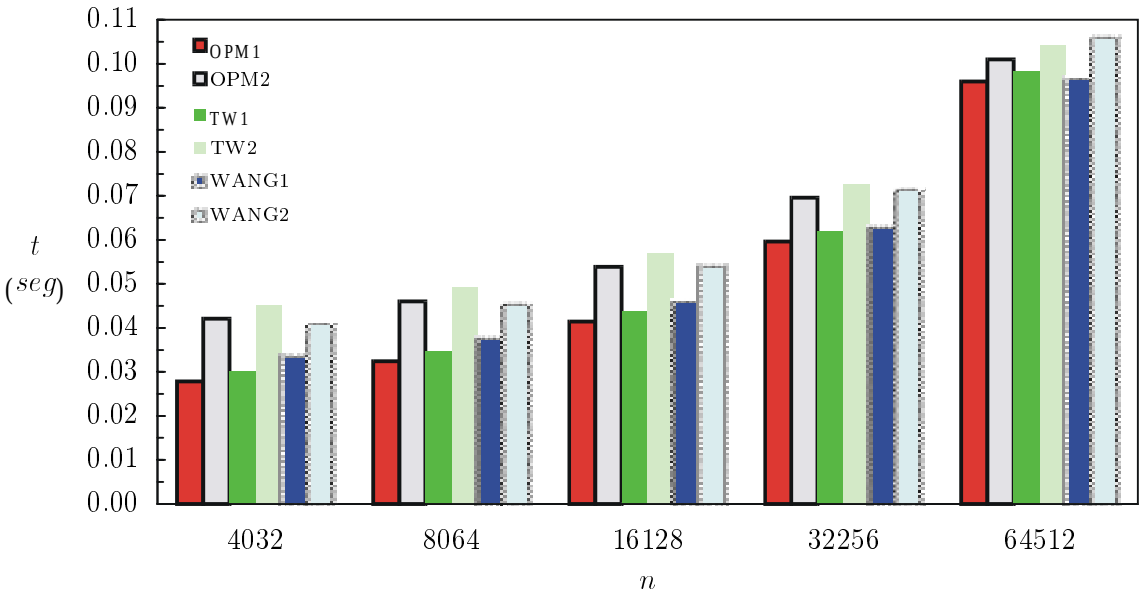


(a) $4096 \leq n \leq 65536$

Cluster de PC's 4 procesadores				
	m			
n	5	10	100	1000
4096	OPM1	OPM1	OPM1	TW1
8192	OPM1	OPM1	OPM1	TW1
16384	OPM1	OPM1	OPM1	TW1
32768	OPM1	OPM1	OPM1	TW1
65536	OPM2	OPM2	OPM2	OPM2

(b) *Algoritmo óptimo*

Figura 5.8: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un cluster de PC's, para 4 procesadores y* $4096 \leq n \leq 65536$.



(a) $4032 \leq n \leq 64512$

Cluster de PC's 6 procesadores				
	m			
n	5	10	100	1000
4032	OPM1	OPM1	OPM1	OPM1
8064	OPM1	OPM1	OPM1	OPM1
16128	OPM1	OPM1	OPM1	OPM1
32256	OPM1	OPM1	OPM1	OPM1
64512	OPM1	OPM1	OPM1	OPM1

(b) *Algoritmo óptimo*

Figura 5.9: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un cluster de PC's, para 6 procesadores y* $4032 \leq n \leq 64512$.

si el objetivo es resolver sistemas de tamaño superior al máximo admitido por un sólo procesador, en esta máquina se pueden resolver, por ejemplo, sistemas de tamaño 387072 $(6 \cdot 64512)$ sólo en paralelo.

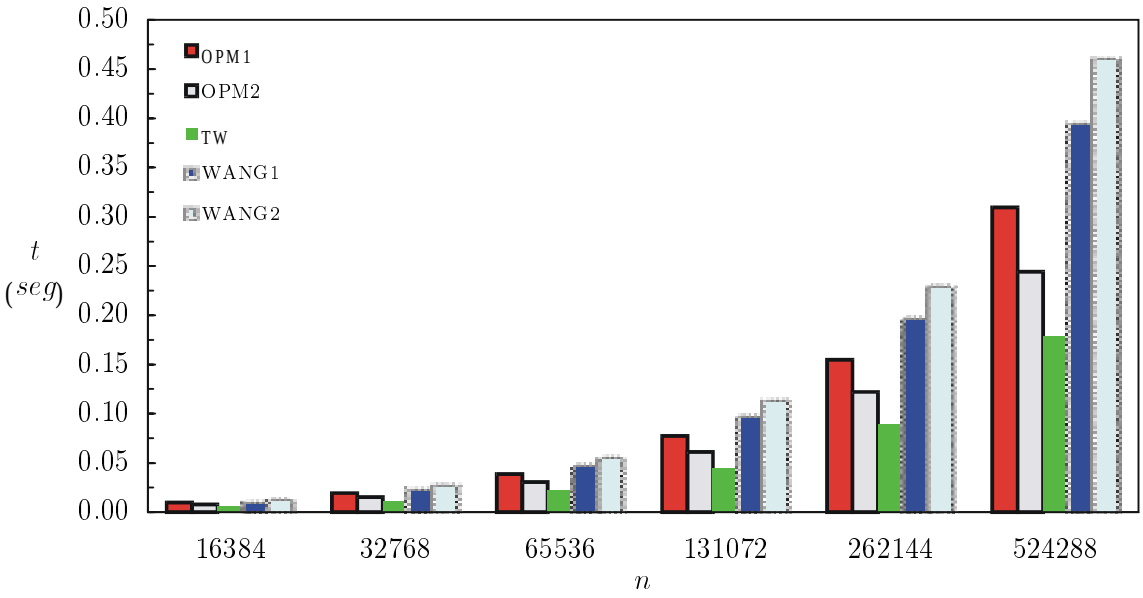
5.3 CRAY T3D

En las figuras 5.10–5.17 se muestran los tiempos teóricos, medidos en segundos, en un CRAY T3D. De nuevo, el algoritmo 3.1 (TW) es el más rápido para 2 procesadores, la diferencia sobre los demás algoritmos varía desde el 36.7% hasta el 159% para $n = 524288$ y $m = 5$. Se recuerda que los porcentajes representan la cantidad de tiempo adicional sobre el que necesita el algoritmo más rápido.

Para 4 procesadores el algoritmo más rápido es el 2.3 (OPM2), con muy poca diferencia sobre el algoritmo 3.3 (TW2). Si el valor de m es 100, es más rápido el algoritmo 3.3 (TW2). No hay una diferencia sustancial en el tiempo invertido por los algoritmos 2.3 (OPM2) y 3.3 (TW2) para valores de m no muy grandes, ahora bien, cuando el valor de m es grande la diferencia es significativa; por ejemplo, para $m = 50000$, $p = 4$ y $n = 524288$ el algoritmo 3.3 (TW2) es el más rápido de todos y tarda 0.2248 segundos mientras que el algoritmo 2.3 (OPM2) tarda 0.2734 segundos, lo que supone un $21,64\%$ de incremento. Para esos mismos valores de m, p y n la resolución del sistema secuencialmente mediante el método de Gauss para sistemas tridiagonales tarda 0.3495 segundos, lo que supone un $55,49\%$ más que con el algoritmo 3.3 (TW2) que es el óptimo.

A medida que va aumentando el número de procesadores, se observa que el algoritmo 3.3 (TW2) es el óptimo para valores de m cada vez más pequeños y que para valores de m grandes el algoritmo 4.2 (WANG1) es el más rápido, a excepción de $p = 255$ donde el óptimo es el algoritmo 4.3 (WANG2) en muchas situaciones.

En esta máquina (véase la tabla 5.1(d)) con valor de g próximo a 1, es más rápido resolver el sistema mediante un algoritmo en paralelo que en secuencial mediante el método de eliminación de Gauss para sistemas tridiagonales. Para 2 procesadores, $n = 524288$ y $m = 5$, el incremento de tiempo necesario en secuencial sobre el algoritmo paralelo más rápido (TW) es del 95.67%, el *speed-up* es muy bueno, $S_2 = 1.96$, al igual que la eficiencia, $E_2 = 97.84\%$. Precisamente este buen comportamiento es el que hace que el

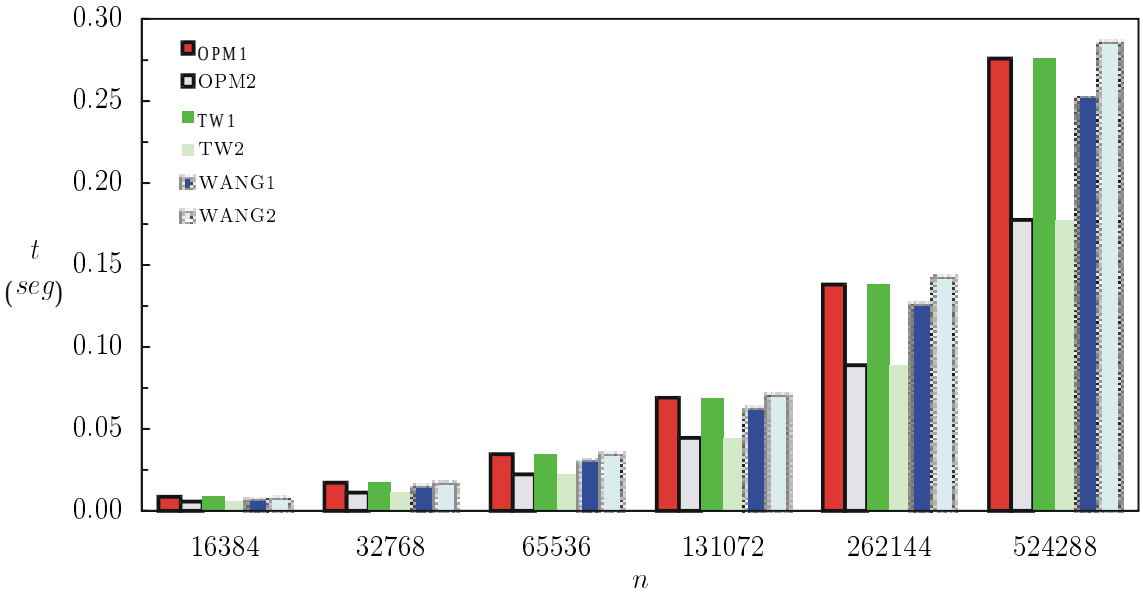


(a) $16384 \leq n \leq 524288$

CRAY T3D 2 procesadores				
	m			
n	5	10	100	1000
16384	TW	TW	TW	TW
32768	TW	TW	TW	TW
65536	TW	TW	TW	TW
131072	TW	TW	TW	TW
262144	TW	TW	TW	TW
524288	TW	TW	TW	TW

(b) *Algoritmo óptimo*

Figura 5.10: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 2 procesadores y* $16384 \leq n \leq 524288$.

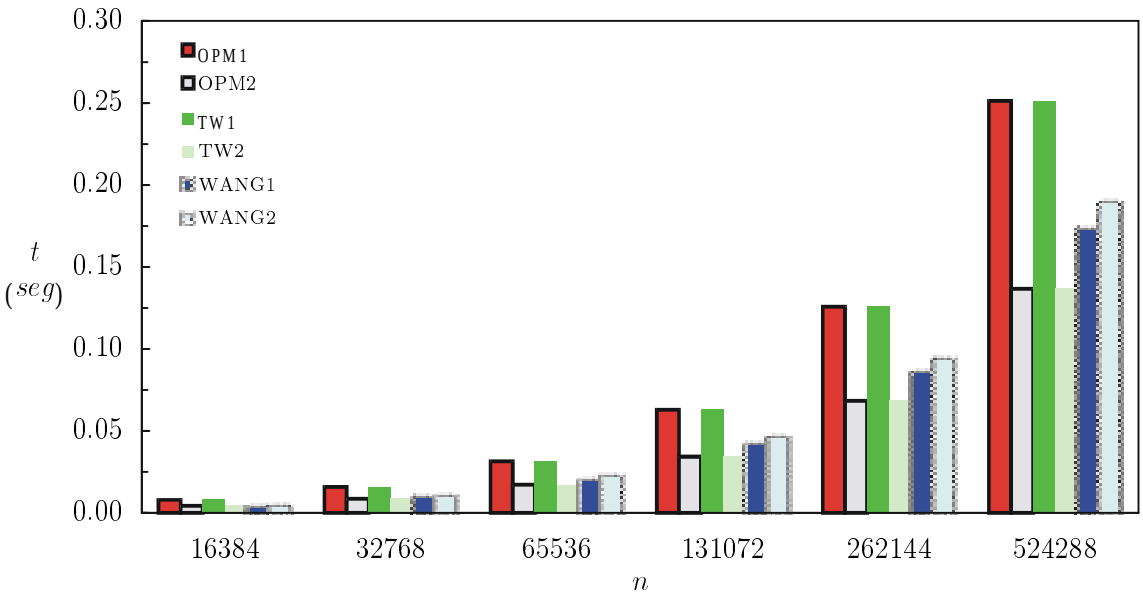


(a) $16384 \leq n \leq 524288$

CRAY T3D 4 procesadores				
	m			
n	5	10	100	1000
16384	OPM2	OPM2	TW2	TW2
32768	OPM2	OPM2	TW2	TW2
65536	OPM2	OPM2	TW2	TW2
131072	OPM2	OPM2	TW2	TW2
262144	OPM2	OPM2	TW2	TW2
524288	OPM2	OPM2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.11: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 4 procesadores y* $16384 \leq n \leq 524288$.

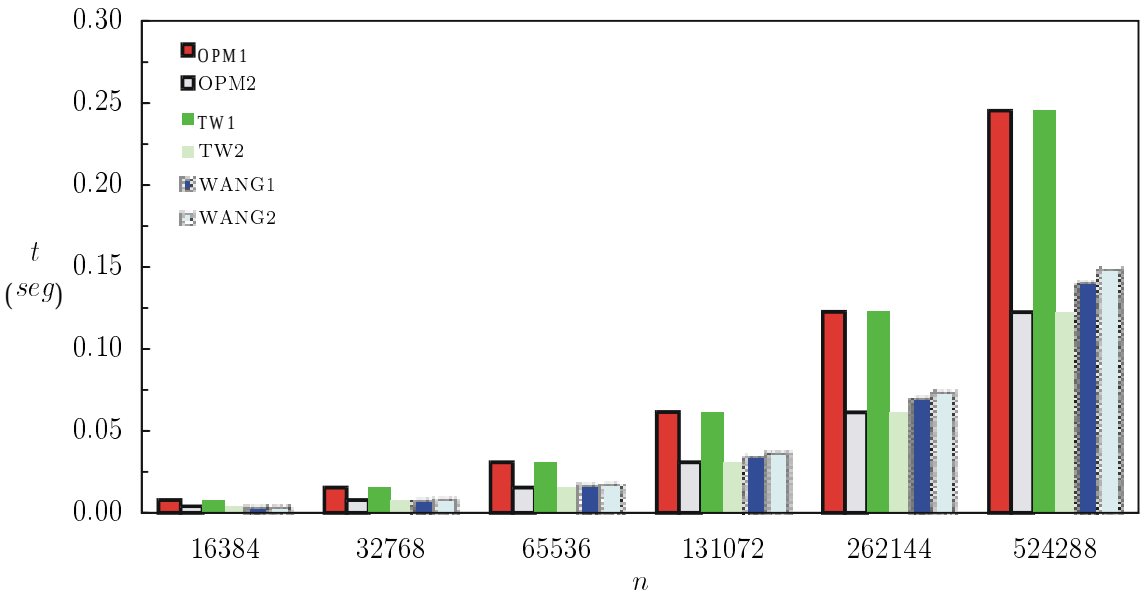


(a) $16384 \leq n \leq 524288$

CRAY T3D 8 procesadores				
	m			
n	5	10	100	1000
16384	OPM2	OPM2	TW2	WANG1
32768	OPM2	OPM2	TW2	TW2
65536	OPM2	OPM2	TW2	TW2
131072	OPM2	OPM2	TW2	TW2
262144	OPM2	OPM2	TW2	TW2
524288	OPM2	OPM2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.12: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 8 procesadores y* $16384 \leq n \leq 524288$.

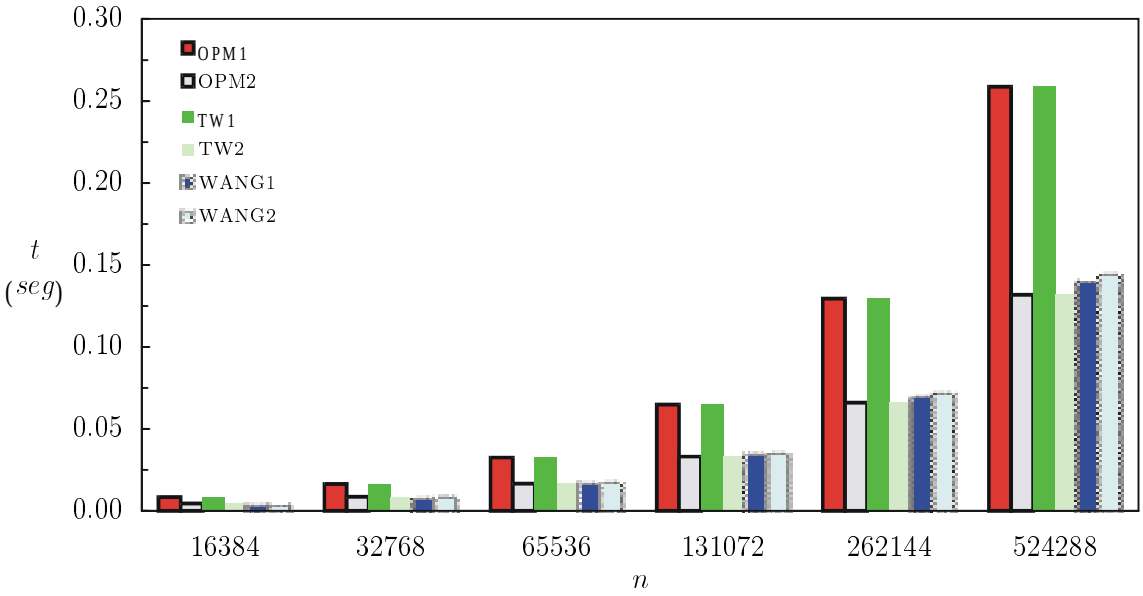


(a) $16384 \leq n \leq 524288$

CRAY T3D 16 procesadores				
	m			
n	5	10	100	1000
16384	OPM2	TW2	TW2	WANG1
32768	OPM2	TW2	TW2	WANG1
65536	OPM2	TW2	TW2	WANG1
131072	OPM2	TW2	TW2	TW2
262144	OPM2	TW2	TW2	TW2
524288	OPM2	TW2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.13: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para* 16 *procesadores y* $16384 \leq n \leq 524288$.

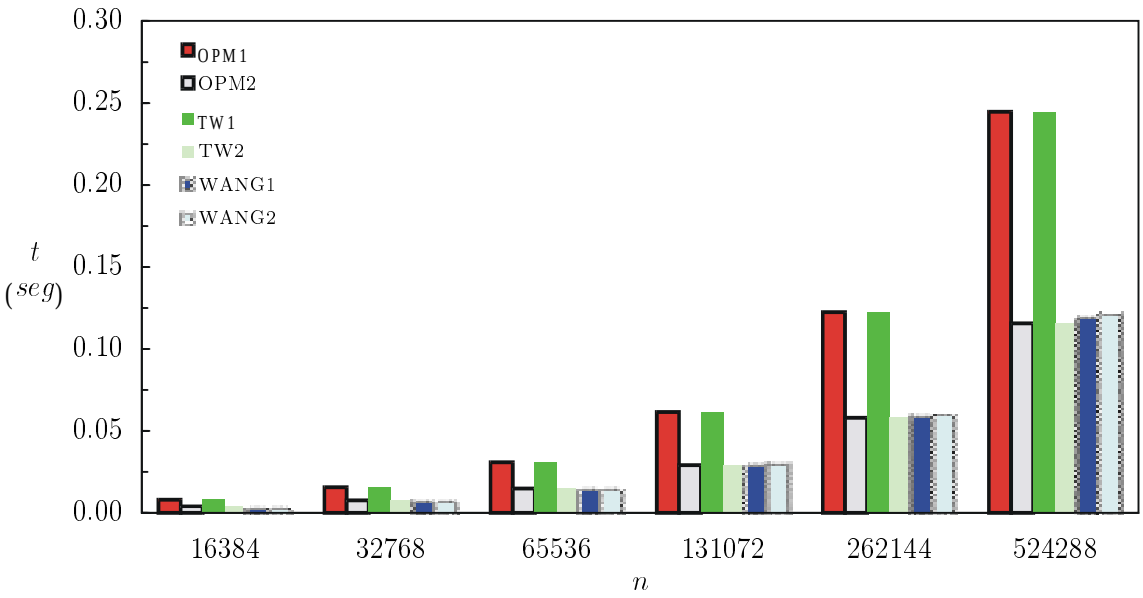


(a) $16384 \leq n \leq 524288$

CRAY T3D 32 procesadores				
	m			
n	5	10	100	1000
16384	TW2	TW2	WANG1	WANG1
32768	TW2	TW2	WANG1	WANG1
65536	TW2	TW2	TW2	WANG1
131072	TW2	TW2	TW2	WANG1
262144	TW2	TW2	TW2	WANG1
524288	TW2	TW2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.14: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para 32 procesadores y* $16384 \leq n \leq 524288$.

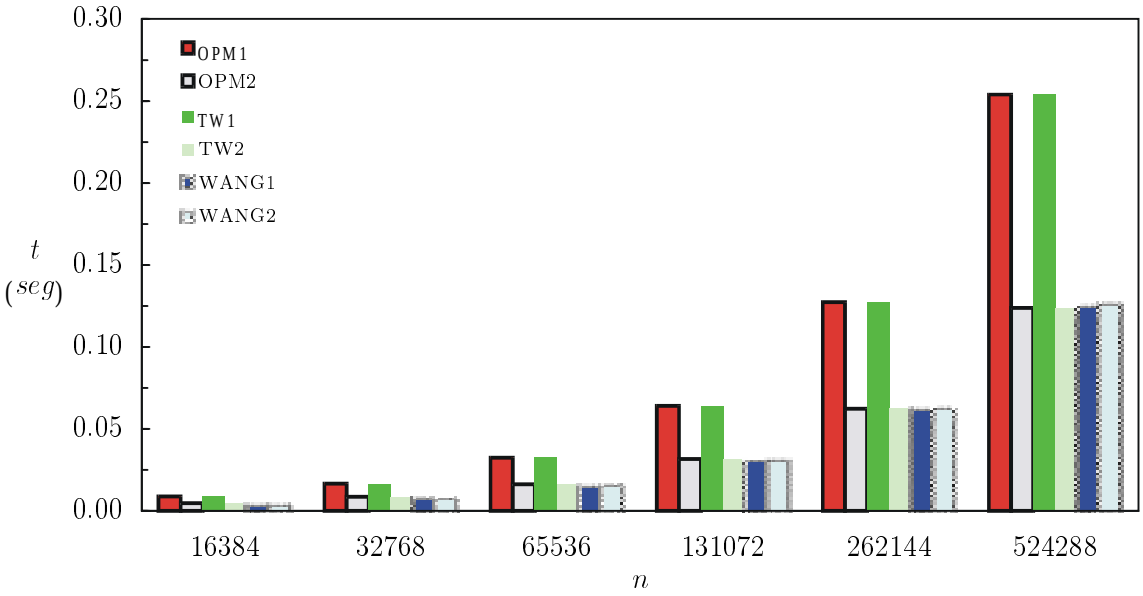


(a) $16384 \leq n \leq 524288$

CRAY T3D 64 procesadores				
	m			
n	5	10	100	1000
16384	TW2	TW2	WANG1	WANG1
32768	TW2	TW2	WANG1	WANG1
65536	TW2	TW2	WANG1	WANG1
131072	TW2	TW2	TW2	WANG1
262144	TW2	TW2	TW2	WANG1
524288	TW2	TW2	TW2	WANG1

(b) *Algoritmo óptimo*

Figura 5.15: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para* 64 *procesadores y* $16384 \leq n \leq 524288$.

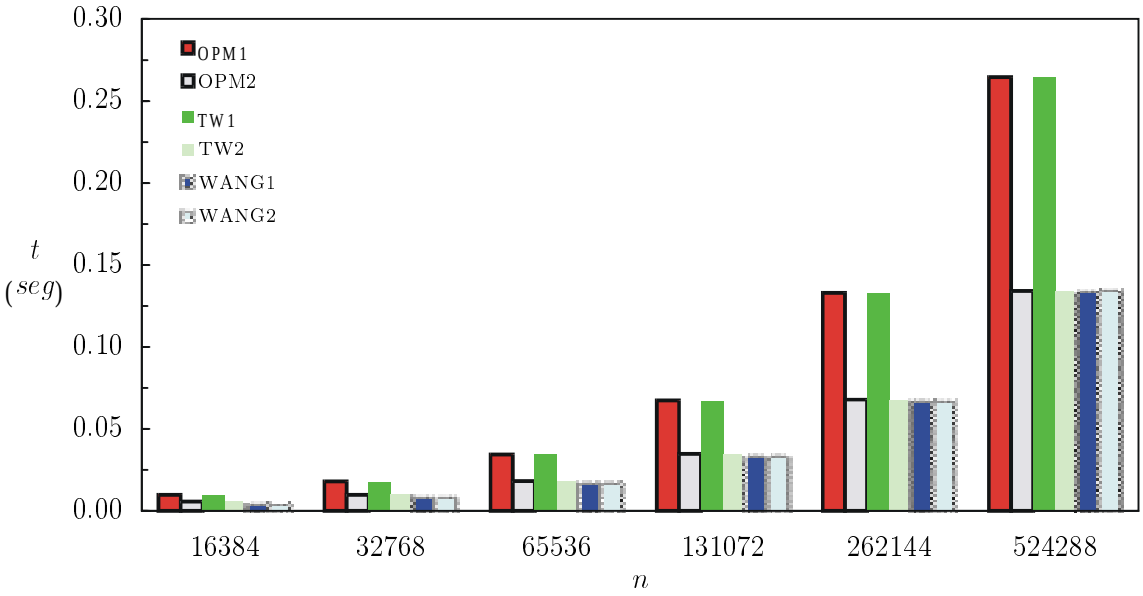


(a) $16384 \leq n \leq 524288$

CRAY T3D 128 procesadores				
	m			
n	5	10	100	1000
16384	WANG2	WANG2	WANG2	WANG2
32768	TW2	WANG1	WANG1	WANG1
65536	TW2	TW2	WANG1	WANG1
131072	TW2	TW2	WANG1	WANG1
262144	TW2	TW2	WANG1	WANG1
524288	TW2	TW2	TW2	WANG1

(b) *Algoritmo óptimo*

Figura 5.16: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para* 128 *procesadores y* $16384 \leq n \leq 524288$.



(a) $16384 \leq n \leq 524288$

CRAY T3D 256 procesadores				
	m			
n	5	10	100	1000
16384	WANG2	WANG2	WANG2	WANG2
32768	WANG2	WANG2	WANG2	WANG2
65536	TW2	WANG2	WANG2	WANG2
131072	TW2	WANG1	WANG1	WANG1
262144	TW2	TW2	WANG1	WANG1
524288	TW2	TW2	WANG1	WANG1

(b) *Algoritmo óptimo*

Figura 5.17: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3D, para* 128 *procesadores y* $16384 \leq n \leq 524288$.

algoritmo 3.3 (TW2) sea el más rápido en muchas situaciones.

En la tabla 5.2 se muestra, para diversos valores de m, el número óptimo de procesadores, si el objetivo es conseguir la mayor rapidez posible en la resolución del sistema tridiagonal. En las columnas etiquetadas con Proc se muestra el número de procesadores óptimo, en las columnas etiquetadas con Alg se muestra el algoritmo con el que se consigue el mejor tiempo y en las columnas etiquetadas con Gauss se muestra el incremento porcentual de tiempo que se produce al resolver el sistema en secuencial por el método de eliminación de Gauss para sistemas tridiagonales sobre el tiempo utilizado en paralelo por el algoritmo más rápido; un 100% de incremento significa que en secuencial se tarda el doble que en paralelo.

5.4 CRAY T3E

En las figuras 5.18–5.22 se muestran los tiempos teóricos, medidos en segundos, en un CRAY T3E.

En esta máquina, el algoritmo 3.1 (TW) también es el más rápido para 2 procesadores, la diferencia sobre los demás algoritmos varía desde el 29.49% hasta el 127.74%, para $n = 524288$ y $m = 5$.

Para 4, 8, 16 y 32 procesadores el algoritmo más rápido es el 2.3 (OPM2), con muy poca diferencia sobre el algoritmo 3.3 (TW2); por ejemplo, para $n = 524288$ y $m = 5$ la máxima diferencia, que se alcanza para 32 procesadores, es del 0.05%. En cambio, si $m = 100$ el algoritmo 3.3 (TW2) es el más rápido, con muy poca diferencia sobre el algoritmo 2.3 (OPM2). La diferencia es sustancial para valores de m mayores; por ejemplo, para $p = 8$, $n = 524288$ y $m = 10000$ el algoritmo 3.3 (TW2) sigue siendo el más rápido, la diferencia con el algoritmo 2.3 (OPM2) es de un $10,18\%$ y con los demás varía desde el $14,97\%$ hasta el $85,07\%$.

Al igual que en el CRAY T3D, en esta máquina el valor de g está próximo a 1 (véase la tabla 5.1(e)) y es más rápido resolver el sistema mediante un algoritmo en paralelo que en secuencial mediante el método de eliminación de Gauss para sistemas tridiagonales. Para 2 procesadores, $n = 524288$ y $m = 5$, el incremento de tiempo necesario en secuencial sobre el algoritmo paralelo más rápido (TW) es del 57.2%, el *speed-up* también es bueno,

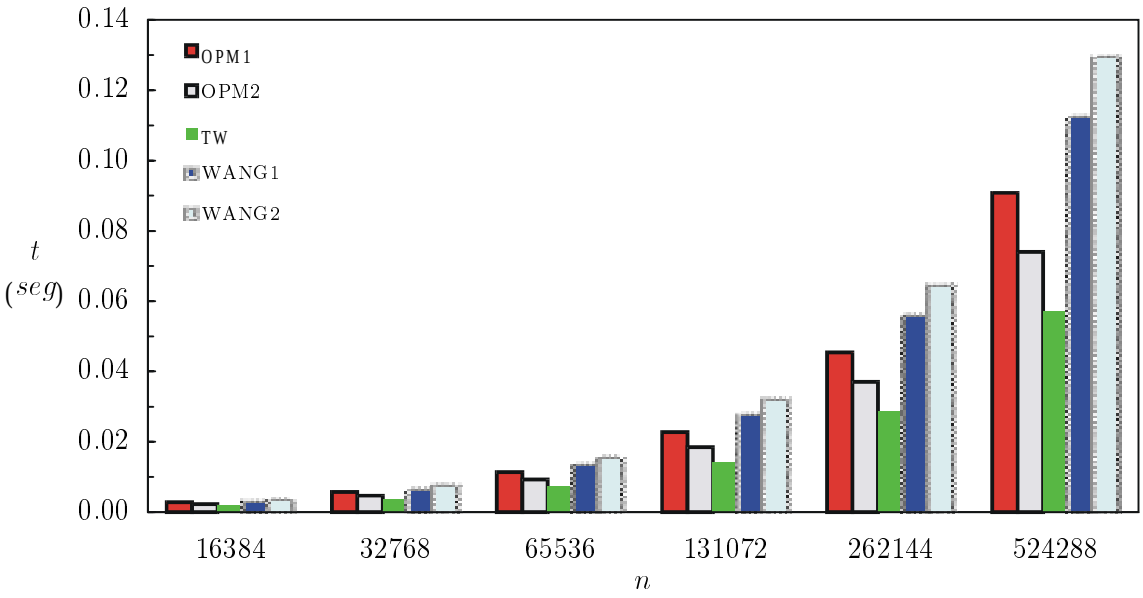
CRAY T3D						
	m					
	5			10		
n	Proc.	Alg.	Gauss	Proc.	Alg.	Gauss
16384	$16p$	OPM2	167.61%	$16p$	TW2	166.45%
32768	$64p$	TW2	184.66%	$64p$	TW2	182.52%
65536	$64p$	TW2	193.98%	$64p$	TW2	192.86%
131072	$64p$	TW2	198.84%	$64p$	TW2	198.27%
262144	$64p$	TW2	201.33%	$64p$	TW2	201.04%
524288	$64p$	TW2	202.59%	$64p$	TW2	202.44%

(a) $m \in \{5, 10\}$

CRAY T3D						
	m					
	100			1000		
n	Proc.	Alg.	Gauss	Proc.	Alg.	Gauss
16384	$64p$	WANG1	159.83%	$64p$	WANG1	159.83%
32768	$64p$	WANG1	174.89%	$64p$	WANG1	174.89%
65536	$64p$	WANG1	183.07%	$64p$	WANG1	183.07%
131072	$64p$	TW2	188.35%	$64p$	WANG1	187.34%
262144	$64p$	TW2	195.94%	$64p$	WANG1	189.52%
524288	$64p$	TW2	199.85%	$64p$	WANG1	190.63%

(b) $m \in \{100, 1000\}$

Tabla 5.2: *Número óptimo de procesadores en un CRAY T3D, para* $16384 \leq n \leq 524288$.

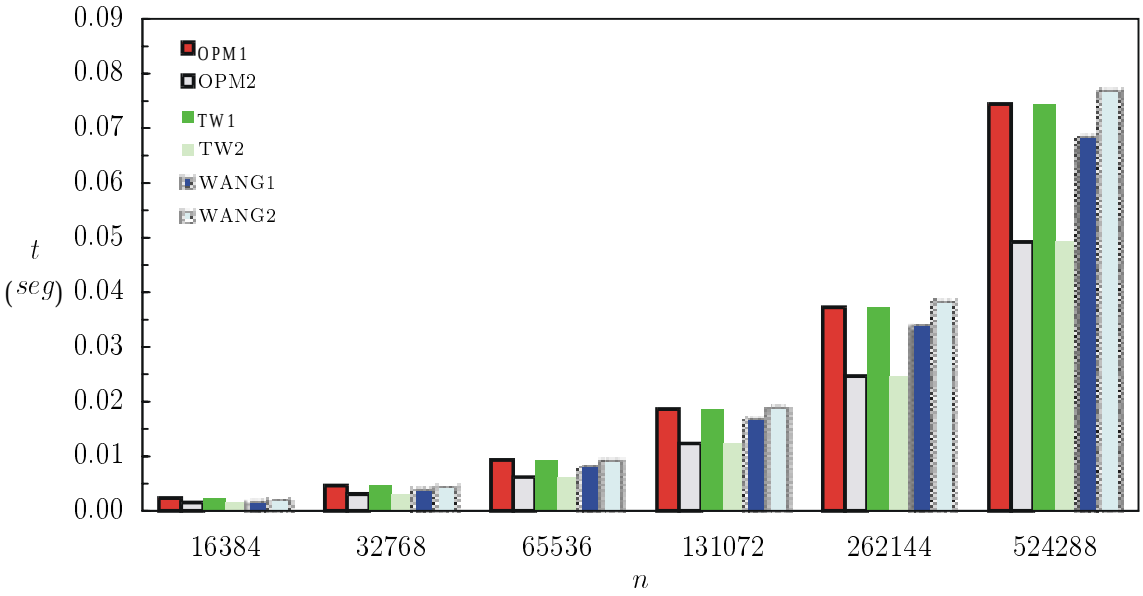


(a) $16384 \leq n \leq 524288$

CRAY T3E 2 procesadores				
	m			
n	5	10	100	1000
16384	TW	TW	TW	TW
32768	TW	TW	TW	TW
65536	TW	TW	TW	TW
131072	TW	TW	TW	TW
262144	TW	TW	TW	TW
524288	TW	TW	TW	TW

(b) *Algoritmo óptimo*

Figura 5.18: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.1 (TW), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para 2 procesadores y* $16384 \leq n \leq 524288$.

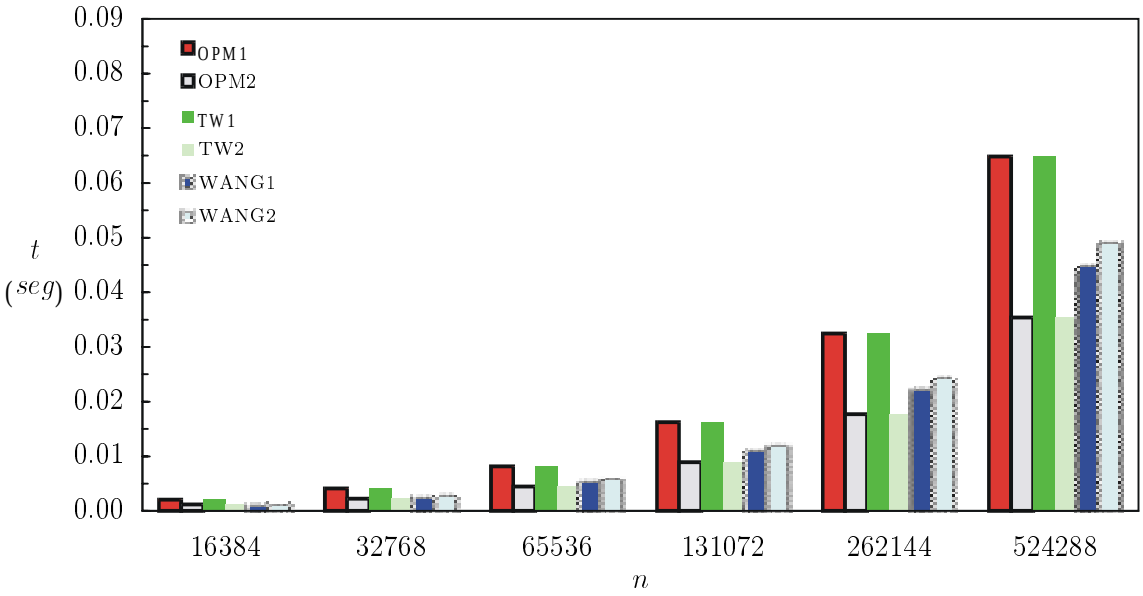


(a) $16384 \leq n \leq 524288$

CRAY T3E 4 procesadores				
	m			
n	5	10	100	1000
16384	OPM2	OPM2	TW2	TW2
32768	OPM2	OPM2	TW2	TW2
65536	OPM2	OPM2	TW2	TW2
131072	OPM2	OPM2	TW2	TW2
262144	OPM2	OPM2	TW2	TW2
524288	OPM2	OPM2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.19: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para* 4 *procesadores y* $16384 \leq n \leq 524288$.

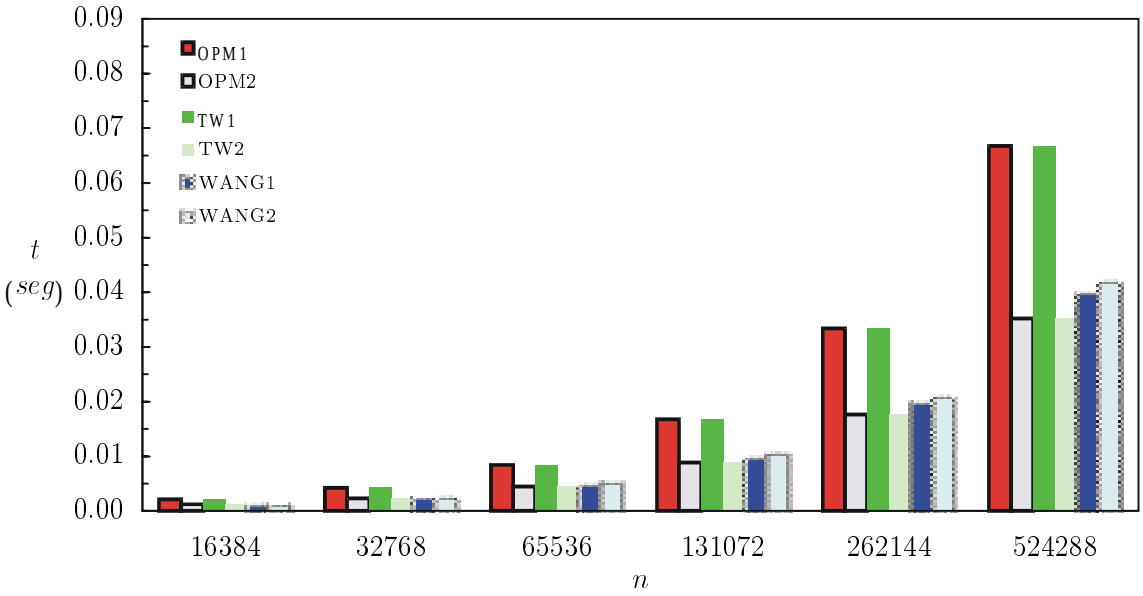


(a) $16384 \leq n \leq 524288$

CRAY T3E 8 procesadores				
	m			
n	5	10	100	1000
16384	OPM2	OPM2	TW2	WANG1
32768	OPM2	OPM2	TW2	TW2
65536	OPM2	OPM2	TW2	TW2
131072	OPM2	OPM2	TW2	TW2
262144	OPM2	OPM2	TW2	TW2
524288	OPM2	OPM2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.20: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para* 8 *procesadores y* $16384 \leq n \leq 524288$.

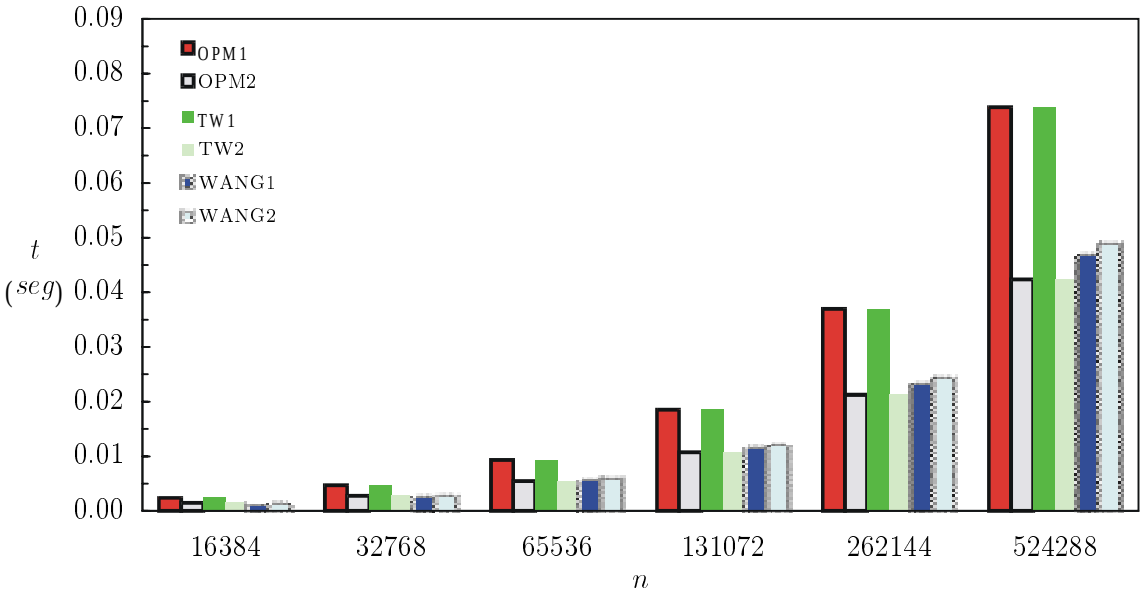


(a) $16384 \leq n \leq 524288$

CRAY T3E 16 procesadores				
	m			
n	5	10	100	1000
16384	OPM2	OPM2	TW2	WANG1
32768	OPM2	OPM2	TW2	WANG1
65536	OPM2	OPM2	TW2	WANG1
131072	OPM2	OPM2	TW2	TW2
262144	OPM2	OPM2	TW2	TW2
524288	OPM2	OPM2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.21: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para* 16 *procesadores y* $16384 \leq n \leq 524288$.



(a) $16384 \leq n \leq 524288$

CRAY T3E 32 procesadores				
	m			
n	5	10	100	1000
16384	OPM2	OPM2	TW2	WANG1
32768	OPM2	OPM2	TW2	WANG1
65536	OPM2	OPM2	TW2	WANG1
131072	OPM2	OPM2	TW2	TW2
262144	OPM2	OPM2	TW2	TW2
524288	OPM2	OPM2	TW2	TW2

(b) *Algoritmo óptimo*

Figura 5.22: *Comparación entre los algoritmos 2.2 (OPM1), 2.3 (OPM2), 3.2 (TW1), 3.3 (TW2), 4.2 (WANG1) y 4.3 (WANG2) en un CRAY T3E, para* 32 *procesadores y* $16384 \leq n \leq 524288$.

$S_2 = 1.57$, al igual que la eficiencia, $E_2 = 78.6\%$. Este buen comportamiento es el que hace que el algoritmo 3.3 (TW2) sea el más rápido en muchas situaciones.

En la tabla 5.3 se muestra, para diversos valores de m, el número óptimo de procesadores, si el objetivo es conseguir la mayor rapidez posible en la resolución del sistema tridiagonal. Al igual que en la tabla 5.2, en las columnas etiquetadas con Proc se muestra el número de procesadores óptimo, en las columnas etiquetadas con Alg se muestra el algoritmo con el que se consigue el mejor tiempo y en las columnas etiquetadas con Gauss se muestra el incremento porcentual de tiempo que se produce al resolver el sistema en secuencial por el método de eliminación de Gauss para sistemas tridiagonales sobre el tiempo utilizado en paralelo por el algoritmo más rápido; un 100% de incremento significa que en secuencial se tarda el doble que en paralelo.

CRAY T3E						
	m					
	5			10		
n	Proc.	Alg.	Gauss	Proc.	Alg.	Gauss
16384	$8p$	OPM2	141.17%	$8p$	OPM2	140.34%
32768	$8p$	OPM2	147.52%	$8p$	OPM2	147.09%
65536	$8p$	OPM2	150.83%	$8p$	OPM2	150.61%
131072	$16p$	OPM2	152.98%	$16p$	OPM2	152.75%
262144	$16p$	OPM2	154.48%	$16p$	OPM2	154.36%
524288	$16p$	OPM2	155.24%	$16p$	OPM2	155.18%

(a) $m \in \{5, 10\}$

CRAY T3E						
	m					
	100			1000		
n	Proc.	Alg.	Gauss	Proc.	Alg.	Gauss
16384	$8p$	TW2	131.80%	$16p$	WANG1	108.02%
32768	$8p$	TW2	142.52%	$16p$	WANG1	115.97%
65536	$8p$	TW2	148.24%	$8p$	TW2	130.21%
131072	$8p$	TW2	151.19%	$8p$	TW2	141.63%
262144	$16p$	TW2	153.19%	$8p$	TW2	147.77%
524288	$16p$	TW2	154.59%	$8p$	TW2	150.96%

(b) $m \in \{100, 1000\}$

Tabla 5.3: *Número óptimo de procesadores en un CRAY T3E, para* $16384 \leq n \leq 524288$.

Conclusiones y líneas futuras

La investigación presentada en esta memoria se ha desarrollado con un doble objetivo, por un lado el análisis del modelo de computación paralela BSP según el modelo de coste, para lo cual se han estudiado y analizado diversos métodos para la resolución de sistemas tridiagonales estrictamente diagonal dominantes y por otro la propuesta de nuevos métodos de resolución de este tipo de sistemas que sean óptimos en determinadas situaciones.

Se han propuesto algoritmos BSP para el método de las particiones superpuestas, el método bidireccional (*two-way*) para dos procesadores y el método de Wang así como un nuevo método bidireccional para un número par de procesadores. Se han obtenido resultados experimentales para los distintos algoritmos BSP en un IBM SP2 con conexión *switch* y *ethernet* y en un *cluster* de PC's, observándose que las mayores diferencias entre el tiempo previsto y el experimental se obtienen en los sistemas de tamaño pequeño en los que suele ser mayor el tiempo experimental que el teórico y que a medida que aumenta el tamaño de sistema el tiempo obtenido experimentalmente se ajusta mejor al esperado; para tamaños grandes la desviación ha oscilado entre el 0% y el 5%.

Para el método de las particiones superpuestas se han propuesto dos algoritmos BSP y se ha observado que, por lo general, a partir de unas 17000 ecuaciones resulta más rápido el algoritmo que calcula m (número de ecuaciones superpuestas) en paralelo.

El método bidireccional (*two-way*) para dos procesadores es siempre el más rápido con un diferencia significativa sobre los demás, además tiene una excelente eficiencia en un CRAY T3D y en un CRAY T3E. Basándose en este método y en el método de las particiones superpuestas se ha propuesto un nuevo método bidireccional para un número par de procesadores, para el mismo se han propuesto dos algoritmos BSP y se ha observado

que también resulta más rápido el algoritmo que calcula m en paralelo cuando el tamaño del sistema está por encima de las 50000 ecuaciones.

Para el método de las particiones de Wang se han propuesto dos algoritmos BSP, uno de ellos supone una variación del método que mejora el tiempo de ejecución cuando el número de procesadores es grande.

En la comparación de los distintos algoritmos entre sí y con el método de eliminación de Gauss para sistemas tridiagonales se han obtenido varias conclusiones. En el IBM SP2 y en el *cluster* de PC's es más rápido resolver el sistema en secuencial mediante el método de eliminación de Gauss para sistemas tridiagonales, por tanto los algoritmos BSP propuestos son de utilidad sólo si el objetivo es resolver sistemas de tamaño superior al máximo admitido por un sólo procesador. Esto no sucede en máquinas con comunicación entre procesadores más rápida que en las anteriores, como es el caso del CRAY T3D o del CRAY T3E. El nuevo método bidireccional para un número par de procesadores propuesto es el óptimo en muchas situaciones, es digno de reseñar que en un CRAY T3E el mejor tiempo se consigue con este método usando sólo 8 procesadores, en este caso el tiempo de ejecución es aproximadamente la tercera parte del necesitado por el mejor algoritmo secuencial.

Futuros trabajos de investigación

En muchas de las aplicaciones de diversos campos de la ingeniería se hace necesaria la resolución de muchos sistemas lineales tridiagonales con la misma matriz de coeficientes, haciendo uso de la descomposición LU de la matriz de coeficientes se puede obtener la solución de cada sistema tridiagonal resolviendo dos sistemas bidiagonales. A la vista de los buenos resultados obtenidos por el método bidireccional para dos procesadores y del tipo de factorización que se hace sobre la matriz de coeficientes del sistema, se estudiará la generalización del método bidireccional para un número par de procesadores a la resolución de múltiples sistemas lineales tridiagonales con la misma matriz de coeficientes.

Otro problema a estudiar, en términos parecidos a los de esta memoria, es la resolución de sistemas lineales por bandas.

Bibliografía

[1] A. Aggarwal, A. K. Chandra y M. Snir. On communication latency in PRAM computation. En *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, páginas 11–21, junio 1989.

[2] A. Aggarwal, A. K. Chandra y M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, **71**: 3–28, marzo 1990.

[3] J.C. Agüí y J. Jiménez. A binary tree implementation of a parallel distributed tridiagonal solver. *Parallel Computing*, **21**: 233–241 (1995).

[4] I. Babuska. Numerical stability in problems of linear algebra. *Journal on Numerical Analysis*, **9**: 53–77 (1972).

[5] I. Bar-On y B. Codenotti. A fast and stable parallel QR algortihm for symmetric tridiagonal matrices. *Linear Algebra and its Applications*, **220**: 63–95 (1996).

[6] I. Bar-On, B. Codenotti y M. Leoncini. A fast parallel Cholesky decomposition algorithm for tridiagonal symmetric matrices. *SIAM Journal on Matrix Analysis and Applications*, **18**: 403–418 (1997).

[7] A. Beguelin, J. Dongarra, A. Geist, W. Jiang, R. Manchek y V. Sunderam. PVM 3 Users guide and reference manual. Report, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, mayo 1994.

[8] A. Beguelin, J. Dongarra, A. Geist, R. Manchek y V. Sunderam. A users' guide to PVM parallel virtual machine. Technical Report CS-91-136, University of Tennessee, julio 1991.

[9] A. Beguelin, J. Dongarra, A. Geist, R. Manchek y V. Sunderam. Recent enhancements to PVM. *International Journal of Supercomputing Aplications and High Performance Computing*, (1995).

[10] M. Bessenrodt y H. Weberpals. A fast vector algorithm for solving tridiagonal linear systems. *Parallel Computing*, **9**: 367–372 (1988/89).

[11] G. Bilardi, K.T. Herley, A. Pietracaprina, G. Pucci y P. Spirakis. BSP vs LogP. En *Proceedings of the 8th Annual Symposium on Parallel Algorithms and Architectures*, páginas 25–32, junio 1996.

[12] G. Bilardi y F.P. Preparata. Horizons of parallel computation. *Journal on Parallel and Distributed Computing*, (1995).

[13] C. Bischof, X. Sun y B. Lang. Parallel tridiagonalization two–step band reduction. En *Proceedings of the scalable high–performance computing conference*, páginas 23–27, Los Alamitos, CA 90720–1264, 1994. IEEE Computer Society.

[14] R.H. Bisseling. Basic techniques for numerical linear algebra on Bulk Synchronous Parallel computers. En *First Wokshop on Numerical Analysis and Applications*, volumen 1196, páginas 46–57. Springer-Verlag, Berlin, 1997.

[15] R.H. Bisseling, M.W. Goudreau, J.M.D. Hill, K. Lang, B. McColl, S.B. Rao, D.C. Stefanescu, T. Suel y T. Tsantilas. The BSP Programming Library. Report, Oxford University Computing Laboratory, mayo 1997. `http://www.bsp-worldwide.org`.

[16] R.H. Bisseling y W.F. McColl. Scientific Computing on Bulk Synchronuos Parallel Architectures. Report, University of Utrecht, 1994. Una versión corta aparece en Proceedings 13th IFIP World Computer Congress. Volumen I (1994). B. Pehrson e I. Simon, editores, Elsevier, páginas 509–514.

[17] S. Bondeli. Divide and conquer: A parallel algorithm for the solution of a tridiagonal linear system of equations. *Parallel Computing*, **17**: 419–434 (1991).

[18] O. Bonorden, B. Juurlink, I. von Otte y I. Rieping. The Paderborn University BSP (PUB) Library-Design, Implementation and Performance. Report TR-RSFB-98-063, University of Paderborn, Computer Science Deptartament, 1998.

[19] C.F. Borges y W.B. Gragg. A parallel divide and conquer algorithm for the generalized real symmetric definite tridiagonal eigenvalue problem. En L. Reichel, A. Ruttan y R.S. Varga, editores, *Numerical Linear Algebra*, páginas 11–29. Walter de Gruyer, Berlin, 1993. Proceedings of the Conference in Numerical Linear Algebra and Scientific Computation, Kent (Ohio), USA. marzo, 1992.

[20] R. Bru y J. Marín. Coste BSP del método del Gradiente Conjugado Precondicionado. En *Actas de las VII Jornadas de Paralelismo*, páginas 163–179, Santiago de Compostela, España, septiembre 1996. Universidad de Santiago de Compostela.

[21] R. L. Burden y J. D. Faires. *Análisis numérico*. Grupo Editorial Iberoamérica, Méjico, 1985.

[22] D. A. Burgess, P. I. Crumpton y J. M. D. Hill. Theory, practice, and a tool for BSP performance prediction. En *EuroPar'96*, páginas 697–705, agosto 1996. Número 1124 de Lecture Notes in Computer Science, Springer-Verlag.

[23] A.W. Burks, H.H. Goldstine y J. von Neumann. *Preliminary discussion of the logical design of an electronic computing instrument*, volumen 1, parte 1. U.S. Army Ordenance Department, The Institute of Advanced Study, Princenton 1946, Primera edición, 28 junio de 1946, Segunda edición, 2 septiembre 1947. También en *Papers of John von Newmann on Computing and Computer Theory*, W. Aspray y A. Burks, editores. Volumen 12 de Charles Babbage Institute Reprint Series for the History of Computing, MIT Press, 1987.

[24] L. Chen, J. He y Q. Miller. Algebraic laws for BSP programming. En *Proceedings of Europar '96*, 1996.

[25] J.J. Climent, L. Tortosa y A. Zamora. A BSP recursive divide and conquer algorithm to compute the inverse of a tridiagonal matrix. *Journal of Parallel and Distributed Computing*. En prensa.

[26] J.J. Climent, L. Tortosa y A. Zamora. Comparing the BSP cost of different algorithms for tridiagonal systems. Sometido para publicación.

[27] J.J. Climent, L. Tortosa y A. Zamora. An hybrid algorithm for solving tridiagonal linear systems in a BSP computer. Sometido para publicación.

[28] J.J. Climent, L. Tortosa y A. Zamora. A note on the recursive decoupling method for solving tridiagonal systems. Sometido para publicación.

[29] J.J. Climent, L. Tortosa y A. Zamora. Coste BSP en distintos métodos de resolución de sistemas tridiagonales. En *Actas de las VIII Jornadas de Paralelismo*, páginas 31–40, Cáceres, septiembre 1997. Universidad de Extremadura.

[30] J.J. Climent, L. Tortosa y A. Zamora. Un algoritmo paralelo para el cálculo de la inversa de una matriz tridiagonal. En *Actas de las IX Jornadas de Paralelismo*, páginas 191–198, San Sebastián, septiembre 1998. Universidad del País Vasco.

[31] J.J. Climent, L. Tortosa y A. Zamora. Un nuevo algoritmo BSP para sistemas tridiagonales. En *Actas de las IX Jornadas de Paralelismo*, páginas 183–190, San Sebastián, septiembre 1998. Universidad del País Vasco.

[32] J.J. Climent, L. Tortosa y A. Zamora. An hybrid parallel algorithm for solving tridiagonal linear systems versus the Wang's method in a CRAY T3D BSP computer. En *Actas de las X Jornadas de Paralelismo*, páginas 79–84, La Manga del Mar Menor (Murcia), septiembre 1999. Universidad de Murcia.

[33] J.J. Climent, L. Tortosa y A. Zamora. A recursive decoupling method for solving tridiagonal linear systems in a BSP computer. En *Actas de las X Jornadas de Paralelismo*, páginas 73–78, La Manga del Mar Menor (Murcia), septiembre 1999. Universidad de Murcia.

[34] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian y T. voon Eicken. LogP: Towards a realistic model of parallel computation. En *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, California, mayo 1993.

[35] B.N. Datta. *Numerical Linear Algebra and Applications*. Brooks/Cole Publishing Company, California, 1995.

[36] D.S. Dodson y S.A. Levin. A tricyclic tridiagonal equation solver. *SIAM Journal on Matrix Analysis and Applications*, **13**: 1246–1254 (1992).

[37] S.R. Donaldson, J.M.D. Hill y A. McEwan. Installation and user guide for the Oxford BSP toolset (v1.3) implementation of BSPlib. Report, Oxford University Computing Laboratory, noviembre 1997. `http://www.bsp-worldwide.org`.

[38] S.R. Donaldson, J.M.D. Hill y A. McEwan. Installation and user guide for the Oxford BSP toolset (v1.4) iplementation of BSPlib. Report, Oxford University Computing Laboratory, septiembre 1998. `http://www.bsp-worldwide.org`.

[39] S.R. Donaldson, J.M.D. Hill y D.B. Skillicorn. BSP clusters: high performance, reliable and very low cost. Report PRG-TR-5-98, Oxford University Computing Laboratory, 1998.

[40] S.J. Farr y C. Walshaw. A two-way parallel partition method dor solving tridiagonal systems. Report 93.25, University of Leeds, junio 1993.

[41] M.J. Flynn. Very high speed computing systems. En *Proceedings IEEE*, volumen 14, páginas 1901–1909, 1966.

[42] S. Fortune y J. Wyllie. Parallelism in random access machines. En *Proceedings of the Tenth Annual Symposium on Theory of Computing*, páginas 114–118, 1978.

[43] K.A. Gallivan, R.J. Plemmons y A.H. Sameh. Parallel algorithms for dense linear algebra Computations. *SIAM Review*, , marzo 1990.

[44] G. A. Geist. PVM3: Beyond network computing. En J. Volkert, editor, *Parallel Computation*, volumen 734 de *Lecture Notes in Computer Science*, páginas 194–203. Springer, 1993.

[45] G. A. Geist, J. A. Kohl y P. M. Papadopoulos. PVM and MPI: a comparison of features. Report, University of Tennessee, 1996.

[46] G.A. Geist. Reduction of a general matrix to tridiagonal form. *SIAM Journal on Matrix Analysis and Applications*, **12**: 362–373 (1991).

[47] A. V. Gerbessiotis y L. G. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing*, **22**: 251–267 (1994).

[48] P. B. Gibbons. A more practical PRAM model. En *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, páginas 158–168, 1989.

[49] G.H. Golub y J.M. Ortega. *Scientific Computing. An introduction with Parallel Computing.* Academic Press, Inc., San Diego, CA, 1993.

[50] G.H. Golub y C. van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, 1989.

[51] M. Goudreau, K. Lang, S. Rao, T. Suel y T. Tsantilas. Towards efficiency and portability: Programming the BSP model. En *Proceedings of the 8th Annual Symposium on Parallel Algorithms and Architectures*, páginas 1–12, junio 1996.

[52] M. W. Goudreau, J.M.D. Hill, K. Lang, B. McColl, S. B. Rao, D. C. Stefanescu, T. Suel y T. Tsantilas. A Proposal for the BSP Worldwide Standard Library. Report, Oxford University Computing Laboratory, julio 1996. `http://www.bsp-worldwide.org`.

[53] M. W. Goudreau, K. Lang, S. B. Rao y T.Tsantilas. The Green BSP library. Report CS-TR-95-11, University of Central Florida, Department of Computer Science, junio 1995.

[54] R.T. Gregory y D.L. Karney. *A collection of matrices for testing computational algorithms.* Wiley–Interscience, New York, 1969.

[55] W. Gropp, E. Lusk y A. Skjellum. *Using MPI: Portable Parallel Programming.* MIT Press, Cambridge, MA, 1994.

[56] J.M.D. Hill. Installation and user guide for the Oxford BSP toolset (v1.1) implementation of BSPlib. Report, Oxford University Computing Laboratory, abril 1996. `http://www.bsp-worldwide.org`.

[57] J.M.D. Hill. Installation and user guide for the Oxford BSP toolset (v1.2) implementation of BSPlib. Report, Oxford University Computing Laboratory, agosto 1997. `http://www.bsp-worldwide.org`.

[58] J.M.D. Hill, P.I. Crumpton y D.A. Burgess. Theory, practice, and a tool for BSP performance prediction. En *Lecture Notes in Computer Science, Springer-Verlag*, volumen 1124, páginas 697–705. EuroPar'96, agosto 1996.

[59] J.M.D. Hill, W.F. McColl y D.B.Skillicorn. Questions and Answers About BSP. Report PRG-TR-15-96, Oxford University Computing Laboratory, Oxford 0X1 3QD, noviembre 1996.

[60] J.M.D. Hill y D.B. Skillicorn. Lessons learned from implementing BSP. Report TR-96-21, Oxford University Computing Laboratory, noviembre 1996.

[61] J.M.D. Hill y D.B. Skillicorn. Practical barrier synchronisation. Report TR-96-16, Oxford University Computing Laboratory, agosto 1996.

[62] R. Hockney. A fast Direct solution of Poisson's equation using Fourier analysis. *Journal of ACM*, **12**: 95–113 (1965).

[63] R. Hockney. Peformance Parameters and benchmarking of supercomputers. *Parallel Computing*, **17**: 1111–1130 (1991).

[64] R. Hockney y C.R. Jesshope. *Parallel Computers* 2. Institute of Physics Publishing, Bristol, 1988.

[65] Y. Huang y W. F. McColl. A two-way BSP algorithm for tridiagonal systems. En *HPCN'97*, Viena, abril 1997. Lecture Notes on Computer Science, Springer-Verlag.

[66] B. H. H. Juurlink y H. A. G. Wijshoff. Communication primitives for BSP computers. *Information Processing Letters*, **58**: 303–310 (1996).

[67] R. M. Karp, M. Luby y F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. En *Proceedings of the Twenty-Fourth Annual ACM Symposium of the Theory of Computing*, páginas 318–326, mayo 1992.

[68] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steele Jr. y M.E. Zosel. *The High Performance Fortran Handbook*. MIT Press, Cambridge MA, 1994.

[69] S. Lakshmiravahan y S.K. Dhall. A new class of parallel algorithm for solving linear tridiagonal systems. En *Proceedings Fall Joint Computer Conference*, páginas 315–324, Dallas, TX, 1986.

[70] B. Lang. A parallel algorithm for reducing symmetric banded matrices to tridiagonal form. *SIAM Journal on Scientific Computing*, **14**: 1320–1338 (1993).

[71] J.L. Larriba. Design and evaluation of tridiagonal solvers for vector and parallel computers. Tesis doctoral, Universitat Politècnica de Catalunya, enero 1995. Dirigida por D. J.J. Navarro.

[72] J.L. Larriba, A. Jorba y J.J. Navarro. A Parallel Tridiagonal Solver for Vector Uniprocessors. En *SIAM International Conference on Parallel Scietific Computation*, páginas 590–597, Norfolk, 1993.

[73] J.L. Larriba, A. Jorba y J.J. Navarro. Solution of strictly diagonal dominant tridiagonal systems on vector computers. Report CEPBA 93/09, Universitat Politècnica de Catalunya, septiembre 1993.

[74] J.L. Larriba, J.J. Navarro, A. Jorba y O. Roig. Review of general and Toeplitz vector bidiagonal solvers. *Parallel Computing*, **22**: 1091–1126 (1996).

[75] D. Lecomber. Abstract Data Types in the Bulk Synchronous Parallel Model. Report, Oxford University Computing Laboratory, 1996.

[76] S. Lele. Compact finite schemes with spectral-like resolution. *Journal of Computational Physics*, **103**: 16–42 (1992).

[77] L.M. Losa, V. Migallón y J. Penadés. Métodos iterativos paralelos en dos etapas. Análisis del coste mediante el modelo BSP. En *Actas de las VII Jornadas de Paralelismo*, páginas 181–198, Santiago de Compostela, España, septiembre 1996. Universidad de Santiago de Compostela.

[78] J. Marín y A. Martínez. Testing PVM versus BSP programming. En *Actas de las VIII Jornadas de Paralelismo*, páginas 153–160, Cáceres, septiembre 1997. Universidad de Extremadura.

[79] A. Martínez. Sobre la eficiencia y estabilidad de los algoritmos básicos del algebra lineal en paralelo. Tesis doctoral, Universidad Politécnica de Valencia, septiembre 1998. Codirigida por D. J. Mas y D. B. Codenotti.

[80] A. Martínez y J. Más. Estudio del coste de diferentes estrategias de pivotamiento en el modelo de computación paralela BSP. En *Actas de las VII Jornadas de Paralelismo*, páginas 147–1162, Santiago de Compostela, España, septiembre 1996. Universidad de Santiago de Compostela.

[81] M. Mascagni. Numerical methods for neuronal modelling. En C. Koch y I. Segev, editores, *Methods in Neuronal Modelling:from Synapse to Networks*, páginas 439–484. MIT Press, Cambridge, MA, 1989.

[82] N. Mattor, T.J. Williams y D.W. Hewett. Algorithm for solving tridiagonal matrix problems in parallel. *Parallel Computing*, **21**: 1769–1782 (1995).

[83] W.F. McColl. General purpose parallel computing. En A.M. Gibbons y P. Spirakis, editores, *Lectures on Parallel Computation*, Cambridge International Series on Parallel Computation, páginas 337–391. Cambridge University Press, Cambridge, MA, 1993.

[84] W.F. McColl. Special purpose parallel computing. En A.M. Gibbons y P. Spirakis, editores, *Lectures on Parallel Computation*, Cambridge International Series on Parallel Computation, páginas 261–336. Cambridge University Press, Cambridge, MA, 1993.

[85] W.F. McColl. BSP Programming. En G.E. Blelloch, K.M. Chandy y S. Jagannathan, editores, *Proceedings of the DIMACS workshop Parallel Processing Specialist Group Workshop*, volumen 18 de *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, páginas 21–35. American Mathematical Society, 1994.

[86] W.F. McColl. Scalable computing. En J. Van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volumen 100 de *Lecture Notes in Computer Science*, páginas 46–61. Springer–Verlag, 1995.

[87] V. Mehrmann. Divide and conquer methods for tridiagonal linear systems. En W. Hackbusch, editor, *Notes on Numerical Fluid Mechanics (Parallel Algorithms for Partial Differential Equations), Proceedings of the Sixth GAMM-Seminar*, 1991.

[88] R. Miller. A library for bulk-synchronous parallel programming. En *Proceedings of the British Computer Society Parallel Processing Specialist Group Workshoop on General Purpose Parallel Computing*, páginas 100–108, diciembre 1993.

[89] R. Miller. Two approaches to architecture-independent parallel computation. Tesis doctoral, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, 1994. Dirigida por D. Phil.

[90] R. Miller y J.L. Reed. The Oxford BSP library users' guide. Technical report, Programming Research Group, University of Oxford, 1993.

[91] J.M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York, 1988.

[92] J.M. Ortega, R.G. Voigt y C.H. Romine. A bibliography on parallel and vector numerical algorithms. En *Parallel algorithms for matrix computations*. SIAM, Philadelphia, 1990.

[93] A. Sameh y D. Kuck. On Stable Parallel Linear Solvers. *J. ACM*, **25**: 81–91 (1978).

[94] D.B. Skillicorn. Foundations of Parallel Programming. En *Cambridge Series in Parallel Computation*. Cambridge University Press, 1994.

[95] G. Spaletta y D.J. Evans. The parallel recursive decoupling algorithm for solving tridiagonal linear systems. *Parallel Computing*, **19**: 563–576 (1993).

[96] H. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of ACM*, **20**: 27–38 (1973).

[97] H.S. Stone. Parallel tridiagonal linear equations solvers. *ACM Transactions on Mahematical Software*, **1**: 289–307 (1975).

[98] X.H. Sun, H. Zhang y L.M. Ni. Efficient tridiagonal solvers on multicomputers. *IEEE Transactions on Computers*, **41**: 286–296 (1992).

[99] A.M. Turing. On computable numbers, with an application to the entscheidungsproblem. En *Proceedings of the London Mathematical Society, Series 2*, volumen 42, páginas 230–265, 1936. Corrections, **43**: 544–546, 1937.

[100] L. G. Valiant. Bulk-synchronous parallel computer. *U.S. Patent No. 5083265*, (1992).

[101] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, **33**: 103–111 (1990).

[102] L.G. Valiant. *General purpose parallel architectures*. Handbook of Theoretical Computer Science. North Holland, 1990.

[103] H. A. van der Vorst. Analysis of a parallel solution method for tridiagonal linear systems. *Parallel Computing*, **5**: 303–311 (1987).

[104] C.H. Walshaw. Diagonal dominance in the parallel partition method for tridiagonal systems. *SIAM Journal on Matrix Analysis and Applications*, **16**: 1086–1099 (1995).

[105] H.H. Wang. A parallel method for tridiagonal equations. *ACM Transactions on Mathematical Software*, **7**: 170–183 (1981).

[106] X. Wang y Z.G. Mou. A divide-and-conquer method for solving tridiagonal systems on hypercube massively parallel computers. En *Third IEEE Symposium on Parallel and Distributed Processing*, páginas 810–817. IEEE Computer Society, 1991.