# Una propuesta para el diseño de métodos de Jacobi unilaterales por bloques para el Problema Simétrico de Valores Propios *

Domingo Giménez Cánovas †

**Resumen**

*En este trabajo se realiza un estudio teórico de algoritmos de Jacobi unilaterales por bloques para la resolución del Problema Simétrico de Valores Propios. Se estudian dos métodos secuenciales y sus versiones paralelas, y se comparan los tiempos de ejecución teóricos con los de un método bilateral por bloques.*

***Palabras clave:*** *métodos de Jacobi, Problema Simétrico de Valores Propios.*

## 1 Introducción

Siguiendo el artículo de Chartres [1] estudiaremos el posible diseño de algoritmos de Jacobi unilaterales por bloques. En dicho artículo se estudian dos métodos que no trabajan por bloques y que llamaremos MI y MII. Hay muchas otras variaciones de métodos de Jacobi unilaterales y aplicaciones a otros problemas como el problema de valores singulares [1, 2, 3, 4, 5, 6, 7, 8], pero estudiaremos sólo los dos métodos del artículo de Chartres por simplicidad.

En los métodos de Jacobi unilaterales se calculan valores y vectores propios de matrices reales simétricas, por lo que compararemos los métodos unilaterales por bloques propuestos con métodos bilaterales por bloques con el fin de estudiar si el uso de bloques en los métodos unilaterales puede dar lugar a algoritmos de altas prestaciones que sigan manteniendo el alto grado de paralelismo que puede hacer a los métodos unilaterales competitivos con los bilaterales en algunos tipos de sistemas, como Máquinas Masivamente Paralelas (MPP) o redes de ordenadores.

## 2 Métodos de Jacobi bilaterales

En esta sección estudiaremos métodos de Jacobi bilaterales con el fin de poder compararlos con los unilaterales que analizaremos en las siguientes secciones. El estudio que realizaremos aquí se puede encontrar más detallado en [9, 10, 11, 12, 13].

### 2.1 Métodos que no trabajan por bloques

Un método de Jacobi trabaja construyendo una secuencia de matrices $\{A_l\}$ por medio de

*Parcialmente subvencionado por la Comisión Interministerial de Ciencia y Tecnología, proyecto TIC96-1062-C03-02; Consejería de Cultura y Educación de Murcia, Dirección General de Universidades, proyecto COM-18/96 MAT; y Acción Integrada Hispano-Lusa HP1996-0007.

†Departamento de Informática y Sistemas. Univ de Murcia. Aptdo 4021. 30001 Murcia. (domingo@dif.um.es)

$$A_{l+1} = Q_l A_l Q_l^t \ , \ l = 1, 2, \ldots \tag{1}$$

donde $A_1 = A$, y $Q_l$ es una rotación de Givens en el plano $(p, q)$, con $1 \leq p, q \leq n$. Bajo ciertas condiciones [14, 15], la secuencia $\{A_l\}$ converge a una matriz diagonal $D$,

$$D = Q_k Q_{k-1} \ldots Q_2 Q_1 A Q_1^t Q_2^t \ldots Q_{k-1}^t Q_k^t \tag{2}$$

cuyos elementos diagonales son los valores propios de $A$, y los vectores propios son las columnas de $Q_1^t Q_2^t \ldots Q_{k-1}^t Q_k^t$.

Cada producto $Q_l A_l Q_l^t$ representa una transformación de semejanza que anula un par de elementos no diagonales, $a_{ij}$ y $a_{ji}$, de la matriz $A_l$. La matriz $Q_l$ coincide con la identidad excepto en los elementos $q_{ii} = c$, $q_{ij} = s$, $q_{ji} = -s$, y $q_{jj} = c$, donde $c = \cos\ \theta$, $s = \sin\ \theta$, y [15]

$$\tan\ 2\theta = \frac{2a_{ij}}{a_{ii} - a_{jj}} \quad . \tag{3}$$

Las distintas maneras de elegir los pares $(i, j)$ han dado lugar a diferentes versiones del método de Jacobi. Con el método cíclico por filas los elementos se anulan en el orden dado por los pares:

$$(1, 2), (1, 3), \ldots, (1, n), (2, 3), (2, 4), \ldots, (2, n), \ldots, (n-1, n) \quad . \tag{4}$$

Son necesarias $\frac{n(n-1)}{2}$ anulaciones en cada barrido, y el coste de cada anulación es de 11 flops en el cálculo de los parámetros de las rotaciones y $6n + 12$ flops en la actualización de la matriz, lo que hace $6n + 23$ flops por anulación y $\frac{(6n+23)n(n-1)}{2}$ flops por barrido para el cálculo de los valores propios, lo que da un valor de $3n^3$ flops para los términos de mayor orden. El cálculo de los valores y vectores propios tendrá un coste por barrido de $6n^3$ flops, pues habrá que acumular las rotaciones.

La actualización de la matriz $A$ pre y postmultiplicando puede producir un alto coste de acceso a memoria. Por ejemplo, si los elementos se almacenan por filas, en la postmultiplicación se actualizan columnas cuyos elementos no están contiguos en la memoria, lo que puede producir una reducción de las prestaciones, principalmente con matrices de gran dimensión. Esto no ocurre en los métodos unilaterales, donde sólo se realizan pre o postmultiplicaciones.

## 2.2 Métodos por bloques

Para mejorar el uso de la memoria es posible diseñar métodos por bloques [16, 17]. Siguiendo [9] analizaremos uno de estos métodos para compararlo posteriormente con los métodos unilaterales propuestos.

Si consideramos la matriz $A$ de tamaño $n \times n$ dividida en filas y columnas de bloques cuadrados de tamaño $t$, es posible hacer un barrido sobre estos bloques usando algún orden de Jacobi (aquí se usará el orden par-impar). De este modo, con $n = 8t$, los bloques se tratan en el orden indicado por los números en la figura 1, donde el trabajo se hace en la parte triangular inferior de la matriz. Dentro de cada bloque el algoritmo trabaja haciendo un barrido sobre los elementos en el bloque, lo que quiere decir que se anulará una vez cada elemento del bloque usando algún orden predeterminado. Pero hay algunos elementos no diagonales que no pertenecen a ninguno de los bloques cuadrados numerados en la figura 1 (los elementos subdiagonales en bloques en la diagonal principal). Para anular estos elementos, los bloques 1, 2, 3 y 4 de la figura 1 se consideran de tamaño $2t \times 2t$ añadiendo a cada uno de estos bloques los dos bloques diagonales adyacentes y el bloque simétrico (figura 2).

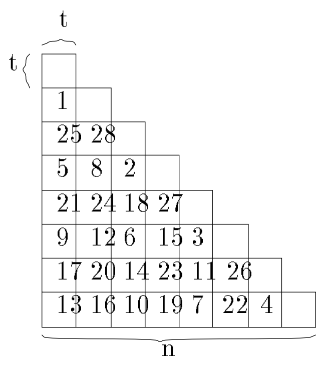


Figura 1: Orden par-impar por bloques.

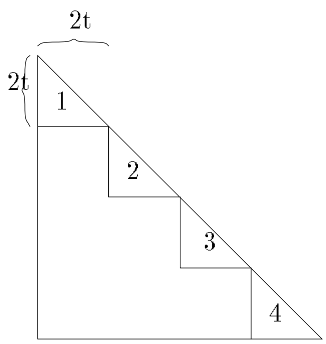


Figura 2: Bloques triangulares.

El algoritmo procesa los bloques en el orden 1, 2, 3, 4, ..., hasta que se completa un barrido, y realiza sucesivos barridos hasta que alcanza la convergencia.

Es posible diseñar un algoritmo rico en operaciones del nivel 3 (en este caso multiplicaciones matriciales) realizando un subbarrido sobre el bloque 1, explotando la simetría de la matriz y usando BLAS 1. Después de esto, si se acumulan las rotaciones sobre una matriz $Q$ de tamaño $2t \times 2t$, se puede actualizar la primera columna de bloques $2t \times 2t$ postmultiplicando $Q$ por esta columna, y esta multiplicación de matrices se puede realizar usando BLAS 3. El algoritmo trabaja sobre los bloques 2, 3 y 4 de manera similar (con los bloques 2 y 3 habrá que hacer pre y postmultiplicaciones por la matriz de rotaciones, y con el bloque 4 sólo premultiplicaciones).

Después de esto se lleva a cabo un movimiento de filas y columnas, de acuerdo con el orden par-impar. Esto es un trabajo adicional pero produce como resultado que los siguientes bloques de tamaño $2t \times 2t$ que se van a anular queden en la diagonal, con lo que será posible trabajar con estos bloques del mismo modo como se había hecho con los bloques 1, 2, 3 y 4, aunque en este caso los elementos a anular están en bloques de tamaño $t \times t$ dentro de bloques diagonales de tamaño $2t \times 2t$.

De este modo, si inicialmente los pares de índices son $\{(1,2),(3,4),(5,6),(7,8)\}$, tras el intercambio de índices se obtiene un nuevo conjunto de pares $\{(1,4),(3,6),(5,8)\}$, e intercambiando filas y columnas de bloques del mismo modo que se hizo con los índices se obtiene una nueva distribución de los bloques (figura 3). Ahora el trabajo se hace sobre los bloques diagonales de tamaño $2t \times 2t$ que aparecen en la figura 3, pero anulando únicamente elementos en los bloques 5, 6 y 7, de tamaño $t \times t$. Después de esto, el algoritmo trabaja de una manera similar con el resto de los bloques hasta que se completa un barrido.

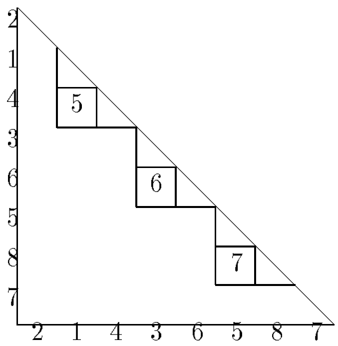


Figura 3: Trabajo sobre bloques cuadrados.

El intercambio de filas y columnas se puede hacer eficientemente usando BLAS 1 para intercambiar y copiar partes de las filas (si la matriz $A$ se almacena por filas) como se muestra en las figuras 4 y 5, donde se muestran los intercambios y las copias que se hacen tras iteraciones impares y pares. En estas figuras una flecha $\longleftrightarrow$ representa un intercambio y una flecha $\rightarrow$ una copia. Parte de este movimiento de datos puede evitarse, ya que consiste en la pre y postmultiplicación de la matriz actualizada $A = QAQ^t$ por una matriz de permutación $P$, y estas multiplicaciones se pueden reorganizar agrupando en la forma $(PQ)A(Q^tP^t)$, de modo que se puede evitar el movimiento de datos en $A$ y hacerlo en $Q$ que tiene casi todos sus elementos nulos, con lo que sólo habría que mover datos en los bloques no nulos de $Q$ (bloques en la diagonal principal de bloques).

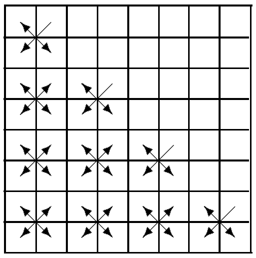

Figura 4: Movimiento de bloques. Paso impar.

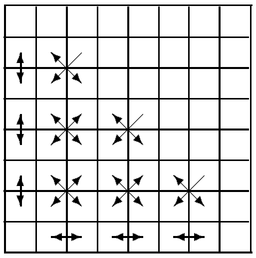

Figura 5: Movimiento de bloques. Paso par.

De este modo, se reduce considerablemente el coste de los accesos a memoria, y no podemos suponer que los métodos unilaterales por bloques sean mejor que los bilaterales en este sentido.

Para calcular los vectores propios es necesario acumular las transformaciones ortogonales, y esto se puede hacer usando multiplicaciones matriciales. Después de un subbarrido sobre un bloque de tamaño $2t \times 2t$, es necesario multiplicar la matriz $2t \times 2t$ donde se han acumulado las transformaciones por las filas de bloques correspondientes de la matriz $Q$ donde se obtendrán los vectores propios. Para hacer esto eficientemente es necesario un movimiento de filas en $Q$ después de cada paso de la ordenación, y este movimiento se hace para agrupar bloques que van a ser tratados juntos en el siguiente paso. Igual que en el cálculo de los valores propios, este movimiento se hace permutando filas en la matriz $2t \times 2t$ donde se acumularon las rotaciones antes de multiplicar la matriz por $Q$.

Según esta explicación, podemos considerar un algoritmo de Jacobi por bloques con el esquema que se muestra en la figura 6. Se realizan sucesivos barridos hasta que se considera que se ha alcanzado la convergencia, en cada barrido se realizan una serie de pasos y en cada paso se calculan matrices de rotaciones realizando un barrido sobre un bloque diagonal y acumulando la matriz de rotaciones, actualizando posteriormente la matriz $A$ utilizando la matriz de rotaciones calculada. Cada paso constará de tantos apartados (cálculo de matriz de rotaciones y actualización de $A$) como pares de índices tenga la ordenación par-impar en ese paso.

Para estudiar el coste del algoritmo sólo consideraremos el número de flops, aunque habrá un tiempo adicional debido al movimiento de datos.

Como se ve en la figura 1, el número de bloques de tamaño $t \times t$ $(q = \frac{n}{t})$ es

```
mientras off(A) > cota
    hasta acabar barrido
        hasta acabar paso
            Realizar barrido sobre bloque
            Actualizar A
            Actualizar matriz de rotaciones
        finpaso
    finbarrido
finmientras
```

Figura 6: Esquema de un algoritmo de Jacobi por bloques.

$$\frac{\frac{n}{t}\left(\frac{n}{t}-1\right)}{2}=\frac{q\,(q-1)}{2} \quad . \tag{5}$$

Los primeros $\frac{q}{2}$ bloques son bloques triangulares de tamaño $2t \times 2t$, con lo que se necesitan $3(2t)^3$ flops para realizar un barrido sobre un bloque y $3(2t)^3$ flops para acumular las rotaciones, con lo que se obtienen $6(2t)^3$ flops.

En el resto de los bloques sólo se anulan $t^2$ elementos, con lo que se necesitan $24t^3$ flops por bloque.

De este modo, el número total de flops necesarios para llevar a cabo un barrido sobre todos los bloques es:

$$\frac{q}{2}6(2t)^3+\left(\frac{q^2}{2}-q\right)24t^3=12n^2t \qquad flops. \tag{6}$$

Los dos valores extremos de $t$ son 1 y $\frac{n}{2}$, con lo que los valores extremos en (6) son $12n^2$ y $6n^3$, pero se obtienen tiempos de ejecución óptimos con valores moderados de $t$.

En el algoritmo se realizan también actualizaciones de filas y columnas de bloques. Después de un barrido sobre un bloque de tamaño $2t \times 2t$, se actualizan matrices de tamaños $2t \times (n-2t-i)$ y $i \times 2t$ (donde $i$ toma los valores $0,\ t,\ 2t, \ldots,\ n-2t$), y estas actualizaciones se realizan pre y postmultiplicando por matrices de tamaño $2t \times 2t$, con lo que se necesitan $2\,(2t)^2\,(n-2t) = 8t^2\,(n-2t)$ flops. Como la cantidad de bloques $t \times t$ es $\frac{q(q-1)}{2}$, el coste de actualización de la matriz es:

$$\frac{q(q-1)}{2}\,8t^2\,(n-2t)=4n^3-12n^2t+8nt^2 \quad flops. \tag{7}$$

De esta manera, sumando las fórmulas (6) y (7) se obtiene el coste del algoritmo (si $t$ es pequeño), pero es importante observar que la fórmula (6) está afectada por un coeficiente correspondiente al nivel 1, y la (7) por un coeficiente correspondiente al nivel 3. Así, se obtiene un coste de $4n^3$ flops, pero este coste es obtenido con operaciones de nivel 3, y es posible que se obtenga una reducción en el tiempo de ejecución con respecto al obtenido con una versión del método de Jacobi que use operaciones de nivel 1. El coste es:

$$4k_3n^3+12\,(k_1-k_3)\,n^2t+8k_3nt^2 \quad flops. \tag{8}$$

donde $k_1$ y $k_3$ son constantes que afectan al coste en flops cuando se usa BLAS 1 y BLAS 3, respectivamente.

La mejora obtenida depende de las implementaciones de BLAS 1 y BLAS 3 disponibles en la máquina donde se ejecute el programa pero, si $t$ es pequeña en comparación con $n$ (como se deduce de los resultados experimentales), el cociente entre los tiempos de ejecución usando BLAS 1 y BLAS 3 será

$$\frac{3k_1}{4k_3} \tag{9}$$

El coste de actualizar la matriz de rotaciones en cada iteración es $2(2t)^2n = 8t^2n$, y son necesarios $\frac{q(q-1)}{2}$ pasos por barrido. Con lo que el coste por barrido de actualizar la matriz de rotaciones es:

$$4n^3 - 4n^2t \qquad flops. \tag{10}$$

Por tanto, el coste por barrido cuando se computan los valores y los vectores propios es:

$$8k_3n^3 + (12k_1 - 16k_3)n^2t + 8k_3nt^2 \qquad flops \tag{11}$$

y en este caso se obtiene el mismo cociente (9) que se obtenía en el cálculo de los valores propios.

Con este esquema se pueden conseguir buenas prestaciones en monoprocesadores y multiprocesadores de Memoria Compartida, y es posible diseñar también algoritmos eficientes para Memoria Distribuida [11].

### 2.3 Métodos paralelos por bloques

Para obtener un algoritmo paralelo para memoria distribuida debemos determinar la distribución de la matriz en el sistema de procesadores. Consideraremos la matriz dividida en bloques y estos bloques agrupados en procesos lógicos tal como se muestra en la figura 7. En esta figura se muestra la división del trabajo en procesos y la memoria reservada en cada proceso, en los pasos impares y pares. En la figura $\frac{n}{t} = 16$ y las filas y columnas de bloques se numeran de 0 a 15, y cada bloque de tamaño $t \times t$ se representa por una $\mathtt{x}$. Estos bloques se agrupan en bloques más grandes, agrupando $2k$ filas y columnas de bloques consecutivas de tamaño $t \times t$ (en la figura $k = 2$).

Para obtener un algoritmo en un sistema concreto habrá que asignar cada uno de los procesos lógicos a un procesador. De esta manera, es posible obtener un algoritmo para malla triangular asignando a cada uno de los procesadores uno de los bloques de tamaño $2tk \times 2tk$ [12], o un algoritmo para malla cuadrada plegando los procesos lógicos sobre los procesadores [9], o en una red de ordenadores asignando un número determinado de procesos a cada procesador intentando balancear la carga [13], o para anillo asignando a cada procesador dos filas de bloques de manera que el trabajo esté balanceado.

El esquema de un algoritmo de Jacobi paralelo por bloques sería el de la figura 8. En el cálculo de las matrices de rotaciones intervienen sólo los procesadores a los que se han asignado procesos diagonales. Después las matrices de rotaciones se envían a los demás procesos en la misma fila y columna de procesos (figura 7). En la actualización de la matriz $A$ y de la matriz de rotaciones deben intervenir todos los procesadores pues estas dos partes son las que producen el mayor coste del algoritmo. Después de las actualizaciones se producirá un movimiento de datos (filas y columnas de bloques) de manera que se obtenga la agrupación de los datos adecuada para llevar a cabo el siguiente paso de acuerdo con la ordenación par-impar que se está usando.

El coste del algoritmo dependerá del sistema donde se mapeen los procesos lógicos. Consideraremos por simplicidad una malla triangular. Además, el uso de una malla triangular nos

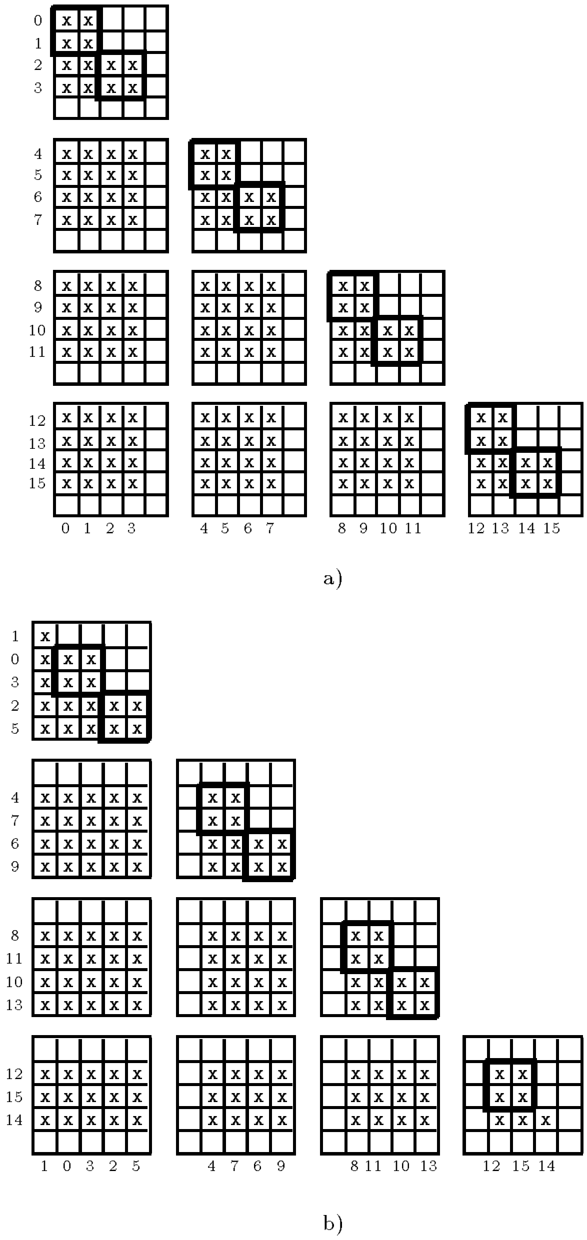


Figura 7: División de la matriz en procesos lógicos en los pasos impares y pares.

```
mientras off(A) > cota
    hasta acabar barrido
        Realizar barrido sobre bloques
        Difundir matrices de rotaciones
        Actualizar A
        Actualizar matriz de rotaciones
        Transferir datos
    finbarrido
finmientras
```

Figura 8: Esquema de un algoritmo de Jacobi paralelo por bloques.

permitirá obtener mejores resultados que si utilizamos una topología que nos obligue a una distribución más irregular de los datos, como por ejemplo una malla cuadrada [12].

En cada paso de un barrido los procesadores en la diagonal principal de la malla de procesadores computan un total de $\frac{n}{2t}$ ó $\frac{n}{2t}-1$ matrices de rotaciones, computando cada uno de estos procesadores $\frac{n}{2rt}=b$ ó $\frac{n}{2rt}-1=b-1$ matrices, con $r=\frac{-1+\sqrt{1+8p}}{2}$ (suponemos $p$ procesadores), y cuando el número de procesadores es grande $r \simeq \sqrt{2}\sqrt{p}$. De esta manera, el coste de computar rotaciones en cada paso es:

$$b48t^3 = \frac{24nt^2}{r} \qquad flops \tag{12}$$

en el primer paso y:

$$b24t^3 = \frac{12nt^2}{r} \qquad flops \tag{13}$$

en los demás pasos.

Con lo que el coste por barrido de computar rotaciones es:

$$k_1\frac{12n^2t}{r} + k_1\frac{12nt^2}{r} \qquad flops. \tag{14}$$

En la difusión las matrices de rotaciones necesitan pasar a través de $r-1$ enlaces hasta llegar a los procesadores destino. Cada procesador en la diagonal principal computa $b$ (o $b-1$) matrices de rotaciones de tamaño $2t \times 2t$, y envía $4bt^2=\frac{2nt}{r}$ datos. Por tanto, el coste de difundir rotaciones en cada paso del algoritmo es:

$$\beta(r-1) + \tau 2nt\left(1-\frac{1}{r}\right) \quad . \tag{15}$$

Donde $\beta$ y $\tau$ representan el tiempo de inicio de una comunicación (start-up time) y de envío de un dato (word-sending time), respectivamente.

Y el coste por barrido es:

$$\beta\frac{n}{t}(r-1) + \tau 2n^2\left(1-\frac{1}{r}\right) \quad . \tag{16}$$

Para la actualización de la matriz, en los pasos impares cada procesador no en la diagonal principal tiene la misma cantidad de trabajo que realizar: la actualización, pre y postmultiplicando, de $\frac{n^2}{r^2}$ datos, lo que se hace en $\frac{8n^2t}{r^2}$ flops.

En los pasos pares, los procesadores en la columna 0 tienen además que actualizar premultiplicando $t\frac{n}{r}$ datos, lo que se hace en $\frac{4t^2n}{r}$ flops, necesitándose $\frac{8n^2t}{r^2} + \frac{4t^2n}{r}$ flops para actualizar la matriz en los pasos pares.

El coste por barrido de actualizar la matriz será:

$$k_3\frac{8n^3}{r^2} + k_3\frac{2n^2t}{r} \qquad flops. \tag{17}$$

En la transferencia de bloques entre los procesadores, en los pasos impares los procesadores que realizan más trabajo envían $\frac{n}{r}t$ datos al oeste, reciben $\frac{n}{r}t$ del este, envían $\left(\frac{n}{r}+t\right)t$ al norte y reciben $\left(\frac{n}{r}+t\right)t$ del sur, con lo que tenemos un coste $4\beta + \tau\left(\frac{4nt}{r}+2t^2\right)$.

Y en los pasos pares los procesadores que realizan más trabajo envían $\left(\frac{n}{r}+t\right)t$ datos al sur, reciben $\left(\frac{n}{r}+t\right)t$ datos del norte, envían $\frac{n}{r}t$ al este y reciben $\frac{n}{r}t$ del oeste, con lo que tenemos un coste $4\beta + \left(\frac{4nr}{t}+2t^2\right)\tau$.

El coste de transferir datos por barrido es por tanto:

$$\frac{4n}{t}\beta + \left(\frac{4n^2}{r} + 2nt\right)\tau \quad . \tag{18}$$

El coste aritmético total por barrido es (sumando 14 y 17):

$$T_a = 8k_3\frac{n^3}{r^2} + (12k_1 + 2k_3)\frac{n^2t}{r} + 12k_1\frac{nt^2}{r} \qquad flops. \tag{19}$$

Cuando el número de procesadores es grande se obtiene un coste:

$$T_a = 4k_3\frac{n^3}{p} + \left(6\sqrt{2}k_1 + \sqrt{2}k_3\right)\frac{n^2t}{\sqrt{p}} + 6\sqrt{2}k_1\frac{nt^2}{\sqrt{p}} \qquad flops. \tag{20}$$

Y el coste de las comunicaciones por barrido es (sumando 16 y 18):

$$T_c = (r+3)\frac{n}{t}\beta + \left(2n^2 + \frac{2n^2}{r} + 2nt\right)\tau \quad . \tag{21}$$

Con lo que la eficiencia teórica del algoritmo paralelo con respecto al algoritmo secuencial que usa BLAS 3 es del 100%.

Si consideremos que se quieren calcular también los vectores propios, habrá que determinar la distribución de la matriz de rotaciones. Consideraremos una distribución dividiendo la matriz de vectores propios en bloques de tamaño $2t \times 2t$ y estos bloques agrupados en bloques de tamaño $\frac{n}{r} \times \frac{n}{r}$, como hacíamos con la matriz $A$. Los bloques se asignarán a los procesadores como se muestra en la figura 9. También en este caso habrá necesidades adicionales de memoria para los pasos pares.

La actualización de la matriz de rotaciones tendrá un coste por barrido de:

$$k_3\frac{8n^3}{r^2} \quad flops \tag{22}$$

y el coste por barrido de la transferencia de datos de la matriz de rotaciones es:

$$4\frac{n}{t}\beta + \left(\frac{4n^2}{r} + 2nt\right)\tau \quad . \tag{23}$$

Sumando las ecuaciones 19 y 22 obtenemos el tiempo aritmético cuando se calculan valores y vectores propios:

$$T_a = 16k_3\frac{n^3}{r^2} + (12k_1 + 2k_3)\,\frac{n^2 t}{r} + 12k_1\frac{nt^2}{r} \qquad flops, \tag{24}$$

y sumando las ecuaciones 21 y 23 el coste de las comunicaciones:

$$T_c = (r+7)\,\frac{n}{t}\beta + \left(2n^2 + \frac{6n^2}{r} + 4nt\right)\tau \quad . \tag{25}$$

| | | | |
|---|---|---|---|
| $P_{00}$ | $P_{10}$ | $P_{20}$ | $P_{30}$ |
| $P_{10}$ | $P_{11}$ | $P_{21}$ | $P_{31}$ |
| $P_{20}$ | $P_{21}$ | $P_{22}$ | $P_{32}$ |
| $P_{30}$ | $P_{31}$ | $P_{32}$ | $P_{33}$ |

Figura 9: Distribución de la matriz de rotaciones.

# 3 Método de Jacobi unilateral. Primera versión.

## 3.1 Método que no trabaja por bloques

Analizaremos el primero de los métodos de Jacobi unilaterales que aparecen en el artículo de Chartres [1], y que llamamos MI. El método trabaja sobre las matrices $B_0 = A$ y $W_0 = I$ y obtiene $B_{r+1} = V_r B_r$, $W_{r+1} = V_r W_r$, con $V_r$ matriz de rotaciones que anula un elemento no diagonal de la matriz $A_r = B_r W_r^t = V_{r-1}V_{r-2}\ldots V_0 B_0 W_0^t V_0^t \ldots V_{r-1}^t$.

Para la anulación del elemento $a_{ij}$ hay que computar $a_{ii}$, $a_{jj}$ y $a_{ij}$, ya que no se trabaja directamente sobre la matriz $A_r$, sino que ésta se conoce por $B_r$ y $W_r$. Estos elementos se calculan por medio de tres productos escalares cuyo coste es $6n-3$ flops. Después se actualizan las filas $i$ y $j$ de $B_r$ y $W_r$, lo que tiene un coste de $12n$ flops, y el coste por barrido es:

$$9n^3 - \frac{21}{2}n^2 + \frac{3}{2}n \qquad flops. \tag{26}$$

Si los elementos diagonales se almacenan en un vector auxiliar no es necesario recalcular cada vez $a_{ii}$ y $a_{jj}$, con lo que el coste por barrido será:

$$7n^3 - \frac{15}{2}n^2 + \frac{n}{2} \qquad flops. \tag{27}$$

Comparando con el método bilateral, vemos que MI es peor en cuanto a tiempo de ejecución, además de que hay que calcular obligatoriamente los vectores propios, lo que no ocurre con el método bilateral. Pero a pesar de tener un mayor coste que el bilateral, el unilateral tiene la ventaja de hacer todo el trabajo por filas (o por columnas), lo que puede propiciar un mejor uso de la memoria y un tiempo de ejecución menor que el bilateral con matrices grandes al disminuir el coste de los accesos a memoria.

En cuanto a la precisión de los resultados los dos métodos se comportan de manera similar, al menos con matrices pequeñas [18].

## 3.2 Método por bloques

Propondremos un método unilateral por bloques combinando las ideas de MI y del método bilateral por bloques presentado. Lo que pretendemos es comprobar si en algún caso un método unilateral por bloques puede ser competitivo con el bilateral.

Hay algunos trabajos previos sobre métodos de Jacobi unilaterales por bloques: en [19] se estudia un método para memoria compartida, y en [20] para memoria distribuida, pero en este método el número de procesadores tiene que ser $p = \frac{n}{2t}$, lo que hace que sea poco flexible en cuanto a poder tomar un tamaño de bloque óptimo independientemente del número de procesadores de que dispongamos.

Consideraremos las matrices $B$ y $W$, de tamaño $n \times n$, divididas en bloques de tamaño $t \times n$, y se tratarán los bloques de la matriz $A = BW$ según la ordenación par-impar, tal como hacíamos con la matriz $A$ en el método bilateral (figura 1).

Inicialmente se tratan los $\frac{n}{2t}$ bloques correspondientes al primer conjunto de Jacobi (los bloques 1, 2, 3 y 4 en la figura 1) haciendo un barrido bilateral sobre los bloques de tamaño $2t \times 2t$ de la matriz $A$. Estos bloques son los bloques correspondientes de la matriz $B$, por lo que habrá que copiarlos en otras posiciones de memoria antes de realizar el barrido. El coste es de $48t^3$ flops ya que hay que acumular las rotaciones. Estas operaciones se hacen con BLAS 1, por lo que el coste del cálculo de las $\frac{n}{2t}$ primeras matrices de rotaciones es $24k_1nt^2$ flops.

Después se actualizan las matrices $B$ y $W$ multiplicando las matrices de rotaciones, de tamaño $2t \times 2t$, por los bloques correspondientes de $B$ y $W$, de tamaño $2t \times n$ ya que en este caso $B$ y $W$ no son simétricas. El coste de cada par de multiplicaciones es $16nt^2$ flops, y el de los $\frac{n}{2t}$ primeros pasos $8n^2t$ flops.

En el método bilateral se realizaba un intercambio de filas y columnas de bloques para obtener la distribución de los datos correspondientes al siguiente conjunto de Jacobi (figura 4). Esto se hacía simplemente cambiando filas en las matrices de rotaciones antes de llevar a cabo la actualización de $A$. En el método unilateral también se podrían hacer los cambios de filas en la matriz de rotaciones y, tras la actualización, $B$ y $W$ quedarían modificadas habiendo intercambiado únicamente las filas de bloques (figura 10).

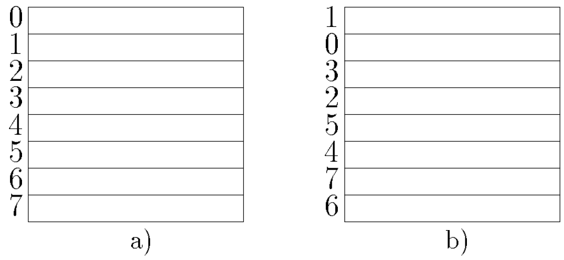


Figura 10: Distribución de las matrices $B$ y $W$: a) inicialmente, b) tras la aplicación del primer conjunto de rotaciones.

En los pasos sucesivos en el método unibilateral habría que obtener $A_{ii}$, $A_{jj}$ y $A_{ij}$, pues no se trabaja directamente con la matriz $A$. Si llamamos $B_i$ y $W_i$ a los $i$-ésimos bloques de $B$ y $W$ en la figura 10.a), se obtiene $A_{ii} = B_iW_i^t$, $A_{jj} = B_jW_j^t$ y $A_{ij} = B_iW_j^t$. El cálculo de $A_{ii}$, $A_{jj}$ y $A_{ij}$ tiene un coste de $6k_3nt^2$ flops. Si se hubieran guardado actualizados los bloques diagonales de $A$ no habría que recalcular $A_{ii}$ y $A_{jj}$, reduciéndose el coste de este paso a $2k_3nt^2$ flops. Como en cada barrido se tratan $\frac{q(q-1)}{2}$ ($\frac{n}{t} = q$) bloques $A_{ij}$, el coste del cálculo de los bloques por barrido (salvo en el primer barrido donde para el primer conjunto de Jacobi no hace falta calcular los bloques pues $B$ contiene inicialmente los elementos de $A$) es:

$$3k_3n^3 - 3k_3n^2t \quad flops \tag{28}$$

si no se guardan los bloques diagonales, y

$$k_3n^3 - k_3n^2t \quad flops \tag{29}$$

si se guardan.

Una vez calculados los bloques $A_{ii}$, $A_{jj}$ y $A_{ij}$ se formará la matriz de tamaño $2t \times 2t$ que los contiene y se hará un barrido bilateral sobre esa matriz acumulando las rotaciones. El coste de cada uno de estos barridos bilaterales será $24k_1t^3$, ya que sólo se anulan los elementos de $A_{ij}$. Y como hay $\frac{q^2}{2} - q$ bloques (sin contar los $\frac{q}{2}$ del primer conjunto) el coste del cálculo de estos bloques será

$$12k_1n^2t - 24k_1nt^2 \quad flops \tag{30}$$

y el coste de los subbarridos sobre los bloques en un barrido es:

$$12k_1n^2t \quad flops. \tag{31}$$

En la actualización de $B$ y $W$ tenemos un coste $16k_3nt^2$ por actualización y un total de $\frac{q(q-1)}{2}$ actualizaciones por barridos, con lo que el coste por barrido es:

$$8k_3n^3 - 8k_3n^2t \quad flops. \tag{32}$$

El coste total por barrido se obtiene sumando las ecuaciones 28, 31 y 32 cuando no se guardan los bloques diagonales, y es:

$$11k_3n^3 + (12k_1 - 11k_3)\, n^2t \quad flops, \tag{33}$$

y cuando se almacenan los bloques principales se obtiene sumando las ecuaciones 29, 31 y 32, y es:

$$9k_3n^3 + (12k_1 - 9k_3)\, n^2t \quad flops. \tag{34}$$

Comparando con la ecuación 11 observamos que el método unilateral tiene mayor coste que el bilateral cuando se calculan valores y vectores propios, tal como pasaba con los métodos que no trabajan por bloques. No se puede considerar que en este caso la ventaja del método unilateral de un mejor uso de la memoria se mantenga, pues el uso de bloques ya nos permite hacer un buen uso de la memoria en el método bilateral.

### 3.3 Método paralelo por bloques

Si consideramos una asignación de las matrices $B$ y $W$ al sistema de procesadores por filas consecutivas de bloques, habría que asignar a cada procesador $k$ bloques consecutivos de tamaño $2t \times 2t$, tal como se muestra en la figura 11, siendo $n = 2tkp$.

Un inconveniente de esta distribución es que dará lugar a un algoritmo para anillo lógico, por lo que previsiblemente tendrá un bajo grado de escalabilidad. Este problema se puede solventar diseñando un algoritmo para malla siendo $n = 2tk\sqrt{p}$.

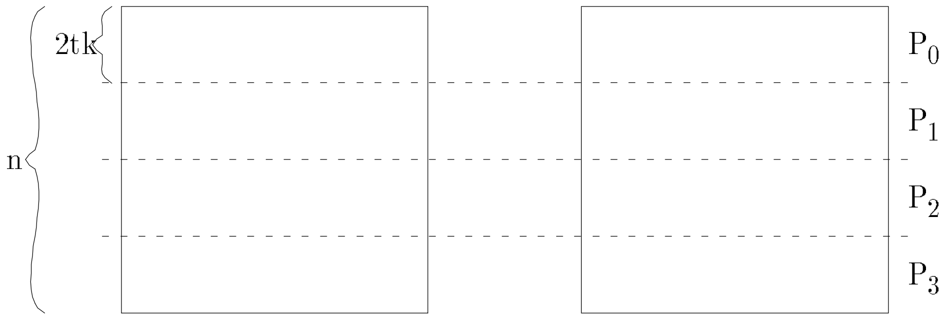


Figura 11: Distribución inicial de las matrices $B$ y $W$ en el sistema de procesadores.

En la figura 11 se muestra sólo la distribución de la matriz pero, tal como ocurría en el algoritmo bilateral, el movimiento de filas de bloques en el sistema de procesadores producirá unas necesidades adicionales de memoria, con lo que en cada procesador habrá que reservar memoria para $(2k+1)tn$ datos de $B$ y otros tantos de $W$.

En el primer paso cada procesador calcula (haciendo un barrido bilateral) $k$ matrices de rotaciones, con coste $24k_1\frac{nt^2}{p}$. En pasos sucesivos los procesadores con más trabajo también calculan $k$ matrices de rotaciones, pero en este caso con coste $12k_1\frac{nt^2}{p}$. El coste total por barrido del cálculo de las matrices de rotaciones es:

$$12k_1\frac{n^2t}{p} + 12k_1\frac{nt^2}{p} \quad flops. \tag{35}$$

Antes de calcular las matrices de rotaciones hay que obtener las $A_{ij}$ (si mantenemos $A_{ii}$ y $A_{jj}$ almacenadas), lo que añade un coste por barrido de:

$$k_3\frac{n^3}{p} \quad flops. \tag{36}$$

La actualización de $B$ y $W$ se hace actualizando cada procesador (al menos los procesadores que más trabajan) $k$ matrices de dimensión $2t \times n$ de $B$, y otras tantas de $W$, en cada paso, con lo que el coste por barrido es:

$$8k_3\frac{n^3}{p} \quad flops. \tag{37}$$

Sumando las ecuaciones 35, 36 y 37 obtenemos el coste aritmético por barrido:

$$T_a = 9k_3\frac{n^3}{p} + 12k_1\frac{n^2t}{p} + 12k_1\frac{nt^2}{p} \quad flops. \tag{38}$$

Con este algoritmo no hay necesidad de difundir las matrices de rotaciones pues cada procesador actualiza las filas de bloques que contiene. Las únicas comunicaciones son las que se hacen para agrupar los datos en los procesadores para que se pueda llevar a cabo el siguiente paso de acuerdo con la ordenación par-impar. En los pasos impares se envía de cada procesador $P_i$ al $P_{i-1}$, con $i = 1, 2, \ldots, p-1$, un bloque $t \times n$ de $B$, otro de $W$, y otro $t \times t$ correspondiente a un bloque diagonal; y en los pasos pares se realizan las mismas comunicaciones pero hacia el sur. De este modo el coste de las comunicaciones por barrido es:

$$T_c = 2\frac{n}{t}\beta + \left(4n^2 + 2nt\right)\tau \quad . \tag{39}$$

Este valor se obtiene suponiendo que se agrupan todos los datos para llevar a cabo la comunicación. Si no se agruparan los datos el término que afecta a $\tau$ sería igual, y sólo se vería afectado por una constante el término en $\beta$.

No podemos comparar las comunicaciones del algoritmo unilateral con las del bilateral pues la topología es distinta, pero lo que sí está claro es que con el unilateral se evita la difusión de las rotaciones, lo que hace que sea más adecuado para programación paralela.

Sólo nos queda decir que, si se almacenan los bloques diagonales, en cada procesador habría que almacenar $\frac{n}{tp}$ bloques $t \times t$, en una memoria de tamaño $\frac{nt}{p} + t^2$ pues hay que reservar memoria para un bloque más debido al movimiento de datos entre los pasos impares y pares.

Para mejorar la escalabilidad se podría usar una malla cuadrada, lo que produciría un desbalanceo de la carga en el cálculo de las matrices de rotaciones, la necesidad de difundir las matrices de rotaciones en cada fila de procesadores, la de obtener cada $A_{ij}$ de manera distribuida interviniendo los procesadores en la misma fila, y en la transferencia de datos habría que transferir filas y columnas. De esta manera habría mayor desbalanceo y más comunicaciones, pero el algoritmo podría ser más escalable al ser para malla y permitir también solapamiento entre computación y comunicación.

# 4 Método de Jacobi unilateral. Segunda versión.

## 4.1 Método que no trabaja por bloques

El segundo método unilateral propuesto en el artículo de Chartres tiene la ventaja de tener un menor coste computacional, pero el inconveniente de tener una menor precisión [18].

El método consiste en la diagonalización de la matriz $B = A^2$ pero sin formar explícitamente $B$. Se trabaja con la matriz $A$ aplicando rotaciones $V$ que anulen elementos $b_{ij}$ de $B$. De este modo, si inicialmente $A_1 = A$ y $B_1 = A_1 A_1^t$, tendremos que $A_{r+1} = V_r A_r$ (con lo que sólo se actualiza $A$ por un lado), y $B_r = A_r A_r^t$ (con lo que habrá que realizar productos escalares para obtener los valores $b_{ii}$, $b_{jj}$ y $b_{ij}$ que se necesitan para obtener la siguiente rotación de Jacobi).

Por tanto, el coste por barrido será aproximadamente $6n^3$ flops si no se almacenan los valores de la diagonal de $B$, y $4n^3$ flops si sí se almacenan.

Además, el método tiene algunos problemas derivados de que no se calculan directamente los valores propios de $A$, sino los de $A^2$ [1, 18]. Los vectores propios se obtienen una vez que se cumple el criterio de parada, que puede ser que no se aplique ninguna rotación en un barrido, y se llega a $A_r = DW$ con $D$ una matriz diagonal y $W$ la matriz cuyas filas son los vectores propios que estamos calculando.

## 4.2 Método por bloques

Para diseñar un método por bloques consideramos la matriz $A$ de tamaño $n \times n$ dividida en bloques contiguos de filas, siendo estos bloques de tamaño $t \times n$. Se tratarán los bloques de la matriz $A^2$ según la ordenación par-impar, tal como hacíamos con la matriz $A$ en el método bilateral. Usando esa ordenación, los bloques de tamaño $t \times t$ de $A^2$ se tratarán en el orden en que se indica en la figura 1.

Dado que el método es similar al estudiado con anterioridad no estudiaremos cada uno de sus pasos con detenimiento, sino que analizaremos su coste.

Antes de cada subbarrido sobre un bloque hay que calcular $B_{ii}$, $B_{jj}$ y $B_{ij}$, o sólo $B_{ij}$ si se mantienen actualizados los bloques diagonales. Aunque se mantengan actualizados los bloques diagonales, en los primeros pasos habrá que calcularlos todos pues no trabajamos sobre $B$, sino sobre $A$. Analizaremos el coste por barrido sin tener en cuenta el trabajo adicional de calcular los bloques diagonales inicialmente, y suponiendo que se mantienen actualizados dichos bloques. El coste de calcular cada matriz $t \times t$ será $2k_3nt^2$ flops, y el coste por barrido del cálculo de las matrices $B_{ij}$:

$$k_3n^3 - k_3n^2t \quad flops. \tag{40}$$

Los primeros $\frac{n}{2t}$ subbarridos se realizan sobre bloques de tamaño $2t \times 2t$, por lo que el coste total de estos subbarridos es de $48k_1t^3\frac{n}{2t} = 24k_1nt^2$ flops. El resto de los subbarridos se realiza sobre bloques $t \times t$ dentro de bloques $2t \times 2t$, y el número de estos subbarridos por barrido es $\frac{n^2}{2t^2} - \frac{n}{t}$. El coste total por barrido de los subbarridos sobre los bloques será:

$$12k_1n^2t \quad flops. \tag{41}$$

En cada actualización de la matriz $A$ se multiplica un bloque $2t \times 2t$ por otro $2t \times n$, con un coste de $8nt^2$ flops. En las actualizaciones se incluirá la multiplicación por la matriz de permutaciones que nos asegure la agrupación de los bloques según la ordenación par-impar. El coste de las actualizaciones por barrido será:

$$4k_3n^3 - 4k_3n^2t \quad flops. \tag{42}$$

Sumando las ecuaciones 40, 41 y 42 se obtiene el coste total por barrido que es:

$$5k_3n^3 + (12k_1 - 5k_3)\, n^2t \quad flops. \tag{43}$$

### 4.3 Método paralelo por bloques

La distribución de la matriz $A$ y de la matriz $D$ de tamaño $n \times t$ donde se almacenan los bloques diagonales sería la que se muestra en la figura 12. Aunque esa es la distribución de los datos en los procesadores, también en este caso se necesitan posiciones adicionales de memoria, con lo que en cada procesador se tendrá memoria de tamaño $\left(\frac{n}{p} + t\right) \times n$ para la matriz $A$ y $\left(\frac{n}{p} + t\right) \times t$ para la matriz $D$.

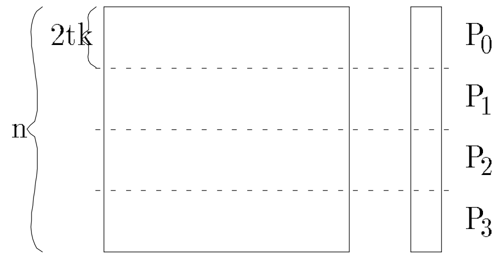


Figura 12: Distribución inicial de las matrices $A$ y $D$ en el sistema de procesadores.

Cada procesador calcula $\frac{n}{2tp}$ bloques $B_{ij}$ (salvo un procesador en los pasos pares), con lo que el cálculo en cada procesador tiene un coste $k_3\frac{n^2t}{p}$ flops, y el coste por barrido es:

$$k_3 \frac{n^3}{p} \qquad flops. \tag{44}$$

Cada procesador realiza $\frac{n}{2tp}$ subbarridos sobre bloques $2t \times 2t$ en el primer paso, y los mismos subbarridos sobre bloques $t \times t$ (salvo un procesador en los pasos pares) en los demás pasos. El coste por barrido de la realización de los subbarridos será:

$$k_1 \left( \frac{12n^2t}{p} + \frac{12nt^2}{p} \right) \qquad flops. \tag{45}$$

En la actualización de la matriz cada procesador (salvo uno en los pasos pares) actualiza $\frac{n}{2tp}$ bloques $2t \times n$, con lo que el coste por barrido es:

$$4k_3 \frac{n^3}{p} \qquad flops. \tag{46}$$

Y el coste aritmético total por barrido es (sumando 44, 45 y 46):

$$T_a = 5k_3 \frac{n^3}{p} + 12k_1 \frac{n^2t}{p} + 12k_1 \frac{nt^2}{p} \qquad flops. \tag{47}$$

En cuanto a las comunicaciones, las únicas que se realizan son las de transferencia de datos, enviándose $t \times (n+t)$ datos hacia el norte en los pasos impares y hacia el sur en los pares, y realizándose en cada procesador no extremo un envío y una recepción de esa cantidad de datos en cada paso. El coste de las comunicaciones será por barrido:

$$T_c = \frac{2n}{t} \beta + \left(2n^2 + 2nt\right) \tau \qquad . \tag{48}$$

# 5 Comparación

En la tabla 1 comparamos los costes de los diversos métodos paralelos por bloques analizados. También incluimos los costes de un método bilateral por bloques en anillo y en malla cuadrada.

| | | |
|---|---|---|
| *anillo bilateral* | $8k_3 \frac{n^3}{p} + (12k_1 - 8k_3) \frac{n^2t}{p} + 12k_1 \frac{nt^2}{p}$ | $(p+3) \frac{n}{t} \beta + 2n^2 \left(1 + \frac{2}{p}\right) \tau$ |
| *malla cuadrada bilateral* | $8k_3 \frac{n^3}{p} + (12k_1 + 2k_3) \frac{n^2t}{\sqrt{p}} + 12k_1 \frac{nt^2}{\sqrt{p}}$ | $(\sqrt{p}+4) \frac{n}{t} \beta + \left(\frac{7}{\sqrt{p}} + 2\right) n^2 \tau$ |
| *malla triangular bilateral* | $16k_3 \frac{n^3}{r^2} + (12k_1 + 2k_3) \frac{n^2t}{r} + 12k_1 \frac{nt^2}{r}$ | $(r+7) \frac{n}{t} \beta + \left(2n^2 + \frac{6n^2}{r} + 4nt\right) \tau$ |
| *MI* | $9k_3 \frac{n^3}{p} + 12k_1 \frac{n^2t}{p} + 12k_1 \frac{nt^2}{p}$ | $2\frac{n}{t} \beta + (4n^2 + 2nt) \tau$ |
| *MII* | $5k_3 \frac{n^3}{p} + 12k_1 \frac{n^2t}{p} + 12k_1 \frac{nt^2}{p}$ | $2\frac{n}{t} \beta + (2n^2 + 2nt) \tau$ |

Tabla 1: Comparación de los costes de diversos métodos paralelos por bloques.

En cuanto al coste aritmético ocurre lo mismo que con los algoritmos que no trabajan por bloques:

- El método MI tiene un mayor coste que el bilateral, aunque en este caso la diferencia es menor que en los algoritmos que no trabajan por bloques, siendo la proporción en aquél caso de 6 a 7 y en éste de 8 a 9.
- El método MII es el que menor coste tiene de todos, mejorando también la relación con respecto al bilateral al trabajar por bloques.

En cuanto a las comunicaciones, en los métodos unilaterales necesitamos de menos comunicaciones al no tener que difundir las matrices de rotaciones. Y en cuanto al número de datos transferidos también es preferible MII ya que trabaja con una única matriz. Por lo tanto, los métodos paralelos unilaterales por bloques pueden ser mejores que los bilaterales en sistemas con un gran número de procesadores, aunque los métodos unilaterales propuestos tienen el inconveniente de estar diseñados para anillo, lo que puede producir una baja escalabilidad. El problema de la baja escalabilidad se podría resolver diseñando algoritmos para malla (siendo en este caso lo más adecuado la cuadrada), pero con estos algoritmos habría que difundir las rotaciones tal como en los métodos bilaterales, y se perdería la principal ventaja de los unilaterales.

El método MII puede ser preferible en términos de coste aritmético, pero es el que más problemas tiene de precisión, aunque este problema se puede resolver en parte utilizando cocientes de Rayleigh [18], lo que conllevaría un trabajo adicional tras considerar que el método converge.

# 6 Propuesta de trabajo

La propuesta de trabajo es obvia:

- Se deberían implementar los métodos MI y MII por bloques propuestos y sus versiones paralelas, con el fin de poder comparar experimentalmente los métodos aquí analizados teóricamente.
- Después de la comparación de los resultados de los métodos para anillo con los métodos bilaterales para malla habría que decidir si puede ser conveniente diseñar métodos unilaterales para malla.
- Un tema que cae fuera del ámbito de este trabajo (donde nos hemos ocupado del coste de los algoritmos) es el de la convergencia y la precisión. En particular está sin estudiar el tema de la velocidad de convergencia de los métodos por bloques y cómo el uso de bloques puede afectar a la precisión de los resultados (posiblemente empeorándola ligeramente al hacerse más operaciones). Para trabajar en estos temas habría que consultar los trabajos de Hari, Mascarenhas, Mathias, Slapničar y Vesselic, entre otros.

# Referencias


[1] B. A. Chartres. Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store. *The Computer Journal*, 5:51–60, 1963.

[2] H. F. Kaiser. The JK method: a procedure for finding the eigenvectors and eigenvalues of a real symmetric matrix. *The Computer Journal*, 15(3):271–273, 1972.

[3] J. C. Nash. A one-sided transformation method for the singular value decomposition and algebraic eigenproblem. *The Computer Journal*, 18(1):74–76, 1972.

[4] P. J. Eberlein. On one-sided Jacobi methods for parallel computation. *SIAM J. Alg. Disc. Math.*, 8(4), 1987.

[5] P. P. M. Rijk. A one-sided Jacobi algorithm for computing the singular value decomposition on a vector computer. *SIAM J. Sci. Stat. Comput.*, 10(2):359–371, 1989.

[6] M. C. Hoff. The use of Jacobi methods for computing eigenvalues of real symmetric matrices. Master of Arts. University of Texas at Austin. 1993.

[7] B. B. Zhou, R. P. Brent y M. Kahn. A one-sided Jacobi algorithm for the symmetric eigenvalue problem. In *Proceedings Euromicro Workshop on Parallel and Distributed Processing*, pages 401–408. IEEE, 1995.

[8] B. B. Zhou y R. P. Brent. On parallel implementation of the one-sided Jacobi algorithm for singular value decompositions. In *Proceedings Third Parallel Computing Workshop*, 1994.

[9] D. Giménez. Métodos de Jacobi para la resolución del Problema Simétrico de Valores Propios en Multicomputadores. Tesis Doctoral. 1995.

[10] D. Giménez, R. van de Geijn, V. Hernández y A. M. Vidal. Exploiting the symmetry on the Jacobi method on a mesh of processors. In *Proceedings of the Fourth Euromicro Workshop on Parallel and Distributed Processing*, pages 377–384. IEEE Computer Society Press, 1996.

[11] D. Giménez, V. Hernández, R. van de Geijn y A. M. Vidal. A block Jacobi method on a mesh of processors. *Concurrency: Practice and Experience*, 9(5):391–411, May 1997.

[12] D. Giménez. A comparison of the solution of the Symmetric Eigenvalue Problem with ScaLAPACK and Jacobi methods. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, CD-ROM*. SIAM, 1997.

[13] D. Giménez, M. J. Majado y I. Verdú. Solving the Symmetric Eigenvalue Problem on Distributed Memory Systems. In H. R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. PDPTA'97*, pages 744–747, 1997.

[14] G. E. Forsythe y P. Henrici. The cyclic Jacobi method for computing the principal values of a complex matrix. *Trans. Amer. Math. Soc.*, 94:1–23, 1960.

[15] G. H. Golub y C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1989. Segunda Edición.

[16] R. Schreiber. Solving eigenvalue and singular value problems on an undersized systolic array. *SIAM J. Sci. Stat. Comput.*, 7(2):441–451, 1986.

[17] C. H. Bischof. Computing the singular value decomposition on a distributed system of vector processors. *Parallel Computing*, 11:171–186, 1989.

[18] Ana J. Viamonte. Métodos de Jacobi para o Cálculo de Valores e Vectores próprios de Matrizes Simétricas. Tesis de Mestrado. Universidade do Minho. 1996.

[19] P. Arbenz y M. Oettli. Block implementations of the symmetric QR and Jacobi algorithms. In L. Bouge, M. Cosnard, Y. Robert, and D. Trystram, editors, *Parallel Processing: CONPAR 92 – VAPP V*, pages 827–828, Berlin, 1992. Springer-Verlag. (Lecture Notes in Computer Science, 634).

[20] P. Arbenz y I. Slapničar. On an Implementation of a One-sided Block Jacobi Method on a Distributed Memory Computer. In *Proceedings of the 3rd International Conference on Industrial and Applied Mathematics. ICIAM95*, Hamburg, 1995.

Una propuesta para el diseño de métodos de Jacobi unilaterales por bloques
para el Problema Simétrico de Valores Propios

Domingo Giménez Cánovas

Departamento de Informática y Sistemas
Universidad de Murcia
1997

*O binómio de Newton é tão belo como a Vénus de Milo.*
*O que há é pouca gente para dar por isso.*

*Fernando Pessoa*