El Problema Lineal de Mínimos Cuadrados. Descomposición QR.

Antonio M. Vidal
Murcia. Enero 2011

Descomposiciones matriciales basadas en transformaciones no ortogonales

Lecturas recomendadas:

"Matrix computations". Golub & van Loan. Capítulos 3 y 4

Descomposición LU: Si la matriz real A, nxn, verifica que $\det(A(1:k,1:k)) \neq 0$, k=1,2,…,n-1 existen matrices reales, nxn, L triangular inferior unidad y U triangular superior, tales que A=LU. Si además A es invertible entonces la descomposición LU es única.

Descomposición LU con pivotamiento: Si la matriz real A, nxn, es invertible, existen matrices reales, nxn, L triangular inferior unidad y U triangular superior, y una matriz de permutación P, tales que PA=LU.

Descomposiciones matriciales basadas en transformaciones no ortogonales

- Si A es simétrica e invertible y tiene descomposición LU existen matrices L, M triangulares inferiores unidad, y una matriz diagonal D, tales que $A=LDM^T$. Si además A es simétrica, M=L, es decir $A=LDL^T$

- Descomposición de Cholesky: Si A es simétrica y definida positiva (*) existe una matriz G triangular inferior tal que $A=GG^T$. G se denomina factor de Cholesky.
- (*)

$$A^T = A \in \Re^{nxn} \text{ es definida positiva si } \forall x \in \Re^n \; ; x^T A x > 0$$

Descomposiciones matriciales basadas en transformaciones no ortogonales

- Si A es simétrica e invertible existe una matriz L, triangular inferior unidad, una matriz diagonal por bloques D, con bloques de tamaño 1x1 o 2x2, y una matriz de permutación P tales que $PAP^T=LDL^T$.

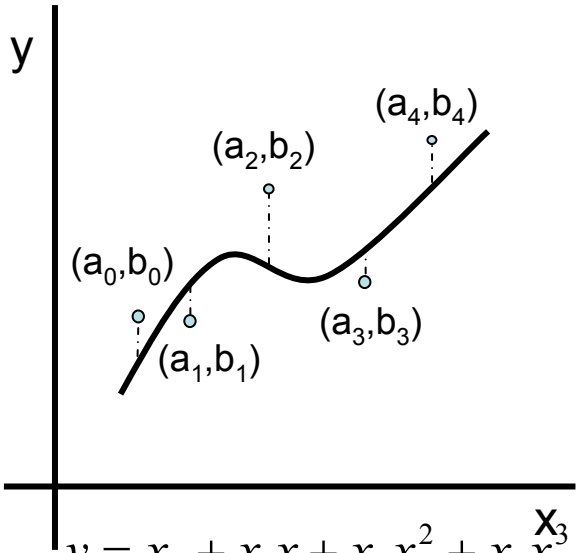
- Si A es simétrica e invertible existe una matriz L, triangular inferior unidad, una matriz tridiagonal, y una matriz de permutación P tales que $PAP^T=LTL^T$.

El problema lineal de mínimos cuadrados

Lecturas recomendadas:

"Matrix computations". Golub & van Loan. Capítulo 5

Ejemplo: Ajuste de una curva a una nube de puntos



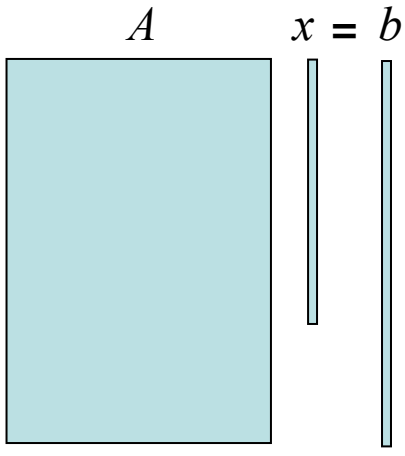
$$y = x_0 + x_1 x + x_2 x^2 + x_3 x^3$$

Imponiendo la condición de que la curva pase por los puntos conocidos se tiene

$$Para\ i = 0,1,\ldots,m-1$$

$$b_i = \sum_{j=0}^{n-1} x_j a_i^{\ j}$$

En forma matricial



con $A \in \Re^{m \times n}, x \in \Re^{n}, b \in \Re^{m}$

y $\quad a_{\mathrm{ij}} = a_i^{\,j},\ x = \begin{bmatrix} x_0 & x_1 & \dots & x_{n-1} \end{bmatrix}^T \in \Re^n, b = \begin{bmatrix} b_0 & b_1 & \dots & b_{m-1} \end{bmatrix}^T \in \Re^m$

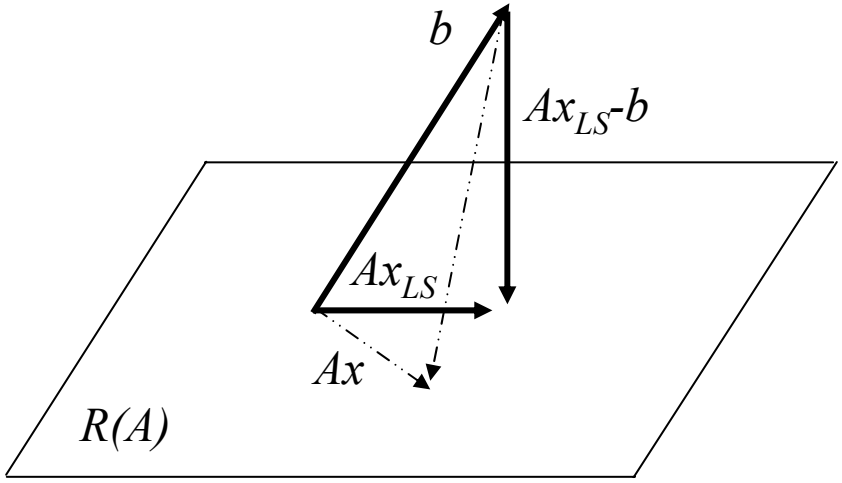
$$\left\| Ax - b \right\|_2 = \sqrt{(Ax-b)_0^{\ 2} + (Ax-b)_1^{\ 2} + \dots + (Ax-b)_{m-1}^{\ 2}}$$

El ajuste de la curva por mínimos cuadrados es equivalente a la resolución de un sistema de ecuaciones sobredeterminado:

$$\text{Resolver } Ax_{LS} = b \Leftrightarrow \boxed{\text{Encontrar } x_{LS} \in \Re^n \ \text{ tal que } \left\| Ax_{LS} - b \right\|_2 = \min_{x \in \Re^n} \left\| Ax - b \right\|_2}$$

Problema Lineal de Mínimos Cuadrados

Ecuaciones normales



Para $i=0,1,\ldots,n-1$

$$a_i^T(Ax_{LS}-b)=0$$

$$A^T(Ax_{LS}-b)=0$$

$$A^TAx_{LS}=A^Tb$$

Ecuaciones Normales del Problema Lineal de Mínimos Cuadrados

Si se conoce $A = QR$, con $Q \in \Re^{m\times m}$, ortogonal $(Q^T = Q^{-1})$ y R, triangular superior,

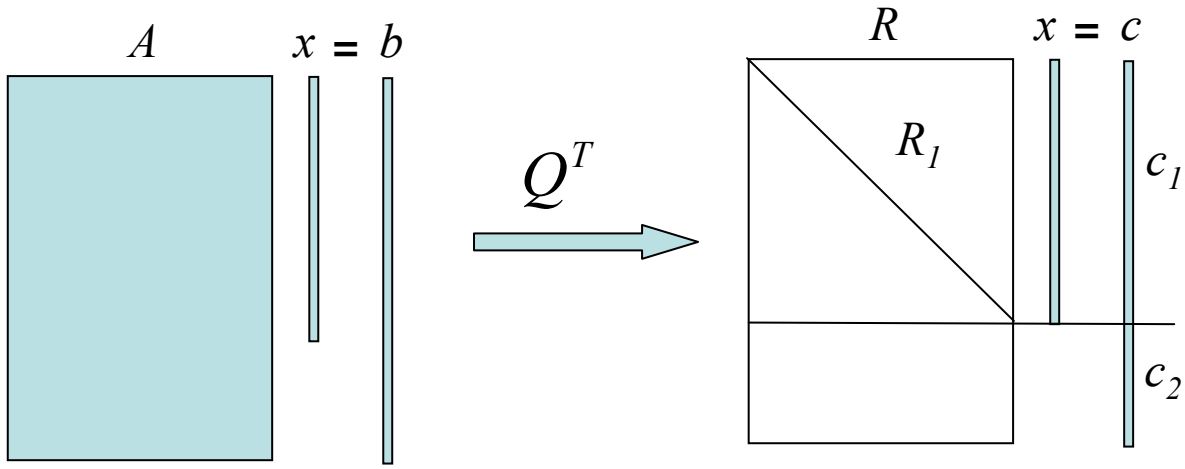
puesto que Q conserva la 2-norma vectorial se tiene

$$\|Ax-b\|_2^{\,2} = \left\|Q^T(Ax-b)\right\|_2^{\,2} = \|Rx-c\|_2^{\,2} = \|R_1x-c_1\|_2^{\,2} + \|-c_2\|_2^{\,2}$$

con $\quad Q^TA = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},\ R_1 \in \Re^{n\times n} \qquad Q^Tb = c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix},\ c_1 \in \Re^n, c_2 \in \Re^{m-n}$

Por tanto el mínimo se alcanza en $x_{LS} = R_1^{-1}c_1$

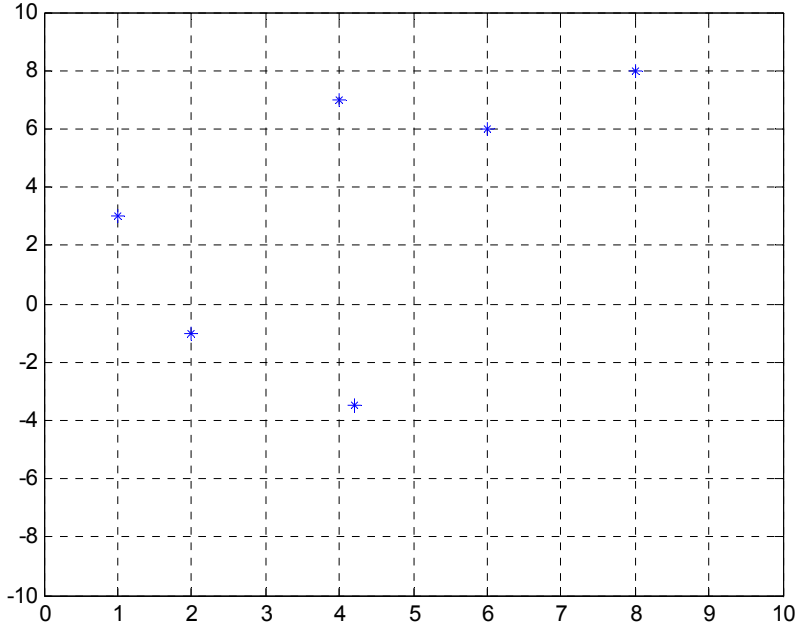
y toma el valor $\min\limits_{x\in\Re^n} \|Ax-b\|_2^{\,2} = \|c_2\|_2^{\,2}$



Aplicaciones

Problemas de mínimos cuadrados; ajuste de funciones a un conjunto de puntos.

Sea el siguiente conjunto de puntos (aleatorios) en el plano: (1,3), (2, -1), (4,7), (4.2,-3.5), (6,6) (8,8)



Ejemplo

Queremos encontrar una función, de algún tipo predefinido, que pase lo mas cerca posible de esos puntos;Por ejemplo, un polinomio de grado 3:

$$ax^3+bx^2+cx+d$$

Para cada punto planteamos la igualdad "deseada":

para (1,3):

$$a\cdot 1^3+b\cdot 1^2+c\cdot 1+d = 3$$

para (2,-1):

$$a\cdot 2^3+b\cdot 2^2+c\cdot 2+d = -1$$

para (4,7):

$$a\cdot 4^3+b\cdot 4^2+c\cdot 4+d = 7$$

Y así para todos los puntos

Aplicaciones

Versión matricial del sistema de ecuaciones:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \\ 64 & 16 & 4 & 1 \\ 512 & 64 & 8 & 1 \\ 216 & 36 & 6 & 1 \\ 74{,}08 & 17{,}64 & 4{,}2 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \\ 7 \\ 8 \\ 6 \\ -3{,}5 \end{pmatrix}$$

o existe solución para este sistema de ecuaciones (a no ser que la matriz ea singular)

n lugar de buscar "la" solución, buscamos la "mejor" solución posible, la ue minimiza $||Ax-b||_2$. ➔ Solución en el sentido de "mínimos cuadrados".

ste problema se puede resolver con las "ecuaciones normales": $A^TAx=A^Tb$; uméricamente es mejor utilizar la QR.

Aplicaciones

Calculamos en Matlab la QR

```
>> [Q,R]=qr(A)

Q =

   -0.0018   -0.0620    0.5609    0.7647   -0.2769    0.1420
   -0.0142   -0.2094    0.6703   -0.2355    0.6068   -0.2880
   -0.1134   -0.5286    0.1279   -0.2898   -0.6458   -0.4363
   -0.9073    0.3573    0.1507   -0.1254   -0.0858    0.0575
   -0.3828   -0.4941   -0.4407    0.4623    0.3589   -0.2694
   -0.1313   -0.5487    0.0527   -0.2154    0.0427    0.7942


R =

 -564.3138  -76.0356  -10.5902   -1.5507
        0  -13.9558   -5.0058   -1.4855
        0        0    1.1961    1.1218
        0        0         0    0.3609
        0        0         0         0
        0        0         0         0
```

Aplicaciones

Se multiplica b por Q^T

```
>> b1=Q'*b
```

$= [-9.8805 \;\; -1.8625 \;\; 0.2852 \;\; 3.0253 \;\; -4.6408 \;\; -6.2758]^T$

Y se resuelve el sistema triangular:

```
>> sol=R(1:4)\b1(1:4)

sol =

   -0.1287
    1.9760
   -7.6248
    8.3838
```

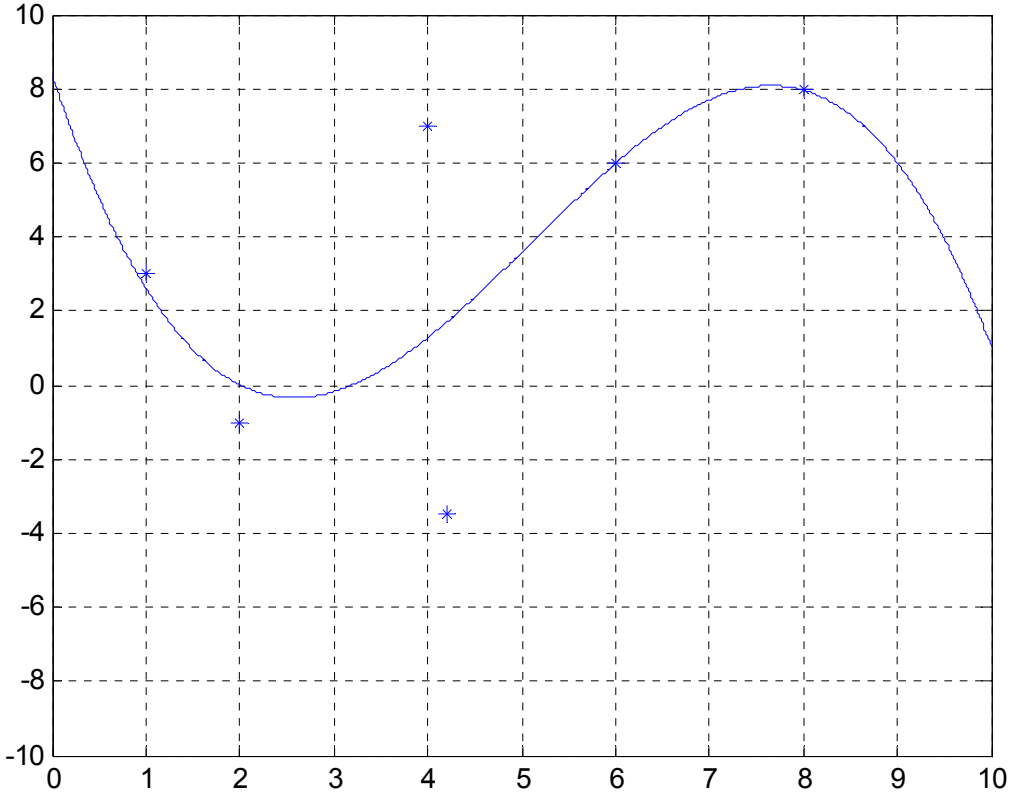
$$= \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

Obtenemos la gráfica del polinomio

$$sol(1)\cdot x^3 + sol(2)\cdot x^2 + sol(3)\cdot x + sol(4)$$

Residuo= $\sqrt{(-4.6408)^2 + (-6.2758)^2} = 7.8053$

Aplicaciones



Descomposición QR

Deseamos obtener una nueva descomposición de una matriz $A \in \Re^{mxn}$, en la forma:

$$A=QR$$

Q es una matriz ortogonal, y R es una matriz triangular

Puede utilizarse también para resolver sistemas de ecuaciones lineales generales

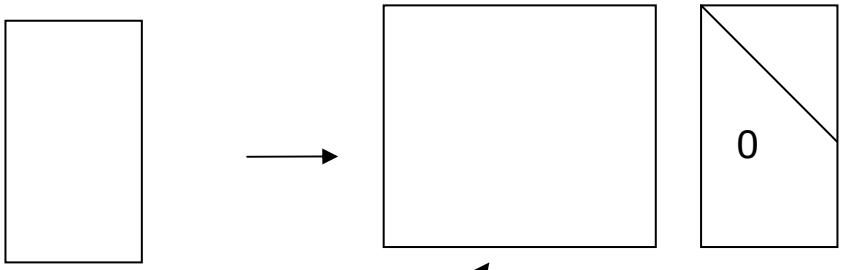
$$Ax = b \Leftrightarrow QRx = b \Leftrightarrow Rx = Q^T b$$

INTRODUCCIÓN

La descomposición QR se puede calcular independientemente de las dimensiones de A

$A \in \Re^{mxn}$,

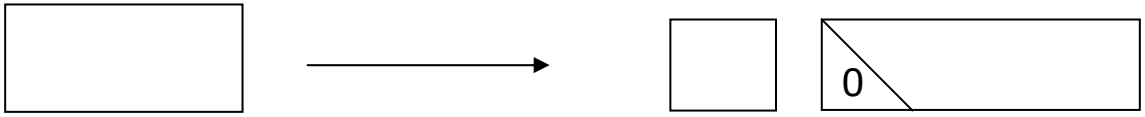
$m>n$



$Q \in \Re^{m\times m}$ $R \in \Re^{m\times n}$

$A \in \Re^{mxn}$,

$m<n$



$Q \in \Re^{m\times m}$ $R \in \Re^{m\times n}$

INTRODUCCIÓN

La descomposición QR se puede calcular de muchas formas:

1) Basada en Reflexiones de Householder

2) Basada en Rotaciones de Givens

3) Basada en la ortonormalización modificada de Gram- Schmidt

4) Basada en rotaciones "rápidas" de Givens

Estudiaremos las dos primeras

1) Reflexiones de Householder

Sea v en $\Re^n$ distinto de 0. Las Reflexiones de Householder son matrices en $\Re^{n\times n}$ de la forma:

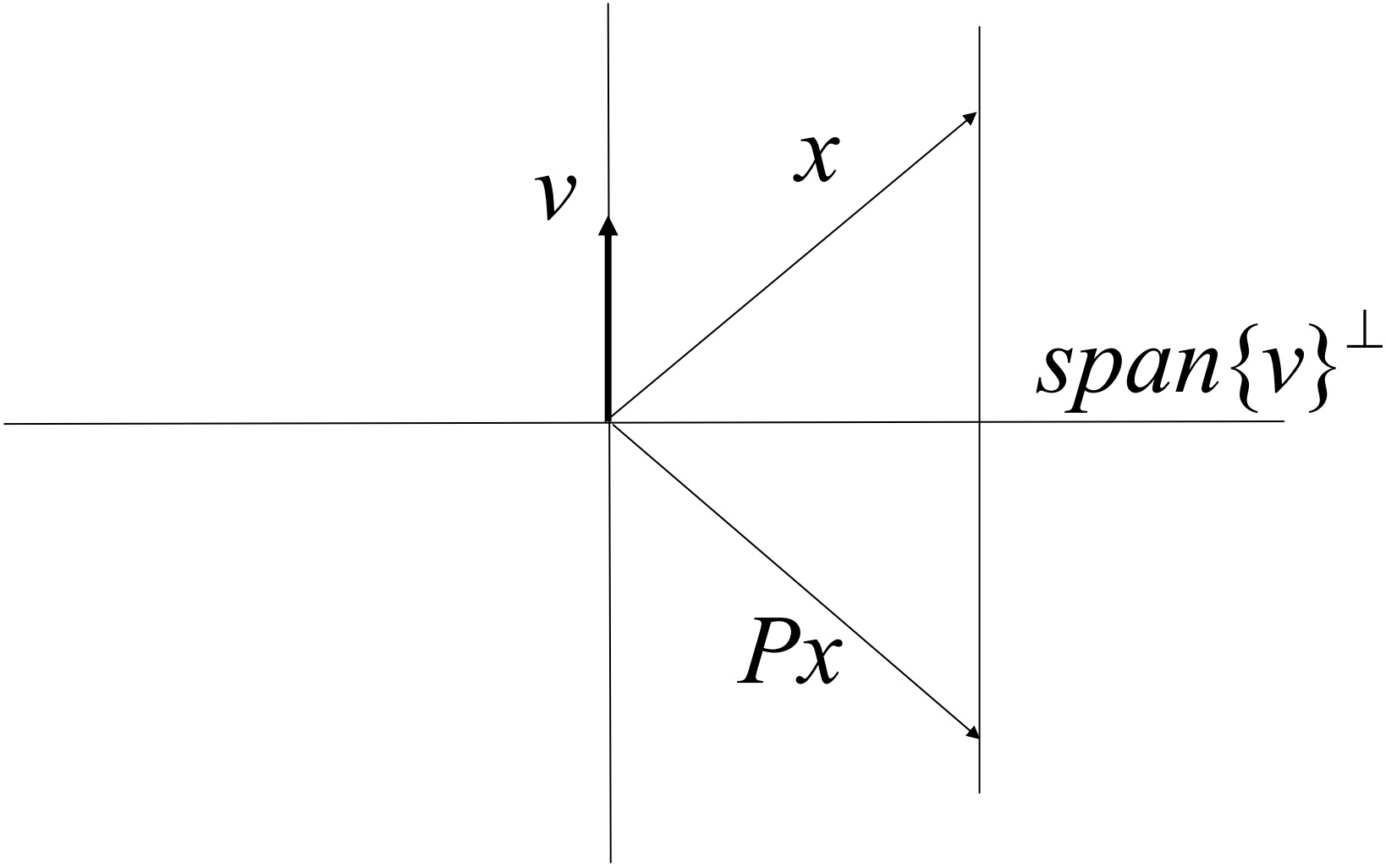
$$P = I - 2\frac{v \cdot v^T}{v^T \cdot v}$$

Matriz de rango 1

Escalar

1)Si y=Px, y se obtiene reflejando x sobre el subespacio Rango$\{v\}^{\perp}$.(Complemento ortogonal de rango de v)

2) Cada P es una modificación de la identidad de rango 1
Simétrica y Ortogonal


v
x
$span\{v\}^{\perp}$
Px

2) Reflexiones de Householder

Utilización para hacer 0s en ciertas componentes de un vector:

Sea $x \in \Re^n$, queremos construir una matriz de Householder P tal que $Px=\alpha \cdot e_1$ (Todas las componentes de Px son 0 excepto la primera).

Escogemos el vector v como: $v=x \pm ||x||_2 e_1$. Es fácil comprobar que, construyendo P con este vector v, Px tiene todas sus componentes iguales a 0, excepto la primera.

3) Reflexiones de Householder

Ejemplo:

$$x = \begin{pmatrix} 3 \\ 1 \\ 5 \\ 1 \end{pmatrix}$$

Tomamos $v = x + \|x\|_2 e_1$ $\longrightarrow$ $v = \begin{pmatrix} 9 \\ 1 \\ 5 \\ 1 \end{pmatrix}$

$$P = I - 2\frac{v \cdot v^t}{v^t \cdot v} = \frac{1}{54}\begin{bmatrix} -27 & -9 & -45 & -9 \\ -9 & 53 & -5 & -1 \\ -45 & -5 & 29 & -5 \\ -9 & -1 & -5 & 53 \end{bmatrix} \longrightarrow Px = \begin{pmatrix} -6 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

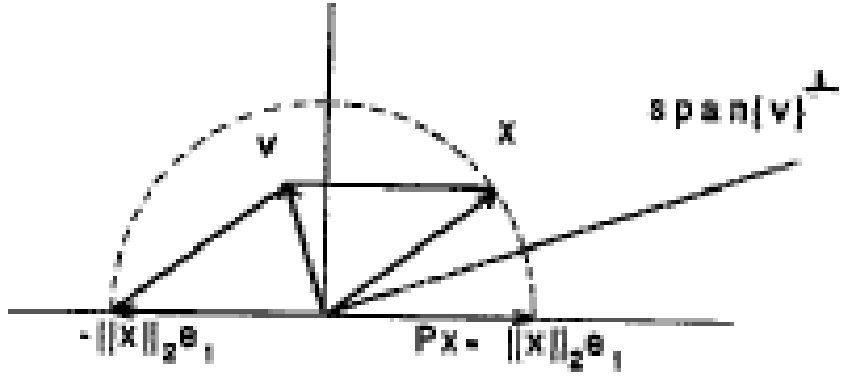
4) Reflexiones de Householder Detalles de implementación.

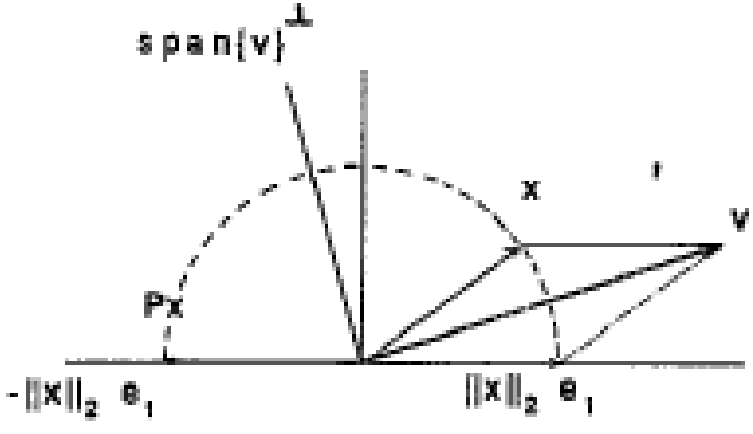
1) Como veremos a continuación, sólo se calcula el vector **v** y el factor $\beta=(v^t v)/2$; en aplicaciones prácticas, NUNCA se calcula la matriz P.

2) En la fórmula $v=x \pm \|x\|_2 e_1$, el signo se selecciona para minimizar el error de redondeo.

3) El vector **v** se puede normalizar para que v(1)=1; de esa forma, se puede almacenar la parte importante del vector v (v(2:n)) en la parte de x donde se han hecho los ceros (x(2:n)). En este caso

$$P = I - \tau * \overline{u} * \overline{u}^T, \quad \text{con } u = v / \|v\|_2, \ \overline{u} = u / u_1, \ \overline{u}_1 = 1, \ \tau = 2 * u_1^{\ 2}$$


span{v}⊥
v
x
-||x||2e1
Px = ||x||2e1


span{v}⊥
x
v
Px
-||x||2 e1
||x||2 e1

Construcción del algoritmo

$v = x + sg(x_1)\,||x||_2 e_1$

$\beta = (v^t v)/2 = sg(x_1)\,||x||_2 (sg(x_1)\,||x||_2 + x_1)$

$$\begin{pmatrix} x & x & x & x & x \\ x & \boxed{x} & x & x & x \\ x & \boxed{x} & x & x & x \\ x & \boxed{x} & x & x & x \\ x & \boxed{x} & x & x & x \end{pmatrix} \Rightarrow \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & 0 & x & x & x \\ x & 0 & x & x & x \\ x & 0 & x & x & x \end{pmatrix}$$

<u>Procedimiento Norm(j)</u>

$$\rho_j = \operatorname{sgn}(a_{jj})\left(\sum_{i=j}^{m-1} a_{ij}^{\,2}\right)^{1/2}$$

$$v_j = [0,0,\ldots,0,a_{jj} + \rho_j, a_{j+1,j},\ldots,a_{m-1,j}]^T$$

$$\beta_j = \rho_j(\rho_j + a_{jj})$$

Coste: 2(m-j) Flops

Resultado de aplicar una Matriz de Householder, P, a otro vector y distinto del x que ha servido para calcularla

$$Py = y - \gamma v \qquad \gamma = (1/\beta)(v^T y) \text{ y } \beta = (v^T v)/2$$

<u>Procedimiento Fact(j,k)</u>

$$\gamma_{jk} = (1/\beta_j)\sum_{i=j}^{m-1}(v_j)_i a_{ik}$$

Coste: 2(m-j) Flops

<u>Procedimiento Cmod(j,k)</u>

$$Para\ i = j, j+1, \ldots, m-1$$

$$a_{ik} = a_{ik} - \gamma_{jk}(v_j)_i$$

Coste: 2(m-j) Flops

$$\begin{pmatrix} x & x & x & x & x \\ 0 & \boxed{x} & x & x & x \\ 0 & \boxed{x} & x & x & x \\ 0 & \boxed{x} & x & x & x \\ 0 & \boxed{x} & x & x & x \end{pmatrix} \Rightarrow \begin{pmatrix} x & x & x & x & x \\ 0 & x & \boxed{x} & \boxed{x} & \boxed{x} \\ 0 & 0 & \boxed{x} & \boxed{x} & \boxed{x} \\ 0 & 0 & \boxed{x} & \boxed{x} & \boxed{x} \\ 0 & 0 & \boxed{x} & \boxed{x} & \boxed{x} \end{pmatrix}$$

Descomposición QR basada en Householder(1)

Partimos de una matriz ejemplo 6*5

$$A = \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix} \longrightarrow H_1 \cdot A = \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix}$$

Mediante una reflexión, hacemos ceros en la primera columna

Descomposición QR basada en Householder(2)

Luego repetimos el mismo proceso, con cada columna para hacerla triangular superior:

$$H_1 \cdot A = \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix} \longrightarrow H_2 \cdot H_1 \cdot A = \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{pmatrix} \longrightarrow \bullet \bullet \bullet$$

$$\longrightarrow H_5 \cdot H_4 \cdot H_3 \cdot H_2 \cdot H_1 \cdot A = \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Descomposición QR basada en Householder(3)

$$H_5 \cdot H_4 \cdot H_3 \cdot H_2 \cdot H_1 \cdot A = \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$H_5 \cdot H_4 \cdot H_3 \cdot H_2 \cdot H_1 = Q^t$$

H_i ortogonales => Q ortogonal

$$Q^t \cdot A = R$$

$$A = QR$$

Q ortogonal

Algoritmo de Triangularización Ortogonal (Householder)

```
Para j = 0,1,...,n - 1
        norm(j)
        Para k = j, j + 1,...,n - 1
                fact(j,k)
                cmod(j,k)
```

Coste : $2n^2(m - \frac{n}{3})$ flops

Estabilidad numérica del algoritmo

The roundoff properties associated with Householder matrices are very favorable. Wilkinson (AEP, pp. 152-62) shows that house produces a Householder vector $\hat{v}$ very near the exact v. If $\hat{P} = I - 2\hat{v}\hat{v}^T/\hat{v}^T\hat{v}$ then

$$\| \hat{P} - P \|_2 = O(\mathbf{u})$$

Moreover, the computed updates with $\hat{P}$ are close to the exact updates with P :

$$fl(\hat{P}A) = P(A+E) \qquad \| E \|_2 = O(\mathbf{u}\| A \|_2)$$

$$fl(A\hat{P}) = (A+E)P \qquad \| E \|_2 = O(\mathbf{u}\| A \|_2)$$

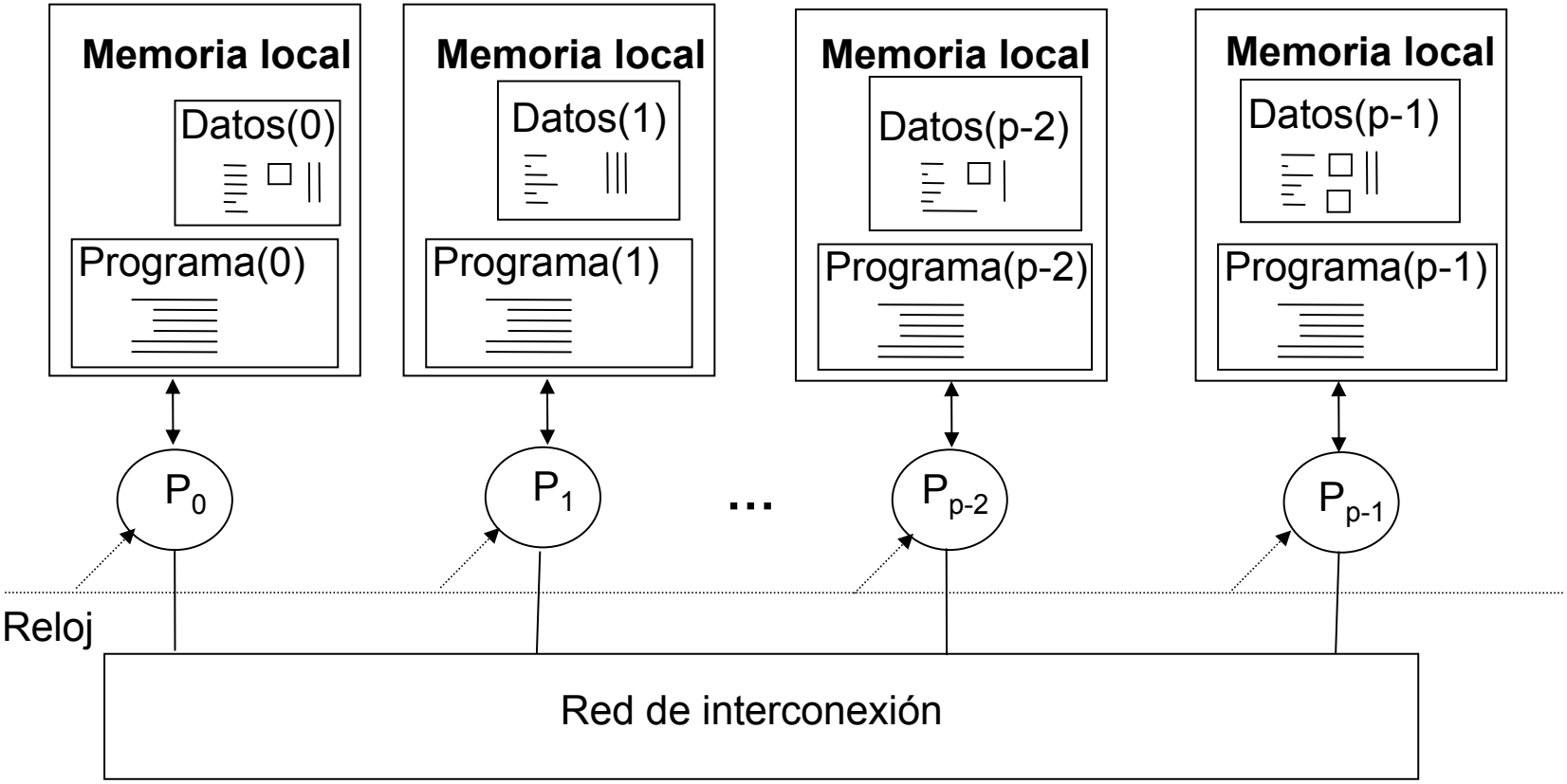
If $A=Q_1R_1$

the corresponding result for the Householder approach is

$$\hat{Q}_1^T\hat{Q}_1 = I + E_H \qquad \| E_H \|_2 \approx \mathbf{u} .$$

Paralelización del Algoritmo de Householder
en el Modelo de Paso de Mensajes

Objetivo:
Diseñar un algoritmo paralelo óptimo para triangularizar una matriz en un multicomputador



Características:

Entorno de paso de mensajes (MPI)

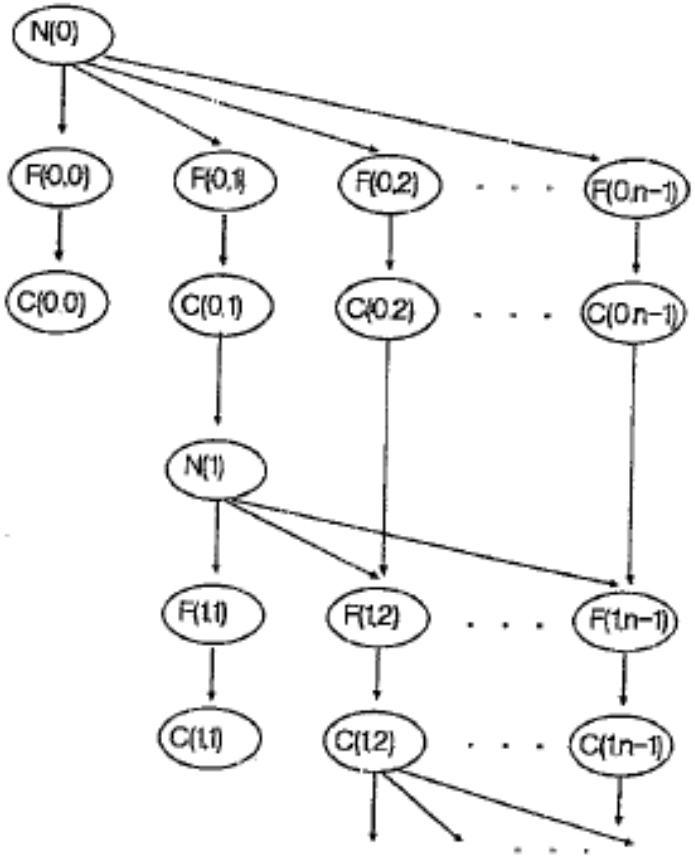
Se conoce el tiempo de ejecución de 1 Flop (operación elemental en coma flotante:+,-,*,:)

Enviar un mensaje de N palabras desde un nodo a otro (o a otros) tiene un coste de $N\tau+\beta$, con τ y β conocidos ($\tau=t_w$ **y** $\beta=t_s$)

$$t_p \cong t_A + t_C - t_{sol} \cong t_A + t_C$$

Grafo de dependencias

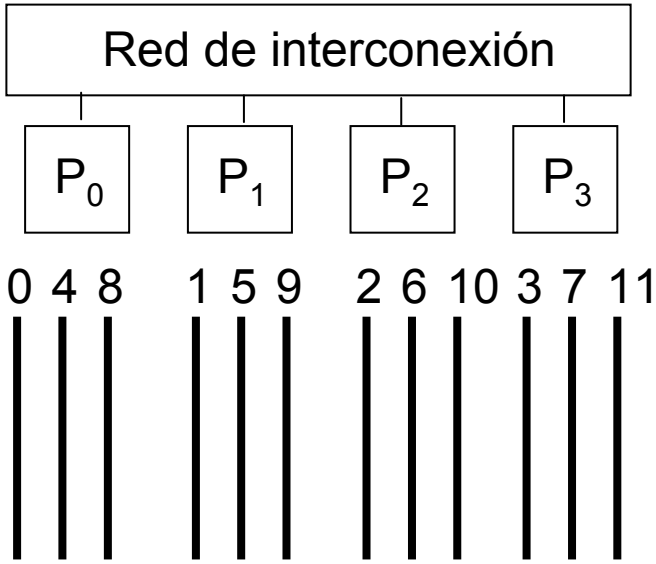
N(j)=Norm(j); F(j,k)=Fact(j,k); C(j,k)=Cmod(j,k)



Distribución cíclica: suponiendo n=k*p

$$Columna\ j \rightarrow P_{jMODp}$$

$$\text{Procesador } i \rightarrow \text{ Columnas } rp+i, r=0,1,\ldots,(n/p)-1$$



Idea básica del algoritmo

Para cada columna

1. Calcular los parámetros de la transformación de Householder (sólo el procesador que la contiene)
2. Difundir los parámetros al resto de procesadores
3. Actualizar las restantes columnas (cada procesador sólo las que contiene)

Algoritmo paralelo

EN PARALELO: PARA pr=0,1,…,p-1

En P_{pr}:

PARA j=0,1,…,n-1

SI jMODp=pr (*columna j pertenece a P_{pr} *)

ENTONCES

Norm(j)

Difunde parámetros (* v_j, β_j*)

EN OTRO CASO

Espera parámetros

FIN SI

PARA k=j,j+1,…,n-1

SI kMODp=pr (*columna k pertenece a P_{pr} *)

ENTONCES

Fact(j,k)

Cmod(j,k)

FIN SI

FIN PARA

FIN PARA

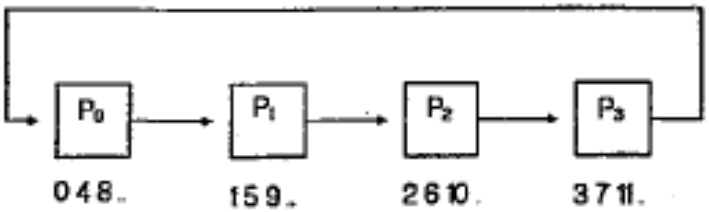
FIN PARA

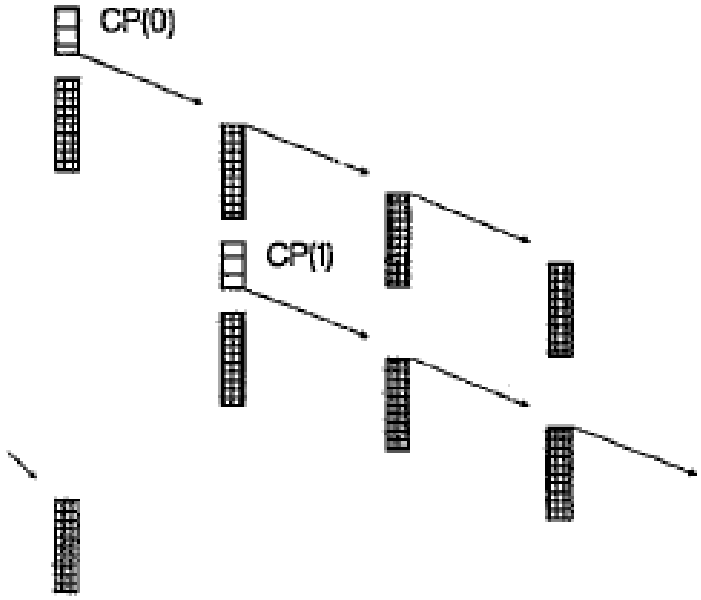
Coste:

$$t_A = (2n^2 / p)(m - n/3)\ Flops$$

$$t_C = p\left[(mn - n^2/2)\tau_{DIF} + n\beta_{DIF}\right]$$

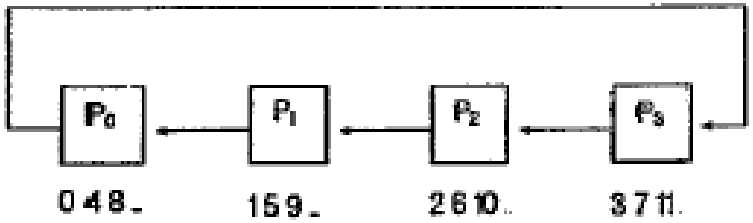
Algoritmo sobre un anillo unidireccional

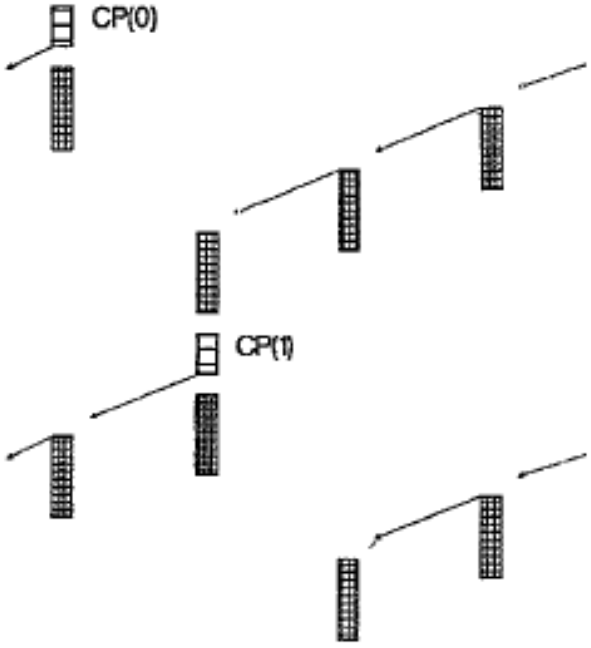




Algoritmo asíncrono, con solapamiento de comunicaciones y operaciones aritméticas

Algoritmo sobre un anillo unidireccional

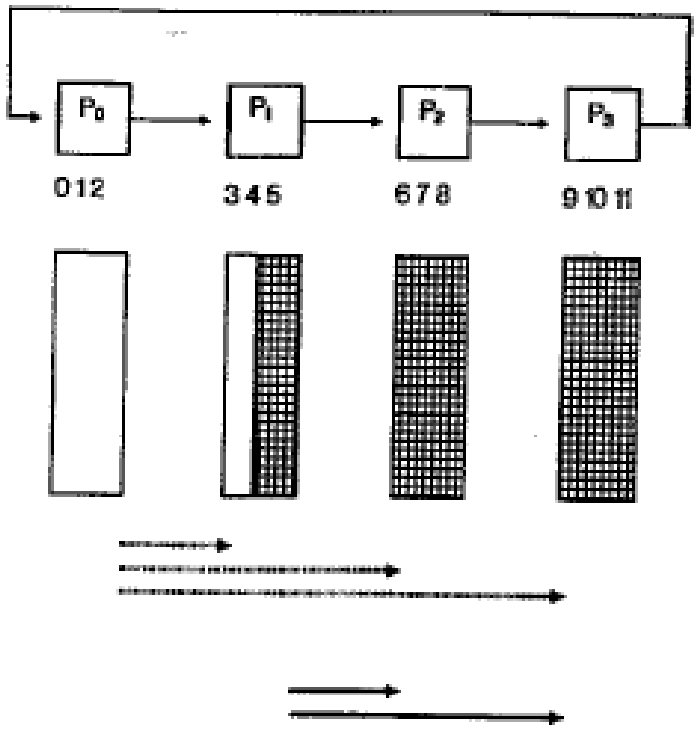




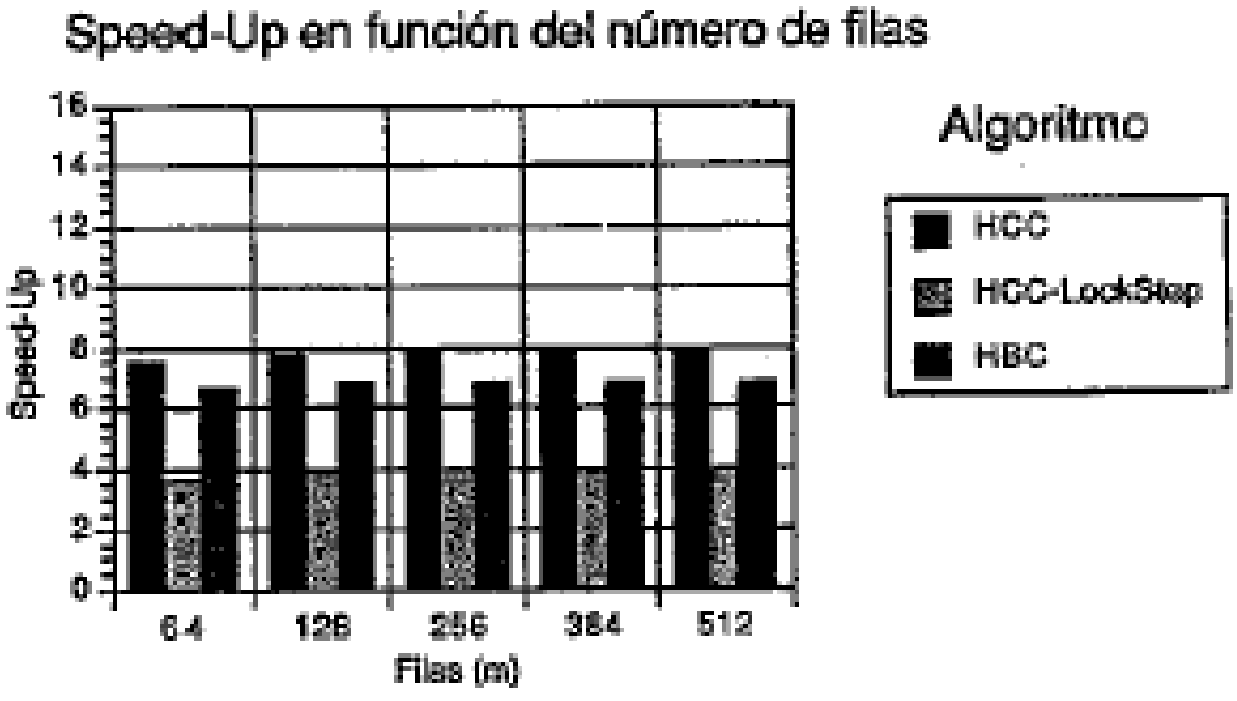
Algoritmo síncrono (lock-step), sin solapamiento de comunicaciones y operaciones aritméticas

Algoritmo con distribución por bloques de columnas consecutivas

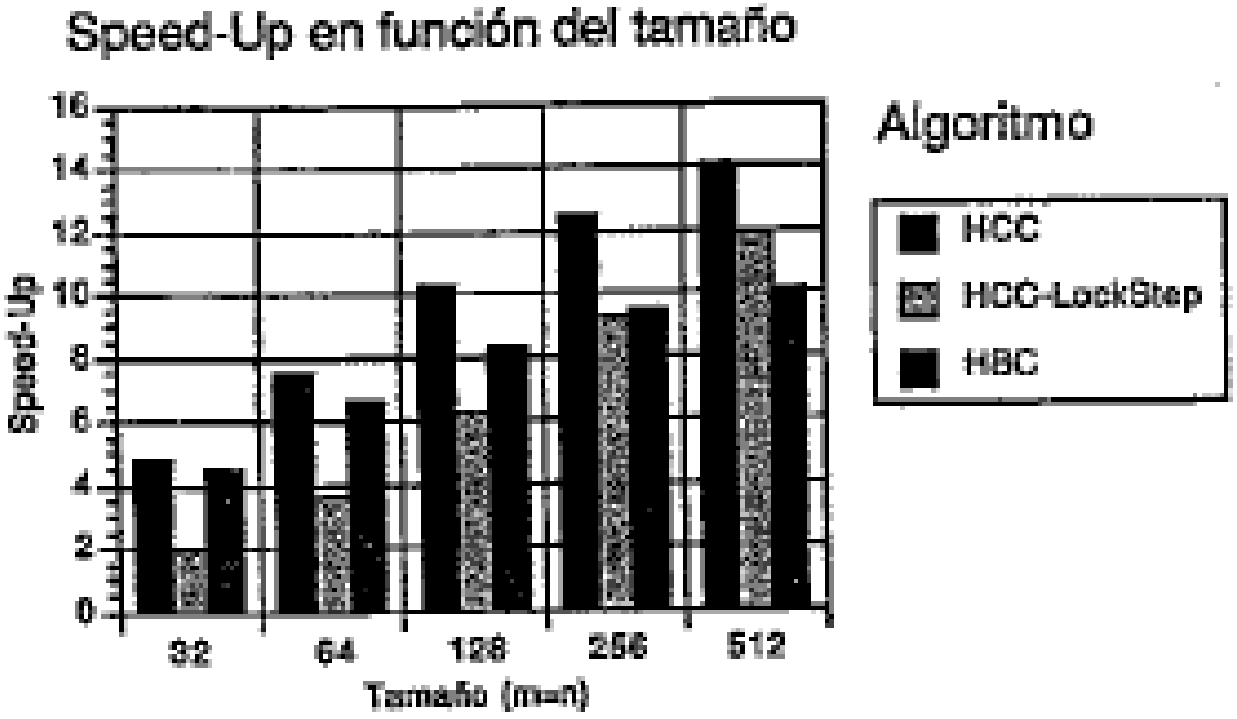
- Distribución cíclica: tiende a equilibrar la carga
- Distribución por bloques de columnas consecutivas: tiende a disminuir el tiempo de comunicación



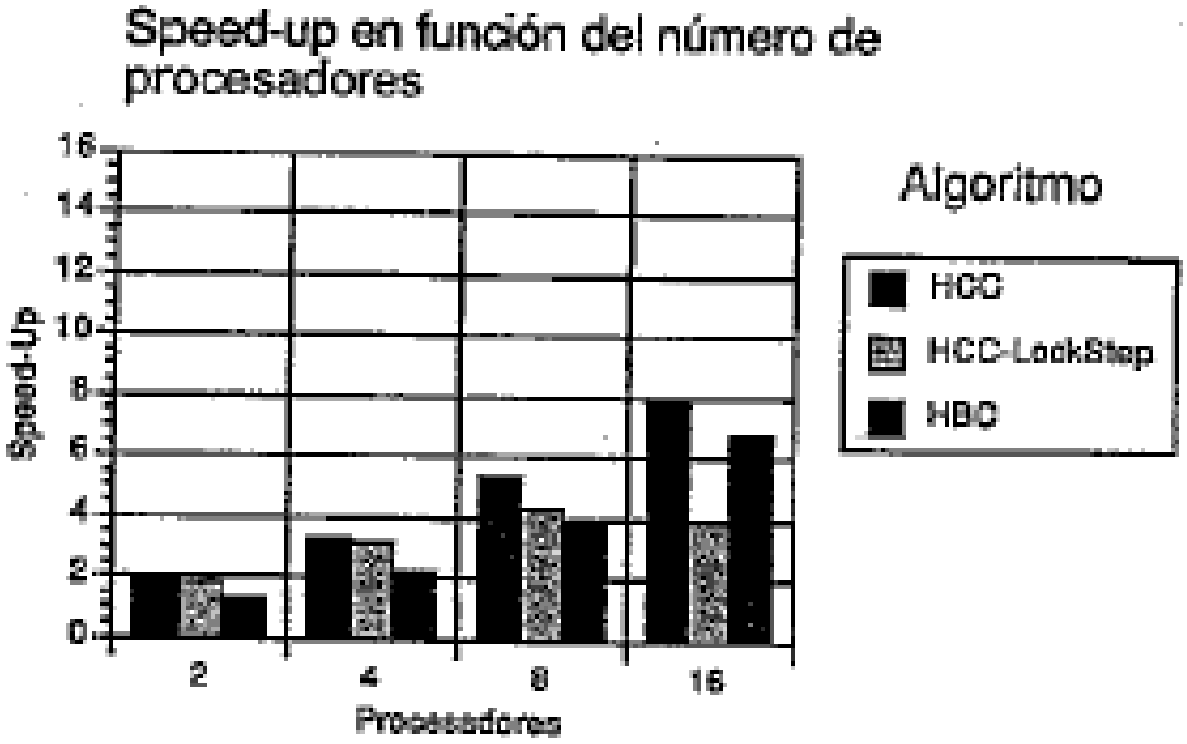
MATRIZ RECTANGULAR mx64
16 PROCESADORES



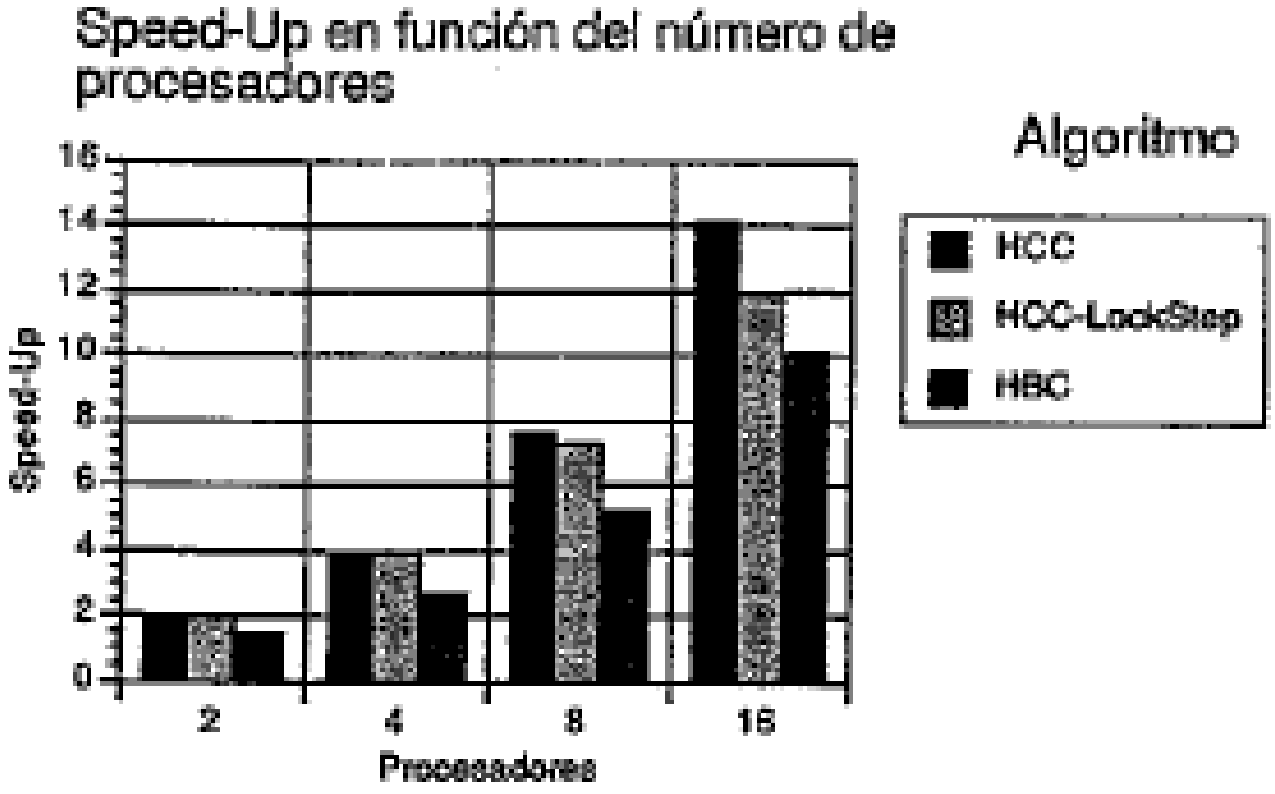
MATRIZ CUADRADA nxn
16 PROCESADORES



MATRIZ RECTANGULAR 512x64



MATRIZ CUADRADA 512X512



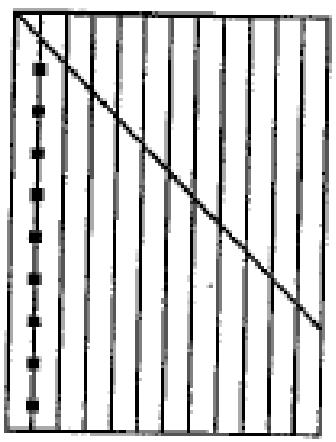
Otros aspectos del Algoritmo Paralelo

Algoritmo general

```
PARA j=0,1,…,n-1
    CalculaParámetros(j)
    PARA k=j,j+1,…,n-1
        ActualizarColumna(j,k)
```

Eliminación Gaussiana

Parámetros=multiplicadores



Actualización:

$$A_k = A_k - a_{jk} * A_j$$

$$\left| \quad \right| = \left| \; - a_{jk} * \; \right|$$

Algoritmo paralelo general

```
EN PARALELO: PARA pr=0,1,…,p-1
En P_pr:
        PARA j=0,1,…,n-1
            SI jMODp=pr (*columna j pertenece a P_pr *)
                ENTONCES
                    CalcularParámetros(j)
                    Difunde parámetros (* ,  *)
                EN OTRO CASO
                    Espera parámetros
            FIN SI
            PARA k=j,j+1,…,n-1
                SI kMODp=pr (*columna k pertenece a P_pr *)
                ENTONCES
                    ActualizarColumna(j,k)
                FIN SI
            FIN PARA
        FIN PARA
FIN PARA
```

BLAS

Operaciones de bajo nivel en Álgebra Lineal ($x^T y$, $x \leftarrow \alpha x$, $y \leftarrow \alpha x+y$, ...) a partir de las cuales se construyen rutinas de nivel superior en Álgebra Lineal ($A= LU$, $L^* L^T$, QR, ...)

BLAS 1:

Operaciones vector-vector $O(n)$ operaciones, $O(n)$ datos
SAXPY: $y \leftarrow \alpha x + y$

BLAS 2:

Operaciones matriz-vector $O(n^2)$ operaciones, $O(n^2)$ datos
SGEMV: $y \leftarrow \alpha Ax + \beta y$

BLAS 3:

Operaciones matriz-matriz $O(n^3)$ operaciones, $O(n^2)$ datos
SGEMM: $C \leftarrow \alpha AB + \beta C$

$$B = \alpha T^{-1} B$$

$$B = \alpha T^{-T} B$$

$$B = \alpha B T^{-1}$$

$$B = \alpha B T^{-T}$$

Here α and β are scalars, A, B and C are rectangular matrices (in some cases square and symmetric), and T is an upper or lower triangular matrix (and nonsingular in (d)).

Analogous operations are in complex arithmetic: conjugate transposition is specified instead of simple transposition and in (b) C is Hermitian and α and β are real.

The results of using the different levels of BLAS on the Alliant FX/8, IBM 3090 with Vector Facility, and the CRAY-2 are shown in the tables below.

Alliant FX/8 (8 Proc.)	MFLOPS
Peak Performance	94
LINPACK Benchmark	7.6
Level 1 BLAS ($y = y + \alpha x$)	14
Level 2 BLAS ($y = \beta y + \alpha A x$)	26
Level 3 BLAS ($C = \beta C + \alpha AB$)	43

Table 5.1: Levels of Performance on the Alliant FX/8

IBM 3090/VF (1 Proc.)	MFLOPS
Peak Performance	108
LINPACK Benchmark	12
Level 1 BLAS ($y = y + \alpha x$)	26
Level 2 BLAS ($y = \beta y + \alpha A x$)	60
Level 3 BLAS ($C = \beta C + \alpha AB$)	80

Table 5.2: Levels of Performance on the IBM 3090/VF

CRAY-2 (1 processor)	MFLOPS
Peak Performance	488
LINPACK Benchmark	15
Level 1 BLAS ($y = y + \alpha x$)	121
Level 2 BLAS ($y = \beta y + \alpha A x$)	350
Level 3 BLAS ($C = \beta C + \alpha AB$)	437

Table 5.3: Levels of Performance on the CRAY-2

following table illustrates the advantage of the Level 3 BLAS:

BLAS	Mem. Ref. Ref:Ops $n = m = k$	Ops	Ratio
Level 1 SAXPY			
$y = y + \alpha x$	$3n$	$2n$	3 : 2
Level 2 SGEMV			
$y = \beta y + \alpha Ax$	$mn + n + 2m$	$2mn$	1 : 2
Level 3 SGEMM			
$C = \beta C + \alpha AB$	$2mn + mk + kn$	$2mnk$	2 : n

Table 5.4: Advantage of the Level 3 BLAS

Organización por bloques

$$P_1 = I - \tau_1 z_1 z_1^{\ T}; P_2 = I - \tau_2 z_2 z_2^{\ T}; A = P_1 P_2 R_{(2)}$$

$$R_{(2)} = \begin{bmatrix} r_{11} & r_{12} & \overline{a}_{13} & \overline{a}_{14} \\ z_{21} & r_{22} & \overline{a}_{23} & \overline{a}_{24} \\ z_{31} & z_{32} & \overline{a}_{33} & \overline{a}_{34} \\ z_{41} & z_{42} & \overline{a}_{43} & \overline{a}_{44} \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ z_{21} & 1 \\ z_{31} & z_{32} \\ z_{41} & z_{42} \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \quad \overline{A} = \begin{bmatrix} \overline{a}_{13} & \overline{a}_{14} \\ \overline{a}_{23} & \overline{a}_{24} \\ \overline{a}_{31} & \overline{a}_{34} \\ \overline{a}_{41} & \overline{a}_{44} \end{bmatrix} = \begin{bmatrix} \overline{A}_1 \\ \overline{A}_2 \end{bmatrix}$$

$$P_1 P_2 = I - ZTZ^T = I - \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} T \begin{bmatrix} Z_1^{\ T} & Z_2^{\ T} \end{bmatrix}, \text{con } T = \begin{bmatrix} t_{11} & \mathrm{t}_{12} \\ 0 & \mathrm{t}_{22} \end{bmatrix}$$

Organización por bloques

Caso de 2 columnas

- $P_1P_2 = I - \tau_1(z_1 z_1^T) - \tau_2(z_2 z_2^T) + \tau_1\tau_2(z_1^T z_2)(z_1 z_2^T)$

- $I - \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} T \begin{bmatrix} Z_1^T & Z_2^T \end{bmatrix} = I - \begin{bmatrix} z_1 & z_2 \end{bmatrix} T \begin{bmatrix} z_1^T \\ z_2^T \end{bmatrix} =$

$$I - \begin{bmatrix} t_{11}z_1 + t_{21}z_2 & t_{12}z_1 + t_{22}z_2 \end{bmatrix} \begin{bmatrix} z_1^T \\ z_2^T \end{bmatrix} =$$

$$= I - t_{11}z_1 z_1^T - t_{21}z_2 z_1^T - t_{12}z_1 z_2^T - t_{22}z_2 z_2^T$$

$$\text{con } T = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}, Z = \begin{bmatrix} z_1 & z_2 \end{bmatrix} \qquad \boxed{\Rightarrow t_{21} = 0}$$

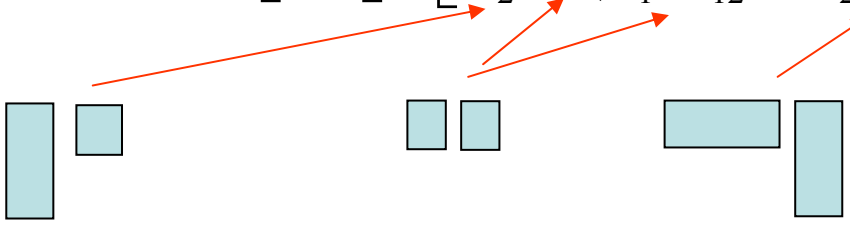
Organización por bloques

$$P_1 = I - \tau_1 z_1 z_1^{\ T}; P_2 = I - \tau_2 z_2 z_2^{\ T}; A = P_1 P_2 R_{(2)}$$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad R_{(2)} = \begin{bmatrix} \overline{R} & \overline{A} \end{bmatrix}$$

$$\overline{A} = (P_1 P_2)^T \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix} = (I - ZTZ^T)^T \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix} = (I - \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} T^T \begin{bmatrix} Z_1^{\ T} & Z_2^{\ T} \end{bmatrix}) \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}$$

$$\overline{A} = \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix} - \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} T^T (Z_1^{\ T} A_{12} + Z_2^{\ T} A_{22}) = \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix} - \begin{bmatrix} Z_1 T^T (Z_1^{\ T} A_{12} + Z_2^{\ T} A_{22}) \\ Z_2 T^T (Z_1^{\ T} A_{12} + Z_2^{\ T} A_{22}) \end{bmatrix}$$

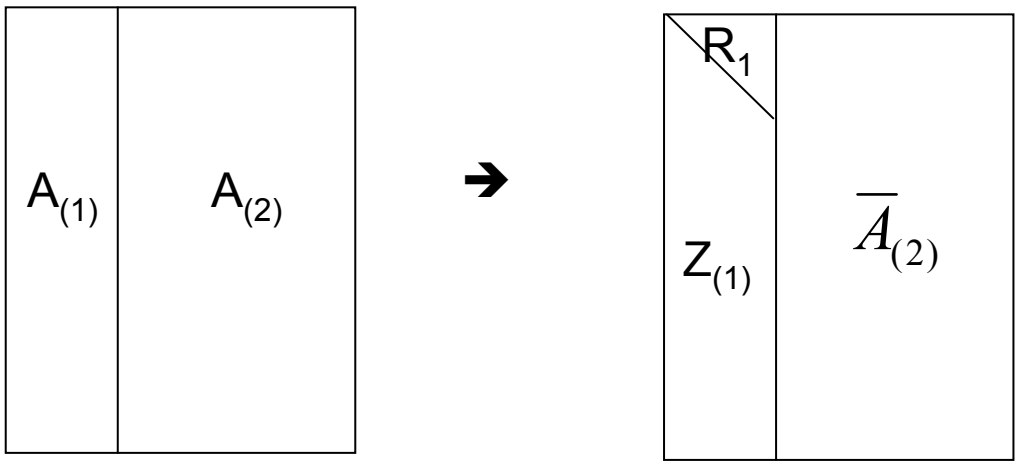


Organización por bloques

Caso general

$$P_1 = I - \tau_1 z_1 z_1^{\ T}; P_2 = I - \tau_2 z_2 z_2^{\ T}; \ldots; P_k = I - \tau_k z_k z_k^{\ T}$$

$$A = P_1 P_2 \ldots P_k R_{(2)}$$

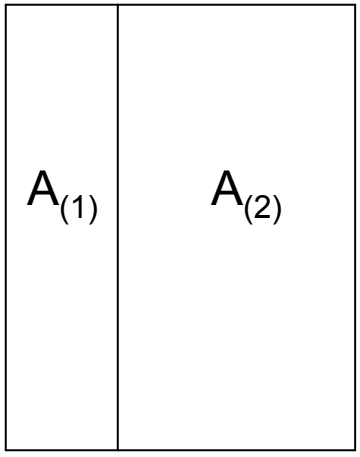


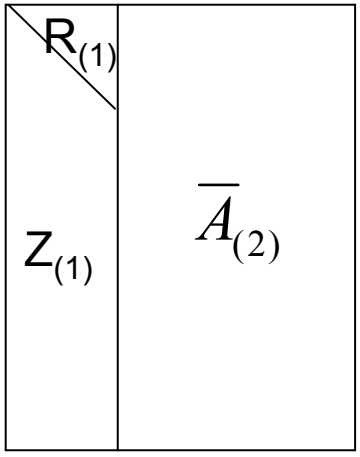
$$P_2 \ldots P_r = I - ZTZ^T = I - \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} T \begin{bmatrix} Z_1^{\ T} & Z_2^{\ T} \end{bmatrix}, \text{con } T = \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1k} \\ & t_{22} & \ldots & t_{2k} \\ & & \ldots & . \\ & & & t_{kk} \end{bmatrix}$$

Organización por bloques

1. Aplicar la triangularización de Householder a $A_{(1)}$: $Calcular\ Z_{(1)}, R_{(1)}, T_{(1}$

2. Actualizar $A_{(2)}$: $\overline{A}_{(2)} = \left(I - Z_{(1)} * T_{(1)} * Z_{(1)}{}^{T}\right)^{T} * A_{(2)}$



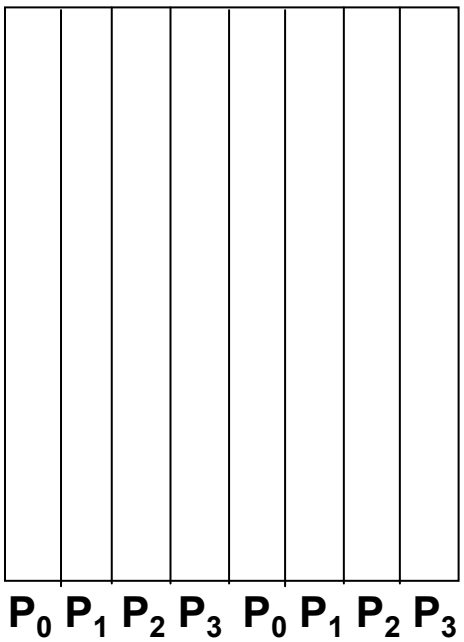


Las operaciones de actualización se pueden hacer utilizando BLAS3 ya que son productos matrizXmatriz

Algoritmo de triangularización de Householder orientado por bloques

$$nb = n / tb, \quad A_j \rightarrow P_{jMODp},$$

$$A = \left[A_0, A_1, \ldots, A_{nb-1}\right] \in \Re^{mxn}$$



$Para\ pr = 0,1,\ldots, p-1$

$En\ P_{pr}:$

$\quad Para\ j = 0,1,\ldots, nb-1$

$\quad\quad Si\ jMODp = pr$

$\quad\quad\quad Triangulariza\ A_j$ y $calcula\ Z_j, T_j$

$\quad\quad\quad Difunde\ Z_j, T_j$

$\quad\quad en\ otro\ caso$

$\quad\quad\quad Espera\ Z_j, T_j$

$\quad\quad finsi$

$\quad\quad Para\ k = j+1, j+2, \ldots, nb-1$

$\quad\quad\quad Si\ kMODp = pr$

$\quad\quad\quad\quad Actualiza\ A_k = \left[I - Z_j T_j Z_j^{\ T}\right] A_k$

$\quad\quad\quad finsi$

$\quad\quad finpara$

$\quad finpara$

Prestaciones del algoritmo

$$T_p = (2 + \frac{1}{2tb})\left[\frac{n^2}{p}\left(m - \frac{n}{3}\right)\right]Flop_{BLAS3} + 2ntb\left(m - \frac{n}{2}\right)Flop + n(m - \frac{n}{2})\tau_{DIF} + \left(\frac{n}{tb}\right)\beta_{DIF}$$

$$\frac{dT_p}{dtb} = 0 \Rightarrow tb = \sqrt{\frac{n\left(m - \frac{n}{3}\right)Flop_{BLAS3} + 2p\beta_{DIF}}{4p\left(m - \frac{n}{2}\right)Flop}}$$

Rotaciones de Givens(1)

Muy similares a las rotaciones en $\Re^{2\times 2}$, pero en $\Re^{n\times n}$

$$G(i,k,\theta)=\begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

,donde c=cos(θ) y s=sen(θ)

1) Son ortogonales

2) Son modificaciones de la identidad de rango 2.

Rotaciones de Givens(2)

Efecto de una rotación de Givens sobre un vector.

$$y = G(i,k,\theta)\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_k \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ c\cdot x_i + s\cdot x_k \\ \vdots \\ -s\cdot x_i + c\cdot x_k \\ \vdots \\ x_n \end{pmatrix}$$

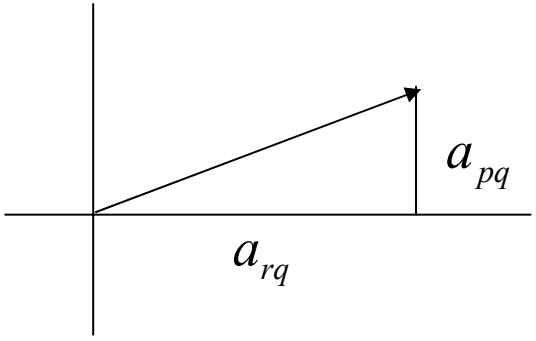
Eligiendo el ángulo θ apropiado, podremos hacer 0 en la componente ***i*** o en la ***k***; por ejemplo, podemos hacer $y_k=0$ tomando:

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}} \quad ; \quad s = \frac{x_k}{\sqrt{x_i^2 + x_k^2}}$$

Procedimiento de cálculo de rotaciones

<u>CalculaRotacion(p,q,r)</u>

$$
\begin{aligned}
&Si\ a_{pq} = 0 \\
&\quad entonces \\
&\qquad c = 1; s = 0 \\
&\quad en\ otro\ caso \\
&\qquad Si\ |a_{pq}| < |a_{rq}| \\
&\qquad\quad entonces \\
&\qquad\qquad t = a_{pq} / a_{rq}; c = 1/\sqrt{1+t^2}; s = c*t; \\
&\qquad\quad en\ otro\ caso \\
&\qquad\qquad t = a_{rq} / a_{pq}; s = 1/\sqrt{1+t^2}; c = s*t;
\end{aligned}
$$



Procedimiento de aplicación de rotaciones

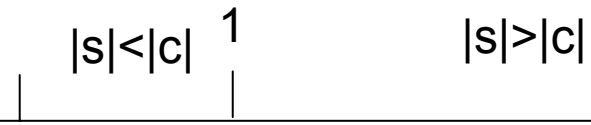
<u>AplicaRotacion(p,q,r,k)</u>

$$
\begin{aligned}
&\mathrm{u} = a_{rk}; \\
&v = a_{pk}; \\
&a_{rk} = c*u + s*v; \\
&a_{pk} = -s*u + c*v;
\end{aligned}
$$

$$
\begin{array}{cc}
\begin{bmatrix} c & s \\ -s & c \end{bmatrix}
\begin{bmatrix}
x & x & x & x & & x & x \\
x & x & x & x & \cdots & x & x
\end{bmatrix}
\begin{array}{c} \leftarrow r \\ \leftarrow p \end{array} \\
\qquad\qquad\qquad\qquad q \qquad\qquad\qquad k
\end{array}
$$

Almacenamiento de las Rotaciones

$$
\begin{aligned}
&\textbf{if } c = 0 \\
&\qquad \rho = 1 \\
&\textbf{elseif } |s| < |c| \\
&\qquad \rho = \text{sign}(c)s/2 \\
&\textbf{else} \\
&\qquad \rho = 2\text{sign}(s)/c \\
&\textbf{end}
\end{aligned}
$$



Recuperación de las Rotaciones

$$
\begin{aligned}
&\textbf{if } \rho = 1 \\
&\qquad c = 0;\ s = 1 \\
&\textbf{elseif } |\rho| < 1 \\
&\qquad s = 2\rho;\ c = \sqrt{1 - s^2} \\
&\textbf{else} \\
&\qquad c = 2/\rho;\ s = \sqrt{1 - c^2} \\
&\textbf{end}
\end{aligned}
$$

Rotaciones de Givens(3)

Algoritmo para calcular una rotación de Givens, sin problemas de overflow. Dados a y b, el algoritmo calcula c (=cos(θ)) y s (=sen(θ))
De forma que:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

```
Function [c,s]=givens(a,b)
If b==0
  c=1;s=0
Else
   If |b|>|a|
```

$$\tau = -\frac{a}{b}; \quad s = \frac{1}{\sqrt{1+\tau^2}}; \quad c = s\tau$$

```
   else
```

$$\tau = -\frac{b}{a}; \quad c = \frac{1}{\sqrt{1+\tau^2}}; \quad s = c\tau$$

```
   end
end
```

El ángulo no se calcula nunca

Rotaciones de Givens(4)
Efecto sobre una matriz

Efecto de una rotación de Givens sobre una matriz $A \in \Re^{m\times n}$ por filas

1) Si $G(i,k,\theta) \in \Re^{m\times m}$ se aplica por la izquierda, de la forma:
$A \leftarrow G(i,k,\theta)^T \cdot A$, sólo se modifican las filas **i** y **k** de A, y requiere exactamente **<u>6n</u>** flops:

```
Function A=appl_givens_f(A,i,k,c,s)
[m,n]=size(A)
For j=1:n
   t1=A(i,j)
   t2=A(k,j)
   A(i,j)=c*t1-s*t2
   A(k,j)=s*t1+c*t2
End
```

Rotaciones de Givens(5)
Efecto sobre una matriz

Efecto de una rotación de Givens sobre una matriz $A \in \Re^{m\times n}$, por columnas

1) Si $G(i,k,\theta) \in \Re^{n\times n}$ se aplica por la derecha, de la forma: $A \leftarrow A\cdot G(i,k,\theta)^T$, sólo se modifican las columnas **i** y **k** de A, y requiere exactamente **<u>6m</u>** flops:

```
Function A=appl_givens_c(A,i,j,c,s)
[m,n]=size(A)
For j=1:m
   t1=A(j,i)
   t2=A(j,k)
   A(j,i)=c*t1-s*t2
   A(j,k)=s*t1+c*t2
End
```

Rotaciones de Givens(6): Propiedades

1)Si, al aplicar una rotación de Givens por filas a un vector columna, los elementos de las filas afectadas son 0, el resultado será 0.

2) Las propiedades de error de las rotaciones de Givens son tan buenas como las de Householder.

The numerical properties of Givens rotations are as favorable as those for Householder reflections. In particular, it can be shown that the computed $\hat{c}$ and $\hat{s}$ in givens satisfy

$$\begin{aligned} \hat{c} &= c(1+\epsilon_c) & \epsilon_c &= O(\mathbf{u}) \\ \hat{s} &= s(1+\epsilon_s) & \epsilon_s &= O(\mathbf{u}). \end{aligned}$$

If $\hat{c}$ and $\hat{s}$ are subsequently used in a Givens update, then the computed update is the exact update of a nearby matrix:

$$\begin{aligned} fl[\hat{G}(i,k,\theta)^T A] &= G(i,k,\theta)^T (A+E) & \| E \|_2 \approx \mathbf{u} \| A \|_2 \\ fl[A\hat{G}(i,k,\theta)] &= (A+E)G(i,k,\theta) & \| E \|_2 \approx \mathbf{u} \| A \|_2. \end{aligned}$$

A detailed error analysis of Givens rotations may be found in Wilkinson (AEP, pp. 131-39).

Descomposición QR basada en Givens(1)

Tomamos la misma matriz de ejemplo que en el caso de Householder, 6*5

Aplicamos una rotación a las filas 5 y 6, para hacer 0 el elemento (6,1)

$$A = \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix} \longrightarrow R^{1}_{5,6} A = \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ \rightarrow x & x & x & x & x \\ \rightarrow 0 & x & x & x & x \end{pmatrix}$$

Descomposición QR basada en Givens(2)

A continuación, aplicamos una rotación a las filas 4 y 5, para hacer 0 el elemento (5,1)

$$R^{1}_{5,6}A = \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix} \longrightarrow R^{1}_{4,5}R^{1}_{5,6}A = \begin{array}{c} \\ \\ \\ \rightarrow \\ \rightarrow \\ \\ \end{array} \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix}$$

Se sigue con filas 3 y 4, luego 2 y 3, hasta 1 y 2:

$$R^{1}_{1,2}\cdots R^{1}_{4,5}R^{1}_{5,6}A = \begin{array}{c} \rightarrow \\ \rightarrow \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix}$$

Descomposición QR basada en Givens(3)

Se sigue con la columna 2, empezando desde abajo hasta la diagonal:

$$R_{1,2}^{1}\cdots R_{4,5}^{1}R_{5,6}^{1}A=\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix}\Rightarrow \begin{array}{c} \\ \\ \\ \\ \rightarrow \\ \rightarrow \end{array}\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \end{pmatrix}\Rightarrow\cdots$$

$$\Rightarrow R_{2,2}^{2}\cdots R_{5,6}^{1}A \begin{array}{c} \\ \rightarrow \\ \rightarrow \\ \\ \\ \\ \end{array}\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{pmatrix}$$

Las filas a las que se aplica la rotación se modifican, salvo en la Columna 1; los ceros permanecen (0·c+0·s=0)

Descomposición QR basada en Givens(4)

Se repite el proceso con todas las columnas hasta la última.

$$R_{5,6}^{5} \cdots R_{4,5}^{1} R_{5,6}^{1} A = \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Obtenemos $R=Q^T \cdot A$, donde Q^T es el producto de todas las rotaciones

Descomposición QR basada en Givens(5)

Algoritmo "simplificado" para calcular la descomposición QR de una matriz $A \in \Re^{m \times n}$; La Q se obtiene explícitamente, normalmente solo se guardan los vectores v.

```
[Q,R]=qr_simpl_Giv(A)

1) Q=I_m
2) For j=1:n
3)     For i=m:-1:j+1
4)         [c,s]=givens(A(i-1,j),A(i,j))
5)         A=appl_givens_f(A,i-1,i,c,s)
6)         Q=appl_givens_f(Q,i-1,i,c,s)
7)     End For
8) End For
```

Coste: $3n^2(m-n/3)$ flops

Estabilidad numérica del algoritmo

Las propiedades de error de las rotaciones de Givens son tan buenas como las de Householder.

We offer some remarks about the propagation of roundoff error in algorithms that involve sequences of Householder/Givens updates. To be precise, suppose $A = A_0 \in \mathbb{R}^{m \times n}$ is given and that matrices $A_1, \ldots, A_p = B$ are generated via the formula

$$A_k = fl(\hat{Q}_k A_{k-1} \hat{Z}_k) \qquad k = 1{:}p .$$

Assume that the above Householder and Givens algorithms are used for both the generation and application of the $\hat{Q}_k$ and $\hat{Z}_k$. Let Q_k and Z_k be the orthogonal matrices that would be produced in the absence of roundoff. It can be shown that

$$B = (Q_p \cdots Q_1)(A + E)(Z_1 \cdots Z_p), \tag{5.1.11}$$

where $\| E \|_2 \leq cu \| A \|_2$ and c is a constant that depends mildly on n, m, and p. In plain English, B is an exact orthogonal update of a matrix near to A.

Descomposición QR basada en Givens: Casos especiales

Reducción a la forma triangular superior de una matriz de Hessenberg superior

$$Q^T \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ & x & x & x \\ & & x & x \end{bmatrix} \rightarrow \begin{bmatrix} x & x & x & x \\ 1 & x & x & x \\ & 2 & x & x \\ & & 3 & x \end{bmatrix}$$

Coste: $3n^2$ flops

Reducción a la forma triangular superior de una matriz tridiagonal

$$Q^T \begin{bmatrix} x & x & & \\ x & x & x & \\ & x & x & x \\ & & x & x \end{bmatrix} \rightarrow \begin{bmatrix} x & x & y & \\ 1 & x & x & y \\ & 2 & x & x \\ & & 3 & x \end{bmatrix}$$

Coste: 25n flops

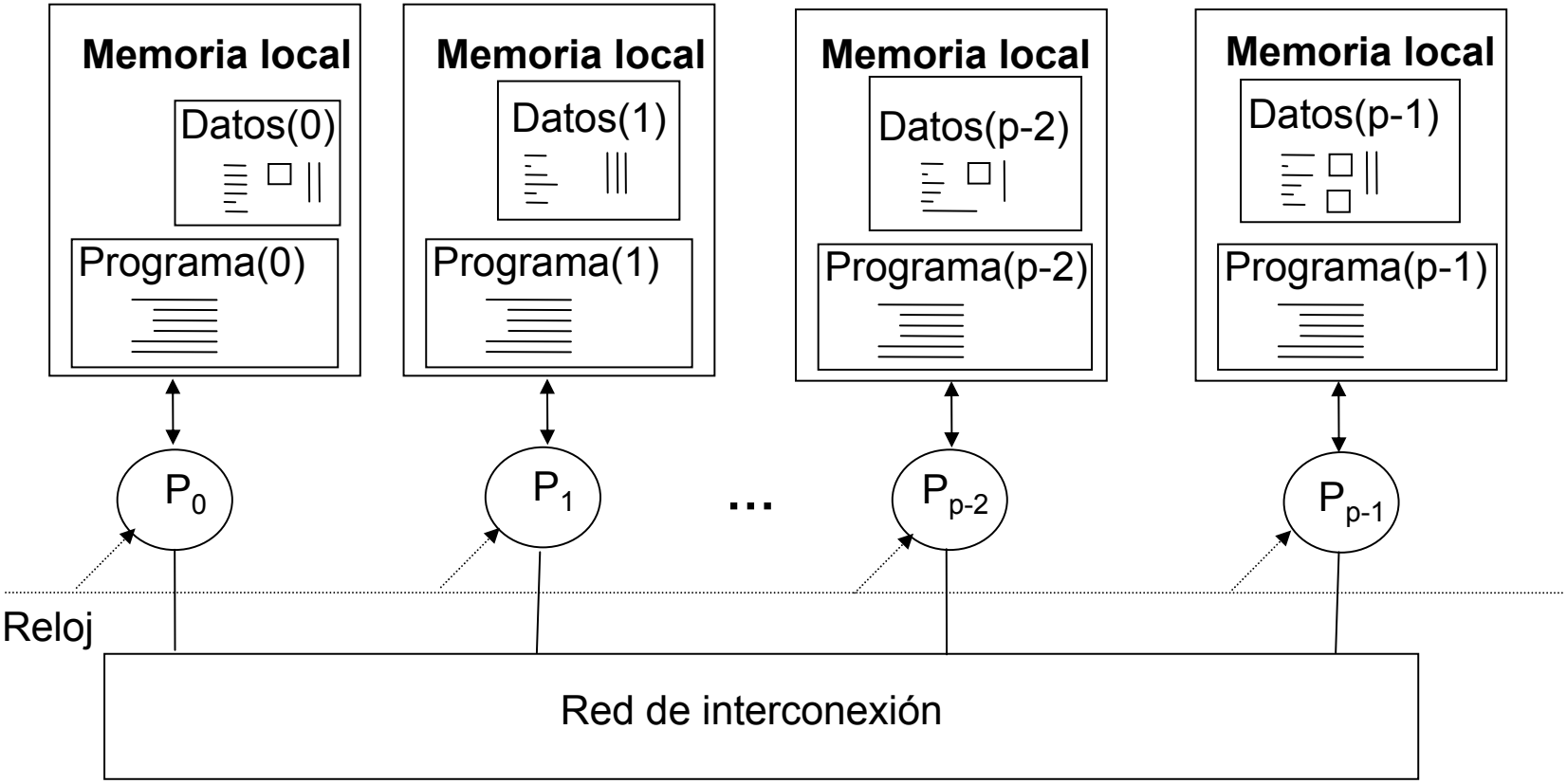
Reducción a la forma triangular inferior de una matriz tridiagonal

$$\begin{bmatrix} x & x & & \\ x & x & x & \\ & x & x & x \\ & & x & x \end{bmatrix} Q^T \rightarrow \begin{bmatrix} x & 1 & & \\ x & x & 2 & \\ y & x & x & 3 \\ & y & x & x \end{bmatrix}$$

Coste: 25n flops

Paralelización del Algoritmo de Givens en el Modelo de Paso de Mensajes

Objetivo:
Diseñar un algoritmo paralelo óptimo para diseñar un algoritmo paralelo óptimo para triangularizar una matriz en un multicomputador utilizando Rotaciones de Givens



Características:

Entorno de paso de mensajes (MPI)

Se conoce el tiempo de ejecución de 1 Flop (operación elemental en coma flotante:+,-,*,:)

Enviar un mensaje de N palabras desde un nodo a otro (o a otros) tiene un coste de $N\tau+\beta$, con τ y β conocidos ($\tau=t_w$ **y** $\beta=t_s$)

$$t_p \cong t_A + t_C - t_{sol} \cong t_A + t_C$$

Paralelismo en las Rotaciones de Givens

Para anular el elemento (p,q), los parámetros c,s sólo dependen de de dos elementos y la anulación sólo implica dos filas

$$
\begin{array}{c}
\begin{array}{cccccc} & & q & p & & \end{array} \\
\begin{bmatrix}
1 & & & & & \\
 & 1 & & & & \\
 & & c & s & & \\
 & & -s & c & & \\
 & & & & 1 & \\
 & & & & & 1
\end{bmatrix}
\begin{bmatrix}
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x
\end{bmatrix}
=
\begin{bmatrix}
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
+ & + & + & + & + \\
x & x & x & x & x \\
+ & + & + & + & + \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x
\end{bmatrix}
\begin{array}{c} \\ \\ \\ q \\ \\ p \\ \\ \\ \\ \end{array}
\end{array}
$$

Clases de paralelismo

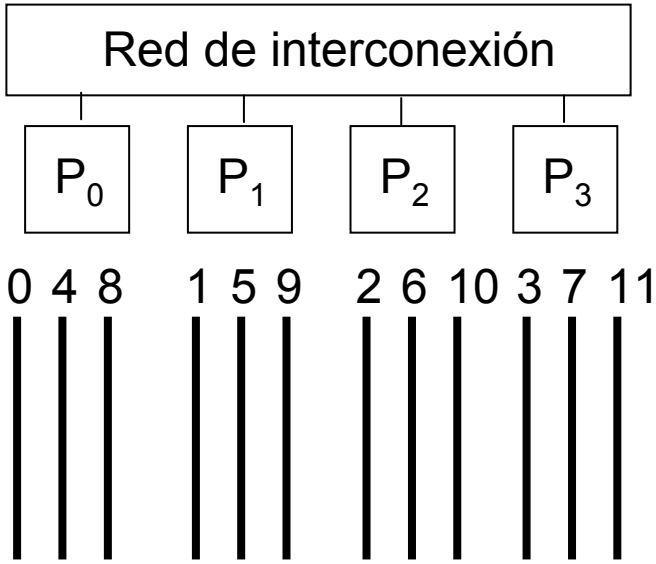
1. Calcular los parámetros de diferentes rotaciones (filas disjuntas) en paralelo
2. Aplicar simultáneamente rotaciones diferentes a pares de filas disjuntas
3. Aplicar en paralelo una rotación a todas las columnas en un par de filas

Algoritmo orientado por columnas

Distribución cíclica: suponiendo n=k*p

$$Columna\ j \rightarrow P_{jMODp}$$

$$\text{Procesador } i \rightarrow \text{ Columnas } rp+i, r=0,1,\ldots,(n/p)-1$$



Idea básica del algoritmo

Para cada columna

1. Calcular los parámetros de las rotaciones de Givens (sólo el procesador que la contiene)
2. Difundir los parámetros al resto de procesadores
3. Actualizar las restantes columnas (cada procesador sólo las que contiene)

Algoritmo paralelo de Givens orientado por columnas

```
EN PARALELO: PARA pr=0,1,…,p-1
En P_pr:
    PARA j=0,1,…,n-1
        PARA i=m-1,m-2,…,j+1
            SI jMODp=pr (*columna j pertenece a P_pr *)
                ENTONCES
                    [c,s]=givens(A,i-1,i)
                    Difunde parámetros c,s
                EN OTRO CASO
                    Espera parámetros c,s
            FIN SI
            PARA k=j,j+1,…,n-1
                SI kMODp=pr (*columna k pertenece a P_pr *)
                ENTONCES
                    u=A(i-1,k);v=A(i,k);
                    A(i-1,k)=c*u+s*v;A(i,k)=-s*u+c*v
                FIN SI
            FIN PARA
        FIN PARA
    FIN PARA
FIN PARA
```

Modelos clásicos de paralelismo

Sameh & Kuck

x	x	x	x	x	x
11	x	x	x	x	x
10	12	x	x	x	x
9	11	13	x	x	x
8	10	12	14	x	x
7	9	11	13	15	x
6	8	10	12	14	16
5	7	9	11	13	15
4	6	8	10	12	14
3	5	7	9	11	13
2	4	6	8	10	12
1	3	5	7	9	11

Greedy

Modi & Clarke

x	x	x	x	x	x
4	x	x	x	x	x
3	6	x	x	x	x
2	5	8	x	x	x
2	4	7	10	x	x
2	4	6	9	12	x
1	3	6	8	11	14
1	3	5	7	10	13
1	3	5	7	9	12
1	2	4	6	8	11
1	2	4	6	8	10
1	2	3	5	7	9

Modelo secuencial

x	x	x	x	x	x
1	x	x	x	x	x
2	3	x	x	x	x
4	5	6	x	x	x
7	8	9	10	x	x
11	12	13	14	15	x
16	17	18	19	20	21
22	23	24	25	26	27
28	29	30	31	32	33
34	35	36	37	38	39
40	41	42	43	44	45
46	47	48	49	50	51

Modelo Paralelo derivado del secuencial

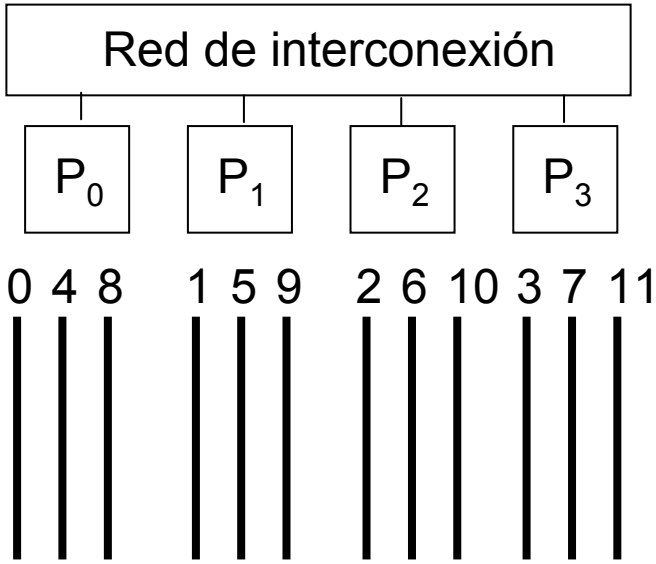
x	x	x	x	x	x
1	x	x	x	x	x
2	3	x	x	x	x
3	4	5	x	x	x
4	5	6	7	x	x
5	6	7	8	9	x
6	7	8	9	10	11
7	8	9	10	11	12
8	9	10	11	12	13
9	10	11	12	13	14
10	11	12	13	14	15
11	12	13	14	15	16

Algoritmo por columnas con optimización de las comunicaciones usando el modelo de paralelismo de las diagonales secundarias

Distribución cíclica: suponiendo n=k*p

$$Columna\ j \rightarrow P_{jMODp}$$

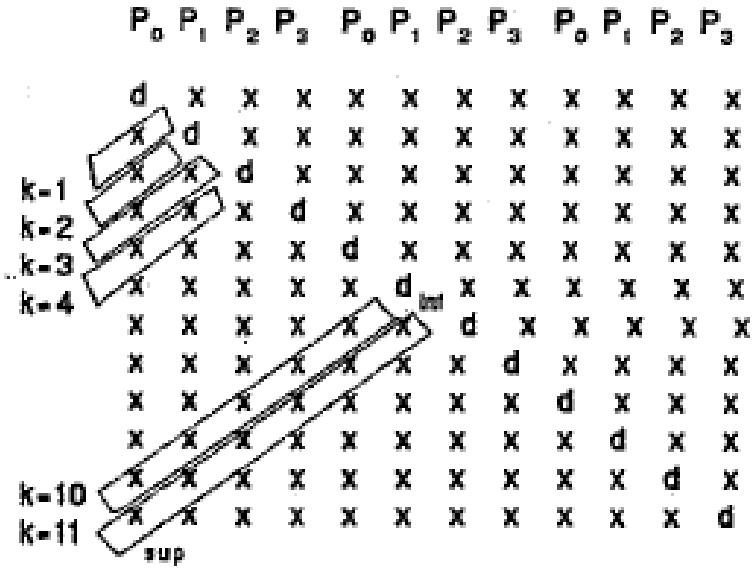
$$\text{Procesador } i \rightarrow \text{ Columnas } rp+i, r=0,1,\ldots,(n/p)-1$$

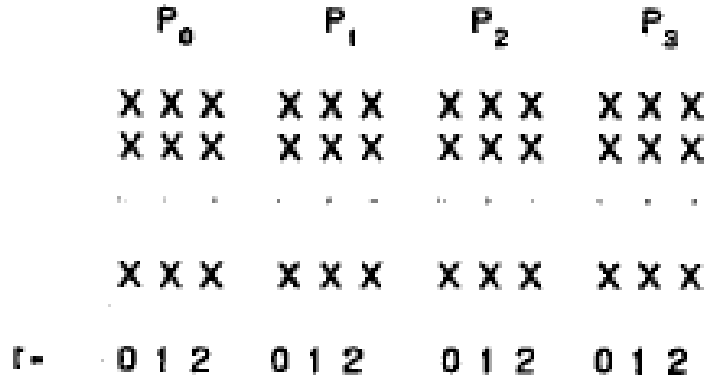


Idea básica del algoritmo

1. En paralelo, calcular los parámetros de las rotaciones de Givens que anulen elementos situados en una diagonal secundaria
2. Difundir los parámetros al resto de procesadores
3. En paralelo, actualizar las restantes columnas (cada procesador sólo las que contiene)

Funcionamiento





Algoritmo paralelo de Givens orientado por columnas y diagonales secundarias

```
EN PARALELO : Para i=0,1,....,p-1
   En Pi:
      Para k=1,2,...,m+n-2
        inf:=max{k-n+1,kDIV2+1}
        sup:=min{k,m-1}
        Para s=inf,inf+1,....,sup
            Si (k-s)MODp=i
              Entonces
                  CalculaRotacion
                      (s,k-s,k-s)

        Para s=inf,....,sup
          Si (k-s)MODp=i
            Entonces
              transferir parametros
              de Rotacion (s,k-s,k-s)
            En otro caso
              esperar parametros
              de Rotacion (s,k-s,k-s)

        Para s=inf,....,sup
          Para r=0,1,...,n/p-1
            Si r*p+i>=k-s
              Entonces
                AplicaRotacion
                  (s,k-s,k-s,r*p+i)
```

Algoritmo paralelo de Givens orientado por filas

Distribución

Por bloques de filas consecutivas

Suponemos $m=p$

fila j ---> $P_{j\,DIV(m/p)}$

P_j ---> $j(m/p)+r$, con $r=0,1,\ldots,(m/p)-1$

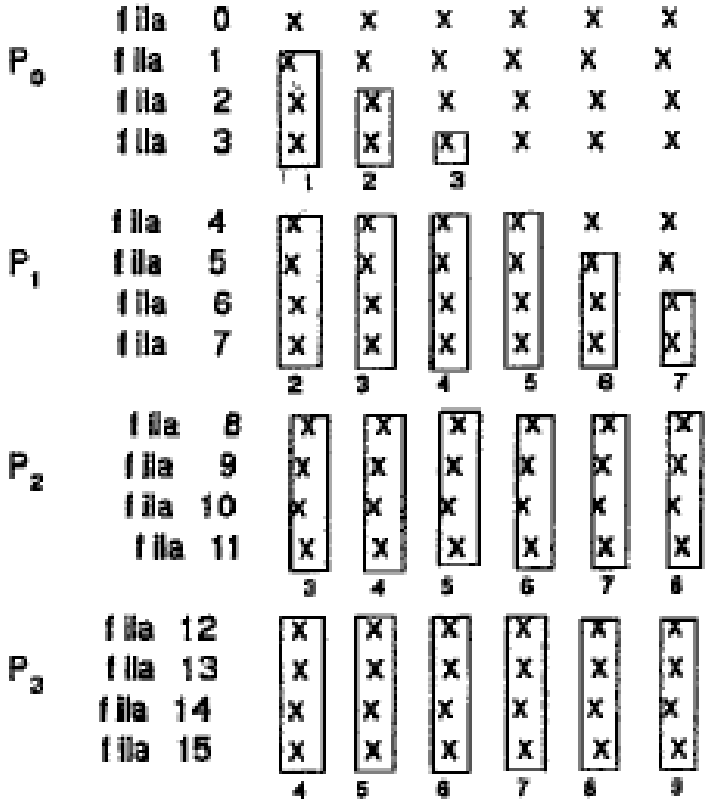
Ideas básicas del Algoritmo

En Paralelo, en cada Procesador:

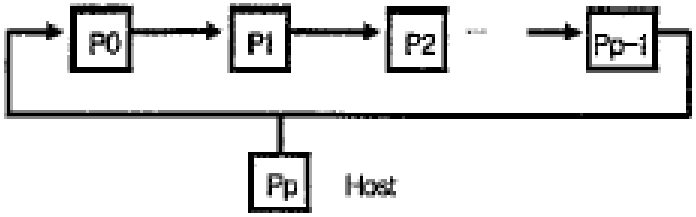
Repetir para las n primeras filas:

1) Esperar fila, si no pertenece al Procesador
2) Anular elementos de la columna del mismo indice que estén en el procesador
3) Transferir fila al siguiente Procesador

Funcionamiento del algoritmo de Givens orientado por filas



Algoritmo orientado por filas




```
EN PARALELO: Para s=0,1,..,p-1
 EN Ps
   PARA i=0,1,..,n-1
        SI i DIV p = s
            ENTONCES
                PARA j=i+1,i+2,..,(s+1)(m/p) - 1
                    CalculaRotación(j,i)
                    PARA k=i+1,..,n-1
                        AplicaRotación(j,i,k)
                Transfiere fila i a Ps+1
            EN OTRO CASO
                SI i < sm/p
                   ENTONCES
                       Espera fila i
                       PARA j=sm/p,(sm/p)+1,..,(s+1)m/p - 1
                           CalculaRotación(j,i)
                           PARA k=i+1,..,n-1
                               AplicaRotación(j,i,k)
                       Transfiere fila i a Ps+1
```

Algoritmo orientado por filas con triangularizaciones parciales

IDEAS BASICAS

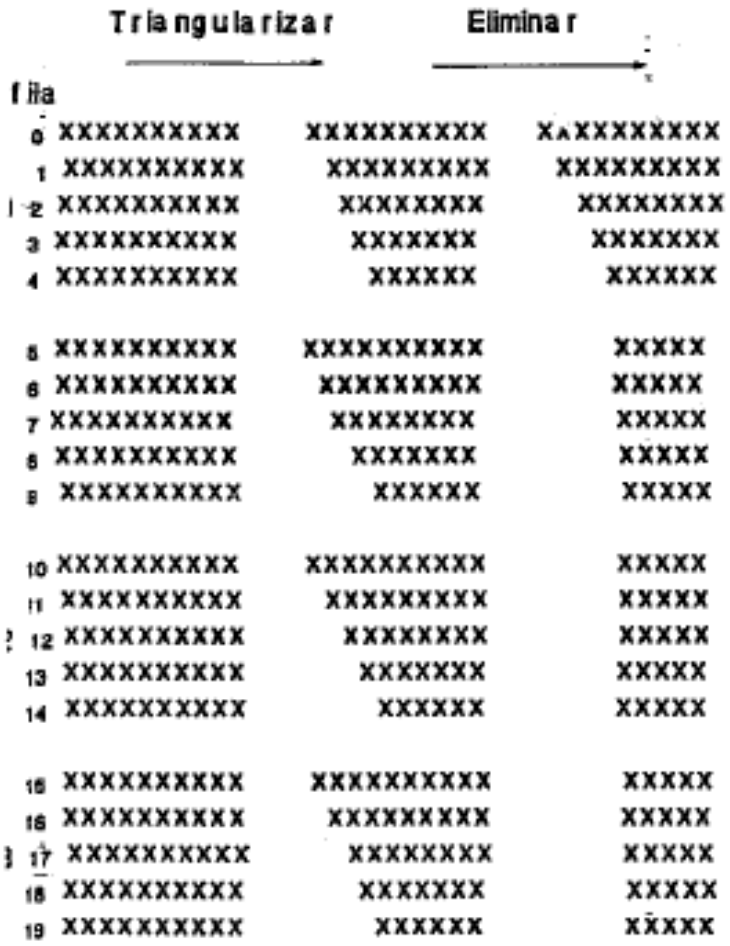
REPETIR

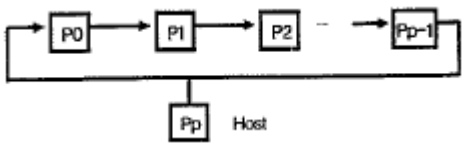
- EN PARALELO: Triangularizar la submatriz contenida en cada procesador

- Eliminar los elementos delanteros de cada submatriz trapezoidal hasta que la submatriz sea rectangular, aplicando el mismo método que en el algoritmo anterior

HASTA que la matriz sea triangular

Algoritmo orientado por filas con triangularizaciones parciales

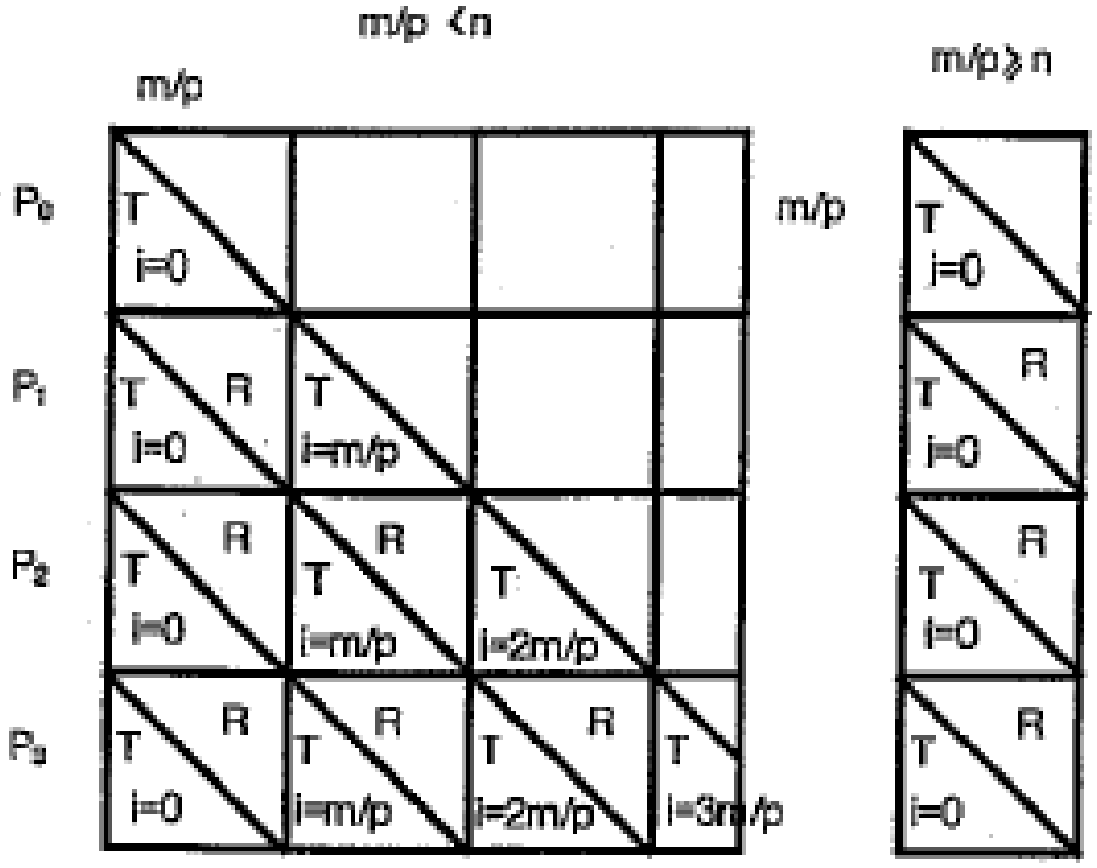




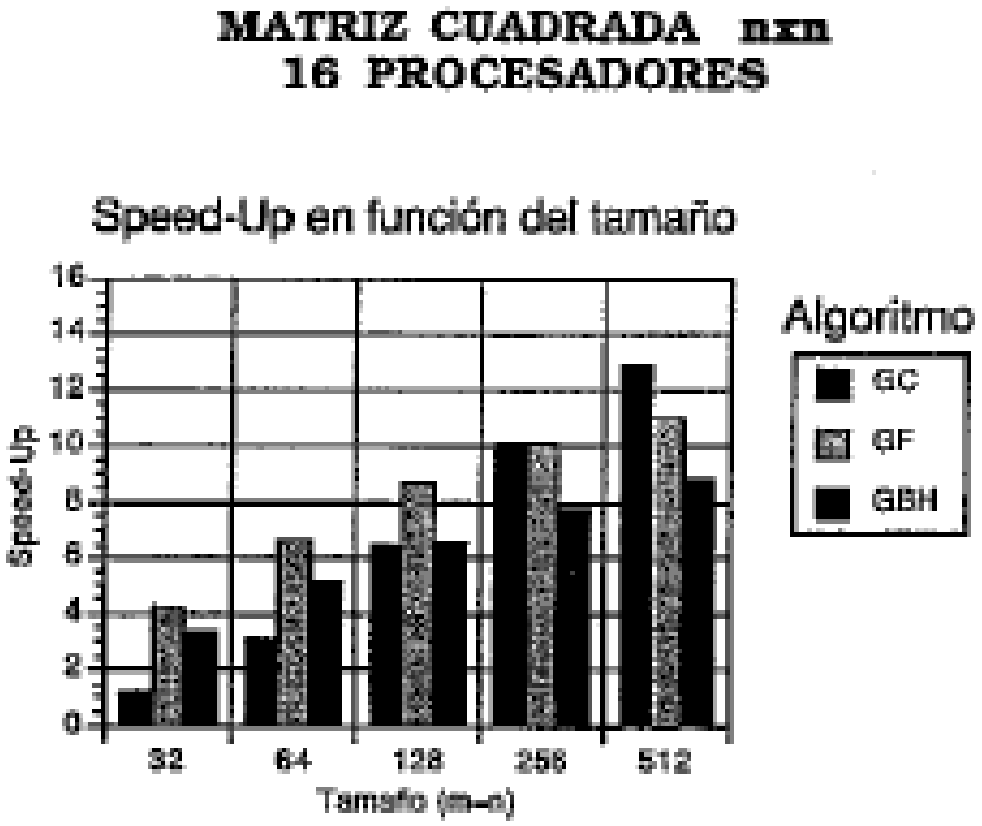
ALGORITMO CON TRIANGULARIZACIONES PARCIALES

```
EN PARALELO: Para s=0,1..p-1
 EN Ps:
    PARA i=0,1..,n-1
          SI i MOD (m/p) = 0   Y   i DIV (m/p)  ≤ s
              ENTONCES
                    Triangulariza submatriz As contenida en Ps
          SI i DIV p = s
              ENTONCES
                    PARA j=i+1,i+2...(s+1)(m/p) - 1
                        CalculaRotación(j,i)
                        PARA k=i+1..,n-1
                            AplicaRotación(j,i,k)
                    Transfiere fila i a Ps+1
              EN OTRO CASO
                    SI i<sm/p
                       ENTONCES
                          Espera fila i
                          PARA j=sm/p,(sm/p)+1...(s+1)(m/p) - 1
                              CalculaRotación(j,i)
                              PARA k=i+1..,n-1
                                  AplicaRotación(j,i,k)
                          Transfiere fila i a Ps+1
```

Funcionamiento

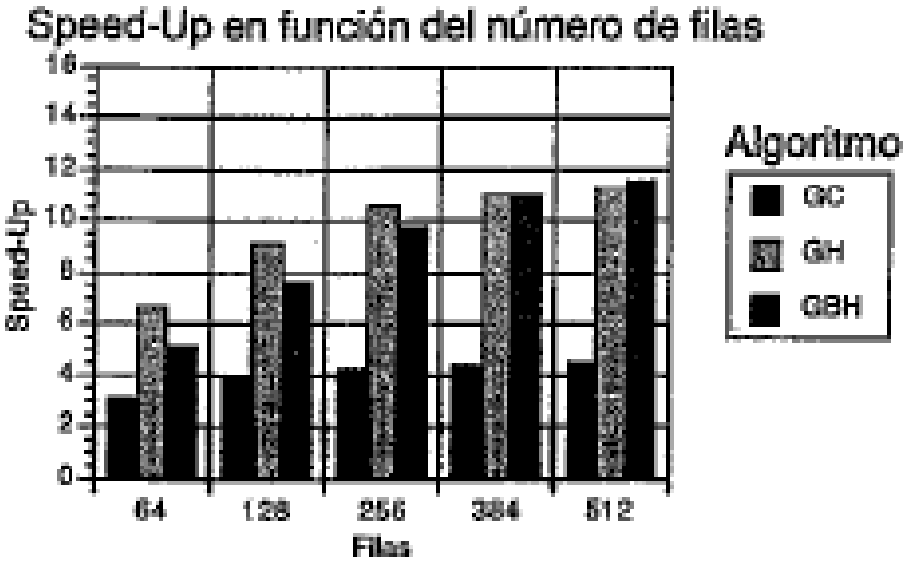


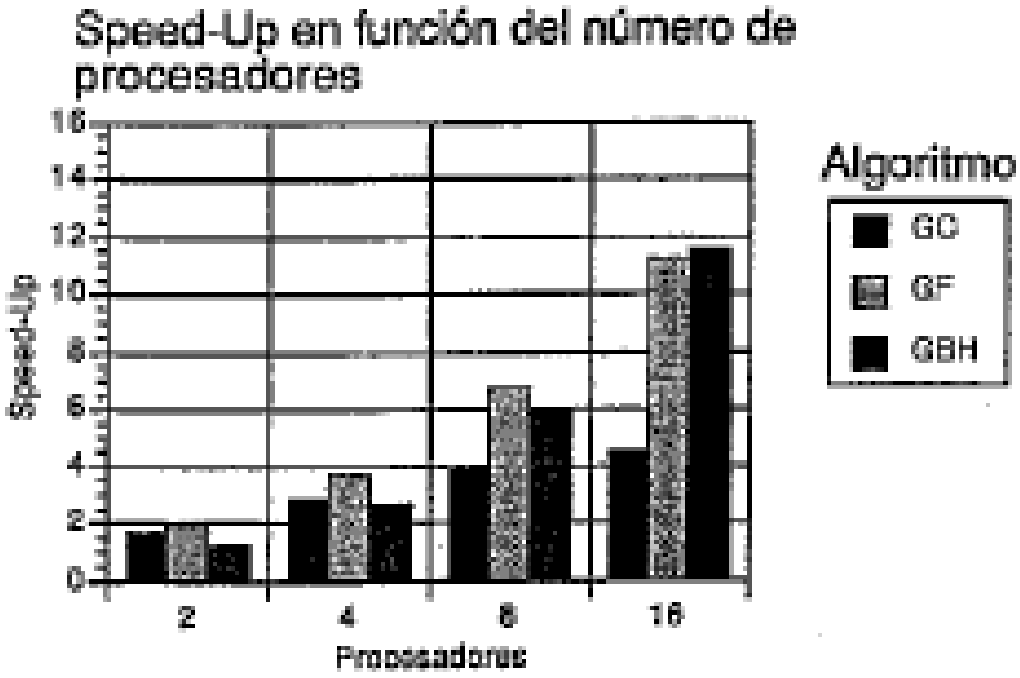
Prestaciones de los algoritmos paralelos basados en Rotaciones de Givens



Prestaciones de los algoritmos paralelos basados en Rotaciones de Givens

MATRIZ RECTANGULAR mx64
16 PROCESADORES



Prestaciones de los algoritmos paralelos basados en Rotaciones de Givens

MATRIZ RECTANGULAR 512x64