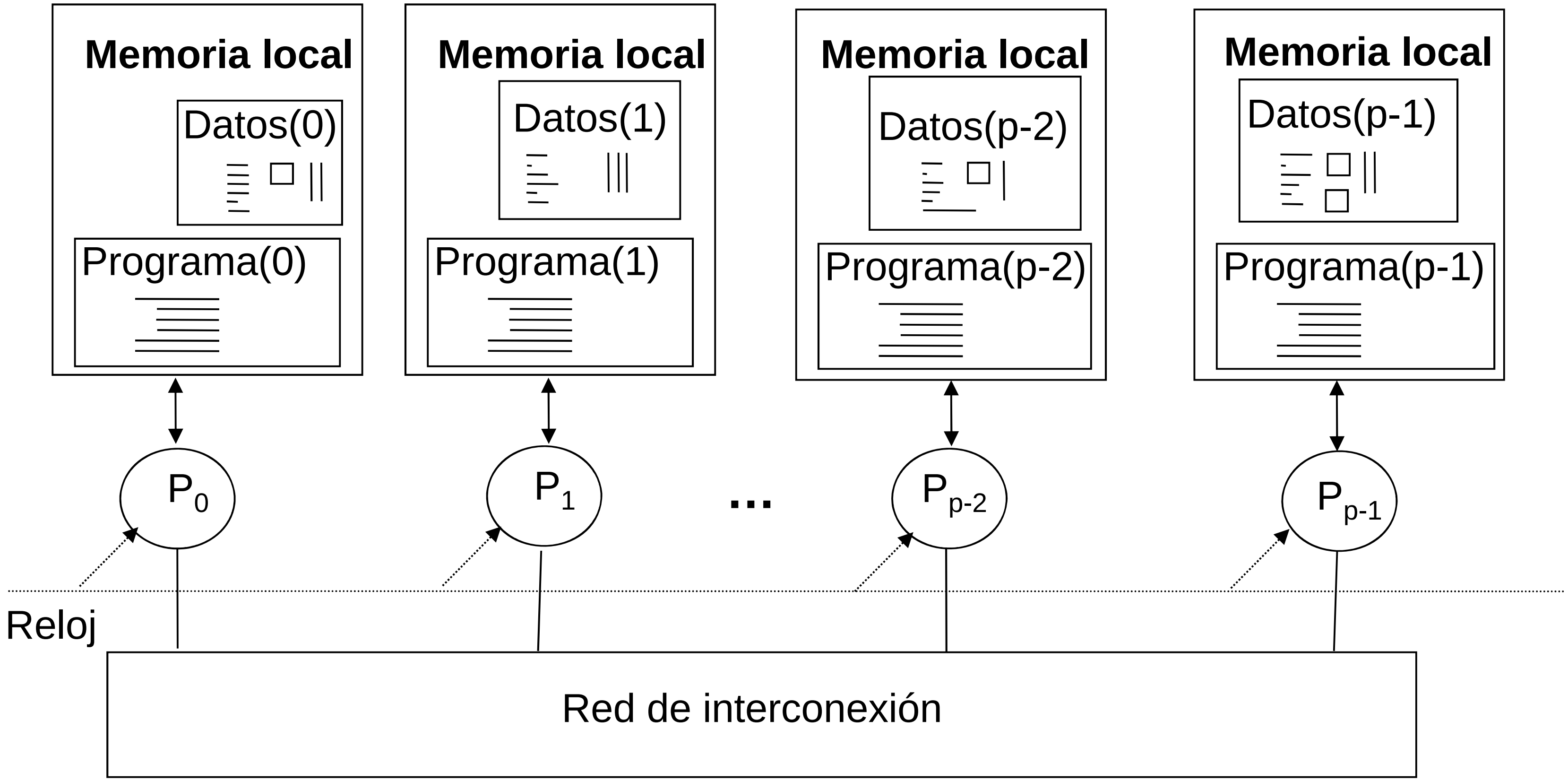
Máster Universitario en Computación Paralela y Distribuida
Algoritmos Paralelos Matriciales en Ingeniería

Tema 2.

La Descomposición QR. Algoritmos paralelos basados en rotaciones de Givens. Método de Gram-Schmidt.

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 **(τ=t_w y β=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

$$
\begin{array}{c}
\begin{array}{ccccccccc} & & & q & & p & & & \end{array} \\
\begin{bmatrix}
1 & & & & & & & & \\
 & .. & & & & & & 0 & \\
 & & 1 & & & & & & \\
 & & & c & & s & & & \\
 & & & & 1 & & & & \\
 & & & -s & & c & & & \\
 & & & & & & 1 & & \\
 & 0 & & & & & & .. & \\
 & & & & & & & & 1
\end{bmatrix}
\end{array}
\begin{bmatrix}
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x
\end{bmatrix}
=
\begin{array}{cc}
\begin{bmatrix}
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
+ & + & + & + & + & + & + & + & + \\
x & x & x & x & x & x & x & x & x \\
+ & + & + & + & + & + & + & + & + \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x \\
x & x & x & x & x & x & x & x & x
\end{bmatrix}
&
\begin{array}{c} \\ \\ \\ q \\ \\ p \\ \\ \\ \\ \end{array}
\end{array}
$$

Clases de paralelismo

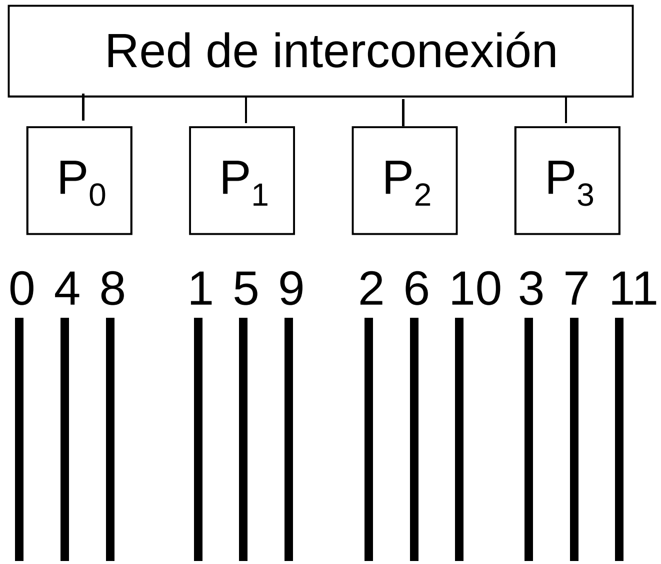
1. Calcular los parámetros de diferentes rotaciones (filas disjuntas) en paralelo
2. Aplicar simultáneamente rotaciones diferentes a pares de filas disjuntas
3. Aplicar en paralelo una rotación a todas las columnas en un par de filas

Algoritmo orientado por columnas

Distribución cíclica: suponiendo n=k*p

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

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



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,j)
                        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

$$
\begin{bmatrix}
x & x & x & x & x & x \\
11 & x & x & x & x & x \\
10 & 12 & x & x & x & x \\
9 & 11 & 13 & x & x & x \\
8 & 10 & 12 & 14 & x & x \\
7 & 9 & 11 & 13 & 15 & x \\
6 & 8 & 10 & 12 & 14 & 16 \\
5 & 7 & 9 & 11 & 13 & 15 \\
4 & 6 & 8 & 10 & 12 & 14 \\
3 & 5 & 7 & 9 & 11 & 13 \\
2 & 4 & 6 & 8 & 10 & 12 \\
1 & 3 & 5 & 7 & 9 & 11
\end{bmatrix}
$$

Greedy
Modi & Clarke

$$
\begin{bmatrix}
x & x & x & x & x & x \\
4 & x & x & x & x & x \\
3 & 6 & x & x & x & x \\
2 & 5 & 8 & x & x & x \\
2 & 4 & 7 & 10 & x & x \\
2 & 4 & 6 & 9 & 12 & x \\
1 & 3 & 6 & 8 & 11 & 14 \\
1 & 3 & 5 & 7 & 10 & 13 \\
1 & 3 & 5 & 7 & 9 & 12 \\
1 & 2 & 4 & 6 & 8 & 11 \\
1 & 2 & 4 & 6 & 8 & 10 \\
1 & 2 & 3 & 5 & 7 & 9
\end{bmatrix}
$$

Modelos clásicos de paralelismo

Secuencial por diagonales secundarias

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

Paralelo por diagonales secundarias

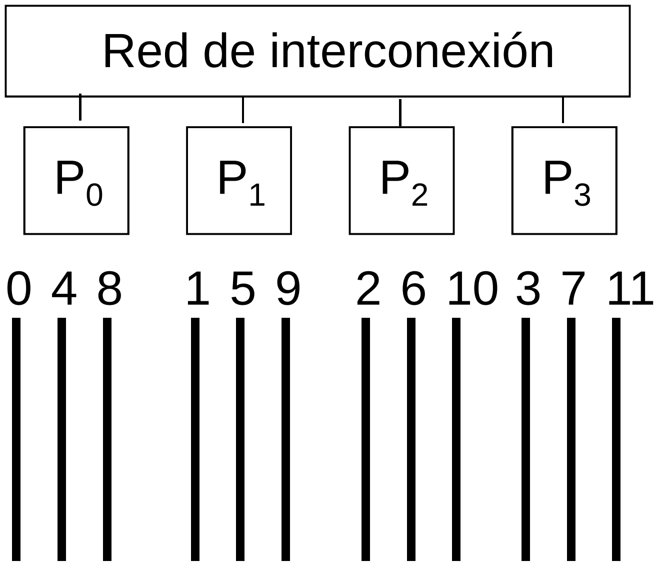
$$
\begin{bmatrix}
x & x & x & x & x & x \\
1 & x & x & x & x & x \\
2 & 3 & x & x & x & x \\
3 & 4 & 5 & x & x & x \\
4 & 5 & 6 & 7 & x & x \\
5 & 6 & 7 & 8 & 9 & x \\
6 & 7 & 8 & 9 & 10 & 11 \\
7 & 8 & 9 & 10 & 11 & 12 \\
8 & 9 & 10 & 11 & 12 & 13 \\
9 & 10 & 11 & 12 & 13 & 14 \\
10 & 11 & 12 & 13 & 14 & 15 \\
11 & 12 & 13 & 14 & 15 & 16
\end{bmatrix}
$$

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

Distribución cíclica: suponiendo n=k*p

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

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



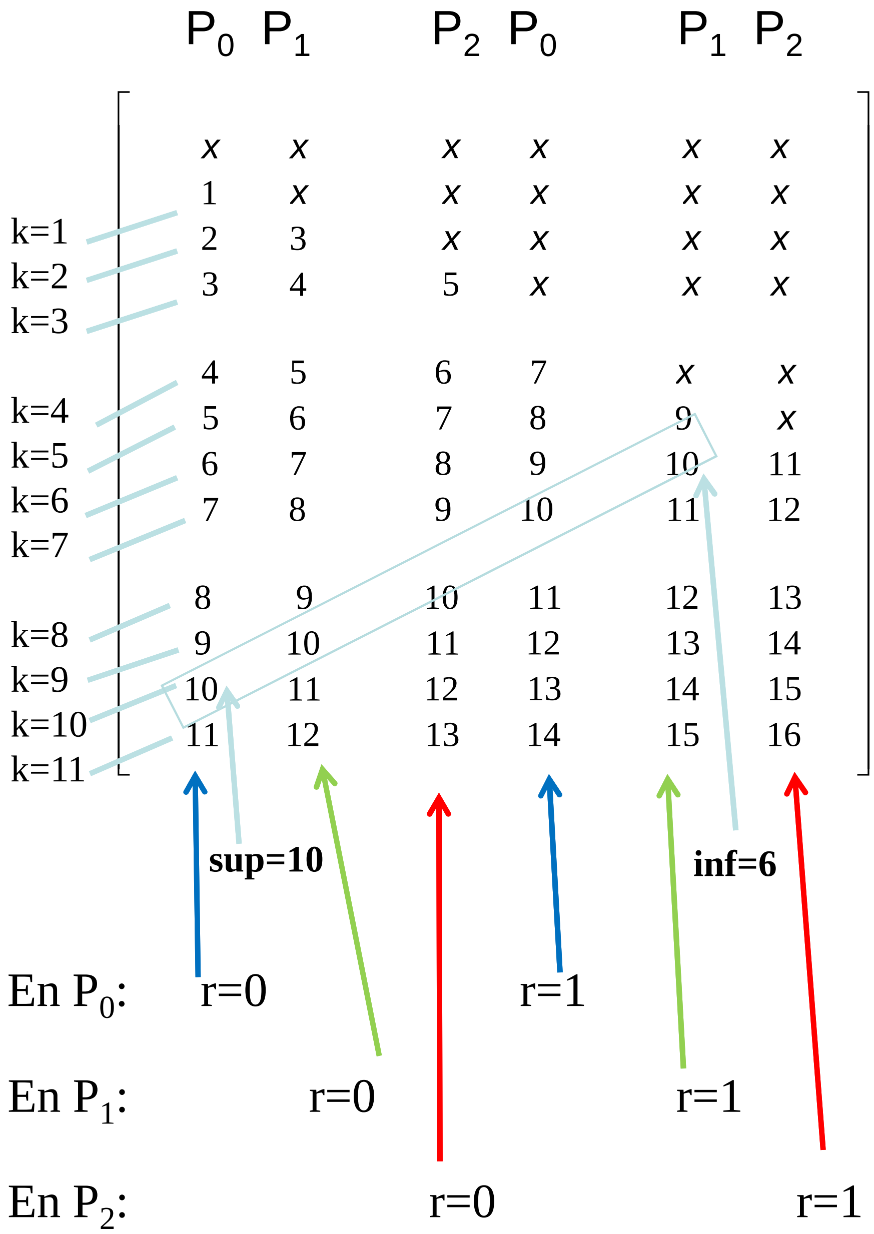
Idea básica del algoritmo

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)

Algoritmo paralelo de Givens orientado por columnas y diagonales secundarias

```
EN PARALELO: Para pr = 0,1,..., p-1
    En P_pr :
        Para k = 1, 2,..., m+n-2
            inf  =  max {k-n+1 ,  k DIV 2  +1 }
            sup  =  min {k , m-1 }
            Para t = inf, inf+1,..., sup
                Si (k-t) MOD p = pr
                    [c_t, s_t] = Givens(A, k-t, t, k-t)
                FinSi
            FinPara
            Para t = inf, inf+1,..., sup
                Si (k-t) MOD p = pr
                    Transfiere parametros [c_t, s_t]
                Si No
                    Espera parametros [c_t, s_t]
                FinSi
            FinPara
            Para t = inf, inf+1,..., sup
                Para r = 0,1,..., n/p - 1
                    Si r * p + pr ≥ k-t
                        AplicaRotacion(k-t, t, k-t, r * p + pr)
                    FinSi
                FinPara
        FinPara
```

Funcionamiento



Algoritmo paralelo de Givens orientado por filas

Distribución: por bloques de filas consecutivas, suponiendo m=k*p

$Fila\ j \;\rightarrow\; P_{j DIV (m/p)}$

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

Ideas básicas del Algoritmo:

En paralelo, en cada Procesador

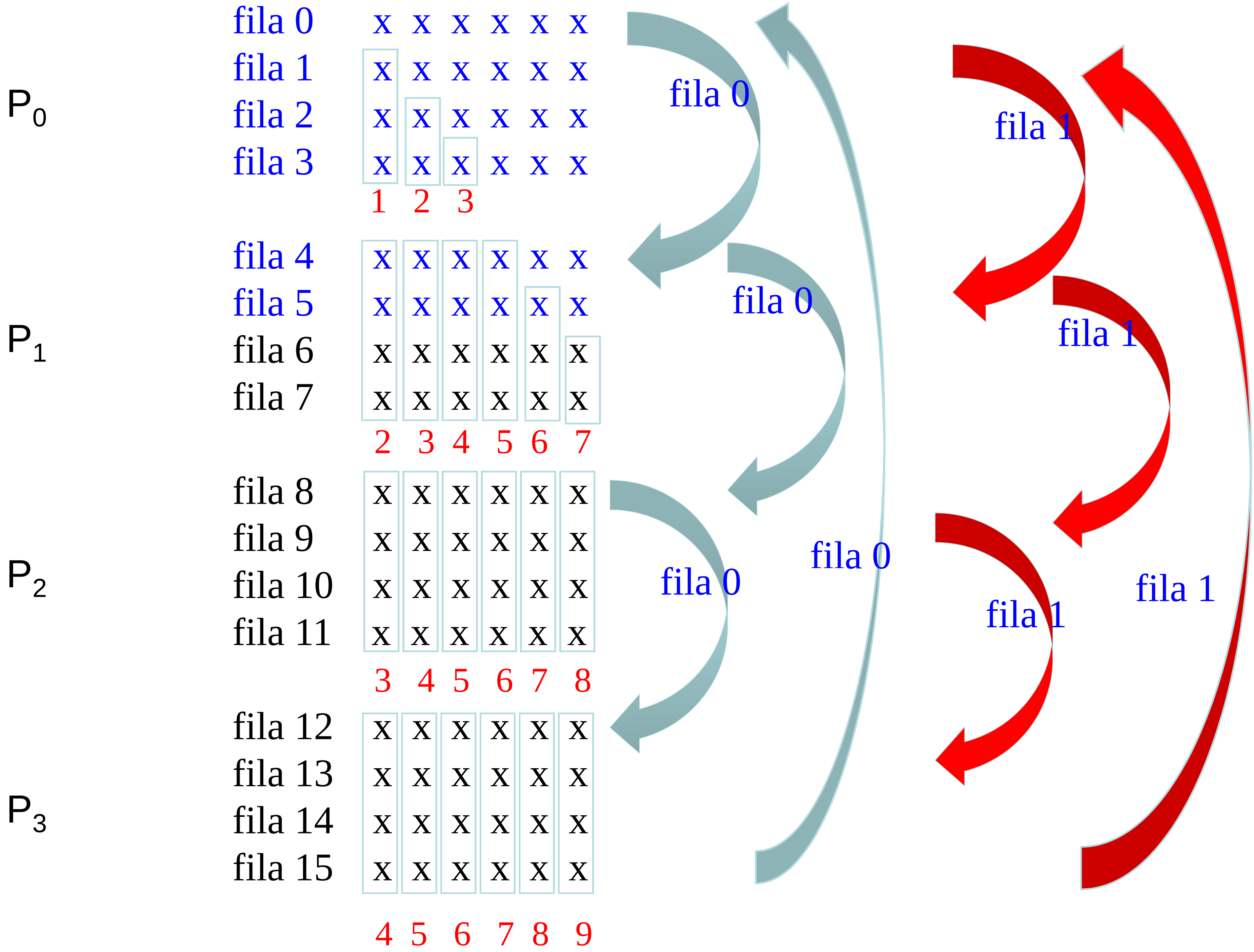
Repetir para las n primeras filas

1) Esperar una fila, si no pertenece al Procesador

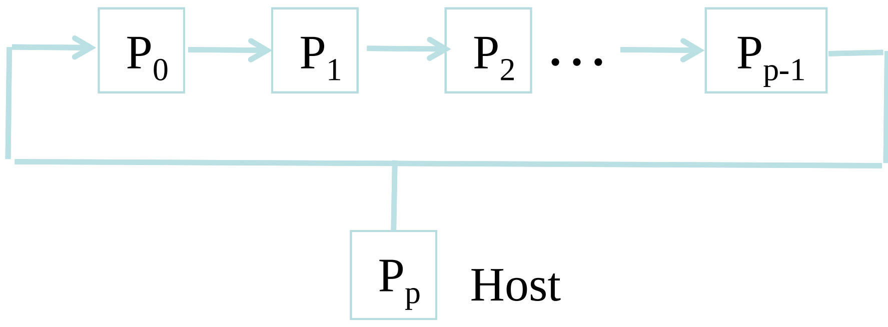
2) Anular los elementos de la columna del mismo índice que están en el Procesador

3) Transferir la fila a Procesador siguiente

Funcionamiento del algoritmo de Givens orientado por filas



Algoritmo orientado por filas



EN PARALELO: Para $pr = 0, 1, \ldots, p\text{-}1$

En P_{pr} :

Para $i = 0, 1, 2, \ldots, n\text{-}1$

Si $i\ DIV\ p\ = pr$

Para j=i+1,i+2,...,(pr +1)*(m/p)-1

$[c, s] = Givens(A, i, j, i)$

Para k=j,j+1,...,n-1

$AplicaRotacion(i, j, i, k)$

FinPara

Transfiere fila i a P_{pr+1}

FinPara

Si No

Si $i < pr * (m / p)$

Espera fila i

Para j=pr*(m/p),pr*(m/p)+1,..,(pr +1)*(m/p)-1

$[c, s] = Givens(A, i, j, i)$

Para k=j,j+1,...,n-1

$AplicaRotacion(i, j, i, k)$

FinPara

FinPara

Transfiere fila i a P_{pr+1}

FinSi

FinSi

FinPara

Algoritmo orientado por filas con triangularizaciones parciales

Ideas básicas

Repetir:

- EN PARALELO: Triangularizar la submatriz contenida en cada procesador

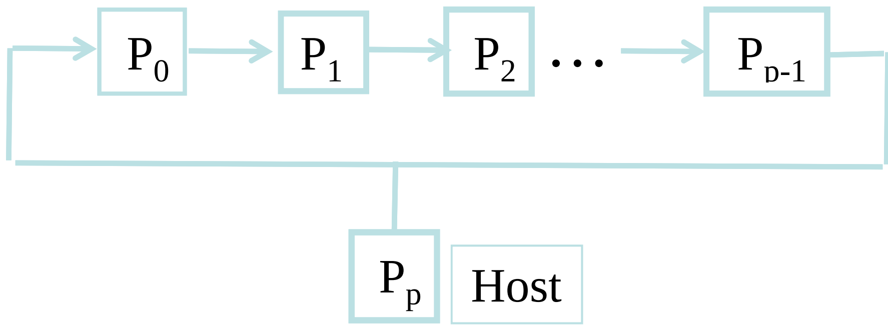
- Aplicando el algoritmo orientado por filas, eliminar los elementos delanteros de cada submatriz hasta que la submatriz sea rectangular.

Hasta que la matriz sea triangular

Funcionamiento del algoritmo de Givens con triangularizaciones parciales, orientado por filas

			Triangularizar			Eliminar		
P_0	fila 0	x x x x x x		fila 0	x x x x x x		fila 0	x x x x x x
	fila 1	x x x x x x		fila 1	x x x x x		fila 1	x x x x x
	fila 2	x x x x x x		fila 2	x x x x	fila 0	fila 2	x x x x
	fila 3	x x x x x x		fila 3	x x x		fila 3	x x x
P_1	fila 4	x x x x x x		fila 4	x x x x x x		fila 4	x x x
	fila 5	x x x x x x		fila 5	x x x x x	fila 0	fila 5	x x x
	fila 6	x x x x x x		fila 6	x x x x		fila 6	x x x
	fila 7	x x x x x x		fila 7	x x x		fila 7	x x x
P_2	fila 8	x x x x x x		fila 8	x x x x x x		fila 8	x x x
	fila 9	x x x x x x		fila 9	x x x x x	fila 0	fila 9	x x x
	fila 10	x x x x x x		fila 10	x x x x		fila 10	x x x
	fila 11	x x x x x x		fila 11	x x x		fila 11	x x x
P_3	fila 12	x x x x x x		fila 12	x x x x x x		fila 12	x x x
	fila 13	x x x x x x		fila 13	x x x x x		fila 13	x x x
	fila 14	x x x x x x		fila 14	x x x x		fila 14	x x x
	fila 15	x x x x x x		fila 15	x x x	fila 0	fila 15	x x x

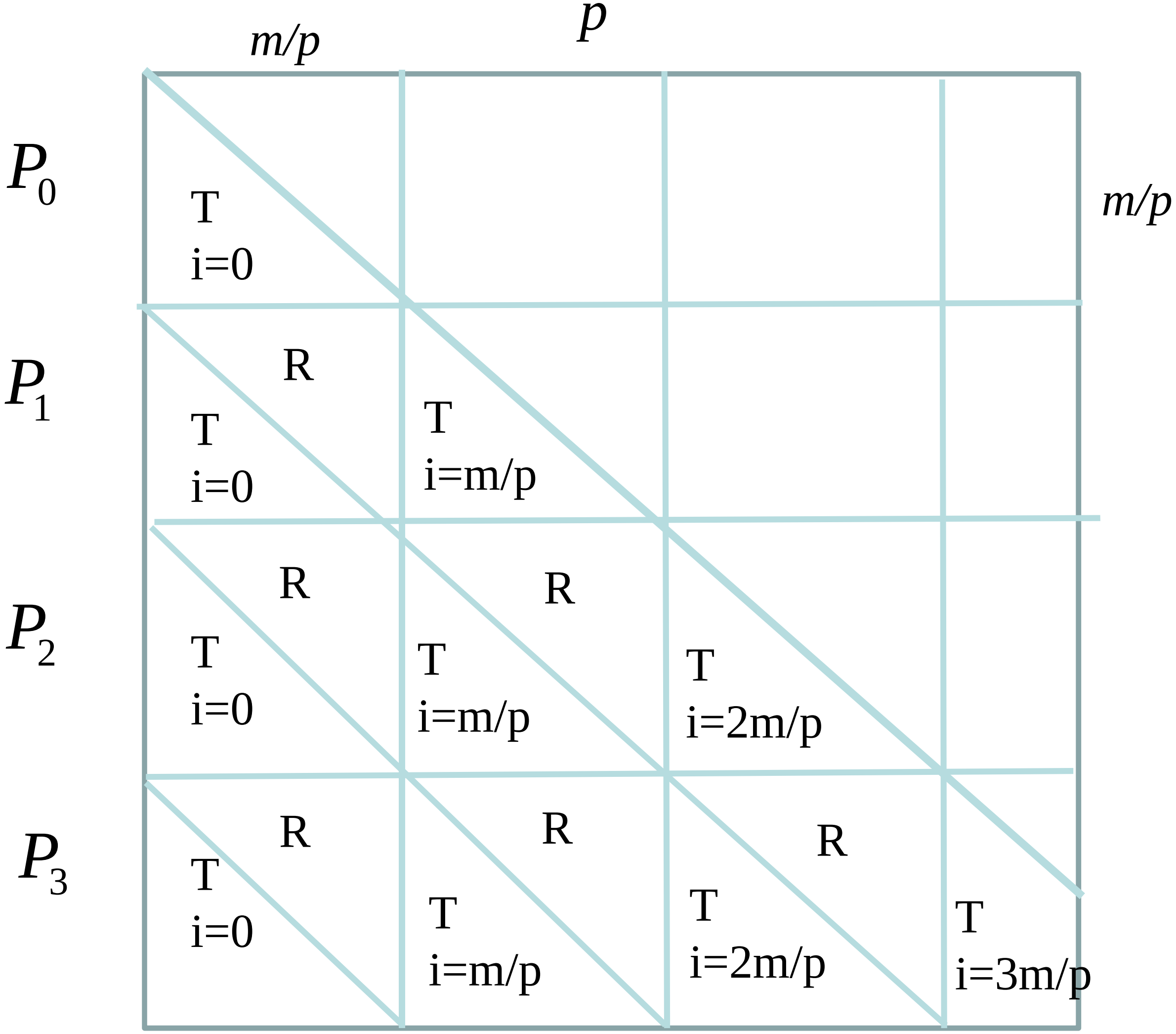
Algoritmo con triangularizaciones parciales orientado por filas



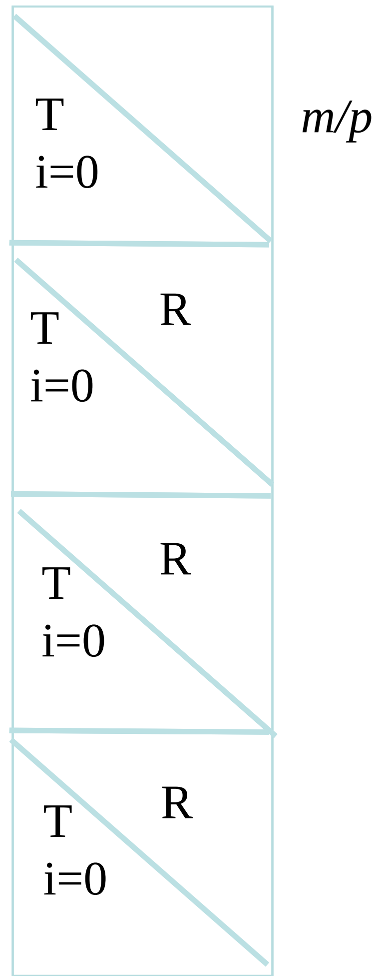

```
EN PARALELO : Para pr = 0,1,..., p - 1
   En P_pr :
      Para i = 0,1,2,...,n-1
         Si iMOD (m/p) =0 and i DIV (m/p) ≤ pr
              Triangulariza A_pr almacenada en P_pr utilizando transformaciones ortogonales
         FinSi
         Si i DIV  p = pr
                  Transfiere  fila i a P_pr+1
         Si No
            Si i < pr * (m / p)
               Espera  fila i
               Para  j=pr*(m/p),pr*(m/p)+1,..,(pr )*(m/p)+i
                  [c,s] = Givens(A,i,j,i)
                  Para k=j,j+1,...,n-1
                     AplicaRotacion(i, j,i,k)
                  FinPara
               FinPara
               Transfiere  fila i a P_pr+1
            FinSi
         FinSi
      FinPara
```

<u>Funcionamiento</u>

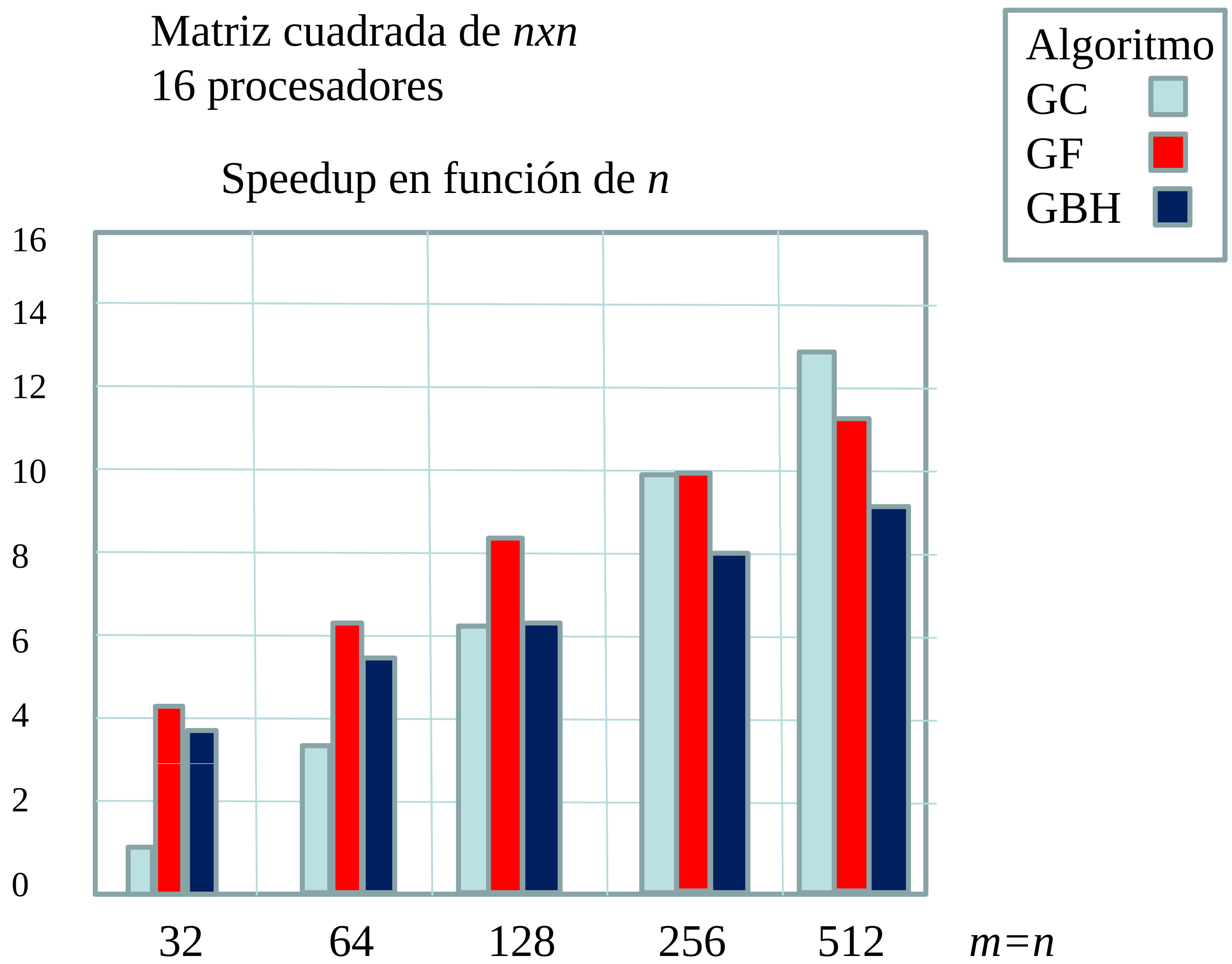
$$\frac{m}{p} < n$$



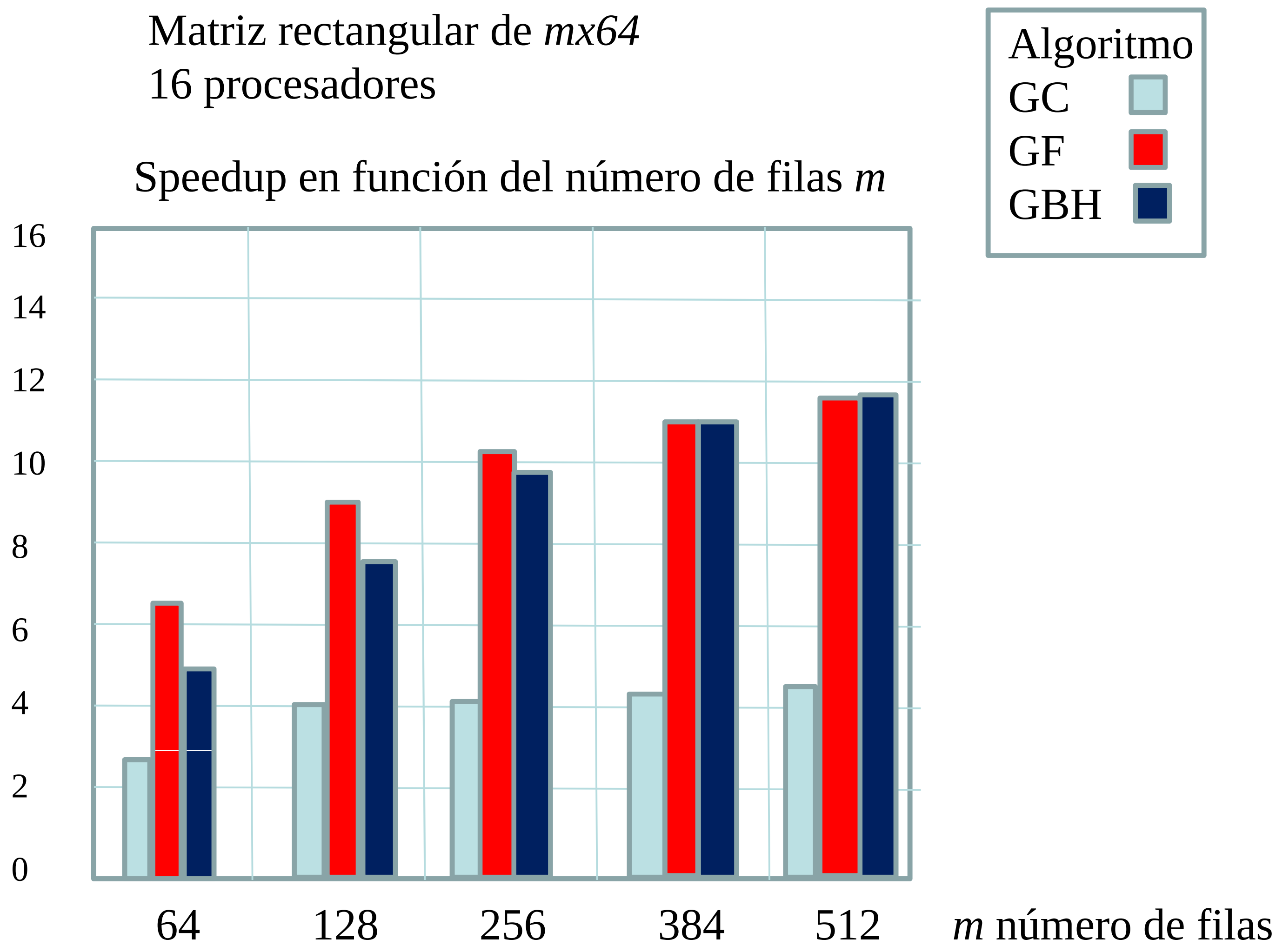
$$\frac{m}{p} \geq n$$



Prestaciones de los algoritmos paralelos basados en Rotaciones de Givens

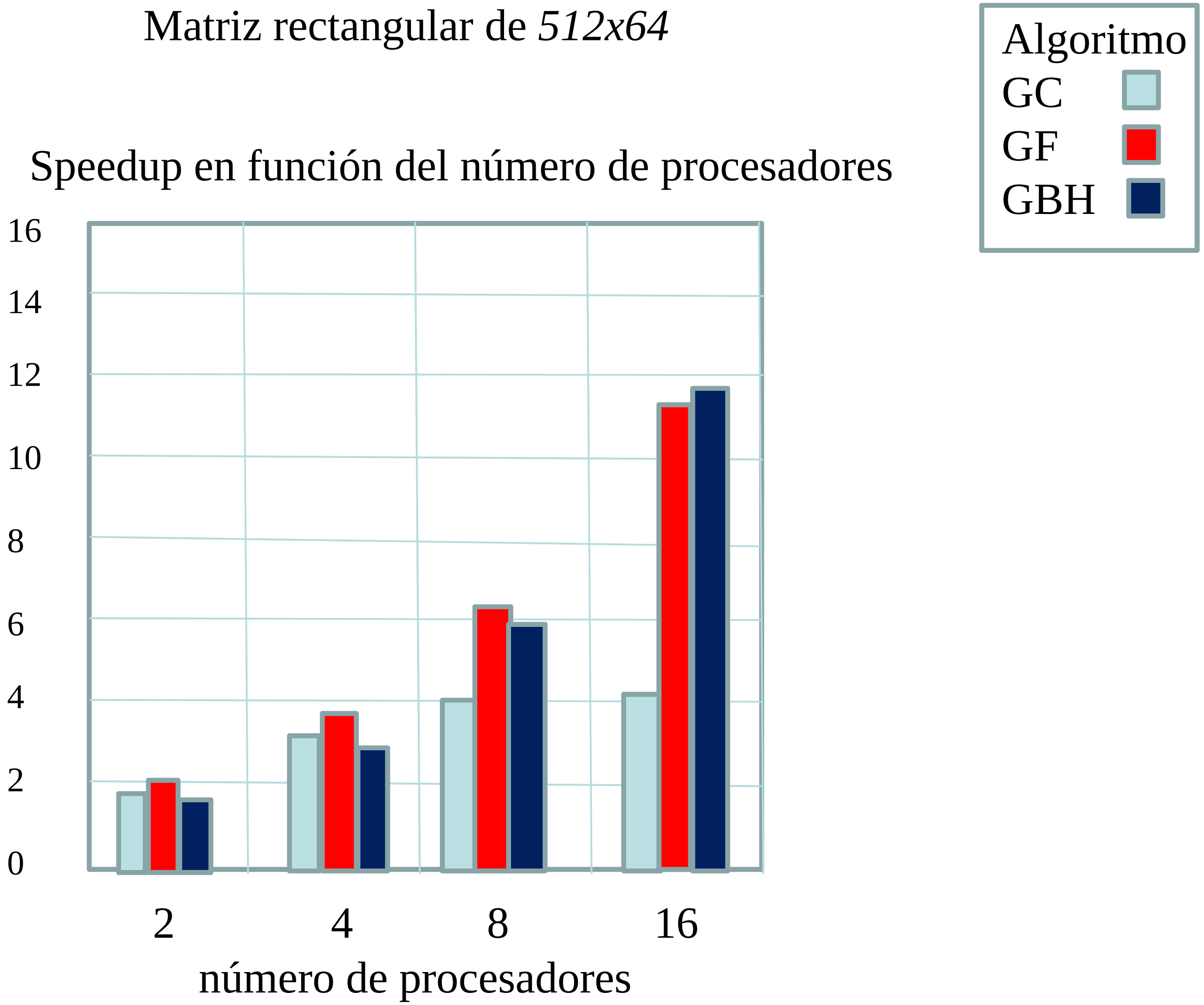


Prestaciones de los algoritmos paralelos basados en Rotaciones de Givens



Prestaciones de los algoritmos paralelos basados en Rotaciones de Givens

Matriz rectangular de *512x64*



Método de Gram-Schmidt

Método de Gram-Schmidt

$$A = QR$$

$$[A_1, A_2, ..., A_j, ..., A_n] = [Q_1, Q_2, ..., Q_j, ..., Q_n] \begin{bmatrix} r_{11} & r_{12} & ... & r_{1j} & ... & r_{1n} \\ 0 & r_{22} & ... & r_{2j} & ... & r_{2n} \\ 0 & 0 & ... & . & ... & . \\ 0 & 0 & ... & 0 & ... & r_{nn} \end{bmatrix}$$

$$Q_i^T Q_j = \delta_{ij}$$

$$A_j = r_{1j}Q_1 + r_{2j}Q_2 + ... + r_{jj}Q_j$$

$$A_j - r_{1j}Q_1 - r_{2j}Q_2 - ... - r_{j-1,j}Q_{j-1} = r_{jj}Q_j$$

$$r_{jj} = \left\| r_{jj}Q_j \right\|_2 ; Q_j = (r_{jj}Q_j) / r_{jj} ;$$

$$Q_i^T A_j = r_{ij}, i = 1,2,...,n-1$$

Algoritmo

$r_{11} = \left\| A_1 \right\|_2 ; Q_1 = A_1 / r_{11} ;$

$Para\ j = 2,3,...,n$

$\quad Q_j = A_j ;$

$\quad Para\ i = 1,2,3,..., j-1$

$\quad\quad r_{ij} = Q_i^T A_j ;$

$\quad\quad Q_j = Q_j - r_{ij} * Q_i ;$

$\quad Finpara$

$\quad r_{jj} = \left\| Q_j \right\|_2 ;$

$\quad Q_j = Q_j / r_{jj} ;$

$Finpara$

Método de Gram-Schmidt Modificado

$$A = QR =$$

$$\left[A_1, A_2, ..., A_j, ..., A_n\right] = \left[Q_1, Q_2, ..., Q_j, ..., Q_n\right] \begin{bmatrix} r_{11} & r_{12} & ... & r_{1j} & ... & r_{1n} \\ 0 & r_{22} & ... & r_{2j} & ... & r_{2n} \\ 0 & 0 & ... & . & ... & . \\ 0 & 0 & ... & 0 & ... & r_{nn} \end{bmatrix}$$

$$A_1 = r_{11}Q_1 \Rightarrow r_{11} = \|A_1\|_2 ; Q_1 = A_1 / r_{11};$$

$$A_2 = r_{12}Q_1 + r_{22}Q_2 \Rightarrow r_{12} = Q_1^T A_2$$

...

$$A_j = r_{1j}Q_1 + r_{2j}Q_2 + ... + r_{jj}Q_j \Rightarrow r_{1j} = Q_1^T A_j$$

...

$$A_n = r_{1n}Q_1 + r_{2n}Q_2 + ... + r_{n-1,n}Q_{j-1} + r_{nn}Q_n \Rightarrow r_{1n} = Q_1^T A_n$$

Método de Gram-Schmidt Modificado

$$A_1 = r_{11}Q_1 \Rightarrow r_{11} = \left\|A_1\right\|_2; Q_1 = A_1 / r_{11};$$

$$A_2 := A_2 - r_{12}Q_1 = r_{22}Q_2 \Rightarrow r_{22} = \left\|A_2\right\|_2; Q_2 = A_2 / r_{22};$$

$$\ldots$$

$$A_j := A_j - r_{1j}Q_1 = r_{2j}Q_2 + \ldots + r_{jj}Q_j$$

$$\ldots$$

$$A_n := A_n - r_{1n}Q_1 = r_{2n}Q_2 + \ldots + r_{n-1,n}Q_{j-1} + r_{nn}Q_n$$

<u>Algoritmo</u>

$Para\ k = 1,2,3,\ldots,n$

$\quad r_{kk} = \left\|A_k\right\|_2; Q_k = A_k / r_{kk};$

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

$\quad\quad r_{kj} = Q_k{}^T A_j;$

$\quad\quad A_j = A_j - r_{kj} * Q_k;$

$\quad Finpara$

$Finpara$

Estabilidad numérica

If one is interested in computing an orthonormal basis for range(A), then the Householder approach requires $2mn^2 - 2n^3/3$ flops to get Q in factored form and another $2mn^2 - 2n^3/3$ flops to get the first n columns of Q. Therefore, for the problem of finding an orthonormal basis for range(A), MGS is about twice as efficient as Householder orthogonalization. However, Björck (1967b) has shown that MGS produces a computed $\hat{Q}_1 = [\,\hat{q}_1, \ldots, \hat{q}_n\,]$ that satisfies

$$\hat{Q}_1^T \hat{Q}_1 \;=\; I \;+\; E_{MGS} \qquad \| E_{MGS} \|_2 \approx \mathbf{u}\kappa_2(A)$$

whereas the corresponding result for the Householder approach is of the form

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

Thus, if orthonormality is critical, then MGS should be used to compute orthonormal bases only when the vectors to be orthogonalized are fairly independent.

Trabajo propuesto

- Implementar en MATLAB un algoritmos para calcular la descomposición QR utilizando el método de Gram-Schmidt modificado y comparar los resultados con los que ofrece la rutina qr de MATLAB
- Implementar en MATLAB un algoritmo para calcular la descomposición QR de matrices Hessenberg superior
- Implementar en MATLAB un algoritmo para calcular la descomposición QR de matrices tridiagonales