

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS

1. Motivación y aspectos de la programación paralela.
2. Tipos de sistemas paralelos. Paradigmas de programación paralela.
3. Conceptos básicos y medidas de paralelismo.
4. Diseño de programas paralelos.
5. La Interface de paso de mensaje: el estándar MPI.
6. Paralelización de algoritmos: ejemplos y aplicaciones.
 - Multiplicaciones matriciales.
 - Multiplicación matriz-matriz por bloques: algoritmo de Cannon.
 - Algoritmo de Floyd-Warshall para la obtención de los caminos más cortos en un grafo.
 - Introducción a los fractales: el conjunto de Mandelbrot.
 - Restauración de imágenes: un caso sencillo.
 - Problema modelo de la ecuación de Laplace.

Multiplicaciones matriciales

(matriz-vector)

- Sea A una matriz de orden $m \times n$ y x un vector de dimensión n . Estableceremos la siguiente notación:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \leftarrow \begin{array}{l} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_m \end{array}$$

$$\begin{array}{cccc} \uparrow & \uparrow & \cdots & \uparrow \\ \mathbf{a}^1 & \mathbf{a}^2 & \cdots & \mathbf{a}^n \end{array}$$

- Es decir, representamos por:
 - a_i el vector fila i de A , $i = 1, 2, \dots, m$
 - a_j el vector columna j de A , $j = 1, 2, \dots, n$

Multiplicaciones matriciales

(matriz-vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 4 \\ 2 & -3 & 5 \\ 1 & 1 & 0 \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$$

El producto $A\boldsymbol{x}$ puede ser visto como 4 productos internos:

$$A\boldsymbol{x} = \begin{bmatrix} (1, 2, 3) \cdot (1, -2, 3) \\ (-1, 0, 4) \cdot (1, -2, 3) \\ (2, -3, 5) \cdot (1, -2, 3) \\ (1, 1, 0) \cdot (1, -2, 3) \end{bmatrix} = \begin{bmatrix} 6 \\ 11 \\ 23 \\ -1 \end{bmatrix} \quad A\boldsymbol{x} = \begin{bmatrix} \sum_{k=1}^n a_{1k}x_k \\ \sum_{k=1}^n a_{2k}x_k \\ \vdots \\ \sum_{k=1}^n a_{mk}x_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1 \cdot \boldsymbol{x} \\ \boldsymbol{a}_2 \cdot \boldsymbol{x} \\ \vdots \\ \boldsymbol{a}_m \cdot \boldsymbol{x} \end{bmatrix}$$

Multiplicaciones matriciales

(matriz-vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 4 \\ 2 & -3 & 5 \\ 1 & 1 & 0 \end{bmatrix} x = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$$

$$Ax = x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} = \sum_{i=1}^n x_i \mathbf{a}^i$$

La segunda forma de realizar el producto Ax consiste en ver la multiplicación matriz-vector como una combinación lineal de las columnas de A , es decir

$$Ax = 1 \cdot \begin{bmatrix} 1 \\ -1 \\ 2 \\ 1 \end{bmatrix} + (-2) \cdot \begin{bmatrix} 2 \\ 0 \\ -3 \\ 1 \end{bmatrix} + 3 \cdot \begin{bmatrix} 3 \\ 4 \\ 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 6 \\ 11 \\ 23 \\ -1 \end{bmatrix}$$

Multiplicaciones matriciales

(matriz-vector)

Tenemos dos formas de ver el producto \mathbf{Ax} :

1. Producto interno de dos vectores:

$$\mathbf{Ax} = \begin{bmatrix} \sum_{k=1}^n a_{1k}x_k \\ \sum_{k=1}^n a_{2k}x_k \\ \vdots \\ \sum_{k=1}^n a_{mk}x_k \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{x} \\ \mathbf{a}_2 \cdot \mathbf{x} \\ \vdots \\ \mathbf{a}_m \cdot \mathbf{x} \end{bmatrix} \quad \mathbf{a}_i \cdot \mathbf{x} = \sum_{k=1}^n a_{ik}x_k$$

2. Combinaciones lineales:

$$\mathbf{Ax} = x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} = \sum_{i=1}^n x_i \mathbf{a}^i$$

Multiplicaciones matriciales (matriz-vector)

Tenemos dos formas de ver el producto \mathbf{Ax} :

1. Producto interno de dos vectores:

Algoritmo ij : (basado en el producto interno)

For $i = 1$ to m

 For $j = 1$ to n

$$y_i = y_i + a_{ij}x_j$$

2. Combinaciones lineales:

Algoritmo ji : (basado en comb. lineales)

For $j = 1$ to n

 For $i = 1$ to m

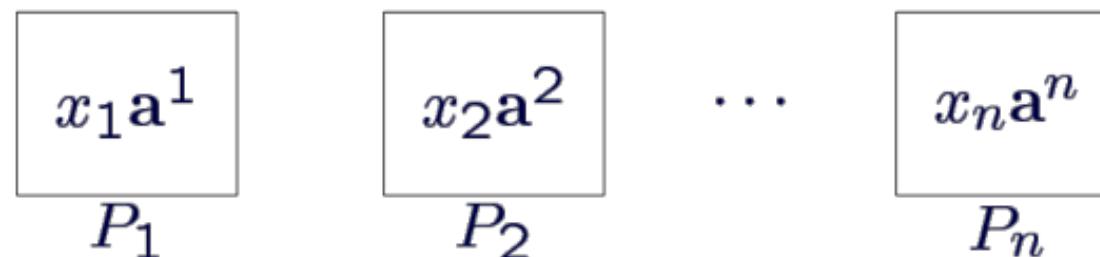
$$y_i = y_i + a_{ij}x_j$$

Multiplicaciones matriciales

(matriz-vector en paralelo:

Algoritmo basado en comb. lineales)

- Supongamos $p = n$. Consideraremos el siguiente almacenamiento de los datos en los proc.:

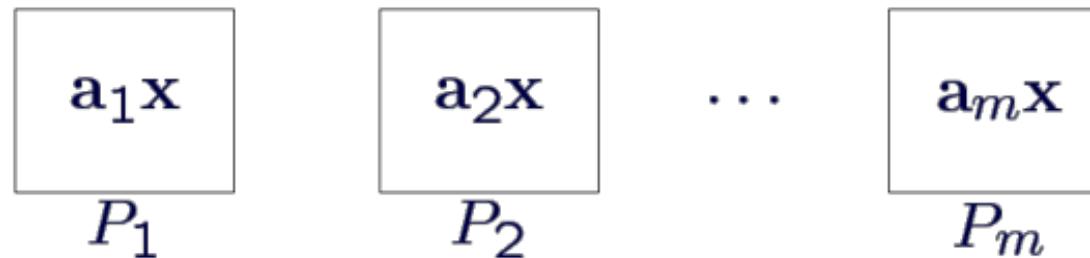


- Todos los productos $x_i a^i$ son realizados con un perfecto grado de paralelismo.
- Sobre memoria distribuida la figura indica que los datos están en las memorias locales.
- Sobre memoria compartida la figura debe interpretarse como asignación de tareas.
- En memoria distribuida, la suma de los resultados de cada procesador se puede realizar con el algoritmo de fan-in aplicado sobre vectores.

Multiplicaciones matriciales

(matriz-vector en paralelo:
Algoritmo basado en producto interno)

- Supongamos $p = m$. Consideremos el siguiente almacenamiento de los datos en los proc.:



- Cada procesador realiza un producto interno con un perfecto grado de paralelismo.
- No hay que aplicar fan-in, ni transferencia de datos.
- La sincronización solo es necesaria al final de los cálculos.

Multiplicaciones matriciales

(matriz-vector en paralelo:

Algoritmo basado en producto interno)

- Pero normalmente n y/o m serán considerablemente mayores que p .
- **Solución:** Asignar a cada procesador algunas filas o columnas.
 - En un caso ideal, para el algoritmo basado en el producto interno, p debe dividir a m , y m/p filas serán asignadas a cada procesador. Matemáticamente,

$$Ax = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix} x = \begin{bmatrix} A_1x \\ A_2x \\ \vdots \\ A_px \end{bmatrix}$$

- donde A_i contiene m/p filas de A . Si A_i y x son asignados al procesador i , entonces los productos A_1x, A_2x, \dots, A_px pueden ser realizados con un perfecto grado de paralelismo.
- Si p no divide a m , entonces se intenta distribuir las filas entre los procesadores tanto más proporcionalmente como sea posible.

Multiplicaciones matriciales

(matriz-vector en paralelo:

Algoritmo basado en producto interno)

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & 2 & 1 \\ \hline -2 & 2 & 0 & 4 \\ \hline 3 & 4 & -1 & 1 \\ \hline 1 & 1 & 2 & 1 \\ -2 & 1 & 1 & 2 \\ 2 & 1 & -1 & 1 \end{bmatrix} \text{ y } \mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$$

Multiplicaciones matriciales

(matriz-vector en paralelo:

Algoritmo basado en producto interno)

P1	P2	P3
$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -2 & 2 & 0 & 4 \\ 3 & 4 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 2 & 1 \\ -2 & 1 & 1 & 2 \\ 2 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$
$\begin{bmatrix} 6 \\ -3 \end{bmatrix}$	$\begin{bmatrix} 6 \\ 13 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 1 \\ 6 \end{bmatrix}$
	$\begin{bmatrix} 6 \\ -3 \\ 6 \\ 13 \\ 2 \\ 1 \\ 6 \end{bmatrix}$	$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & 2 & 1 \\ -2 & 2 & 0 & 4 \\ 3 & 4 & -1 & 1 \\ 1 & 1 & 2 & 1 \\ -2 & 1 & 1 & 2 \\ 2 & 1 & -1 & 1 \end{bmatrix} \text{ y } x = \begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$
		236

Multiplicaciones matriciales

(matriz-vector en paralelo:

Algoritmo basado en comb. lineales)

- Pero normalmente n y/o m serán considerablemente mayores que p .
- **Solución:** Asignar a cada procesador algunas columnas.
 - En un caso ideal, para el algoritmo basado en combinaciones lineales, p debe dividir a n , y n/p columnas serán asignadas a cada procesador. Matemáticamente,

$$Ax = \begin{bmatrix} A_1 & A_2 & \dots & A_p \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = A_1x_1 + A_2x_2 + \dots + A_px_p$$

- donde A_i contiene n/p filas de A . Si A_i y x_i son asignados al procesador i , entonces los productos $A_1x_1, A_2x_2, \dots, A_px_p$ pueden ser realizados con un perfecto grado de paralelismo.
- Si p no divide a n , entonces se intenta distribuir las columnas entre los procesadores tanto más proporcionalmente como sea posible.

Multiplicaciones matriciales

(matriz-vector en paralelo:

Algoritmo basado en comb. lineales)

$$A = \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 0 & -1 & 2 & 1 \\ -2 & 2 & 0 & 4 \\ 3 & 4 & -1 & 1 \\ 1 & 1 & 2 & 1 \\ -2 & 1 & 1 & 2 \\ 2 & 1 & -1 & 1 \end{array} \right] \text{ y } \mathbf{x} = \left[\begin{array}{c} 1 \\ 2 \\ -1 \\ 1 \end{array} \right]$$

Multiplicaciones matriciales

(matriz-vector en paralelo:

Algoritmo basado en comb. lineales)

$P_1 \left[\begin{array}{cc} 1 & 2 \\ 0 & -1 \\ -2 & 2 \\ 3 & 4 \\ 1 & 1 \\ -2 & 1 \\ 2 & 1 \end{array} \right] \left[\begin{array}{c} 1 \\ 2 \end{array} \right]$	$\left[\begin{array}{c} 5 \\ -2 \\ 2 \\ 11 \\ 3 \\ 0 \\ 4 \end{array} \right]$	
$P_2 \left[\begin{array}{cc} 3 & 4 \\ 2 & 1 \\ 0 & 4 \\ -1 & 1 \\ 2 & 1 \\ 1 & 2 \\ -1 & 1 \end{array} \right] \left[\begin{array}{c} -1 \\ 1 \end{array} \right]$	$\left[\begin{array}{c} 1 \\ -1 \\ 4 \\ 2 \\ -1 \\ 1 \\ 2 \end{array} \right]$	

$$\left[\begin{array}{c} 5 \\ -2 \\ 2 \\ 11 \\ 3 \\ 0 \\ 4 \end{array} \right] + \left[\begin{array}{c} 1 \\ -1 \\ 4 \\ 2 \\ -1 \\ 1 \\ 2 \end{array} \right] = \left[\begin{array}{c} 6 \\ -3 \\ 6 \\ 13 \\ 2 \\ 1 \\ 6 \end{array} \right]$$

$$A = \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 0 & -1 & 2 & 1 \\ -2 & 2 & 0 & 4 \\ 3 & 4 & -1 & 1 \\ 1 & 1 & 2 & 1 \\ -2 & 1 & 1 & 2 \\ 2 & 1 & -1 & 1 \end{array} \right] \quad \text{y } x = \left[\begin{array}{c} 1 \\ 2 \\ -1 \\ 1 \end{array} \right]$$

239

Multiplicaciones matriciales

(matriz-matriz)

- Sean \mathbf{A} y \mathbf{B} dos matrices de orden $m \times n$ y $n \times q$ respectivamente. Seguiremos denotando por:

	A	B
fila	\mathbf{a}_i	\mathbf{b}_i
columna	\mathbf{a}^i	\mathbf{b}^i

- El producto \mathbf{AB} puede ser visto como q multiplicaciones matriz-vector:
 - $\mathbf{Ab}^i, \quad i = 1, 2, \dots, q$
- Es decir,
 - $\mathbf{AB} = [\mathbf{Ab}^1 \ \mathbf{Ab}^2 \ \dots \ \mathbf{Ab}^q]$
- Podremos utilizar cualquiera de los dos algoritmos anteriores prestando atención al almacenamiento de los elementos.

Multiplicaciones matriciales

(matriz-matriz en paralelo:
Algoritmo basado en comb. lineales)

- Supongamos $p = n$. El almacenamiento será el siguiente:

	P_1	P_2	...	P_n
Para calcular Ab^1	$b_{11}a^1$	$b_{21}a^2$...	$b_{n1}a^n$
Para calcular Ab^2	$b_{12}a^1$	$b_{22}a^2$...	$b_{n2}a^n$
...
Para calcular Ab^q	$b_{1q}a^1$	$b_{2q}a^2$...	$b_{nq}a^n$
Almacenamiento	b_1, a^1	b_2, a^2	...	b_n, a^n

- Cada procesador obtendrá una matriz de orden $m \times q$.
- Todas ellas deben ser sumadas para obtener el resultado final, por ejemplo con el algoritmo de fan-in aplicado a matrices.

Multiplicaciones matriciales

(matriz-matriz en paralelo:
Algoritmo basado en prod. interno)

- Supongamos $p = m$. El almacenamiento será el siguiente:

	P_1	P_2	...	P_m
Para calcular Ab^1	$a_1 b^1$	$a_2 b^1$...	$a_m b^1$
Para calcular Ab^2	$a_1 b^2$	$a_2 b^2$...	$a_m b^2$
...
Para calcular Ab^q	$a_1 b^q$	$a_2 b^q$...	$a_m b^q$
Almacenamiento	a_1, B	a_2, B	...	a_m, B

- En este caso, el resultado está almacenado de manera que cada procesador posee una de las m filas de la matriz producto AB .
- Todo esto también es valido si disponemos de menos procesadores que el numero de filas y columnas.

Multiplicaciones matriciales

- Consideraremos ahora algoritmos paralelos basados en particiones de las matrices \mathbf{A} y \mathbf{B} . Supongamos entonces que \mathbf{A} y \mathbf{B} están particionadas de la forma:

$$C = AB = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1r} \\ A_{21} & A_{22} & \cdots & A_{2r} \\ \vdots & \vdots & & \vdots \\ A_{s1} & A_{s2} & \cdots & A_{sr} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1t} \\ B_{21} & B_{22} & \cdots & B_{2t} \\ \vdots & \vdots & & \vdots \\ B_{r1} & B_{r2} & \cdots & B_{rt} \end{bmatrix} = \left(\sum_{k=1}^r A_{ik} B_{kj} \right)$$

- Por ejemplo, si $p = st$, el numero de bloques en \mathbf{C} , entonces estos st bloques pueden ser calculados en paralelo teniendo en cuenta una buena asignación de los bloques de \mathbf{A} y \mathbf{B} a los procesadores.
- Casos especiales de este son cuando $s = 1$, es decir cuando \mathbf{A} esta particionada en grupos de columnas, y cuando $t = 1$, es decir, cuando \mathbf{B} está particionada en grupos de filas.

Multiplicaciones matriciales

- Otro caso sería cuando $r = 1$, es decir, cuando A está particionada en s bloques formados por grupos de filas y B está particionada en t bloques formados por grupos de columnas.

$$C = AB = \begin{bmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{s1} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1t} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} & \cdots & A_{11}B_{1t} \\ A_{21}B_{11} & A_{21}B_{12} & \cdots & A_{21}B_{1t} \\ \vdots & \vdots & & \vdots \\ A_{s1}B_{11} & A_{s1}B_{12} & \cdots & A_{s1}B_{1t} \end{bmatrix}$$

- Si usamos $p = st$ procesadores con los bloques de A y B almacenados tal y como indica la propia estructura de los bloques de la matriz resultado C , tenemos que cada uno de estos bloques resultado pueden ser calculados en paralelo en un procesador.

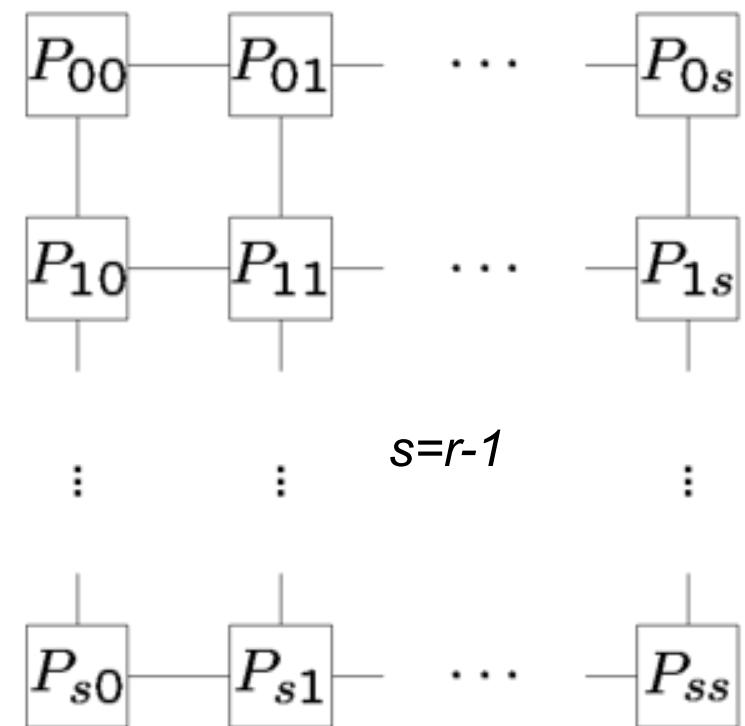
Multiplicación matriz-matriz por bloques: algoritmo de Cannon

Consideremos las matrices A y B particionadas en $r \times r$ bloques de idéntico tamaño.

Supongamos que disponemos de una **malla abierta** de $r \times r$ procesadores:

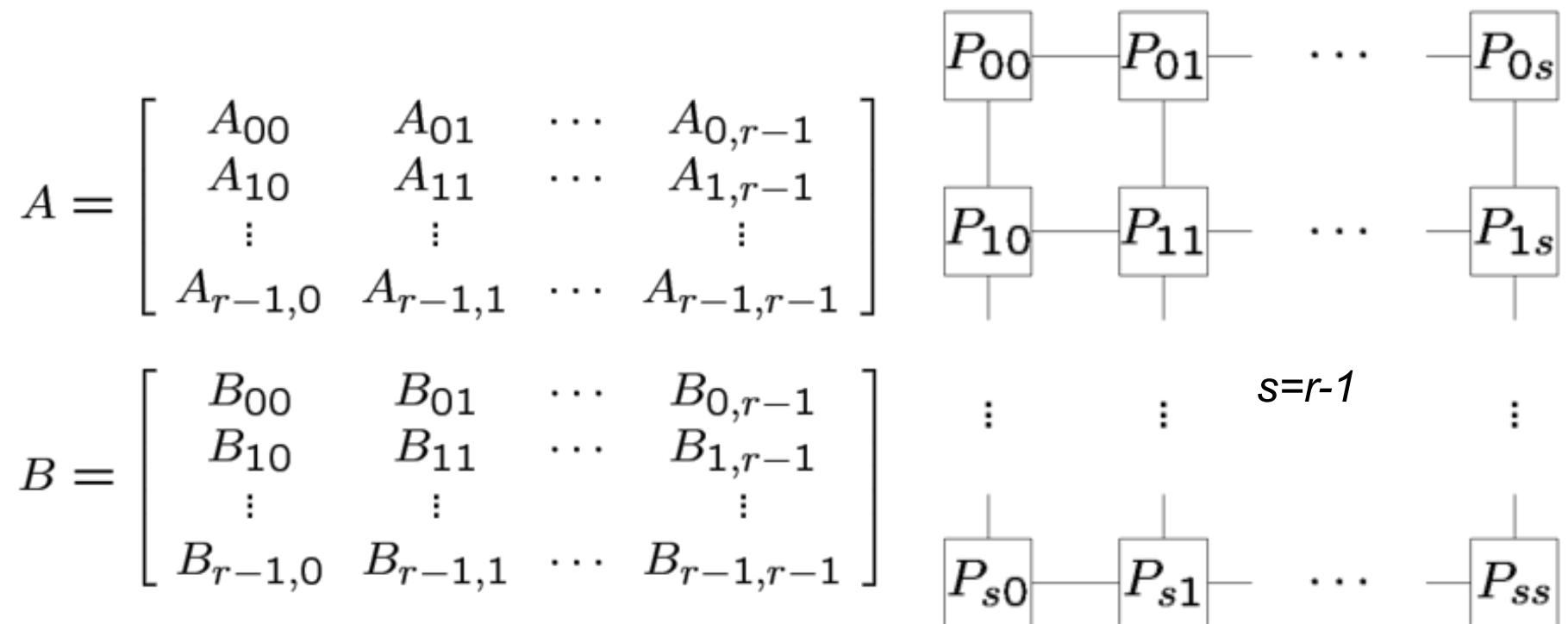
$$A = \begin{bmatrix} A_{00} & A_{01} & \cdots & A_{0,r-1} \\ A_{10} & A_{11} & \cdots & A_{1,r-1} \\ \vdots & \vdots & & \vdots \\ A_{r-1,0} & A_{r-1,1} & \cdots & A_{r-1,r-1} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{00} & B_{01} & \cdots & B_{0,r-1} \\ B_{10} & B_{11} & \cdots & B_{1,r-1} \\ \vdots & \vdots & & \vdots \\ B_{r-1,0} & B_{r-1,1} & \cdots & B_{r-1,r-1} \end{bmatrix}$$



Multiplicación matriz-matriz por bloques: algoritmo de Cannon

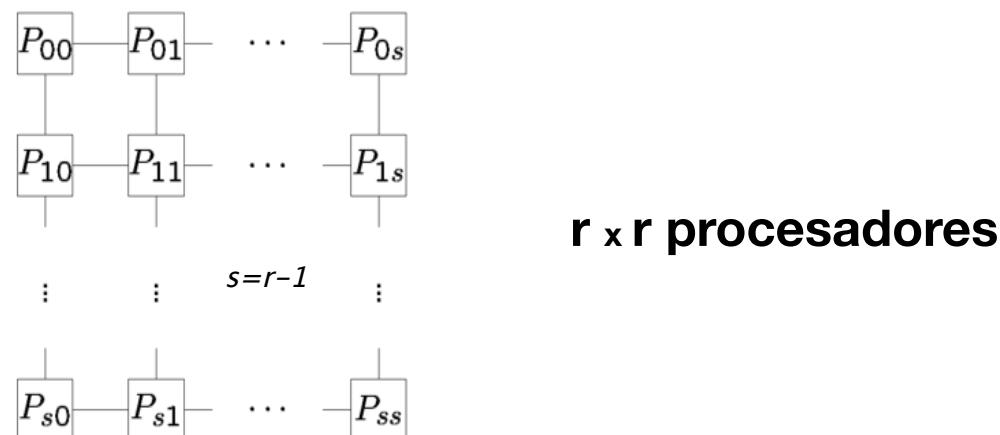
Este algoritmo es especialmente apropiado para sistemas de paso de mensajes, y la arquitectura de paso de mensajes que más se ajusta a las matrices es una malla. Realmente, aunque la arquitectura física no sea una malla. Lógicamente cualquier arquitectura puede representarse como una malla.



Multiplicación matriz-matriz por bloques: algoritmo de Cannon

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix} =$$

$$\begin{bmatrix} A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20} & A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21} & A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22} \\ A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20} & A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21} & A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22} \\ A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20} & A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} & A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$



Multiplicación matriz-matriz por bloques: algoritmo de Cannon

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix} =$$

$$\begin{bmatrix} A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20} & A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21} & A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22} \\ A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20} & A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21} & A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22} \\ A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20} & A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} & A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

$$C_{i,j} = \sum_{k=0}^{r-1} A_{i,k} B_{k,j} = A_{i,0} B_{0,j} + A_{i,1} B_{1,j} + \cdots + A_{i,i-1} B_{i-1,j} + A_{i,i} B_{i,j} + A_{i,i+1} B_{i+1,j} + \cdots + A_{i,r-1} B_{r-1,j}$$

**Fila i de bloques A
por columna j de bloques B**

Multiplicación matriz-matriz por bloques: algoritmo de Cannon

$$C_{i,j} = \sum_{k=0}^{r-1} A_{i,k} B_{k,j} = A_{i,0} B_{0,j} + A_{i,1} B_{1,j} + \cdots + A_{i,i-1} B_{i-1,j} + A_{i,i} B_{i,j} + A_{i,i+1} B_{i+1,j} + \cdots + A_{i,r-1} B_{r-1,j}$$

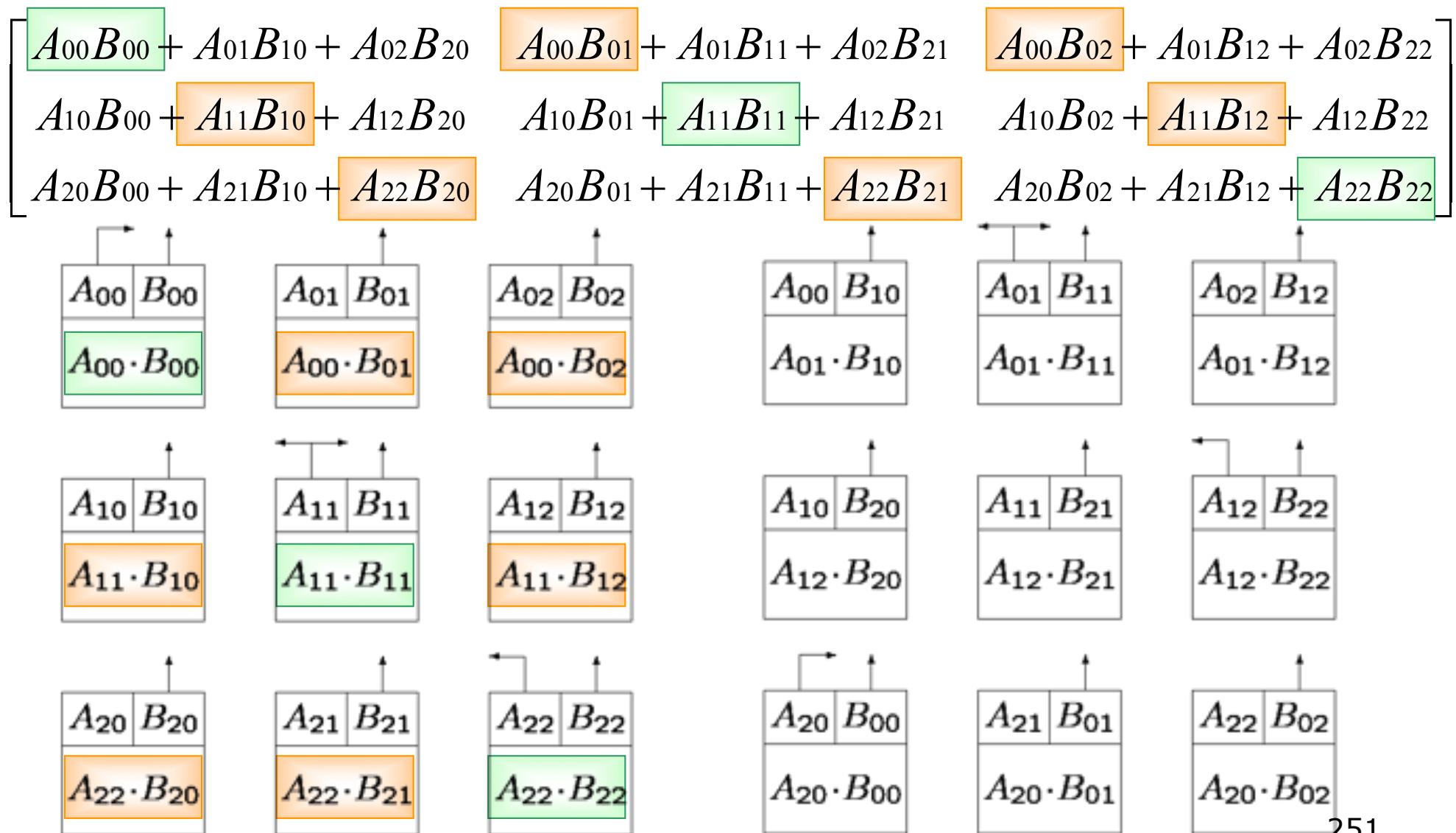
Etapa	Calcular	Fila i de bloques A por columna j de bloques B
0	$C_{i,j} = A_{i,j} B_{i,j}$	
1	$C_{i,j} = C_{i,j} + A_{i,i+1} B_{i+1,j}$	
...	...	
	$C_{i,j} = C_{i,j} + A_{i,r-1} B_{r-1,j}$	
	$C_{i,j} = C_{i,j} + A_{i,0} B_{0,j}$	
	$C_{i,j} = C_{i,j} + A_{i,1} B_{1,j}$	
...	...	
r-1	$C_{i,j} = C_{i,j} + A_{i,i-1} B_{i-1,j}$	

Multiplicación matriz-matriz por bloques: algoritmo de Cannon

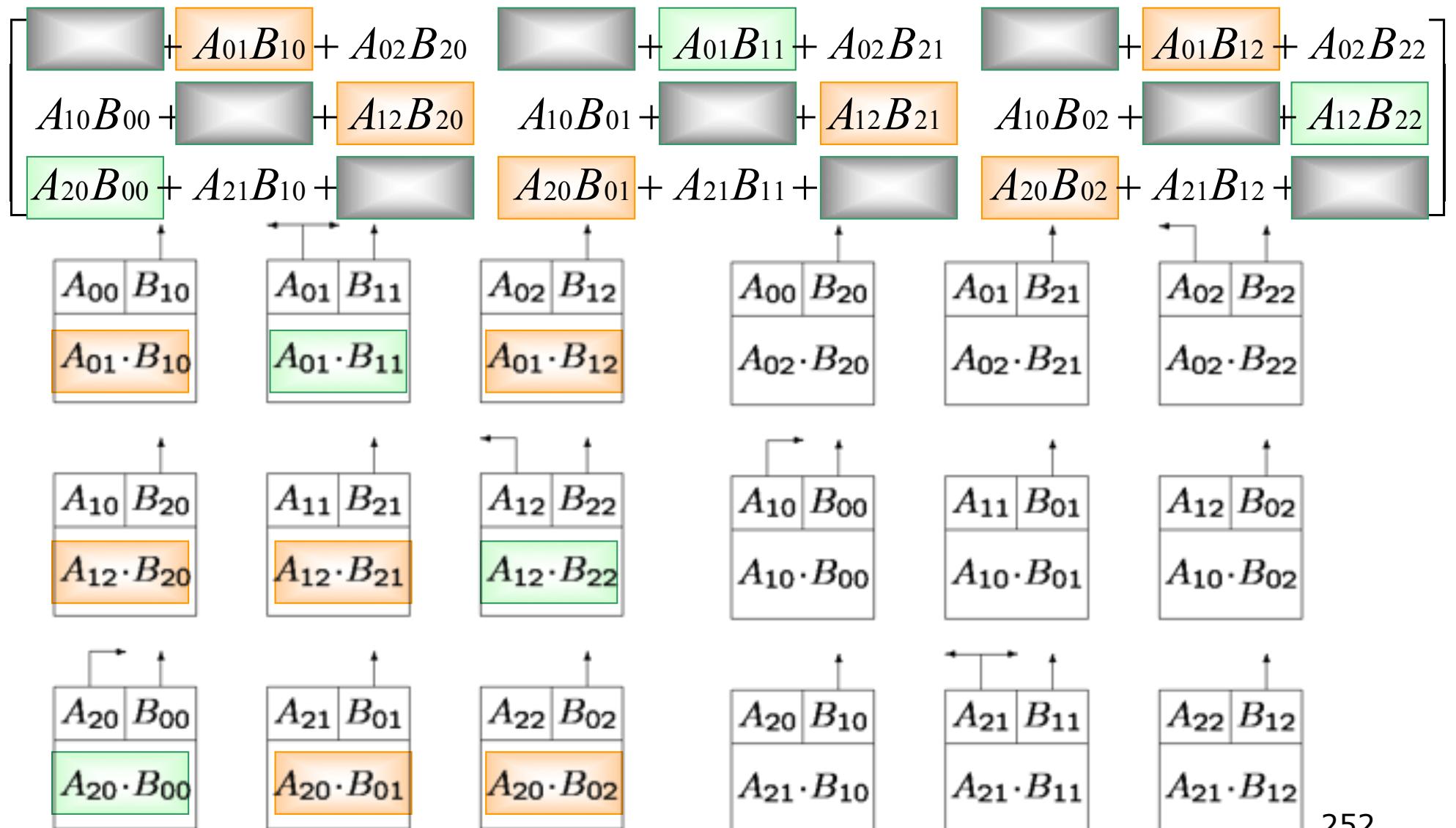
Etapa	Calcular
0	$C_{i,j} = A_{i,i}B_{i,j}$
1	$C_{i,j} = C_{i,j} + A_{i,i+1}B_{i+1,j}$
...	...
	$C_{i,j} = C_{i,j} + A_{i,q-1}B_{q-1,j}$
	$C_{i,j} = C_{i,j} + A_{i,0}B_{0,j}$
	$C_{i,j} = C_{i,j} + A_{i,1}B_{1,j}$
...	...
r-1	$C_{i,j} = C_{i,j} + A_{i,i-1}B_{i-1,j}$

Etapa	Cada proceso calcula
0	<ul style="list-style-type: none"> ○ proceso (i,j) tiene $A_{i,j}, B_{i,j}$ pero necesita $A_{i,i}$ ○ proceso (i,i) broadcasts $A_{i,i}$ a la fila de procesadores i ○ proceso (i,j) calcula $C_{i,j} = A_{i,i}B_{i,j}$
1	<ul style="list-style-type: none"> ○ proceso (i,j) tiene $A_{i,j}, B_{i,j}$, pero necesita $A_{i,i+1}, B_{i+1,j}$ <ul style="list-style-type: none"> ○ shift el bloque columna j of B por el bloque de arriba (bloque 0 va al bloque $r - 1$) ○ proceso $(i, i + 1)$ broadcasts $A_{i,i+1}$ a la fila de procesadores i ○ proceso (i,j) calcula $C_{i,j} = C_{i,j} + A_{i,i+1}B_{i+1,j}$
	Similar en las etapas siguientes

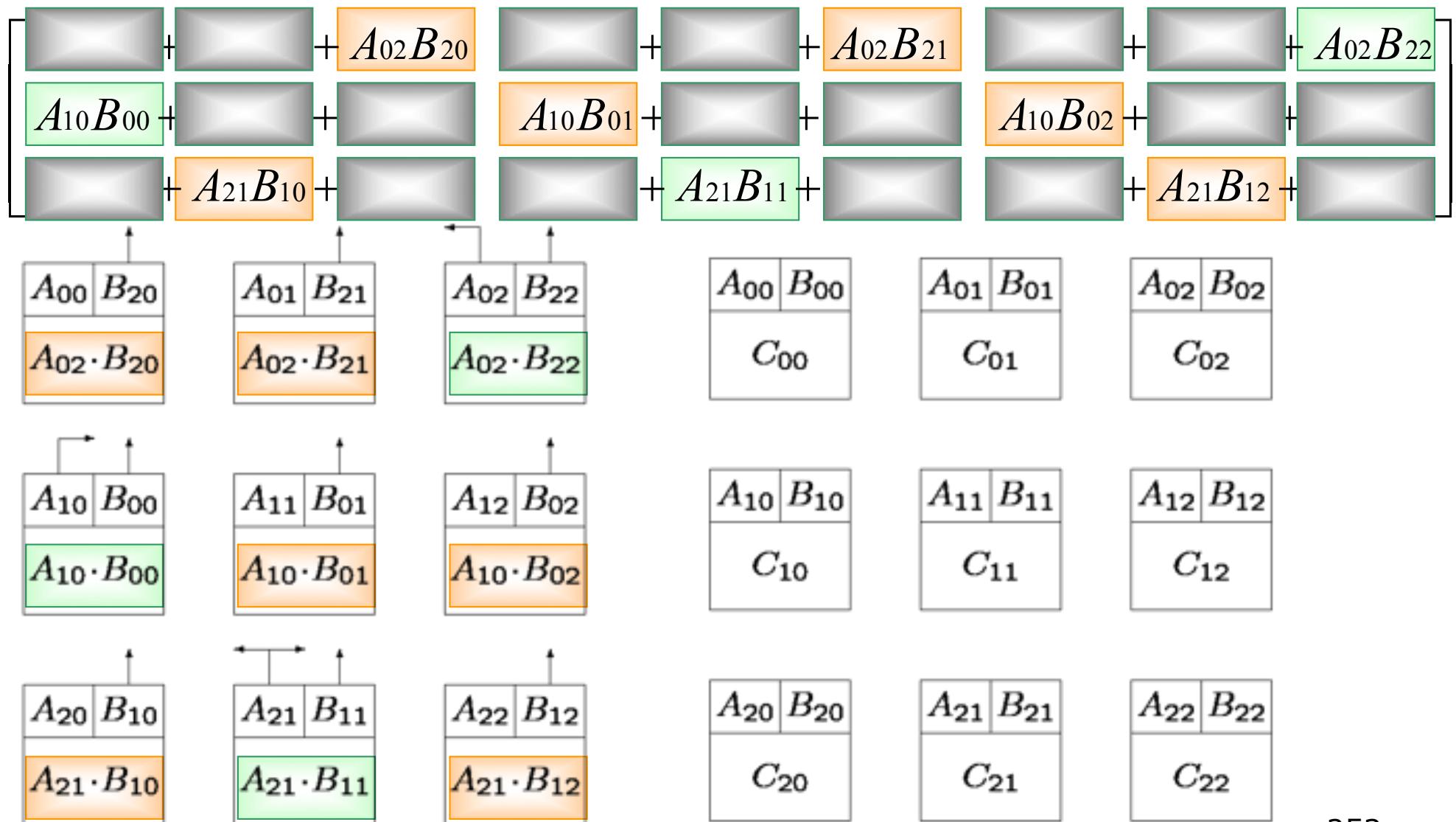
Multiplicación matriz-matriz por bloques: algoritmo de Cannon



Multiplicación matriz-matriz por bloques: algoritmo de Cannon



Multiplicación matriz-matriz por bloques: algoritmo de Cannon



Multiplicación matriz-matriz por bloques: algoritmo de Cannon

ALGORITMO

Inicialmente P_{ij} contiene A_{ij} y B_{ij} . Cada procesador calculará el bloque correspondiente C_{ij} del producto $C = AB$. Internamente estos bloques tendrán el mismo nombre, es decir

$$\mathcal{A} \leftarrow A_{ij} \quad \mathcal{B} \leftarrow B_{ij} \quad \mathcal{C} \leftarrow C_{ij}$$

Cada procesador P_{ij} estará identificado por su *fila* ($= i$) y su *columna* ($= j$).

Multiplicación matriz-matriz por bloques: algoritmo de Cannon

Para $k = 0, 1, \dots, r - 1$

- Si columna=mod(fila+k,r):

Mandar \mathcal{A} a todos los procesadores de mi fila.

$$\mathcal{C} = \mathcal{C} + \mathcal{A} * \mathcal{B}$$

Si no:

Recibir \mathcal{A} del procesador que envía en mi fila y almacenarlo en $ATMP$.

$$\mathcal{C} = \mathcal{C} + ATMP * \mathcal{B}$$

- Hacer una rotación de los bloques columna de B , es decir, P_{ij} manda su bloque \mathcal{B} a $P_{i-1,j}$. El proceso P_{0j} manda su bloque \mathcal{B} a $P_{r-1,j}$.

Después de estas r iteraciones, C contiene el producto AB distribuido entre los procesadores y los bloques de B han sufrido una rotación entre las columnas, volviendo a estar almacenados como al inicio del algoritmo.

Multiplicación matriz-matriz por bloques: algoritmo de Cannon

r=3

	fila	0	1	2
k=0	mod(fila+0,3)	0	1	2
	(fila,columna)	(0,0)	(1,1)	(2,2)
	fila	0	1	2
k=1	mod(fila+1,3)	1	2	0
	(fila,columna)	(0,1)	(1,2)	(2,0)
	Fila	0	1	2
k=2	mod(fila+2,3)	2	0	1
	(fila,columna)	(0,2)	(1,0)	(2,1)

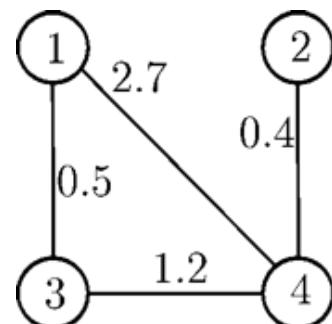
Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

- Sea $G = (V, A)$ un grafo ponderado finito tal que $V = \{v_1, v_2, \dots, v_n\}$. El grafo G puede ser representado mediante su **matriz de peso** definida de la siguiente forma:

$$\Omega = [a_{ij}] / a_{ij} = \begin{cases} \omega_{ij} & \text{si } \{v_i, v_j\} \in A \text{ (si } (v_i, v_j) \in A, \text{ en el caso dirigido)} \\ \infty & \text{si } \{v_i, v_j\} \notin A \text{ (si } (v_i, v_j) \notin A, \text{ en el caso dirigido)} \end{cases}$$

donde ω_{ij} representa el peso de la arista o arco que une los vértices v_i y v_j .

- EJEMPLO:**



$$\begin{bmatrix} \infty & \infty & 0.5 & 2.7 \\ \infty & \infty & \infty & 0.4 \\ 0.5 & \infty & \infty & 1.2 \\ 2.7 & 0.4 & 1.2 & \infty \end{bmatrix}$$

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

- Llamaremos u_{ij} al peso del camino más corto de i a j .
- Utilizaremos las variables:
- $u_{ij}^{(m)}$: peso del camino más corto del vértice i al j con la restricción que no contenga los vértices $m, m + 1, \dots, n$ (exceptuando los extremos i y j en su caso).
- Estas variables pueden calcularse recursivamente utilizando las ecuaciones:

$$\begin{aligned} u_{ij}^{(1)} &= w_{ij} \quad \forall i, j \\ u_{ij}^{(m+1)} &= \min \left\{ u_{ij}^{(m)}, u_{im}^{(m)} + u_{mj}^{(m)} \right\} \quad \forall i, j, \\ &\quad m = 1, 2, \dots, n \end{aligned}$$

- Y es posible ver que $u_{ij} = u_{ij}^{(n+1)}$, con lo que tendremos los pesos de los caminos más cortos entre todos los pares de vértices.

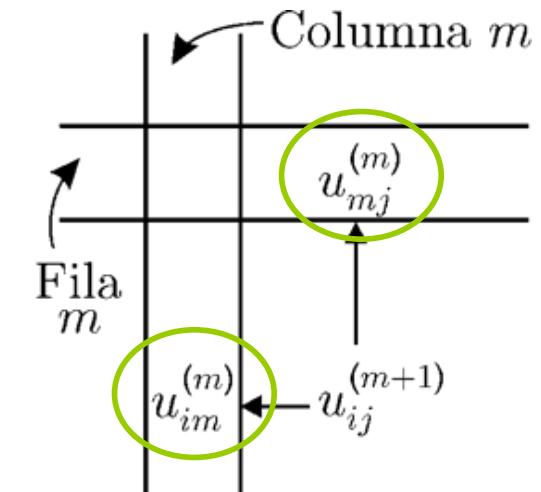
Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

$$\begin{aligned}
 u_{ij}^{(1)} &= w_{ij} \quad \forall i, j \\
 u_{ij}^{(m+1)} &= \min_{m=1,2,\dots,n} \left\{ u_{ij}^{(m)}, u_{im}^{(m)} + u_{mj}^{(m)} \right\} \quad \forall i, j,
 \end{aligned}$$

Para actualizar un elemento que ocupe la fila i y columna j en la iteración $m+1$, hemos de calcular:

El mínimo entre el mismo elemento de la iteración anterior m y la suma de dos elementos:

- el que ocupa la misma fila i y la columna de la iteración m ,
- el que ocupa la misma columna j y la fila de la iteración m .



Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

EJEMPLO:

$W =$

	1	2	3	4	5	6
1	∞	3	∞	2	∞	∞
2	∞	∞	10	6	∞	∞
3	∞	1	∞	9	∞	3
4	1	8	∞	∞	4	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	1	∞	4	∞

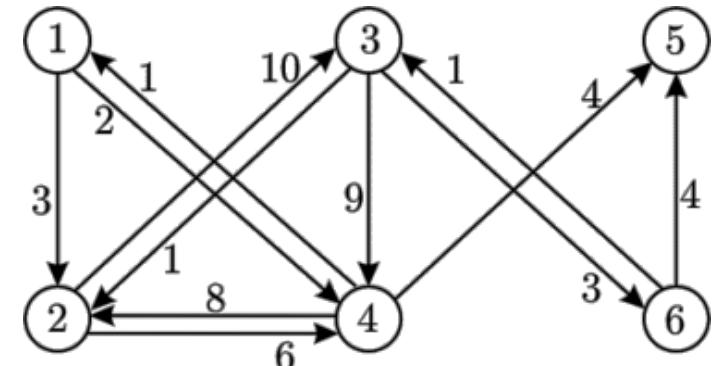
(m=2)

	1	2	3	4	5	6
1	∞	3	∞	2	∞	∞
2	∞	∞	10	6	∞	∞
3	∞	1	∞	9	∞	3
4	1	[4]	∞	[3]	4	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	1	∞	4	∞

$$u_{ij}^{(1)} = w_{ij} \quad \forall i, j$$

$$u_{ij}^{(m+1)} = \min \left\{ u_{ij}^{(m)}, u_{im}^{(m)} + u_{mj}^{(m)} \right\} \quad \forall i, j,$$

$$m = 1, 2, \dots, n$$



(m=3)

	1	2	3	4	5	6
1	∞	3	[13]	2	∞	∞
2	∞	∞	10	6	∞	∞
3	∞	1	[11]	[7]	∞	3
4	1	4	[14]	3	4	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	1	∞	4	∞

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

EJEMPLO:

	1	2	3	4	5	6
1	∞	3	[13]	2	∞	∞
2	∞	∞	10	6	∞	∞
3	∞	1	[11]	[7]	∞	3
4	1	4	[14]	3	4	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	1	∞	4	∞

(m=3)

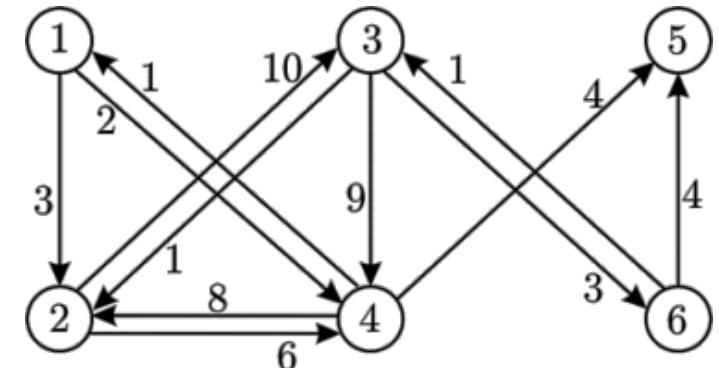
	1	2	3	4	5	6
1	∞	3	13	2	∞	[16]
2	∞	[11]	10	6	∞	[13]
3	∞	1	11	7	∞	3
4	1	4	14	3	4	[17]
5	∞	∞	∞	∞	∞	∞
6	∞	[2]	1	[8]	4	[4]

(m=4)

$$u_{ij}^{(1)} = w_{ij} \quad \forall i, j$$

$$u_{ij}^{(m+1)} = \min \left\{ u_{ij}^{(m)}, u_{im}^{(m)} + u_{mj}^{(m)} \right\} \quad \forall i, j,$$

$$m = 1, 2, \dots, n$$



(m=5)

	1	2	3	4	5	6
1	[3]	3	13	2	[6]	16
2	[7]	[10]	10	6	[10]	13
3	[8]	1	11	7	[11]	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	[9]	2	1	8	4	4

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

EJEMPLO:

	1	2	3	4	5	6
1	[3]	3	13	2	[6]	16
2	[7]	[10]	10	6	[10]	13
3	[8]	1	11	7	[11]	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	[9]	2	1	8	4	4

(m=5)

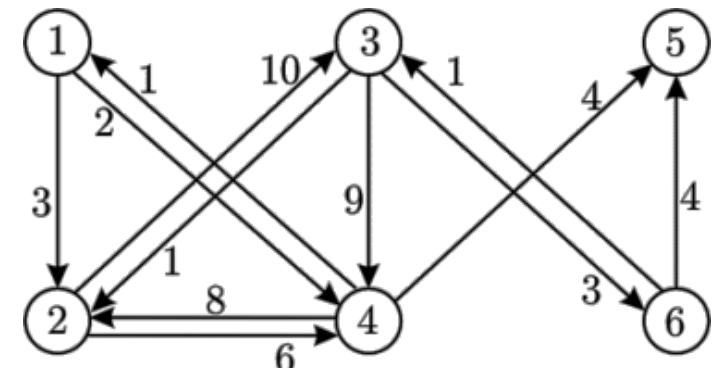
	1	2	3	4	5	6
1	3	3	13	2	6	16
2	7	10	10	6	10	13
3	8	1	11	7	11	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	9	2	1	8	4	4

(m=6)

$$u_{ij}^{(1)} = w_{ij} \quad \forall i, j$$

$$u_{ij}^{(m+1)} = \min \left\{ u_{ij}^{(m)}, u_{im}^{(m)} + u_{mj}^{(m)} \right\} \quad \forall i, j,$$

$$m = 1, 2, \dots, n$$



	1	2	3	4	5	6
1	3	3	13	2	6	16
2	7	10	10	6	10	13
3	8	1	11	7	11	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	9	2	1	8	4	4

(m=7)

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

Para facilitar la construcción de los caminos más cortos una vez calculados sus pesos, se puede utilizar otra matriz:

$$\Theta^{(m)} = [\theta_{ij}^{(m)}]$$

en la que $\theta_{ij}^{(m)}$ representa el vértice anterior al j en el camino más corto de i a j en la iteración m .

Inicialmente

$$\theta_{ij}^{(1)} = i \text{ si } u_{ij}^{(1)} < +\infty$$

y:

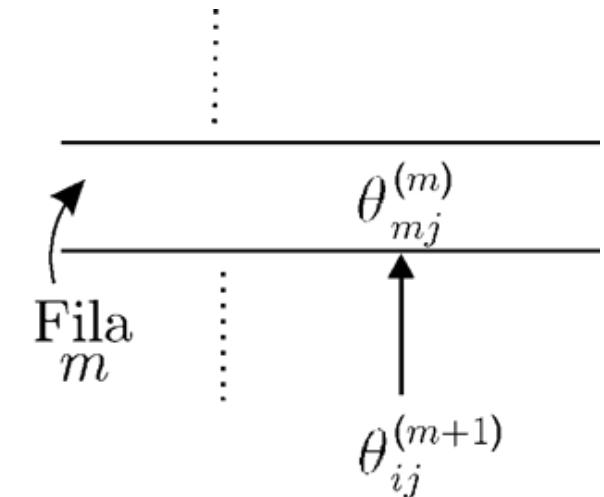
$$\theta_{ij}^{(m+1)} = \begin{cases} \theta_{ij}^{(m)} & \text{si } u_{ij}^{(m+1)} = u_{ij}^{(m)} \\ \theta_{mj}^{(m)} & \text{si } u_{ij}^{(m+1)} < u_{ij}^{(m)} \end{cases}$$

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

$$\theta_{ij}^{(m+1)} = \begin{cases} \theta_{ij}^{(m)} \\ \theta_{mj}^{(m)} \end{cases} \text{ si } u_{ij}^{(m+1)} = u_{ij}^{(m)} \\ \text{ si } u_{ij}^{(m+1)} < u_{ij}^{(m)} \end{cases}$$

Si un elemento de la matriz de pesos no se modifica en la iteración **$m+1$** , entonces el correspondiente elemento de la matriz $\Theta^{(m+1)}$ tampoco se modifica.

Si un elemento de la matriz de pesos se modifica en la iteración **$m+1$** , entonces el correspondiente elemento de la matriz $\Theta^{(m+1)}$ se modifica por el que ocupa su misma columna, y fila **m** .



Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

EJEMPLO:

	1	2	3	4	5	6
1	∞	3	∞	2	∞	∞
2	∞	∞	10	6	∞	∞
3	∞	1	∞	9	∞	3
4	1	8	∞	∞	4	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	1	∞	4	∞

	1	2	3	4	5	6
1	∞	3	∞	2	∞	∞
2	∞	∞	10	6	∞	∞
3	∞	1	∞	9	∞	3
4	1	[4]	∞	[3]	4	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	1	∞	4	∞

	1	2	3	4	5	6
1	∞	3	[13]	2	∞	∞
2	∞	∞	10	6	∞	∞
3	∞	1	[11]	[7]	∞	3
4	1	4	[14]	3	4	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	1	∞	4	∞

	1	2	3	4	5	6
1		1	1			
2			2	2		
3				3	3	3
4				4		4
5						
6				6	6	

	1	2	3	4	5	6
1		1	1			
2			2	2		
3				3	3	3
4				[1]	[1]	4
5						
6				6	6	

	1	2	3	4	5	6
1		1	[2]	1		
2			2	2		
3				3	[2]	3
4				4	[2]	1
5						
6				6	6	

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

EJEMPLO:

	1	2	3	4	5	6
1	∞	3	[13]	2	∞	∞
2	∞	∞	10	6	∞	∞
3	∞	1	[11]	[7]	∞	3
4	1	4	[14]	3	4	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	1	∞	4	∞

	1	2	3	4	5	6
1	∞	3	13	2	∞	[16]
2	∞	[11]	10	6	∞	[13]
3	∞	1	11	7	∞	3
4	1	4	14	3	4	[17]
5	∞	∞	∞	∞	∞	∞
6	∞	[2]	1	[8]	4	[4]

	1	2	3	4	5	6
1	[3]	3	13	2	[6]	16
2	[7]	[10]	10	6	[10]	13
3	[8]	1	11	7	[11]	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	[9]	2	1	8	4	4

	1	2	3	4	5	6
1		1	[2]	1		
2			2	2		
3				[2]	[2]	3
4	4	1	[2]	1	4	
5						
6					6	6

	1	2	3	4	5	6
1		1	2	1		[3]
2		[3]	2	2		[3]
3			3	2	2	3
4	4	1	2	1	4	[3]
5						
6		[3]	6	[2]	6	[3]

	1	2	3	4	5	6
1	[4]	1	2	1	[4]	3
2	[4]	[1]	2	2	[4]	3
3	[4]	3	2	2	[4]	3
4	4	1	2	1	4	3
5						
6	[4]	3	6	2	6	3

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

EJEMPLO:

	1	2	3	4	5	6
1	[3]	3	13	2	[6]	16
2	[7]	[10]	10	6	[10]	13
3	[8]	1	11	7	[11]	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	[9]	2	1	8	4	4

	1	2	3	4	5	6
1	3	3	13	2	6	16
2	7	10	10	6	10	13
3	8	1	11	7	11	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	9	2	1	8	4	4

	1	2	3	4	5	6
1	3	3	13	2	6	16
2	7	10	10	6	10	13
3	8	1	[4]	7	[7]	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	9	2	1	8	4	4

	1	2	3	4	5	6
1	[4]	1	2	1	[4]	3
2	[4]	[1]	2	2	[4]	3
3	[4]	3	2	2	[4]	3
4	4	1	2	1	4	3
5						
6	[4]	3	6	2	6	3

	1	2	3	4	5	6
1	4	1	2	1	4	3
2	4	1	2	2	4	3
3	4	3	2	2	4	3
4	4	1	2	1	4	3
5						
6	4	3	6	2	6	3

	1	2	3	4	5	6
1	4	1	2	1	4	3
2	4	1	2	2	4	3
3	4	3	[6]	2	[6]	3
4	4	1	2	1	4	3
5						
6	4	3	6	2	6	3

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

EJEMPLO:

	1	2	3	4	5	6
1	3	3	13	2	6	16
2	7	10	10	6	10	13
3	8	1	[4]	7	[7]	3
4	1	4	14	3	4	17
5	∞	∞	∞	∞	∞	∞
6	9	2	1	8	4	4

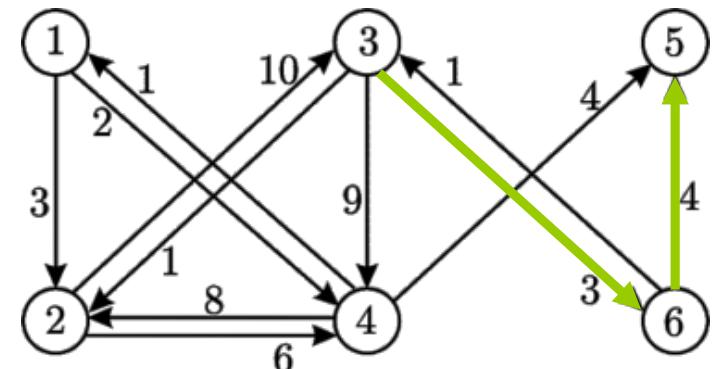
(m=7)

	1	2	3	4	5	6
1	4	1	2	1	4	3
2	4	1	2	2	4	3
3	4	3	[6]	2	6]	3
4	4	1	2	1	4	3
5						
6	4	3	6	2	6	3

IDENTIFICACIÓN DE CAMINOS:

Camino más corto de 3 a 5:

1. Peso: $u_{35}^{(7)} = 7$
2. Camino:
 - Vértice anterior al 5: $\theta_{35}^{(7)} = 6$
 - Vértice anterior al 6: $\theta_{36}^{(7)} = 3$



Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

OBSERVACIONES SOBRE LA IMPLEMENTACIÓN

- El bucle principal del algoritmo sería:

```
for (k=0; k<n; k++)
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (dist[i][k] + dist[k][j] < dist[i][j]) {
                dist[i][j] = dist[i][k] + dist[k][j];
                caminos[i][j] = caminos[k][j];
            }
        }
```

- Ahora bien, desde el punto de vista de la implementación es conveniente **almacenar como 0** el caso en el que no hay arco entre un par de vértices (**en lugar de infinito**), con lo que el bucle principal debe ser modificado adecuadamente.

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

OBSERVACIONES SOBRE LA IMPLEMENTACIÓN

- Con esta consideración, el bucle principal del algoritmo sería:

```
for (k=0; k<n; k++)
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (dist[i][k] * dist[k][j] != 0)
                if ((dist[i][k] + dist[k][j] < dist[i][j]) || (dist[i][j] == 0))
                    dist[i][j] = dist[i][k] + dist[k][j];
                    caminos[i][j] = caminos[k][j];
```

Solo entramos cuando $\text{dist}[i][k]$ y $\text{dist}[k][j]$ son distintos de INFINITO, es decir, cuando $\text{dist}[i][k]$ y $\text{dist}[k][j]$ sean $\neq 0$

Si $\text{dist}[i][j]$ fuese infinito seguro que se actualizaría por la suma $\text{dist}[i][k]+\text{dist}[k][j]$ que ya nos hemos asegurado que será no INFINITO. El equivalente es “SI $\text{dist}[i][j] == 0$ ”

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

IMPLEMENTACIÓN EN PARALELO

- Una posible versión paralela del algoritmo de Floyd-Warshall está basada en una descomposición unidimensional por **bloques de filas consecutivas** de las matrices intermedias en cada iteración.
- Se asume, sin pérdida de generalidad, que el número de vértices n es múltiplo del número de procesos p .
- Podremos utilizar n procesadores como máximo.
- Cada tarea será responsable de una o más filas adyacentes y ejecutará el siguiente código:

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

IMPLEMENTACIÓN EN PARALELO

```

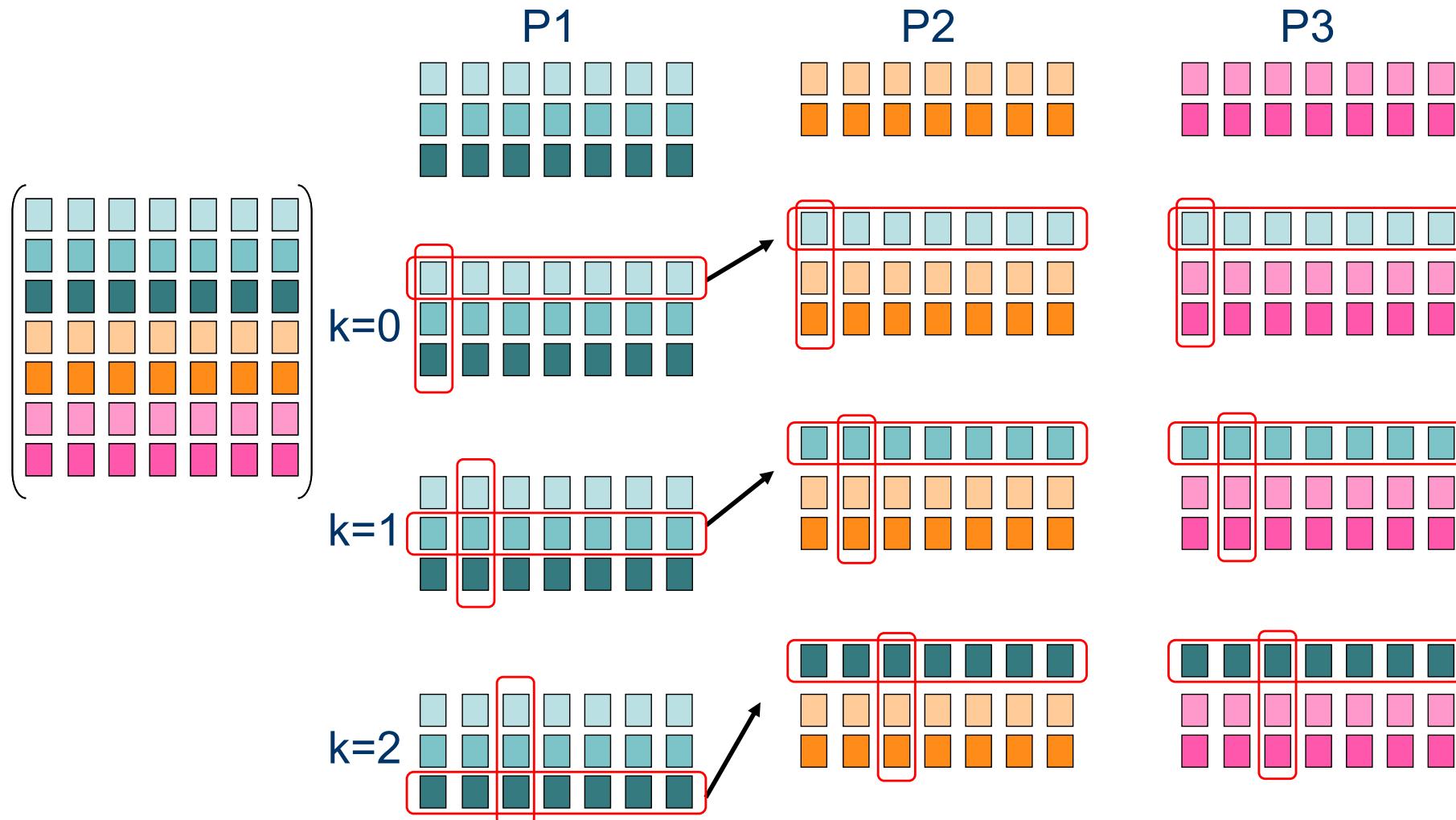
for (k=0; k<n; k++)
    Localizar el proceso que posee la fila k (sender)
    Hacer aux[ ] = dist[fila k][ ]
    Hacer auxC[ ] = caminos[fila k][ ]
    MPI_Bcast(&aux[0],n,MPI_FLOAT, sender, MPI_COMM_WORLD);
    MPI_Bcast(&auxC[0],n,MPI_INTEGER, sender, MPI_COMM_WORLD);
        for (i = 0; i < filas_por_proceso; i++)
            for (j = 0; j < n; j++)
                if (dist[i][k] * aux[j] != 0) {
if ((dist[i][k] + aux[j] | < dist[i][j]||(dist[i][j] == 0)){
                    dist[i][j] = dist[i][k] + aux[j] ;
                    caminos[i][j] = auxC[j] ;

```

- En la iteración k , cada tarea, además de sus datos locales, necesita los valores de la fila k , tanto de la matriz de pesos o distancias ($dist$) como de la matriz de caminos ($caminos$). Por ello la tarea que tenga asignada la fila k deberá difundirla (MPI_Bcast) a todas las demás.

Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

IMPLEMENTACIÓN EN PARALELO



Algoritmo de Floyd-Warshall para la obtención de los c.m.c. cortos en un grafo

IMPLEMENTACIÓN EN PARALELO

