

# 并行计算

——结构·算法·编程

主讲教师：谢磊

## 第三篇 并行数值算法

第八章 基本通讯操作

第九章 稠密矩阵运算

第十章 线性方程组的求解

第十一章 快速傅里叶变换

# 第九章 稠密矩阵运算

## 9.1 矩阵的划分

## 9.2 矩阵转置

## 9.3 矩阵-向量乘法

## 9.4 矩阵乘法

## 9.1 矩阵的划分

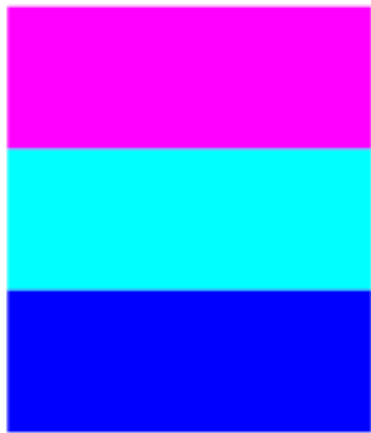
### 9.1.1 带状划分

### 9.1.2 棋盘划分

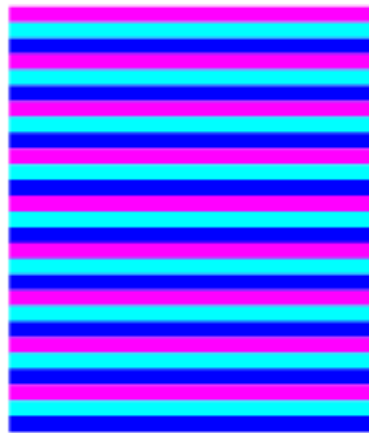


# 带状划分

\* 示例： $p=3$ ， $27 \times 27$ 矩阵的3种带状划分






(a) block



(b) cyclic



(c) block-cyclic

  $P_1$   
  $P_2$   
  $P_3$

Striped row-major mapping of a  $27 \times 27$  matrix on  $p = 3$  processors.

## 9.1 矩阵的划分

### 9.1.1 带状划分

### 9.1.2 棋盘划分

# 棋盘划分

\*  $8 \times 8$ 阶矩阵,  $p=16$

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)	(0, 7)
<b>P<sub>0</sub></b>	<b>P<sub>1</sub></b>	<b>P<sub>2</sub></b>	<b>P<sub>3</sub></b>				
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)	(1, 7)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)	(2, 7)
<b>P<sub>4</sub></b>	<b>P<sub>5</sub></b>	<b>P<sub>6</sub></b>	<b>P<sub>7</sub></b>				
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)	(3, 7)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)	(4, 7)
<b>P<sub>8</sub></b>	<b>P<sub>9</sub></b>	<b>P<sub>10</sub></b>	<b>P<sub>11</sub></b>				
(5, 0)	(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)	(5, 7)
(6, 0)	(6, 1)	(6, 2)	(6, 3)	(6, 4)	(6, 5)	(6, 6)	(6, 7)
<b>P<sub>12</sub></b>	<b>P<sub>13</sub></b>	<b>P<sub>14</sub></b>	<b>P<sub>15</sub></b>				
(7, 0)	(7, 1)	(7, 2)	(7, 3)	(7, 4)	(7, 5)	(7, 6)	(7, 7)

(a)  
块棋盘划分

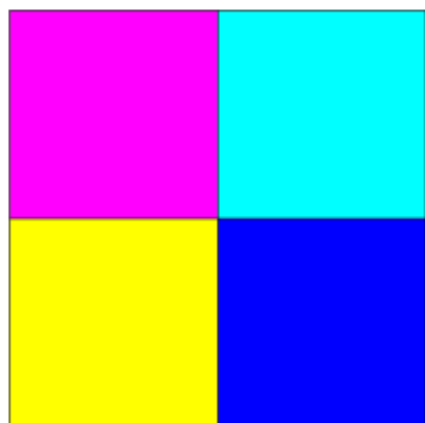
(0, 0)	(0, 4)	(0, 1)	(0, 5)	(0, 2)	(0, 6)	(0, 3)	(0, 7)
<b>P<sub>0</sub></b>	<b>P<sub>1</sub></b>	<b>P<sub>2</sub></b>	<b>P<sub>3</sub></b>				
(4, 0)	(4, 4)	(4, 1)	(4, 5)	(4, 2)	(4, 6)	(4, 3)	(4, 7)
(1, 0)	(1, 4)	(1, 1)	(1, 5)	(1, 2)	(1, 6)	(1, 3)	(1, 7)
<b>P<sub>4</sub></b>	<b>P<sub>5</sub></b>	<b>P<sub>6</sub></b>	<b>P<sub>7</sub></b>				
(5, 0)	(5, 4)	(5, 1)	(5, 5)	(5, 2)	(5, 6)	(5, 3)	(5, 7)
(2, 0)	(2, 4)	(2, 1)	(2, 5)	(2, 2)	(2, 6)	(2, 3)	(2, 7)
<b>P<sub>8</sub></b>	<b>P<sub>9</sub></b>	<b>P<sub>10</sub></b>	<b>P<sub>11</sub></b>				
(6, 0)	(6, 4)	(6, 1)	(6, 5)	(6, 2)	(6, 6)	(6, 3)	(6, 7)
(3, 0)	(3, 4)	(3, 1)	(3, 5)	(3, 2)	(3, 6)	(3, 3)	(3, 7)
<b>P<sub>12</sub></b>	<b>P<sub>13</sub></b>	<b>P<sub>14</sub></b>	<b>P<sub>15</sub></b>				
(7, 0)	(7, 4)	(7, 1)	(7, 5)	(7, 2)	(7, 6)	(7, 3)	(7, 7)

(b)  
循环棋盘划分

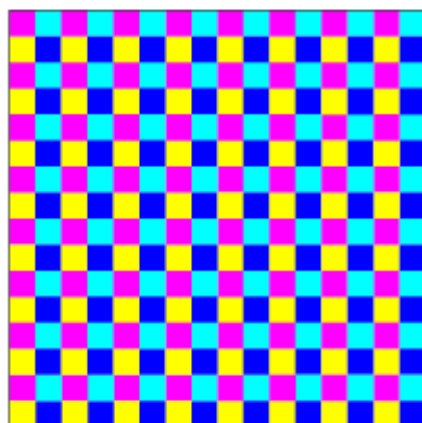
图9.2

# 棋盘划分

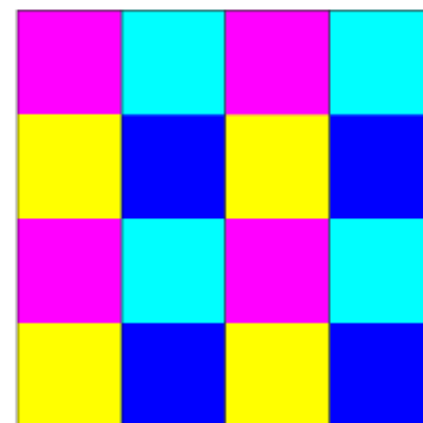
\* 示例： $p=4$ ， $16 \times 16$ 矩阵的3种棋盘划分



(a) block



(b) cyclic



(c) block cyclic



Checkerboard mapping of a  $16 \times 16$  matrix on  $p = 2 \times 2$  processors.

# 第九章 稠密矩阵运算

9.1 矩阵的划分

9.2 矩阵转置

9.3 矩阵-向量乘法

9.4 矩阵乘法

# 单处理机上的矩阵转置算法

\* 算法9.1 单处理机上的矩阵转置算法

\* 输入:  $A_{n \times n}$

\* 输出:  $A^T_{n \times n}$

Begin

for i=2 to n do

for j=1 to i-1 do

$a_{i,j} \leftrightarrow a_{j,i}$

endfor

endfor

End

运行时间:  $(n^2 - n)/2 = O(n^2)$

## 9.2 矩阵转置

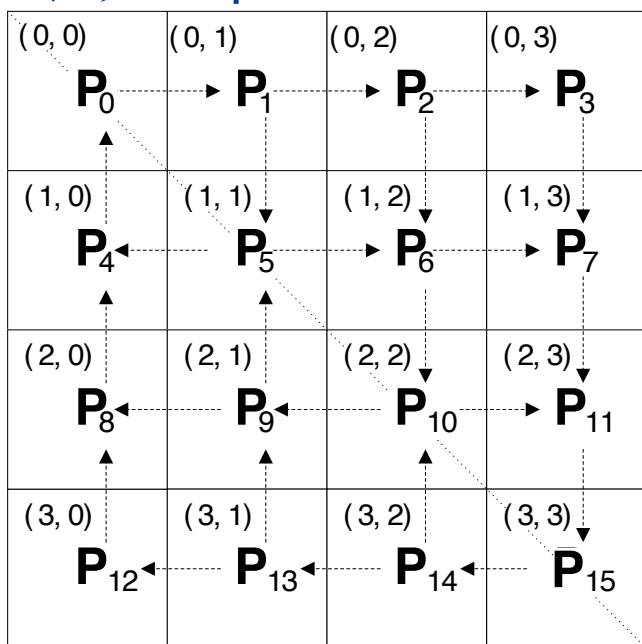
### 9.2.1 棋盘划分的矩阵转置

### 9.2.2 带状划分的矩阵转置

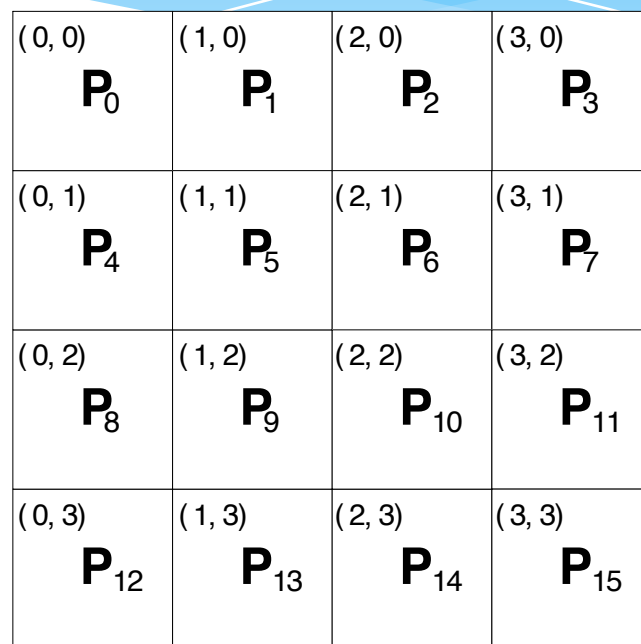
# 棋盘划分的矩阵转置

## \* 网孔连接

\* 情形1:  $p=n^2$ 。



( a )



( b )

通讯步

图9.3

转置后

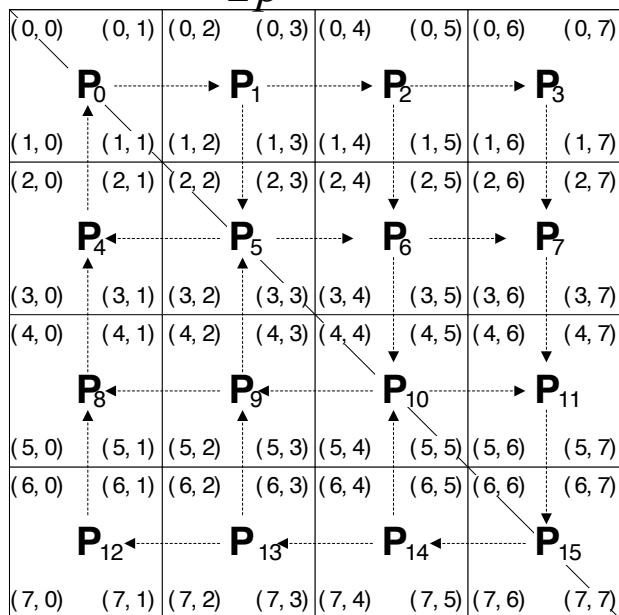
# 棋盘划分的矩阵转置

\* 情形2:  $p < n^2$ 。

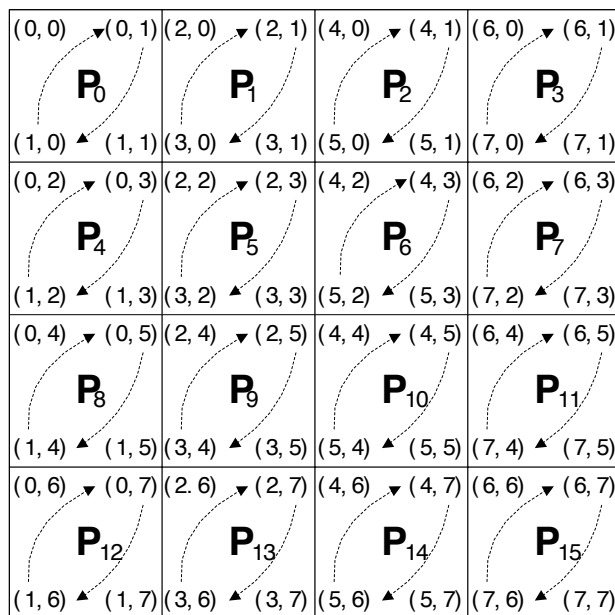
- 划分:  $A_{n \times n}$  划分成  $p$  个大小为  $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$  子块

- 算法: ①按mesh连接进行块转置(不同处理器间) //  $2\sqrt{p}(t_s + t_w n^2 / p) \dots$  通讯  
 ②进行块内转置(同一处理器内) //  $\frac{n^2}{2p} \dots$  计算

$$T_p = \frac{n^2}{2p} + 2t_s \sqrt{p} + 2t_w n^2 / \sqrt{p} \dots \dots \text{运行时间}$$



通讯步



转置后

图9.4

# 棋盘划分的矩阵转置

## \* 超立方连接

\* 划分:  $A_{n \times n}$  划分成  $p$  个大小为  $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$  子块

\* 算法:

① 将  $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$  转置为  $\begin{pmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{pmatrix}$

② 对  $A_{ij}$  递归应用①进行转置, 直至分块矩阵的元素处于同一处理器;

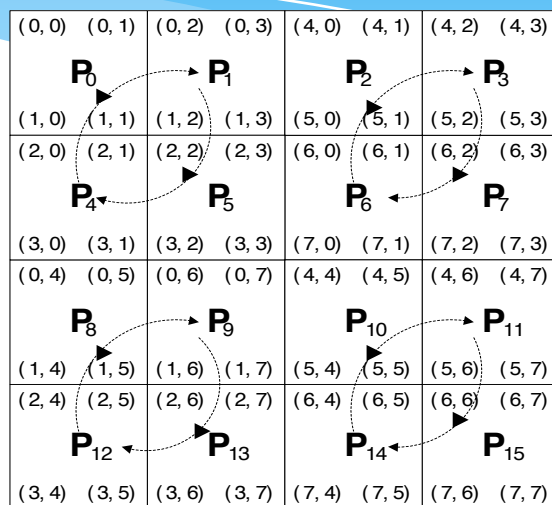
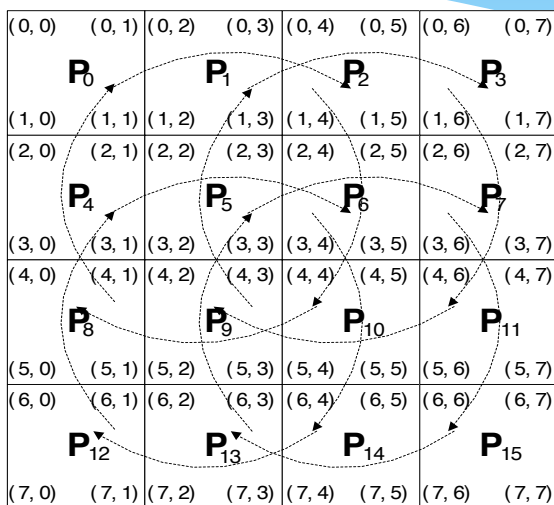
③ 进行同一处理器的内部转置。

\* 运行时间:

$$T_p = \frac{n^2}{2p} + 2(t_s + t_w \frac{n^2}{p}) \log \sqrt{p} \quad // \text{内部转置 } \frac{n^2}{2p}, \text{选路: } 2(t_s + t_w \frac{n^2}{p}), \text{递归步: } \log \sqrt{p}$$
$$= \frac{n^2}{2p} + (t_s + t_w \frac{n^2}{p}) \log p$$

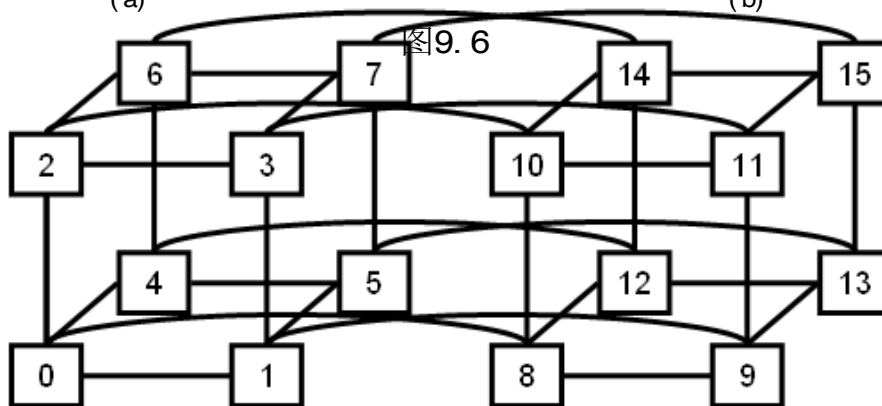
# 棋盘划分的矩阵转置

\* 超立方连接：示例



(a)

(b)



## 9.2 矩阵转置

### 9.2.1 棋盘划分的矩阵转置

### 9.2.2 带状划分的矩阵转置

# 带状划分的矩阵转置

\* 划分:  $An \times n$  分成  $p$  个  $(n/p) \times n$  大小的带

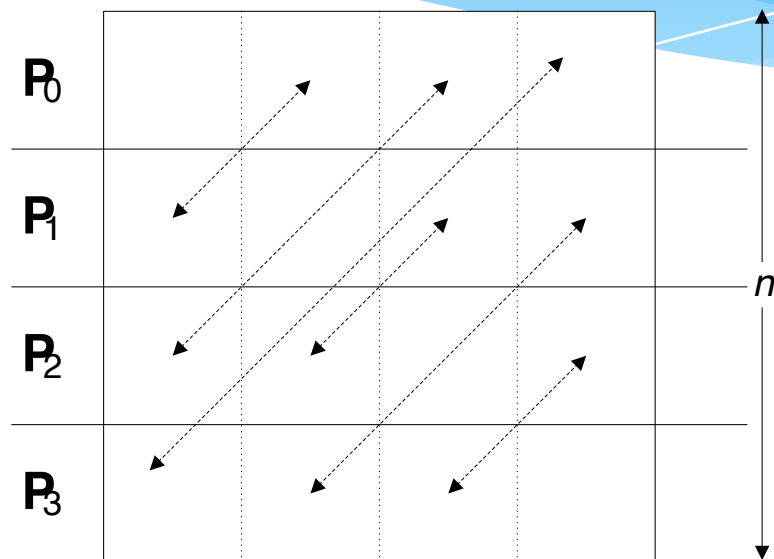


图9.7

\* 算法:

- ①  $P_i$  有  $p-1$  个  $(n/p) \times (n/p)$  大小子块发送到另外  $p-1$  个处理器中;
- ② 每个处理器本地交换相应的元素

# 第九章 稠密矩阵运算

9.1 矩阵的划分

9.2 矩阵转置

9.3 矩阵-向量乘法

9.4 矩阵乘法

# 单处理机上的矩阵-向量乘法

- \* 算法9.2单处理机上的矩阵-向量乘法
- \* 输入:  $A_{n \times n}$ ,  $X_{n \times 1}$
- \* 输出:  $Y_{n \times 1}$

Begin

for i=0 to n-1 do

$y_i = 0$

for j=0 to n-1 do

$y_i = y_i + a_{i,j} \times x_j$

endfor

endfor

End

- \* 算法运行时间为 $O(n^2)$

## 9.3 矩阵-向量乘法

### 9.3.1 带状划分的矩阵-向量乘法

### 9.3.2 棋盘划分的矩阵-向量乘法

# 带状划分的矩阵-向量乘法

\* 划分(行带状划分):  $P_i$  存放  $x_i$  和  $a_{i,0}, a_{i,1}, \dots, a_{i,n-1}$ , 并输出  $y_i$

\* 算法: 对  $p=n$  情形

① 每个  $P_i$  向其他处理器播送  $x_i$  (多到多播送);

② 每个  $P_i$  计算;

\* 注: 对  $p < n$  情形, 算法中  $P_i$  要播送  $X$  中相应的  $n/p$  个分量

(1) 超立方连接的计算时间

$$T_p = \frac{n^2}{p} + t_s \log p + \frac{n}{p} t_w (p-1) \quad // \text{前1项是乘法时间, 后2项是多到多的播送时间}$$

$$= \frac{n^2}{p} + t_s \log p + n t_w \quad // p \text{ 充分大时}$$

(2) 网孔连接的计算时间

$$T_p = \frac{n^2}{p} + 2(\sqrt{p}-1)t_s + \frac{n}{p} t_w (p-1) \quad // \text{前1项是乘法时间, 后2项是多到多的播送时间}$$

$$= \frac{n^2}{p} + 2t_s(\sqrt{p}-1) + n t_w \quad // p \text{ 充分大时}$$

# 带状划分的矩阵-向量乘法

\* 示例

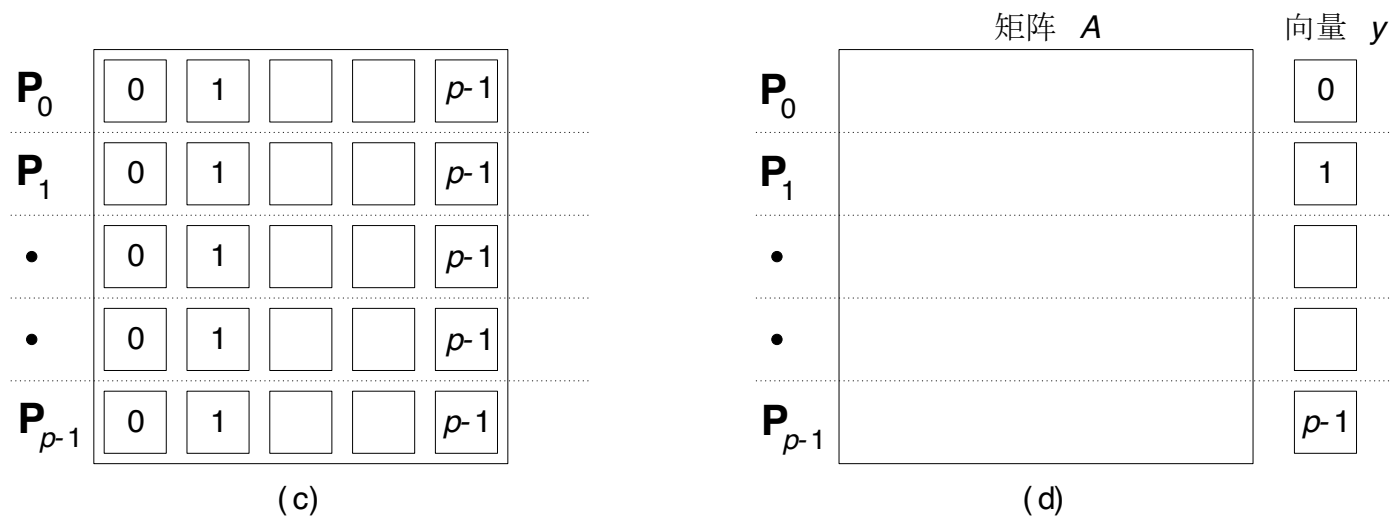
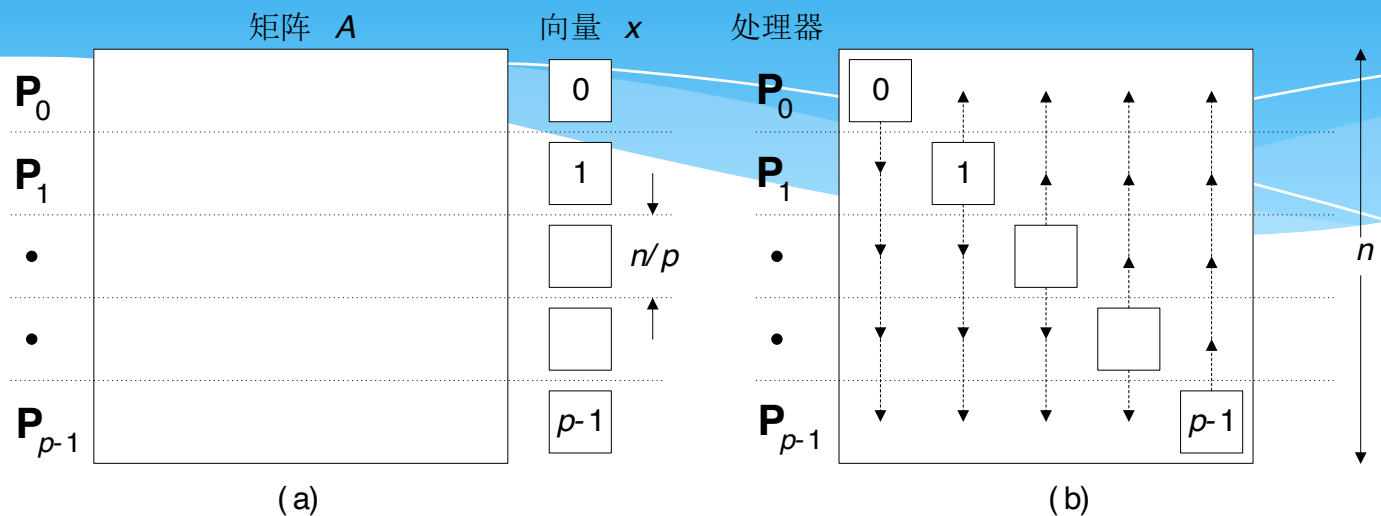


图9.8

## 9.3 矩阵-向量乘法

9.3.1 带状划分的矩阵-向量乘法

9.3.2 棋盘划分的矩阵-向量乘法

# 棋盘划分的矩阵-向量乘法

\* 划分(块棋盘划分):  $P_{ij}$  存放  $a_{i,j}$ ,  $x_i$  置入  $P_{i,i}$  中

\* 算法: 对  $p=n^2$  情形

① 每个  $P_{i,i}$  向  $P_{j,i}$  播送  $x_i$  (一到多播送);

② 按行方向进行乘-加与积累运算, 最后一列  $P_{i,n-1}$  收集的结果为  $y_i$ ;

\* 注: 对  $p < n^2$  情形,  $p$  个处理器排成  $\sqrt{p} \times \sqrt{p}$  的二维网孔,

算法中  $P_{i,i}$  向  $P_{j,i}$  播送  $X$  中相应的  $n/\sqrt{p}$  个分量

(1) 网孔连接的计算时间  $T_p$  (CT):  $t_s + \frac{n}{\sqrt{p}} t_w + t_h \sqrt{p}$

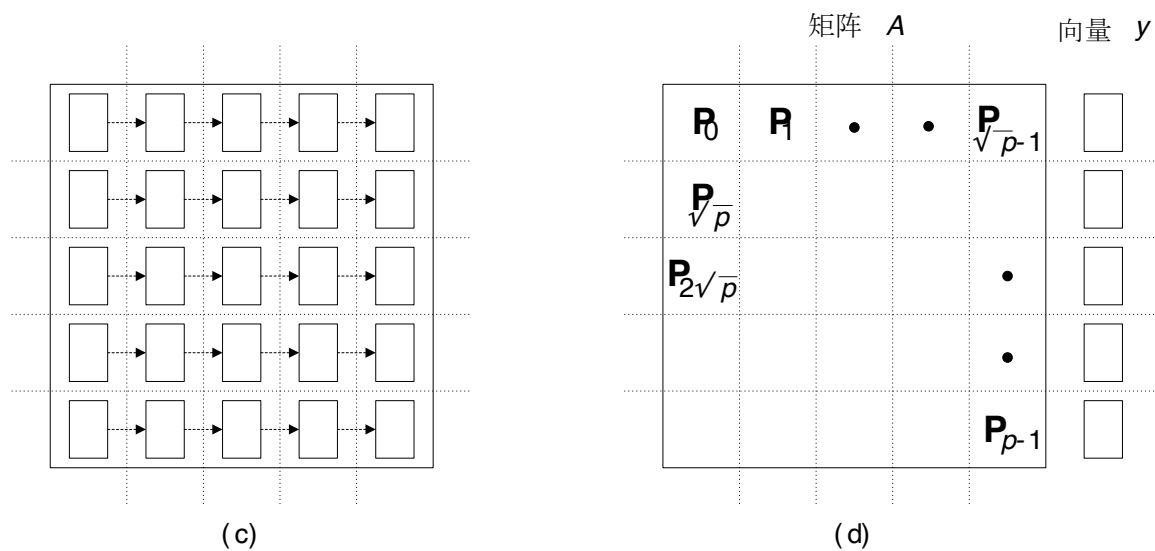
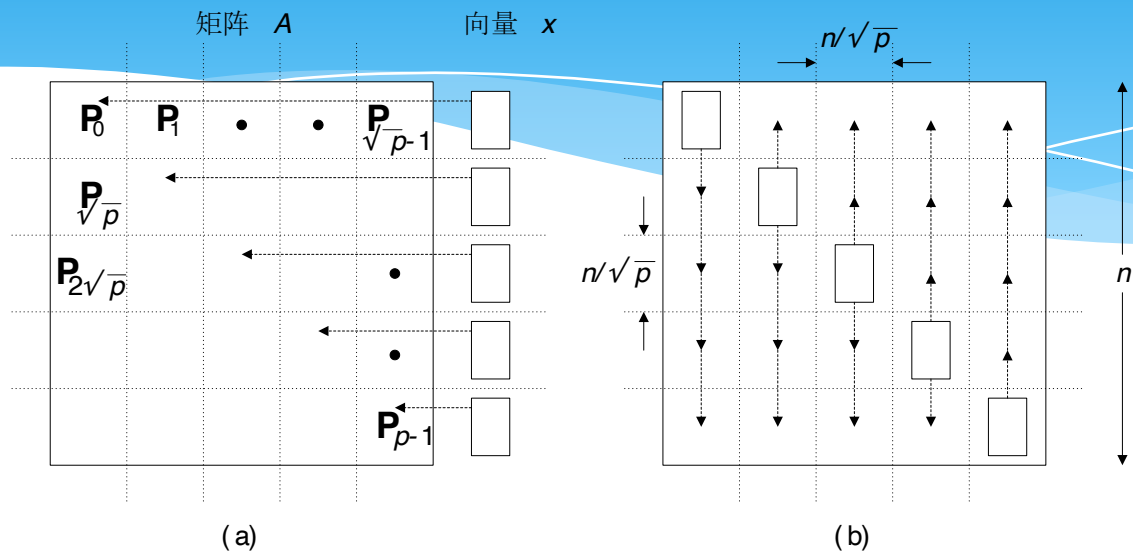
. 按列一到多播送时间:  $(t_s + \frac{n}{\sqrt{p}} t_w) \log \sqrt{p} + t_h (\sqrt{p} - 1)$

. 按行单点积累的时间:  $(t_s + \frac{n}{\sqrt{p}} t_w) \log \sqrt{p} + t_h (\sqrt{p} - 1)$

. 总运行时间为:  $\therefore T_p \approx \frac{n^2 \sqrt{p}}{p} + t_s \log p + \frac{n}{\sqrt{p}} t_w \log p + 3t_h \sqrt{p}$

# 棋盘划分的矩阵-向量乘法

\* 示例



# 带状与棋盘划分比较

\*以网孔链接为例

\*网孔上带状划分的运行时间

$$T_p = \frac{n^2}{p} + 2t_s(\sqrt{p}-1) + nt_w \quad (9.5)$$

\*网孔上棋盘划分的运行时间

$$T_p \approx \frac{n^2}{p} + t_s \log p + \frac{n}{\sqrt{p}} t_w \log p + 3t_h \sqrt{p} \quad (9.6)$$

\*棋盘划分要比带状划分快。

# 第九章 稠密矩阵运算

9.1 矩阵的划分

9.2 矩阵转置

9.3 矩阵-向量乘法

9.4 矩阵乘法

## 9.4 矩阵乘法

### 9.4.1 简单并行分块乘法

### 9.4.2 Cannon乘法

### 9.4.3 Fox乘法

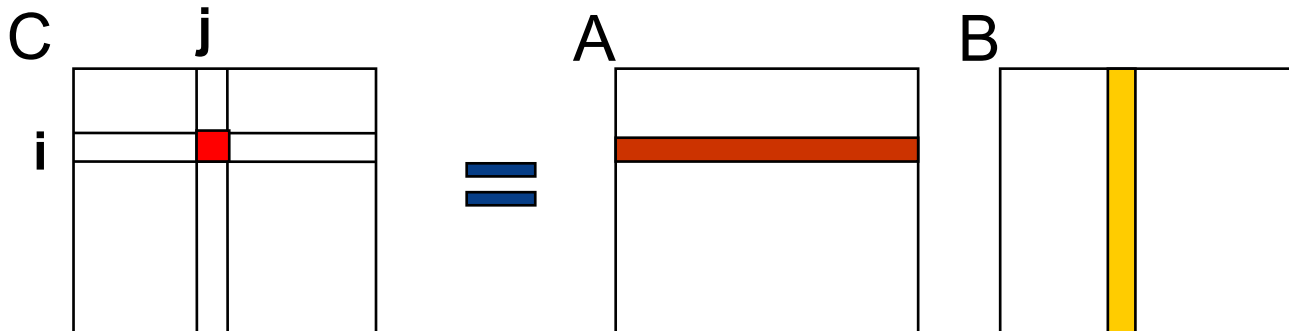
### 9.4.4 Systolic乘法

### 9.4.5 DNS乘法

# 矩阵乘法符号及定义

设  $A = (a_{ij})_{n \times n}$   $B = (b_{ij})_{n \times n}$   $C = (c_{ij})_{n \times n}$ ,  $C = A \times B$

$$\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,n-1} \\ c_{1,0} & c_{1,1} & & c_{1,n-1} \\ \vdots & \vdots & & \vdots \\ c_{n-1,0} & c_{n-1,1} & \cdots & c_{n-1,n-1} \end{pmatrix} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix} \cdot \begin{pmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & & b_{1,n-1} \\ \vdots & \vdots & & \vdots \\ b_{n-1,0} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{pmatrix}$$



$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

A中元素的第1下标与B中元素的第2下标相一致 (对准)

# 单处理机上分块矩阵相乘

- \* 算法9.4单处理机上分块矩阵相乘算法
- \* 输入:  $A_{n \times n}, B_{n \times n}$ , 子块大小为  $n/q * n/q$
- \* 输出:  $C_{n \times n}$

Begin

for  $i=0$  to  $q-1$  do

for  $j=1$  to  $q-1$  do

$C_{i,j} = 0$

for  $k=0$  to  $q-1$  do

$C_{i,j} = C_{i,j} + A_{i,k} \times B_{k,j}$

endfor

endfor

End

- \* 算法运行时间为  $O(n^3)$ .

# 矩阵乘法并行实现方法

- \* 计算结构：二维阵列
- \* 空间对准(元素已加载到阵列中)  
Cannon's, Fox's, DNS
- \* 时间对准(元素未加载到阵列中)

Systolic

$A_{0,0}$ $B_{0,0}$	$A_{0,1}$ $B_{0,1}$	$A_{0,2}$ $B_{0,2}$	$A_{0,3}$ $B_{0,3}$
$A_{1,0}$ $B_{1,0}$	$A_{1,1}$ $B_{1,1}$	$A_{1,2}$ $B_{1,2}$	$A_{1,3}$ $B_{1,3}$
$A_{2,0}$ $B_{2,0}$	$A_{2,1}$ $B_{2,1}$	$A_{2,2}$ $B_{2,2}$	$A_{2,3}$ $B_{2,3}$
$A_{3,0}$ $B_{3,0}$	$A_{3,1}$ $B_{3,1}$	$A_{3,2}$ $B_{3,2}$	$A_{3,3}$ $B_{3,3}$

# 简单并行分块乘法

\* 分块: A、B和C分成  $p = \sqrt{p} \times \sqrt{p}$  的方块阵  $A_{i,j}$ 、 $B_{i,j}$ 和 $C_{i,j}$ , 大小均为  $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$   
p个处理器编号为  $(P_{0,0}, \dots, P_{0,\sqrt{p}-1}, \dots, P_{\sqrt{p}-1,\sqrt{p}-1})$   $P_{i,j}$  存放  $A_{i,j}$ 、 $B_{i,j}$ 和 $C_{i,j}$ 。

\* 算法:

① 通讯: 每行处理器进行A矩阵块的多到多播送(得到  $A_{i,k}$ ,  $k=0 \sim \sqrt{p}-1$ )

每列处理器进行B矩阵块的多到多播送(得到  $B_{k,j}$ ,  $k=0 \sim \sqrt{p}-1$ )

② 乘-加运算:  $P_{i,j}$  做  $C_{ij} = \sum_{k=0}^{\sqrt{p}-1} A_{ik} \cdot B_{kj}$

\* 运行时间

(1) 超立方连接:

① 的时间  $t_1 = 2(t_s \log \sqrt{p} + t_w \frac{n^2}{\sqrt{p}} (\sqrt{p} - 1))$

② 的时间  $t_2 = \sqrt{p} \times (\frac{n}{\sqrt{p}})^3 = n^3 / p$

# 简单并行分块乘法

## \* 运行时间

(2) 二维环绕网孔连接:

① 的时间:  $t_1 = 2(t_s + \frac{n^2}{p}t_w)(\sqrt{p}-1) = 2t_s\sqrt{p} + 2t_w\frac{n^2}{\sqrt{p}}$

② 的时间  $t_2 = n^3/p$

$$\therefore T_p = \frac{n^3}{p} + 2t_s\sqrt{p} + 2t_w\frac{n^2}{\sqrt{p}}$$

## \* 注

(1) 本算法的缺点是对处理器的存储要求过大

每个处理器有  $2\sqrt{p}$  个块, 每块大小为  $n^2/p$ ,

所以需要  $O(n^2/\sqrt{p})$ ,  $p$  个处理器共需要  $O(n^2\sqrt{p})$ ,

是串行算法的  $\sqrt{p}$  倍

(2)  $p=n^2$  时,  $t(n)=O(n)$ ,  $c(n)=O(n^3)$

## 9.4 矩阵乘法

9.4.1 简单并行分块乘法

9.4.2 Cannon乘法

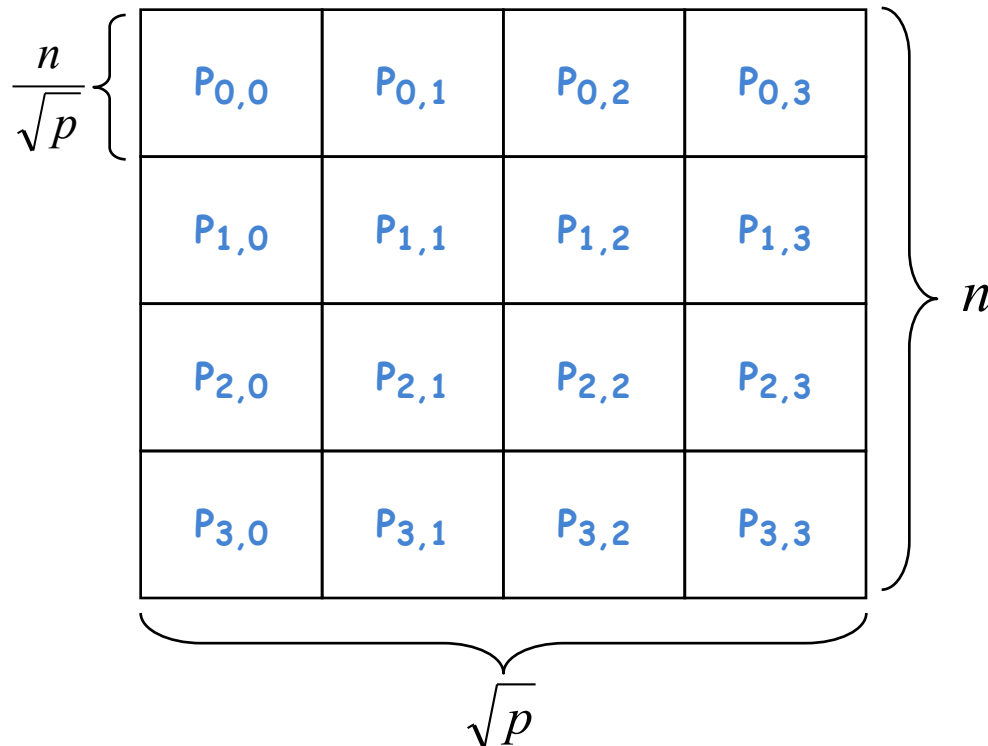
9.4.3 Fox乘法

9.4.4 Systolic乘法

9.4.5 DNS乘法

# Cannon 乘法

- \* 分块: A、B和C分成  $p = \sqrt{p} \times \sqrt{p}$  的方块阵  $A_{i,j}$ 、 $B_{i,j}$ 和 $C_{i,j}$ , 大小  $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$  均为  $p$  个处理器编号为  $(P_{0,0}, \dots, P_{0,\sqrt{p}-1}, \dots, P_{\sqrt{p}-1,\sqrt{p}-1})$ ,  $P_{i,j}$  存放  $A_{i,j}$ 、 $B_{i,j}$ 和 $C_{i,j}$  ( $n \gg p$ )



# Cannon乘法

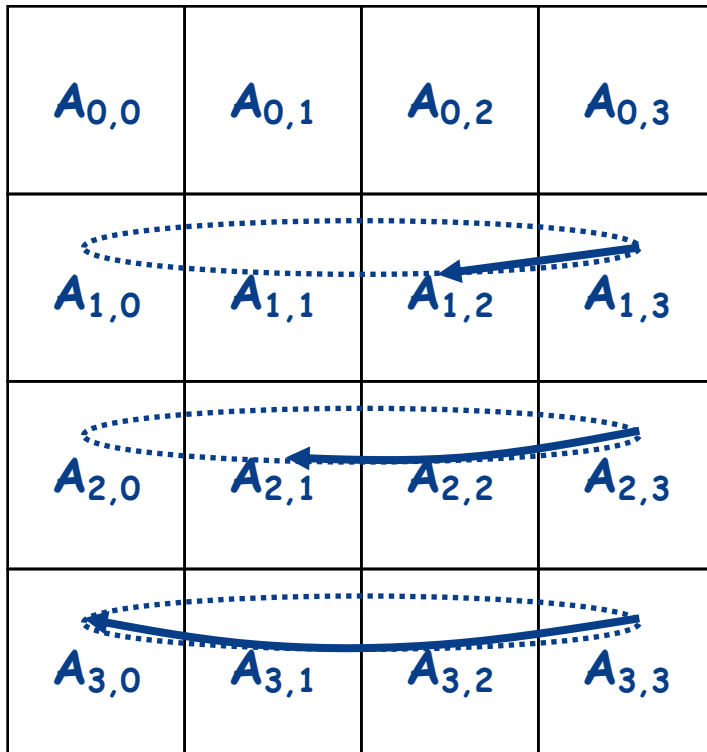
## \* 算法原理 (非形式描述)

- ① 所有块  $A_{i,j}$  ( $0 \leq i, j \leq \sqrt{p}-1$ ) 向左循环移动  $i$  步 (按行移位);  
所有块  $B_{i,j}$  ( $0 \leq i, j \leq \sqrt{p}-1$ ) 向上循环移动  $j$  步 (按列移位);
- ② 所有处理器  $P_{i,j}$  做执行  $A_{i,j}$  和  $B_{i,j}$  的乘-加运算;
- ③ A 的每个块向左循环移动一步;  
B 的每个块向上循环移动一步;
- ④ 转②执行  $\sqrt{p}-1$  次;

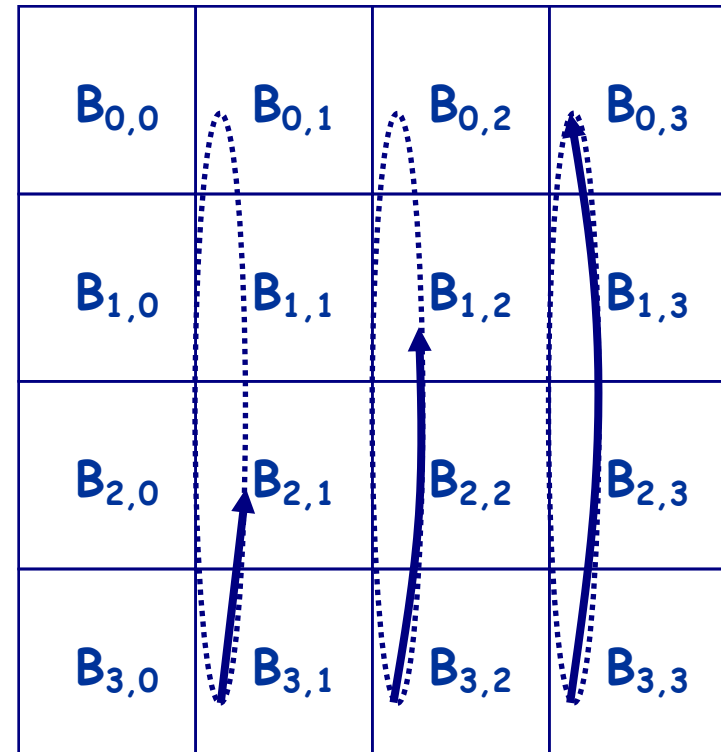
# Cannon 乘法

\* 示例:  $A_{4 \times 4}$ ,  $B_{4 \times 4}$ ,  $p=16$

Initial alignment of A



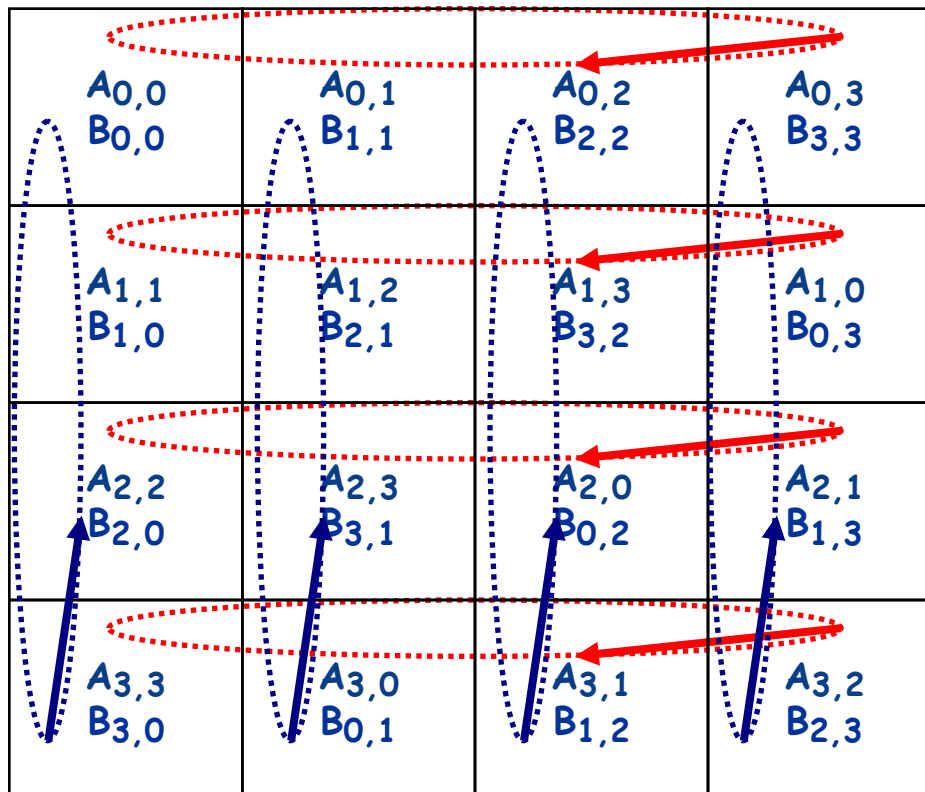
Initial alignment of B



# Cannon 乘法

\* 示例:  $A_{4 \times 4}$ ,  $B_{4 \times 4}$ ,  $p=16$

A and B after initial alignment and shifts after every step



# Cannon 乘法

\* 示例:  $A_{4 \times 4}$ ,  $B_{4 \times 4}$ ,  $p=16$

After first shift

After second shift

After third shift

$A_{0,1}$ $B_{1,0}$	$A_{0,2}$ $B_{2,1}$	$A_{0,3}$ $B_{3,2}$	$A_{0,0}$ $B_{0,3}$
$A_{1,2}$ $B_{2,0}$	$A_{1,3}$ $B_{3,1}$	$A_{1,0}$ $B_{0,2}$	$A_{1,1}$ $B_{1,3}$
$A_{2,3}$ $B_{3,0}$	$A_{2,0}$ $B_{0,1}$	$A_{2,1}$ $B_{1,2}$	$A_{2,2}$ $B_{2,3}$
$A_{3,0}$ $B_{0,0}$	$A_{3,1}$ $B_{3,1}$	$A_{3,2}$ $B_{2,2}$	$A_{3,3}$ $B_{3,3}$

$A_{0,2}$ $B_{2,0}$	$A_{0,3}$ $B_{3,1}$	$A_{0,0}$ $B_{0,2}$	$A_{0,1}$ $B_{1,3}$
$A_{1,3}$ $B_{3,0}$	$A_{1,0}$ $B_{0,1}$	$A_{1,1}$ $B_{1,2}$	$A_{1,2}$ $B_{2,3}$
$A_{2,0}$ $B_{0,0}$	$A_{2,1}$ $B_{1,1}$	$A_{2,2}$ $B_{2,2}$	$A_{2,3}$ $B_{3,3}$
$A_{3,1}$ $B_{1,0}$	$A_{3,2}$ $B_{2,1}$	$A_{3,3}$ $B_{3,2}$	$A_{3,0}$ $B_{0,3}$

$A_{0,3}$ $B_{3,0}$	$A_{0,0}$ $B_{0,1}$	$A_{0,1}$ $B_{1,2}$	$A_{0,2}$ $B_{2,3}$
$A_{1,0}$ $B_{0,0}$	$A_{1,1}$ $B_{1,1}$	$A_{1,2}$ $B_{2,2}$	$A_{1,3}$ $B_{3,3}$
$A_{2,1}$ $B_{1,0}$	$A_{2,2}$ $B_{2,1}$	$A_{2,3}$ $B_{3,2}$	$A_{2,0}$ $B_{0,3}$
$A_{3,2}$ $B_{2,0}$	$A_{3,3}$ $B_{3,1}$	$A_{3,0}$ $B_{0,2}$	$A_{3,1}$ $B_{1,3}$

# Cannon 乘法

## \* 算法描述: Cannon 分块乘法算法

//输入:  $A_{n \times n}$ ,  $B_{n \times n}$ ; 输出:  $C_{n \times n}$

Begin

(1) for  $k=0$  to  $\sqrt{p}-1$  do

for all  $P_{i,j}$  par-do

(i) if  $i > k$  then

$A_{i,j} \leftarrow A_{i,(j+1) \bmod \sqrt{p}}$   
endif

(ii) if  $j > k$  then

$B_{i,j} \leftarrow B_{(i+1) \bmod \sqrt{p}, j}$   
endif

endfor

endfor

(2) for all  $P_{i,j}$  par-do  $C_{i,j} = 0$  endfor

(3) for  $k=0$  to  $\sqrt{p}-1$  do  
for all  $P_{i,j}$  par-do  
(i)  $C_{i,j} = C_{i,j} + A_{i,j} B_{i,j}$   
(ii)  $A_{i,j} \leftarrow A_{i,(j+1) \bmod \sqrt{p}}$   
(iii)  $B_{i,j} \leftarrow B_{(i+1) \bmod \sqrt{p}, j}$   
endfor  
endfor

End

## 时间分析:

$$T_p(n) = T_1 + T_2 + T_3$$

$$= O(\sqrt{p}) + O(1) + O(\sqrt{p} \cdot (n/\sqrt{p})^3)$$

$$= O(n^3 / p)$$

## 9.4 矩阵乘法

9.4.1 简单并行分块乘法

9.4.2 Cannon乘法

9.4.3 Fox乘法

9.4.4 Systolic乘法

9.4.5 DNS乘法

# FOX乘法

\* 分块：同Cannon分块算法

\* 算法原理

①  $A_{i,j}$  向所在行的其他处理器进行一到多播送；

② 各处理器将收到的A块与原有的B块进行乘-加运算；

③ B块向上循环移动一步；

④ 如果  $A_{i,j}$  是上次第  $i$  行播送的块，本次选择  $A_{i,(j+1) \bmod \sqrt{p}}$  向所在行的其他处理器进行一到多播送；

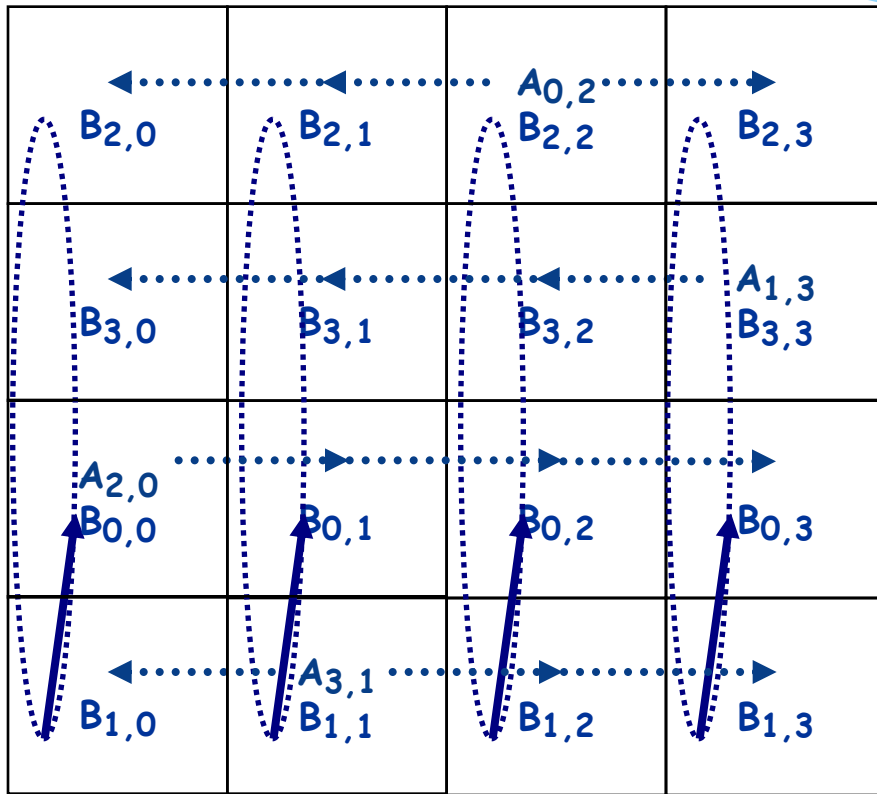
⑤ 转②执行  $\sqrt{p} - 1$  次；

$A_{0,0}$ $B_{0,0}$	$A_{0,1}$ $B_{0,1}$	$A_{0,2}$ $B_{0,2}$	$A_{0,3}$ $B_{0,3}$
$A_{1,0}$ $B_{1,0}$	$A_{1,1}$ $B_{1,1}$	$A_{1,2}$ $B_{1,2}$	$A_{1,3}$ $B_{1,3}$
$A_{2,0}$ $B_{2,0}$	$A_{2,1}$ $B_{2,1}$	$A_{2,2}$ $B_{2,2}$	$A_{2,3}$ $B_{2,3}$
$A_{3,0}$ $B_{3,0}$	$A_{3,1}$ $B_{3,1}$	$A_{3,2}$ $B_{3,2}$	$A_{3,3}$ $B_{3,3}$

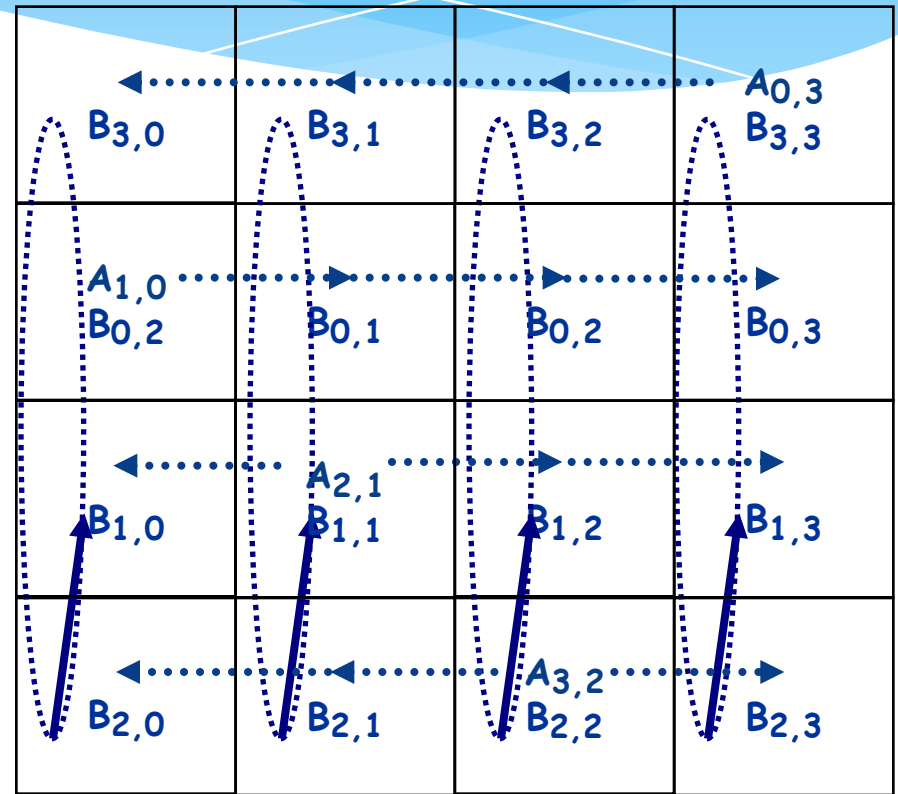


# FOX乘法

\* 示例：  $A_{4 \times 4}$ ,  $B_{4 \times 4}$ ,  $p=16$



(c)



(d)

## 9.4 矩阵乘法

9.4.1 简单并行分块乘法

9.4.2 Cannon乘法

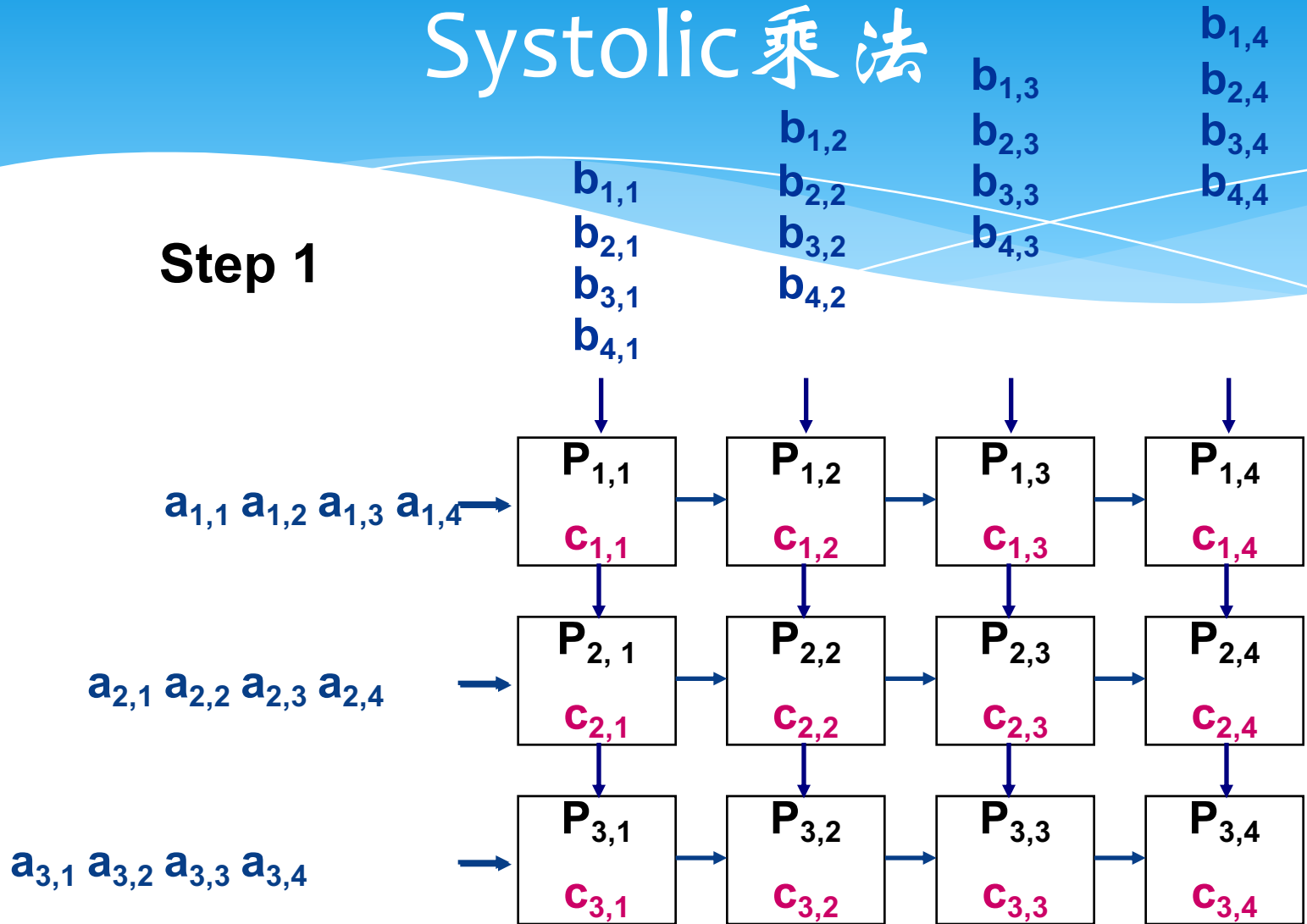
9.4.3 Fox乘法

9.4.4 Systolic乘法

9.4.5 DNS乘法

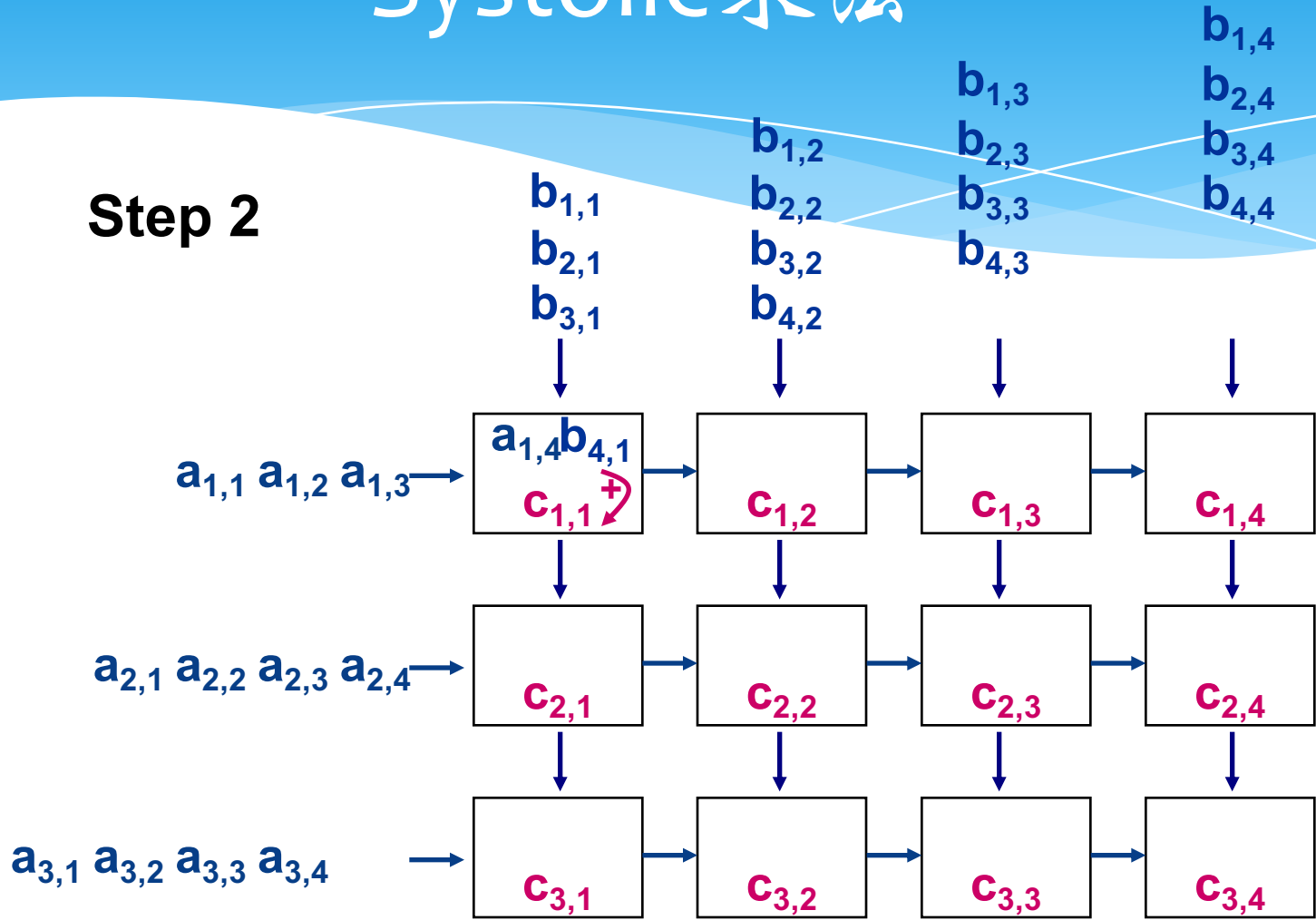
# Systolic 乘法

Step 1



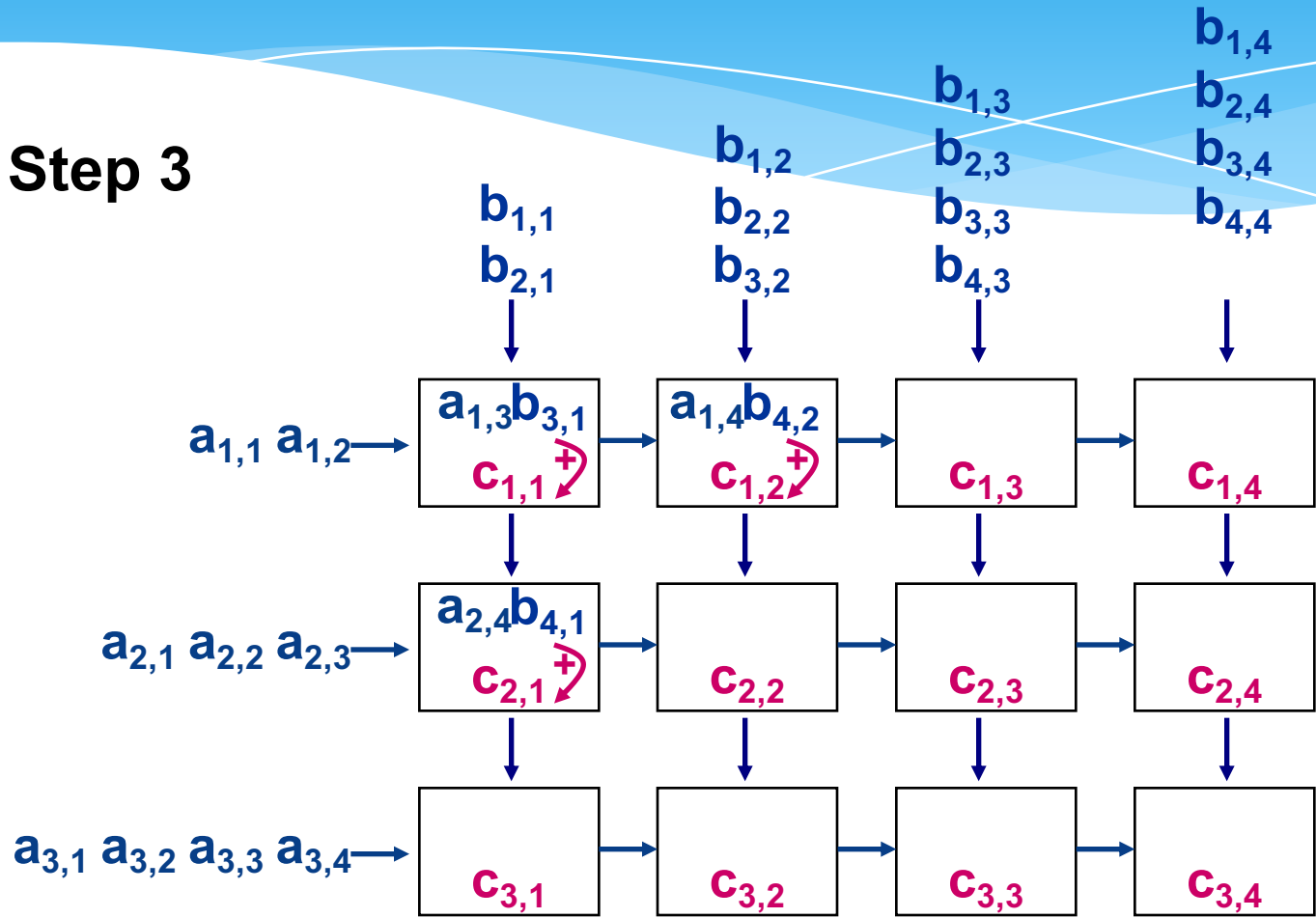
# Systolic 乘法

Step 2



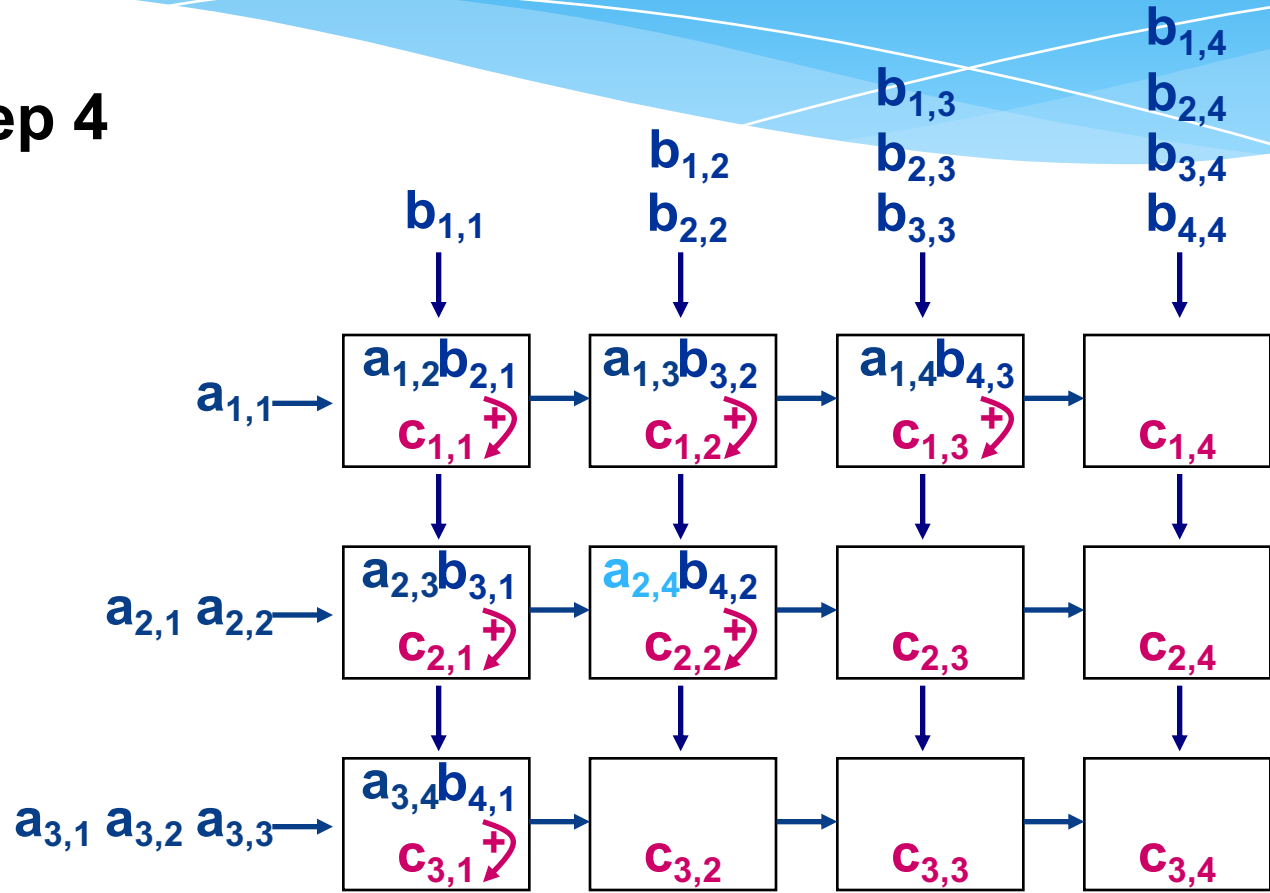
# Systolic 乘法

Step 3



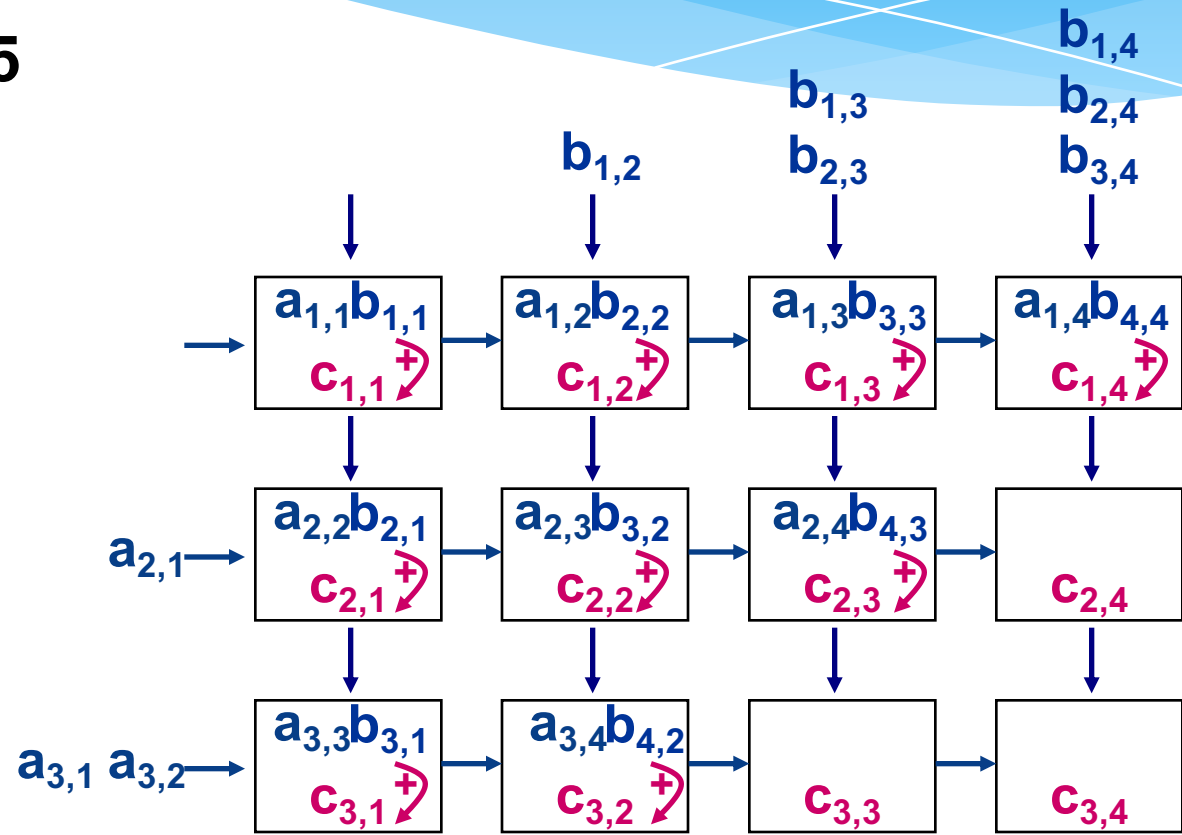
# Systolic 乘法

## Step 4



# Systolic 乘法

## Step 5

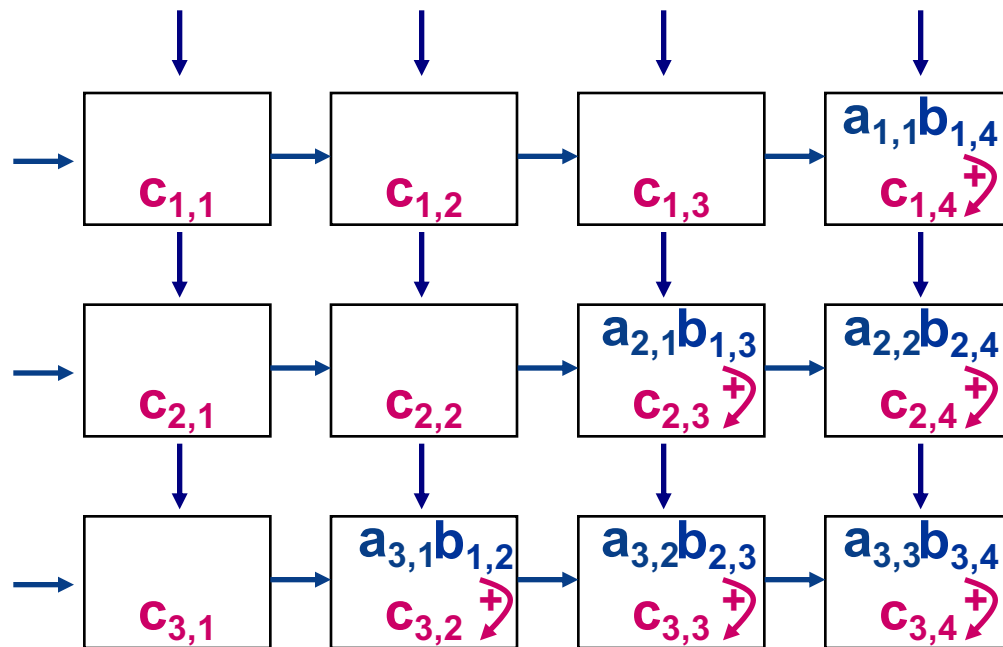






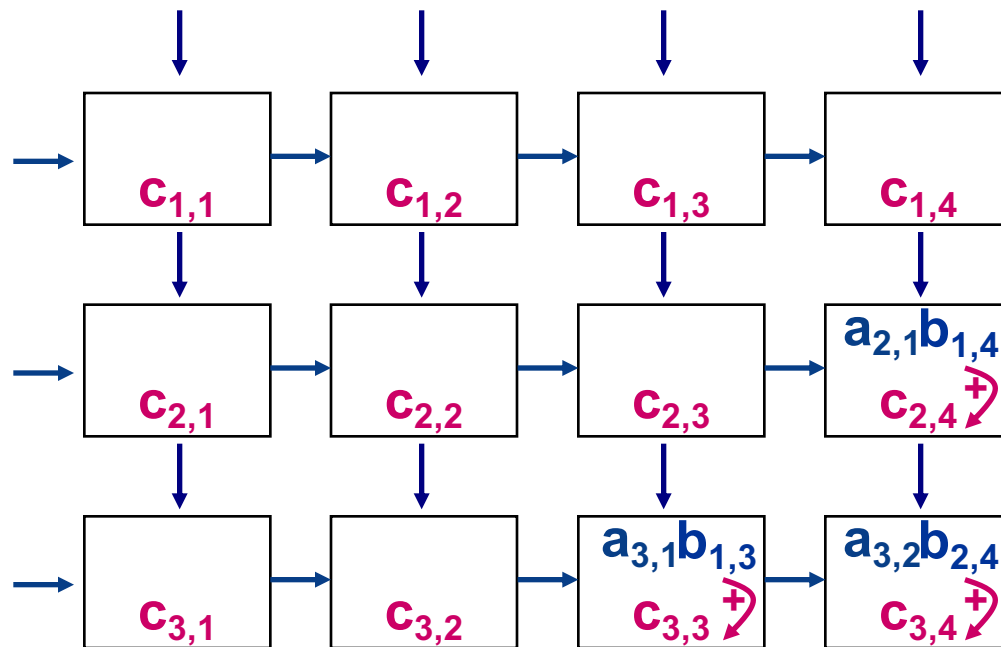
# Systolic 乘法

## Step 8



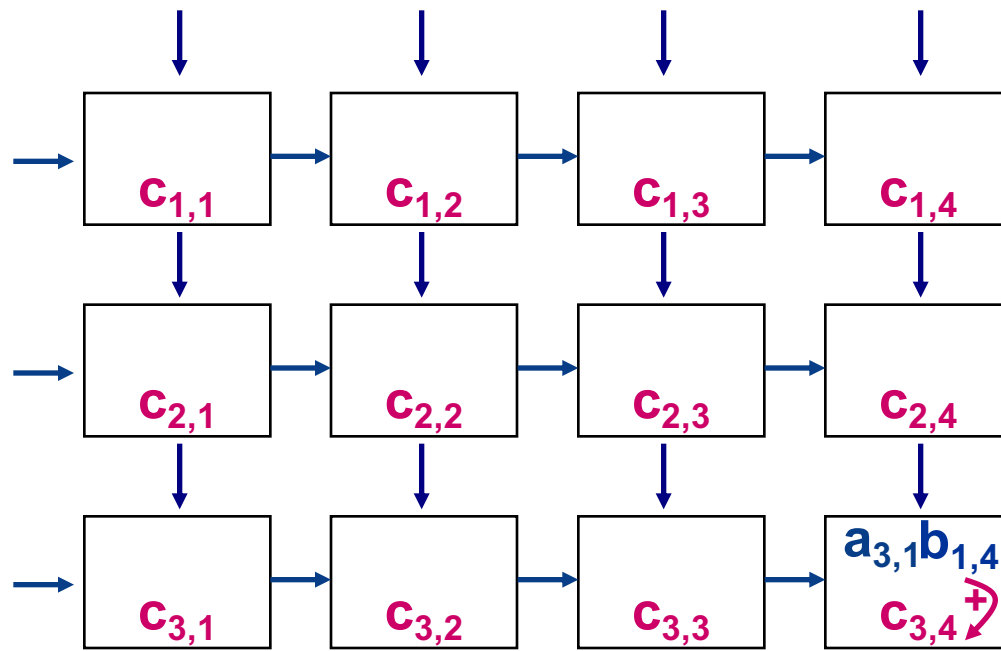
# Systolic 乘法

## Step 9



# Systolic 乘法

## Step 10



# Systolic 乘法

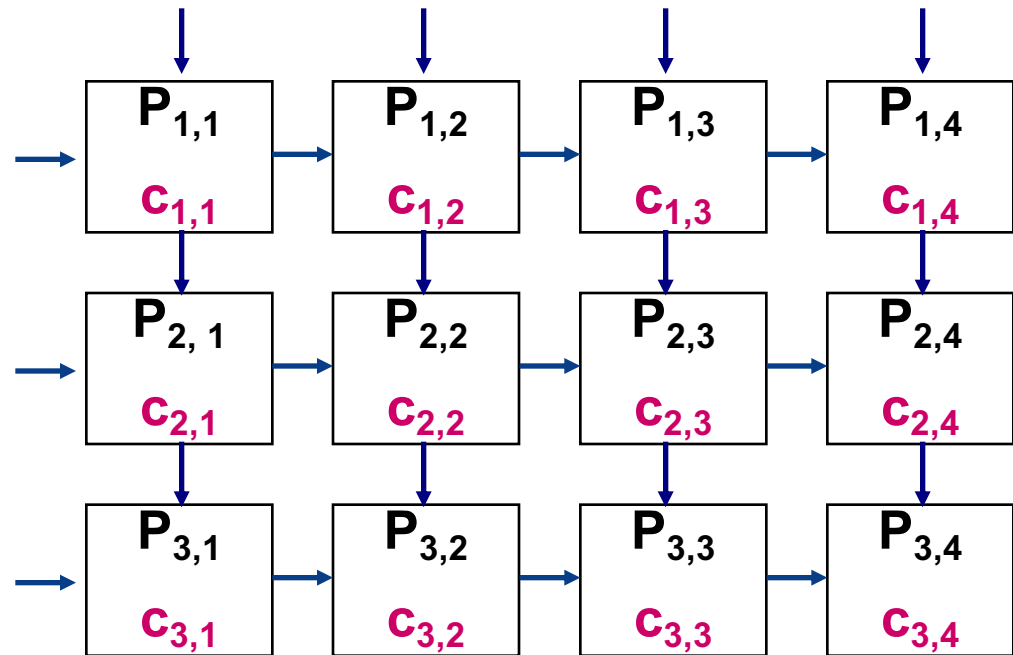
$$c_{1,1} = a_{1,1} b_{1,1} + a_{1,2} b_{2,1} + a_{1,3} b_{3,1} + a_{1,4} b_{4,1}$$

$$c_{1,2} = a_{1,1} b_{1,2} + a_{1,2} b_{2,2} + a_{1,3} b_{3,2} + a_{1,4} b_{4,2}$$

.....

$$c_{3,4} = a_{3,1} b_{1,4} + a_{3,2} b_{2,4} + a_{3,3} b_{3,4} + a_{3,4} b_{4,4}$$

Over



# Systolic 乘法

## ■ Systolic 算法

// 输入:  $A_{m \times n}$ ,  $B_{n \times k}$ ; 输出:  $C_{m \times k}$

Begin

for  $i=1$  to  $m$  par-do

for  $j=1$  to  $k$  par-do

(i)  $c_{i,j} = 0$

(ii) while  $P_{i,j}$  收到  $a$  和  $b$  时 do

$c_{i,j} = c_{i,j} + ab$

if  $i < m$  then 发送  $b$  给  $P_{i+1,j}$  endif

if  $j < k$  then 发送  $a$  给  $P_{i,j+1}$  endif

endwhile

endfor

endfor

End

## 9.4 矩阵乘法

9.4.1 简单并行分块乘法

9.4.2 Cannon乘法

9.4.3 Fox乘法

9.4.4 Systolic乘法

9.4.5 DNS乘法

# DNS乘法

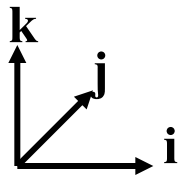
- \* **背景:** 由Dekel、Nassimi和Sahni提出的SIMD-CC上的矩阵乘法, 处理器数目为 $n^3$ , 运行时间为 $O(\log n)$ , 是一种速度很快的算法。
- \* **基本思想:** 通过一到一和一到多的播送办法, 使得处理器 $(k,i,j)$ 拥有 $a_{i,k}$ 和 $b_{k,j}$ , 进行本地相乘, 再沿 $k$ 方向进行单点积累求和, 结果存储在处理器 $(0,i,j)$ 中。
- \* **处理器编号:** 处理器数 $p=n^3=(2^q)^3=2^{3q}$ , 处理器 $P_r$ 位于位置 $(k,i,j)$ , 这里 $r=kn^2+in+j$ , ( $0 \leq i, j, k \leq n-1$ )。位于 $(k,i,j)$ 的处理器 $P_r$ 的三个寄存器 $A_r, B_r, C_r$ 分别表示为 $A[k,i,j]$ ,  $B[k,i,j]$ 和 $C[k,i,j]$ , 初始时均为0。
- \* **算法:** 初始时 $a_{i,j}$ 和 $b_{i,j}$ 存储于寄存器 $A[0,i,j]$ 和 $B[0,i,j]$ ;
  - ① **数据复制:**  $A, B$ 同时在 $k$ 维复制(一到一播送);  
 $A$ 在 $j$ 维复制(一到多播送);  $B$ 在 $i$ 维复制(一到多播送);
  - ② **相乘运算:** 所有处理器的 $A, B$ 寄存器两两相乘;
  - ③ **求和运算:** 沿 $k$ 方向进行单点积累求和;

\* 示例

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$B = \begin{pmatrix} -5 & -6 \\ 7 & 8 \end{pmatrix}$$

求  $C = A \times B$



$C_{00} = 1 \times (-5) + 2 \times 7 = 9$   
 $C_{01} = 1 \times (-6) + 2 \times 8 = 10$   
 $C_{10} = 3 \times (-5) + 4 \times 7 = 13$   
 $C_{11} = 3 \times (-6) + 4 \times 8 = 14$

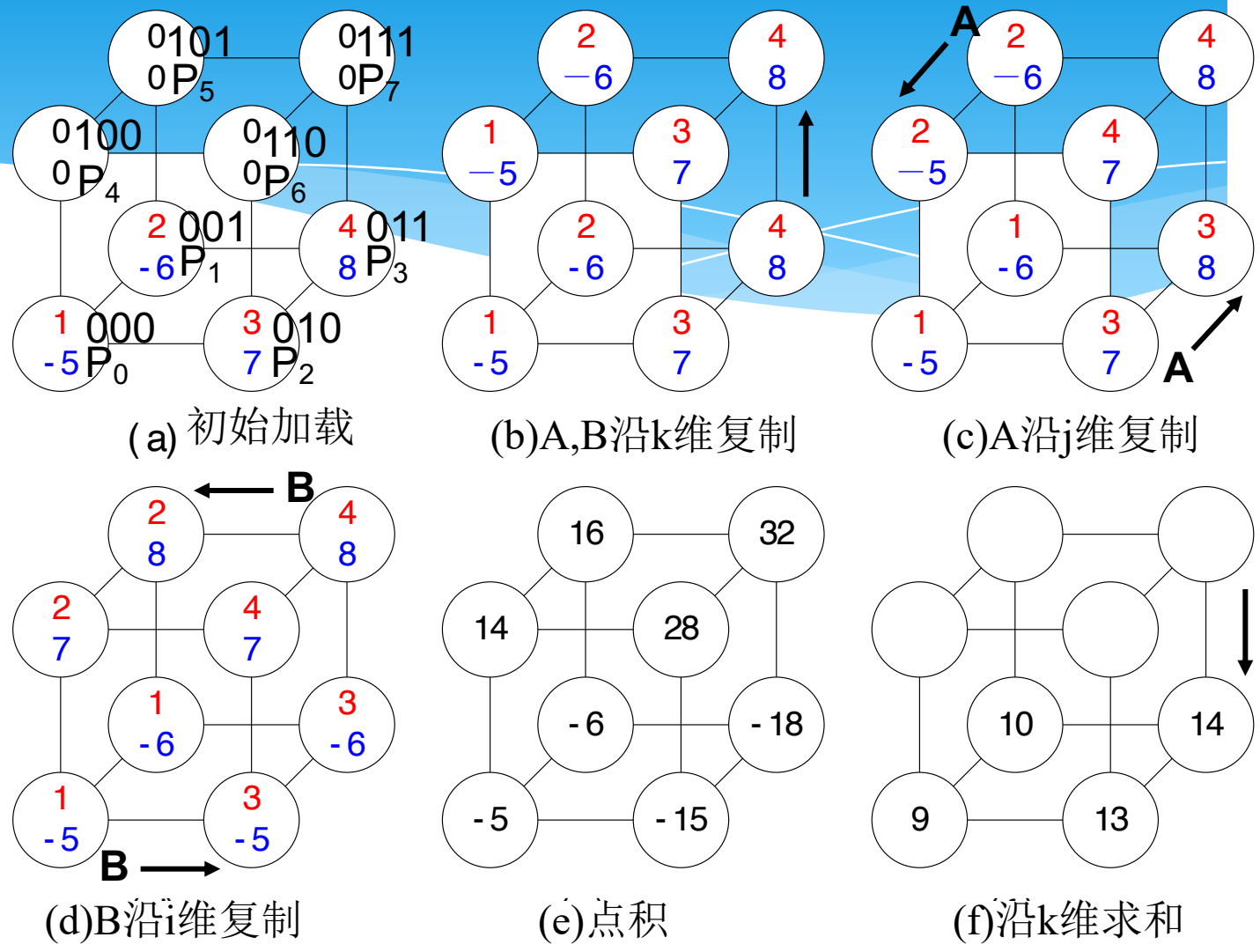


图9.12

# DNS乘法

## \* 算法描述:

//令 $r^{(m)}$ 表示 $r$ 的第 $m$ 位取反;

// $\{p, r_m=d\}$ 表示 $r(0 \leq r \leq p-1)$ 的集合, 这里 $r$ 的二

//进制第 $m$ 位为 $d$ ;

//输入:  $A_{n \times n}, B_{n \times n}$ ; 输出:  $C_{n \times n}$

Begin //以 $n=2, p=8=2^3$ 举例,  $q=1, r=(r_2 r_1 r_0)_2$

(1)for  $m=3q-1$  to  $2q$  do //按 $k$ 维复制 $A, B, m=2$

for all  $r$  in  $\{p, r_m=0\}$  par-do // $r_2=0$ 的 $r$

(1.1)  $A_{r^{(m)}} \leftarrow A_r$  // $A(100) \leftarrow A(000)$ 等

(1.2)  $B_{r^{(m)}} \leftarrow B_r$  // $B(100) \leftarrow B(000)$ 等

endfor

endfor

(2)for  $m=q-1$  to  $0$  do //按 $j$ 维复制 $A, m=0$

for all  $r$  in  $\{p, r_m=r_{2q+m}\}$  par-do // $r_0=r_2$ 的 $r$

$A_{r^{(m)}} \leftarrow A_r$  // $A(001) \leftarrow A(000), A(100) \leftarrow A(101)$

endfor // $A(011) \leftarrow A(010), A(110) \leftarrow A(111)$

endfor

(3)for  $m=2q-1$  to  $q$  do //按 $i$ 维复制 $B, m=1$

for all  $r$  in  $\{p, r_m=r_{q+m}\}$  par-do // $r_1=r_2$ 的 $r$

$B_{r^{(m)}} \leftarrow B_r$  // $B(010) \leftarrow B(000), B(100) \leftarrow B(110)$

endfor // $B(011) \leftarrow B(001), B(101) \leftarrow B(111)$

endfor

(4)for  $r=0$  to  $p-1$  par-do //相乘, all  $P_r$

$C_r = A_r \times B_r$

endfor

(5)for  $m=2q$  to  $3q-1$  do //求和,  $m=2$

for  $r=0$  to  $p-1$  par-do

$C_r = C_r + C_{r^{(m)}}$

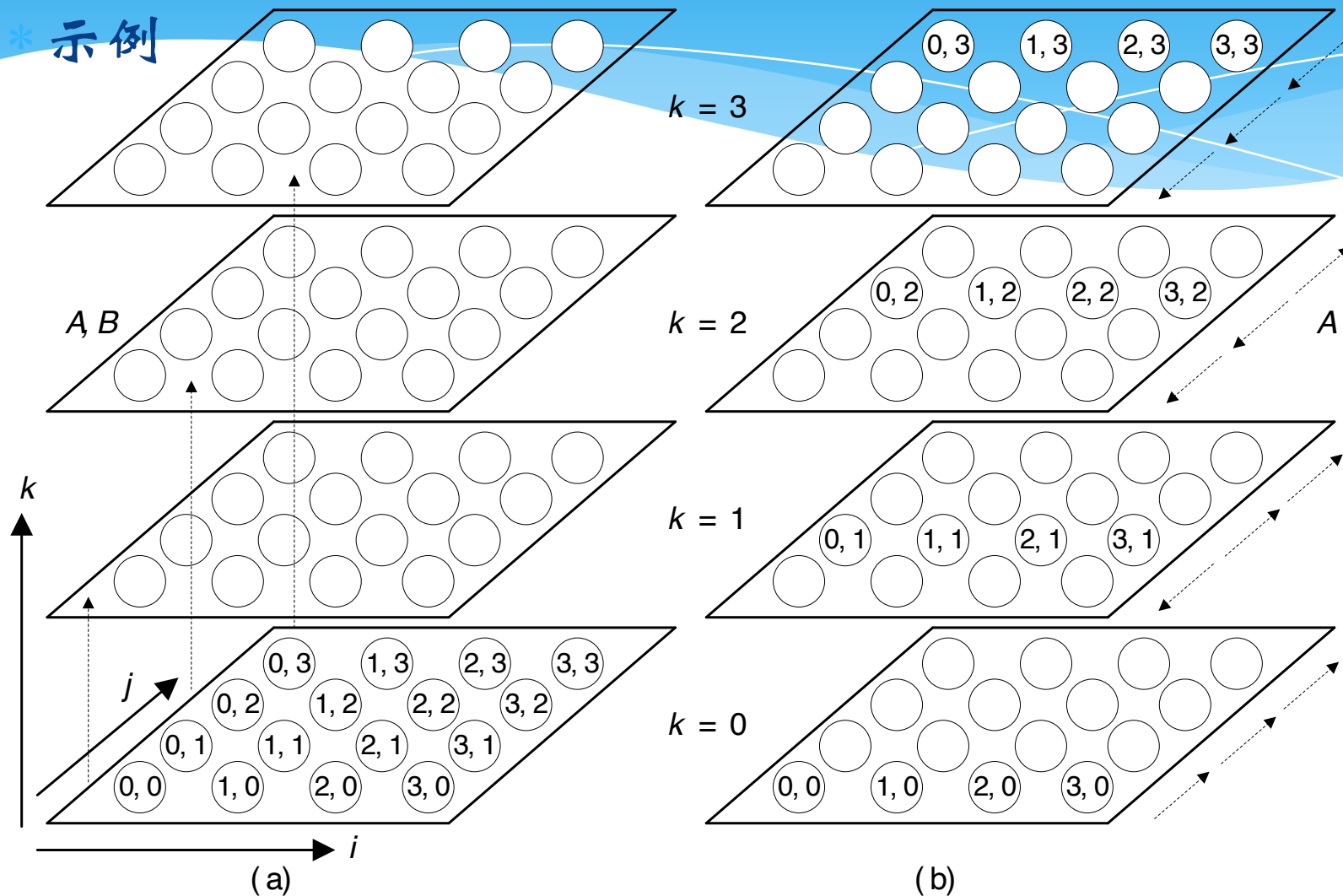
endfor

endfor

End

# DNS乘法

\* 示例



# DNS乘法

