



第七章

并行算法及性能分析

哈尔滨工业大学

张伟哲

2025, Fall Semester

目录

- 通信开销模型
- 不同拓扑的通信开销
- 矩阵-向量相乘

目录

- 通信开销模型
- 不同拓扑的通信开销
- 矩阵-向量相乘

通信开销

- 在并行程序中，除了空闲 (idling) 和争用 (contention) 之外，通信 (communication) 也是主要的开销
- 通信成本取决于各种特征，包括编程模型语义 (programming model semantics)、网络拓扑 (network topology)、数据处理 (data handling) 和路由 (routing) 以及相关的软件协议 (software protocols)

消息传递开销

- 通过网络传输信息的总时间包含以下时间：
- 启动时间 (Startup time) t_s : 在发送和接收节点所花费的时间 (执行路由算法、编程路由器等)
- 每跳时间 (Per-hop time) t_h : 跳数的函数, 包括交换机延迟、网络延迟等因素
- 每字传输时间 (Per-word transfer time) t_w : 由消息长度决定的所有开销, 包括链路带宽、错误检查和纠正等

存储转发路由

- **存储转发路由 (Store-and-Forward Routing)**
- 信息被发送到一个中间站, 被保存起来, 并在以后发送到最终目的地或另一个中间站
- 一条大小为 m 个字的消息通过 l 个通信链路 (communication links) 的总通信成本为

$$t_{comm} = t_s + (mt_w + t_h)l.$$

- 在大多数平台上, t_h 很小, 上面的表达式可以近似为

$$t_{comm} = t_s + mlt_w.$$

直通路由

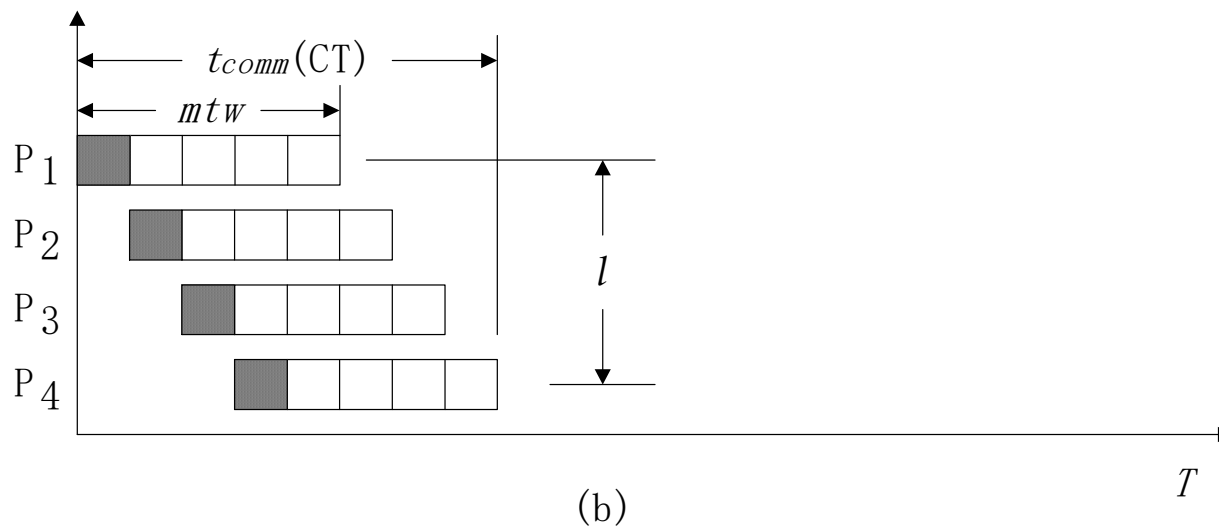
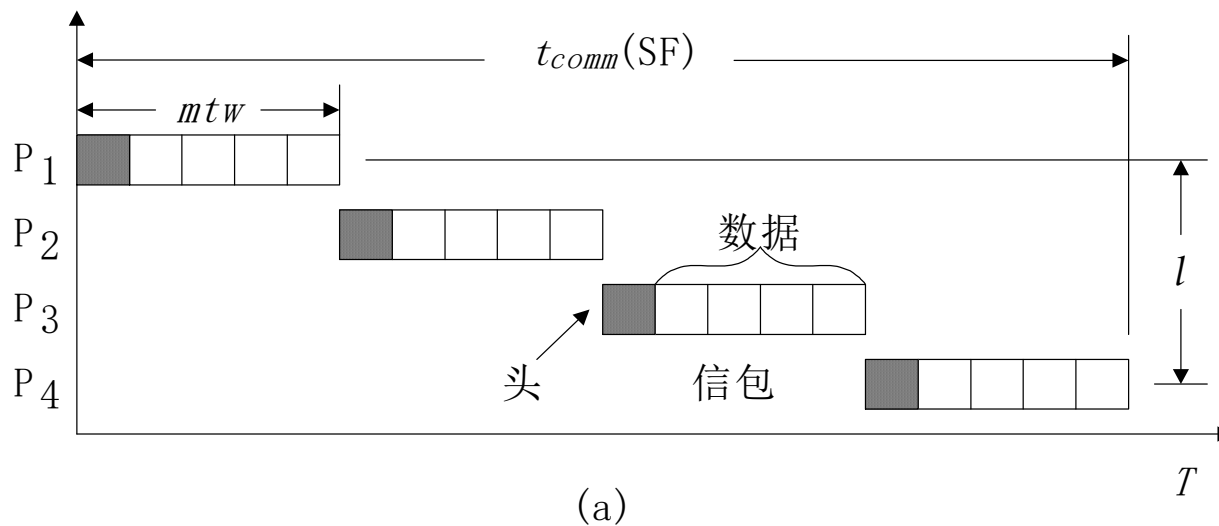
- **直通路由** (Cut-Through Routing)
- 在传递一个消息之前，就为它建立一条从源结点到目的结点的物理通道。在传递的全部过程中，线路的每一段都被占用，当消息的尾部经过网络后，整条物理链路才被废弃
- 总通信时间近似为：

$$t_{comm} = t_s + t_h l + t_w m.$$

- t_h 通常远小于 t_s 和 t_w 的，总通信时间进一步简化

$$t_{comm} = t_s + t_w m.$$

SF和CT模式的时空图



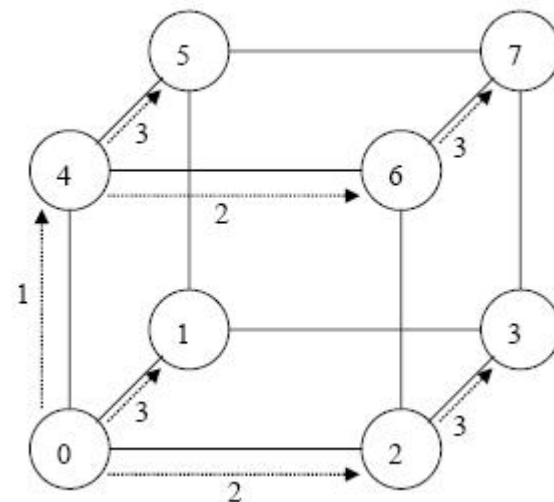
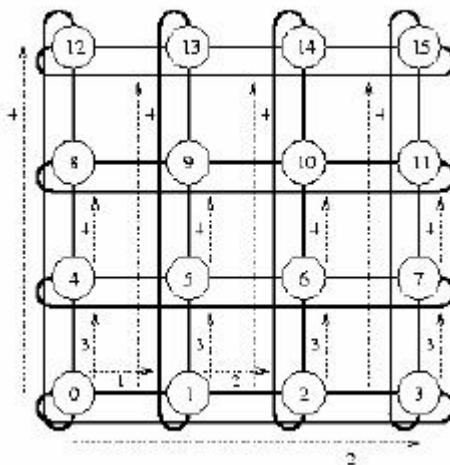
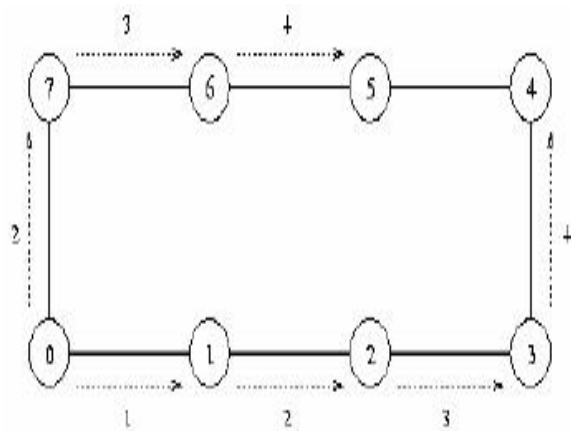
目录

- 通信开销模型
- **不同拓扑的通信开销**
- 矩阵-向量相乘

点对点通信

■ 距离 l 的计算： 对于 p 个处理器

- 一维环形： $l \leq \lfloor p/2 \rfloor$
- 带环绕Mesh： $l \leq 2 \lfloor \sqrt{p}/2 \rfloor$
- 超立方： $l \leq \log p$



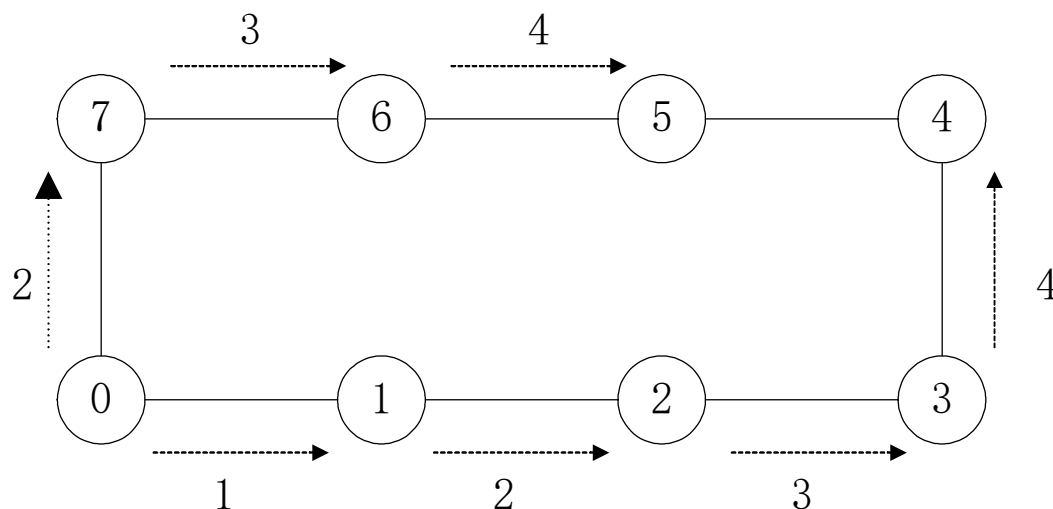
点对点通信

■ 两个网络处理器之间的数据通信

拓扑	存储转发路由 Store-and-Forward Routing	直通路由 Cutting-Through
环 (Ring)	$t_s + mt_w \lfloor p/2 \rfloor$	$t_s + mt_w$
网格 (Grid —torus)	$t_s + 2mt_w \lfloor \sqrt{p}/2 \rfloor$	$t_s + mt_w$
超立方 (Hypercube)	$t_s + mt_w \log_2 p$	$t_s + mt_w$

One-to-All Broadcast

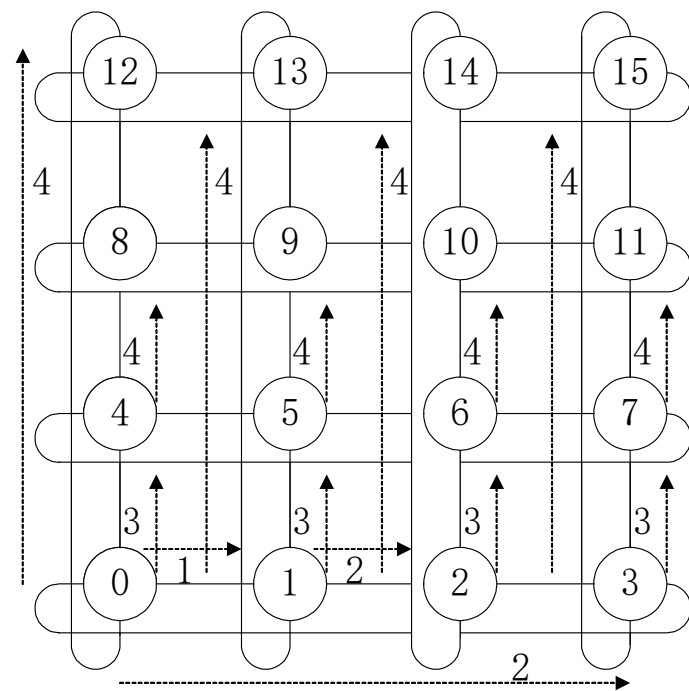
- 一到全广播，SF模式，环
- 步骤：①先左右邻近传送;②再左右二个方向同时播送
- 示例：



- 通信时间: $t_{one-to-all}(SF) = (t_s + mt_w) \lceil p/2 \rceil$

One-to-All Broadcast

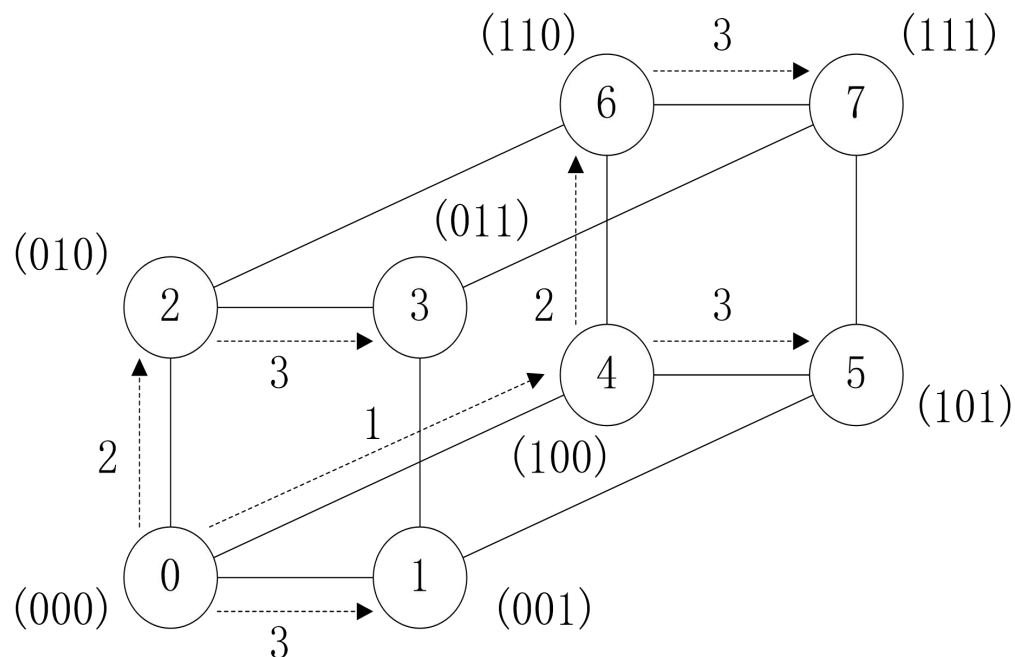
- 一到全广播，SF模式，环绕网孔
- 步骤：①先完成一行中的播送;②再同时进行各列的播送
- 示例：共4步(2步行、2步列)



- 通信时间: $t_{one-to-all}(SF) = 2(t_s + mt_w) \left\lceil \frac{\sqrt{p}}{2} \right\rceil$

One-to-All Broadcast

- 一到全广播，SF模式，超立方
- 步骤：从低维到高维，依次进行播送；
- 示例：



- 通信时间： $t_{one-to-all}(SF) = (t_s + mt_w) \log p$

One-to-All Broadcast

■ 一到全广播，CT模式，环

■ 步骤：

- (1) 先发送至 $p/2$ 远的处理器；
- (2) 再同时发送至 $p/2^2$ 远的处理器；
-
- (i) 再同时发送至 $p/2^i$ 远的处理器；

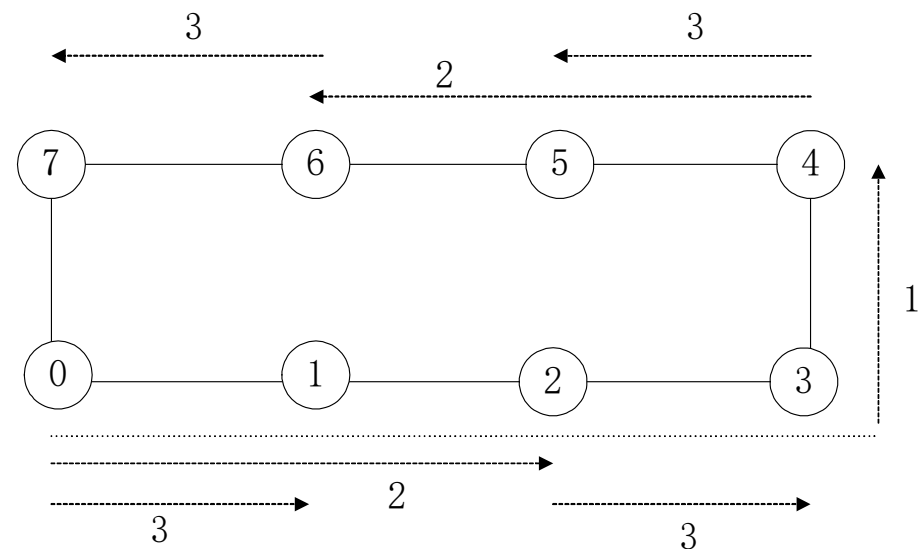
■ 通讯时间：

$$t_{one-to-all}(CT) = \sum_{i=1}^{\log p} (t_s + mt_w + t_h p / 2^i)$$

$$= t_s \log p + mt_w \log p + t_h (p - 1)$$

$$\approx (t_s + mt_w) \log p \quad (t_h \text{可忽略时})$$

并行计算



One-to-All Broadcast

■ 一到全广播，CT模式，网孔

■ 步骤：

➤ (1) 先进行行播送；

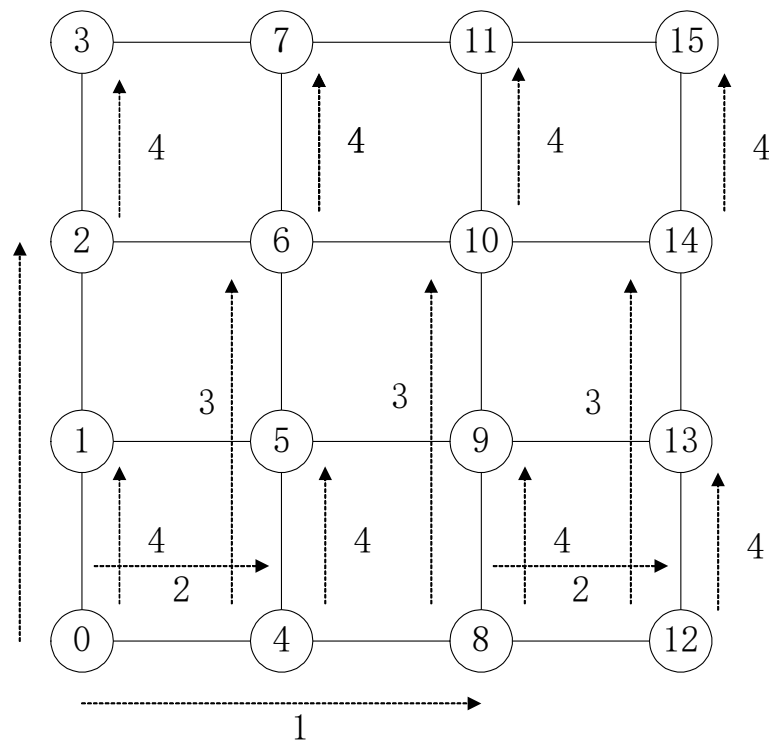
$$// t_s \log \sqrt{p} + mt_w \log \sqrt{p} + t_h(\sqrt{p} - 1)$$

➤ (2) 再同时进行列播送；

$$// t_s \log \sqrt{p} + mt_w \log \sqrt{p} + t_h(\sqrt{p} - 1)$$

■ 通信时间：

$$\begin{aligned} t_{one-to-all}(CT) &= 2(t_s \log \sqrt{p} + mt_w \log \sqrt{p} + t_h(\sqrt{p} - 1)) \\ &= \boxed{(t_s + mt_w) \log p} + 2t_h(\sqrt{p} - 1) \end{aligned}$$



One-to-All Broadcast

- 一到全广播, CT模式, 超立方
- 步骤: 依次从低维到高维播送, d-立方, $d=0,1,2,3,4\dots$;
- 通信时间:

$$t_{one-to-all}(CT) = (t_s + mt_w) \log p$$

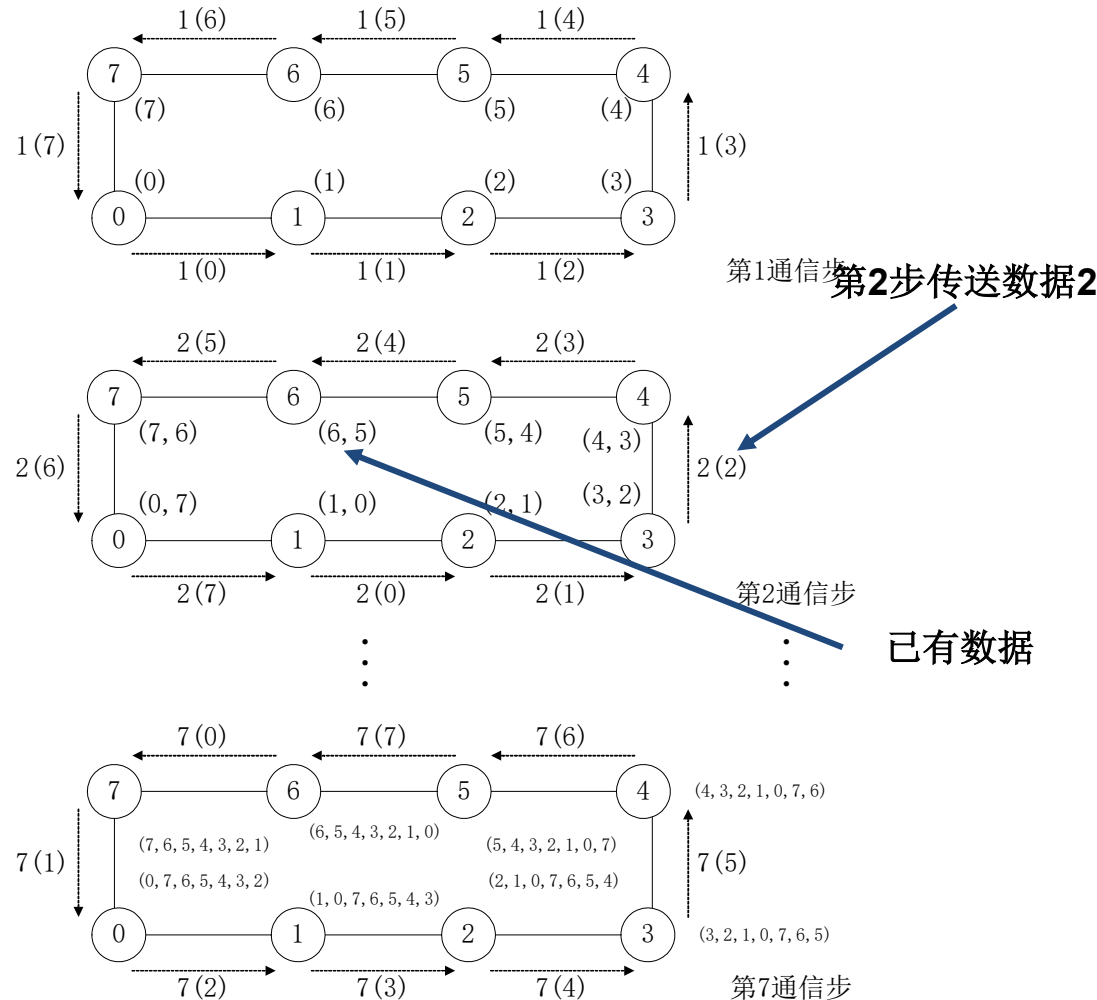
All-to-All Broadcast

- 全到全广播，SF模式，环

- 步骤：同时向右(或左)播送刚接收到的信包

- 通信时间：

$$t_{all-to-all}(SF) = (t_s + mt_w)(p-1)$$



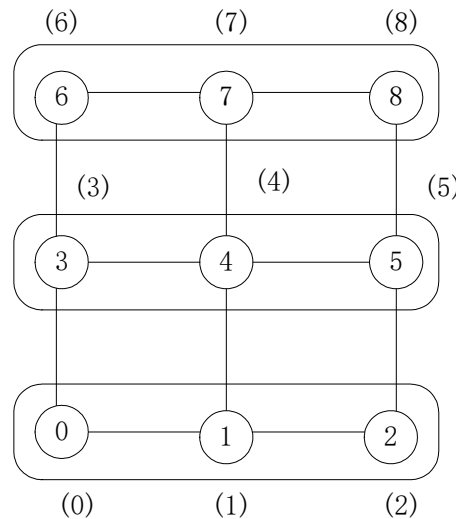
All-to-All Broadcast

■ 全到全广播，SF模式，环绕网孔

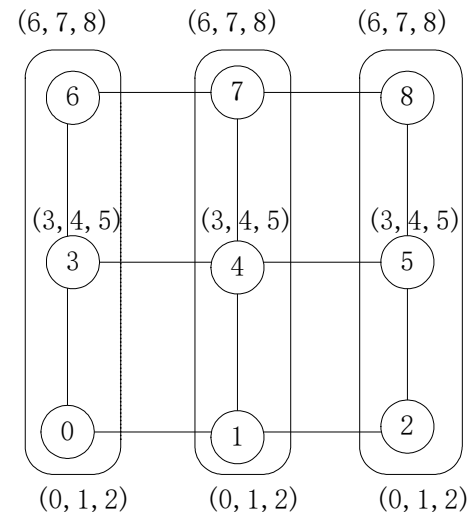
■ 步骤：

- (1) 先进行行的播送；
- (2) 再进行列的播送；

■ 通信时间：



(a)

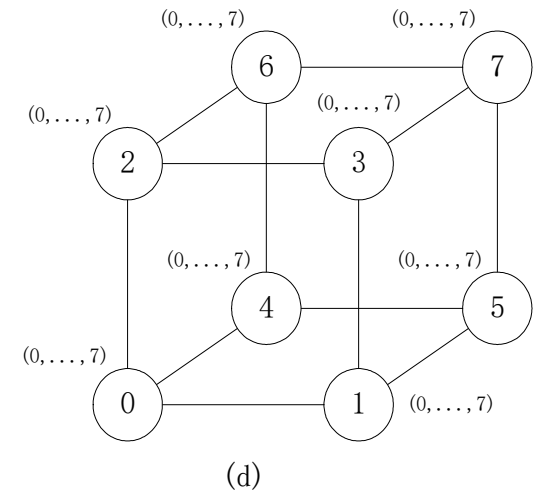
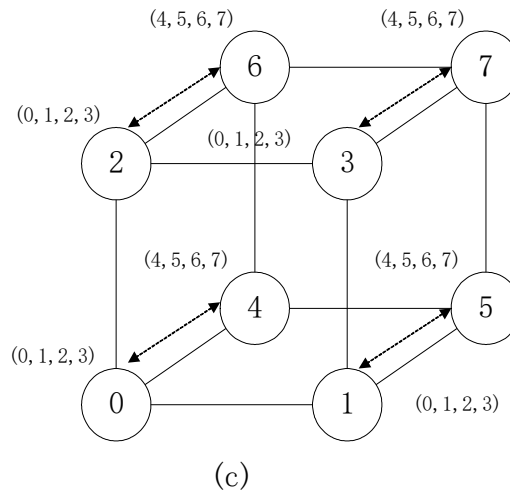
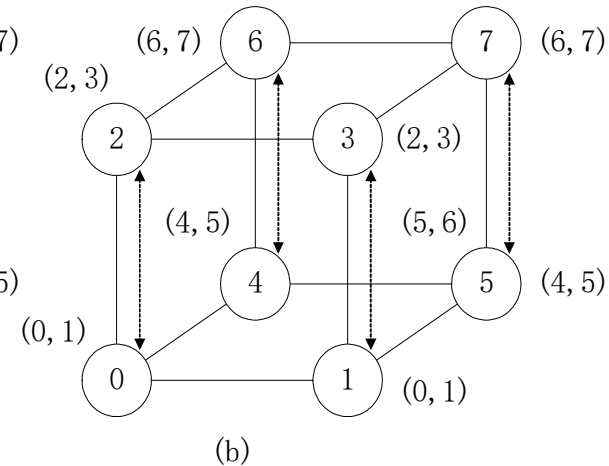
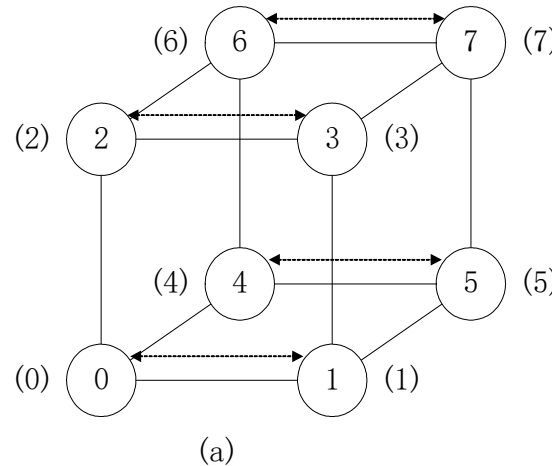


(b)

$$\begin{aligned}
 t_{all-to-all}(SF) &= (t_s + mt_w)(\sqrt{p} - 1) + (t_s + m\sqrt{p} \cdot t_w)(\sqrt{p} - 1) \\
 &= 2t_s(\sqrt{p} - 1) + mt_w(p - 1)
 \end{aligned}$$

All-to-All Broadcast

- **全到全广播, SF 模式, 超立方体**
- **步骤: 依次按维进行, 多到多的播送;**



- **通信时间:**

$$t_{all-to-all}(SF) = \sum_{i=1}^{\log p} (t_s + 2^{i-1} m t_w)$$

$$= t_s \log p + m t_w (p-1)$$

并行计算

All-to-All Broadcast

- 全到全广播，CT模式

$$t_{all-to-all}(CT) = t_{all-to-all}(SF)$$

目录

- 通信开销模型
- 不同拓扑的通信开销
- 矩阵-向量相乘

矩阵的划分——带状划分

■ 16×16 阶矩阵, $p=4$

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

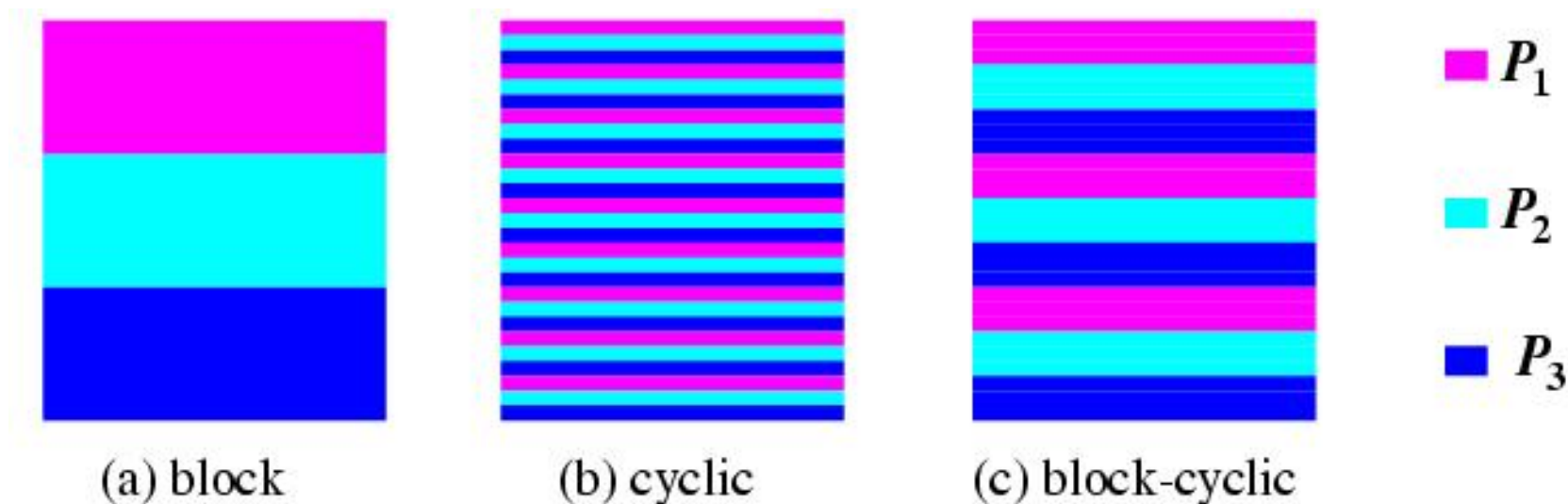
(a)

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

(b)

矩阵的划分——带状划分

■ 示例： $p = 3$, 27×27 矩阵的3种带状划分



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

矩阵的划分——棋盘划分

■ 8×8 阶矩阵, $p=16$

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)	(0, 7)
P_0		P_1		P_2		P_3	
(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)
P_4		P_5		P_6		P_7	
(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)
P_8		P_9		P_{10}		P_{11}	
(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)
P_{12}		P_{13}		P_{14}		P_{15}	
(7, 0)	(7, 1)	(7, 2)	(7, 3)	(7, 4)	(7, 5)	(7, 6)	(7, 7)

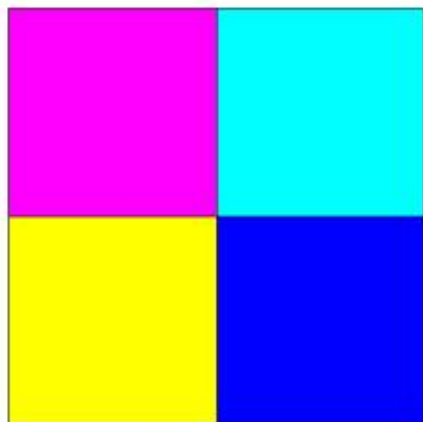
(a)

(0, 0)	(0, 4)	(0, 1)	(0, 5)	(0, 2)	(0, 6)	(0, 3)	(0, 7)
P_0		P_1		P_2		P_3	
(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)
P_4		P_5		P_6		P_7	
(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)
P_8		P_9		P_{10}		P_{11}	
(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)
P_{12}		P_{13}		P_{14}		P_{15}	
(7, 0)	(7, 4)	(7, 1)	(7, 5)	(7, 2)	(7, 6)	(7, 3)	(7, 7)

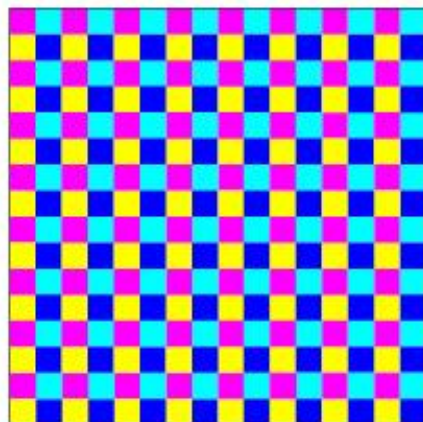
(b)

矩阵的划分——棋盘划分

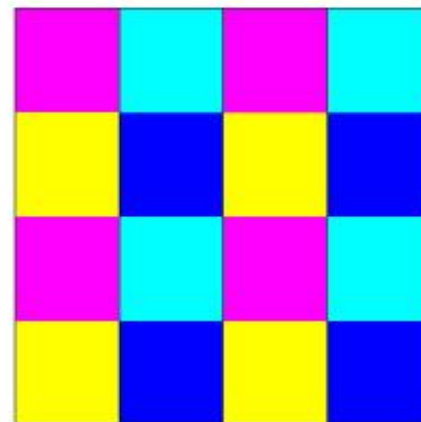
■ 示例： $p = 4$, 16×16 矩阵的3种棋盘划分



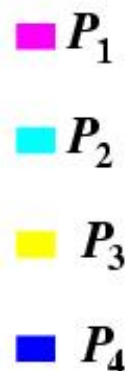
(a) block



(b) cyclic



(c) block cyclic



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

矩阵-向量相乘

■ 矩阵-向量相乘 (Matrix-Vector Multiplication)

$$c = A \cdot b$$



$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} a_{0,0}, a_{0,1}, \dots, a_{0,n-1} \\ \vdots \\ a_{m-1,0}, a_{m-1,1}, \dots, a_{m-1,n-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

矩阵-向量乘法可以转化m个A矩阵行向量和列向量b的内积

$$c_i = (a_i, b) = \sum_{j=0}^{n-1} a_{ij} b_j \quad 0 \leq i < m$$

矩阵-向量相乘

- 若 $m=n$ ，将 $n \times n$ 矩阵 A 与 $n \times 1$ 向量相乘，得到 $n \times 1$ 的结果向量 y
- 串行算法需要 n^2 次乘法和加法运算

$$W = n^2.$$

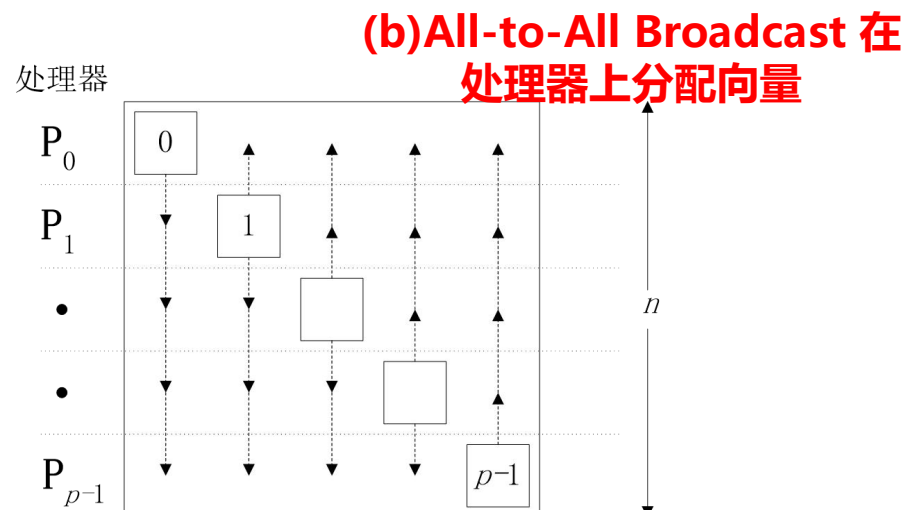
矩阵-向量相乘：带状划分

- $n \times n$ 矩阵被分布到 p 个进程上，每个进程存储矩阵的完整行
- $n \times 1$ 列向量也被分布到 p 个进程上，每个进程上存储列向量的部分元素 (n/p 个元素)

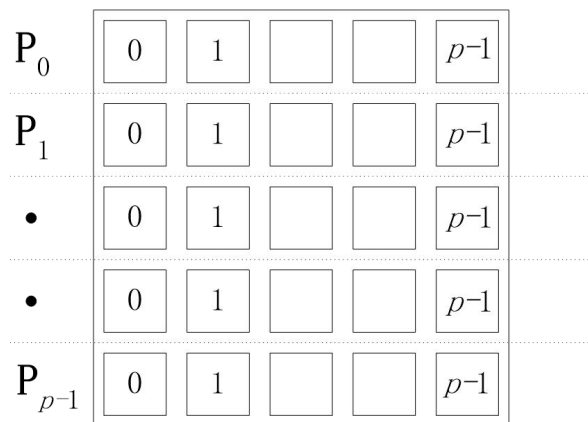
矩阵-向量相乘：带状划分



(a) 矩阵A和列向量x的初始化划分



(b)



(c) 广播通讯之后，每个进程分配完整向量



(d) 矩阵和结果向量y的最终分布情况

矩阵-向量相乘：带状划分

- 每进程初始时拥有向量x的部分元素（n/p个元素），之后使用All-to-All广播通信将所有元素分布到所有进程
- 则进程 P_i 计算

$$y[i] = \sum_{j=0}^{n-1} (A[i, j] \times x[j])$$

矩阵-向量相乘：带状划分

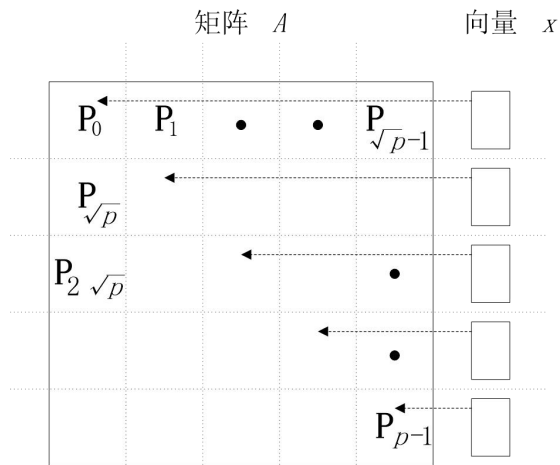
- 现在考虑 $P < n$ 的情况
- 每个进程最初存储 n/p 矩阵的完整行和大小为 n/p 的向量的一部分
- 在所有进程上利用All-to-All广播通讯，所传递消息的大小为 n/p
- 然后进行 n/p 次的局部向量点积 (dot product)
- 超立方连接，并行运行时间为：

$$\begin{aligned} T_p &= \frac{n^2}{p} + t_s \log p + \frac{n}{p} t_w (p - 1) \\ &= \frac{n^2}{p} + t_s \log p + n t_w \end{aligned}$$

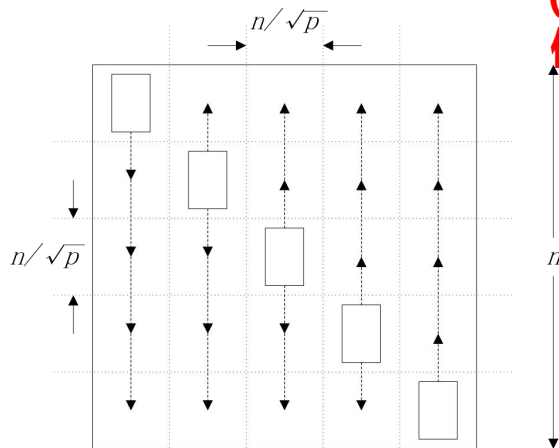
矩阵-向量相乘：棋盘划分

- $n \times n$ 矩阵在 $p = n^2$ 个进程分布，每个进程处理一个元素
- $n \times 1$ 向量只分布在最后一列的 n 个进程上

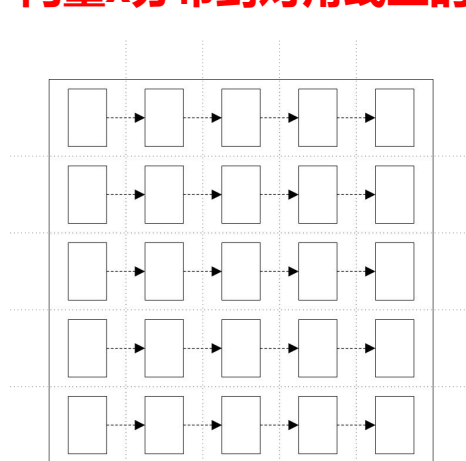
矩阵-向量相乘：棋盘划分



(a) 数据初始化分布及通信，将列向量 x 分布到对角线上的进程

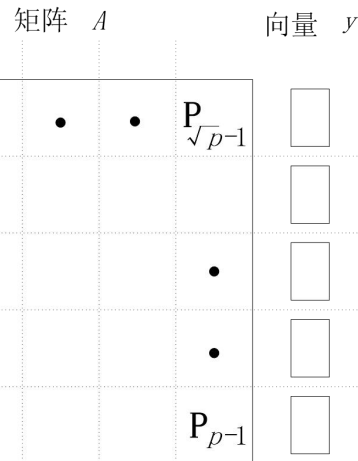


(b) one-to-all广播通信，将列向量 x 的部分在每个进程列内广播



(c) all-to-one规约通信，在每个进程行内规约得到部分结果

并行计算



(d) 计算结果的最终分布情况

矩阵-向量相乘：棋盘划分

- 当进程数量 $P < n^2$ 的时，每个进程处理的元素的个数为： $(n/\sqrt{p}) \times (n/\sqrt{p})$
- 列向量在进程列上分布，每个部分包含元素的个数为： n/\sqrt{p}
- 对齐操作、广播通信、规约通信的消息大小的元素数为： n/\sqrt{p}
- 计算操作是 $(n/\sqrt{p}) \times (n/\sqrt{p})$ 子矩阵和 n/\sqrt{p} 子向量的乘积

矩阵-向量相乘：棋盘划分

■ 对齐操作所需时间

$$t_s + t_w n / \sqrt{p}$$

■ 广播和规约通信所需时间

$$(t_s + t_w n / \sqrt{p}) \log(\sqrt{p})$$

■ 局部子矩阵-子向量乘法所需时间

$$t_c n^2 / p$$

■ 总运行时间

$$T_P \approx \frac{n^2}{p} + t_s \log p + t_w \frac{n}{\sqrt{p}} \log p$$

Reference

- 并行计算课程，陈国良，中国科学技术大学
- Introduction to Parallel Computing, Ananth Grama, Purdue University, https://www.cs.purdue.edu/homes/ayg/CS525_SPR17/chap2_slides.pdf
- Introduction to Parallel Computing, Ananth Grama, Purdue University, https://www.cs.purdue.edu/homes/ayg/CS525_SPR17/chap8_slides.pdf

附录：矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

- 考虑 $n \times n$ 稠密方阵乘法, 记作 $C = A \times B$
- 串行计算的复杂度的是 $O(n^3)$
- 分块操作: $n \times n$ 矩阵可以视作 $q \times q$ 个分块子矩阵的矩阵, 每块记作 A_{ij} ($0 \leq i, j < q$), 维度是 $(n/q) \times (n/q)$
- 需要执行 q^3 次矩阵乘法, 每个矩阵乘法都是两个 $(n/q) \times (n/q)$ 矩阵相乘

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

- 再考虑将 $n \times n$ 矩阵 A 、 B 划分为 p 块, A_{ij} and B_{ij} ($0 \leq i, j < \sqrt{p}$), 每块维度是 $(n/\sqrt{p}) \times (n/\sqrt{p})$
- 进程 P_{ij} 初始存储 A_{ij} 和 B_{ij} , 并计算结果子矩阵 C_{ij}
- 计算结果子矩阵 C_{ij} 需要所有子矩阵 A_{ik} 和 B_{kj} ($0 \leq k < \sqrt{p}$)
- 使用All-to-All广播通信, 在行方向广播 A 的块, 在列方向广播 B 的块
- 最后执行局部子矩阵乘法

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

- 两个广播通信所需时间:

$$2(t_s \log(\sqrt{p}) + t_w(n^2/p)(\sqrt{p} - 1))$$

- 需要 \sqrt{p} 次子矩阵惩罚, 子矩阵维度为 $(n/\sqrt{p}) \times (n/\sqrt{p})$
- 并行运行时间约为:

$$T_P = \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}.$$

- 这种算法的主要缺点是没有访存优化

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ Cannon算法

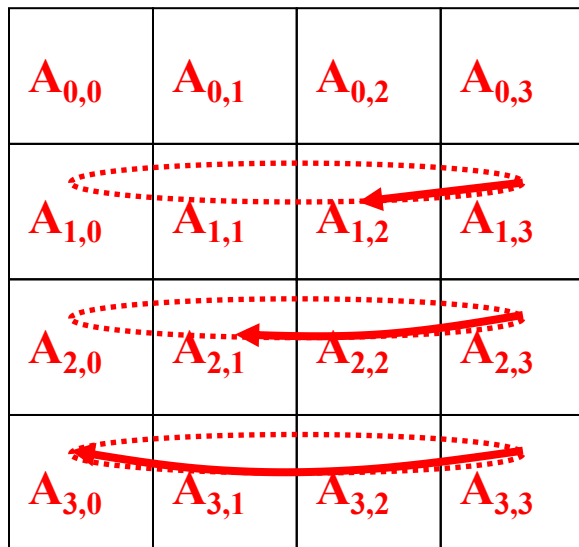
- 在该算法中，对第 i 行中的 \sqrt{p} 个进程进行调度，即在同一时间每个进程在使用不同的 $A_{i,k}$ 分块
- 在每次子矩阵乘法之后，这些分块子矩阵在进程之间循环移动，以使每个进程都可以得到一个新的 $A_{i,k}$

矩阵-矩阵相乘

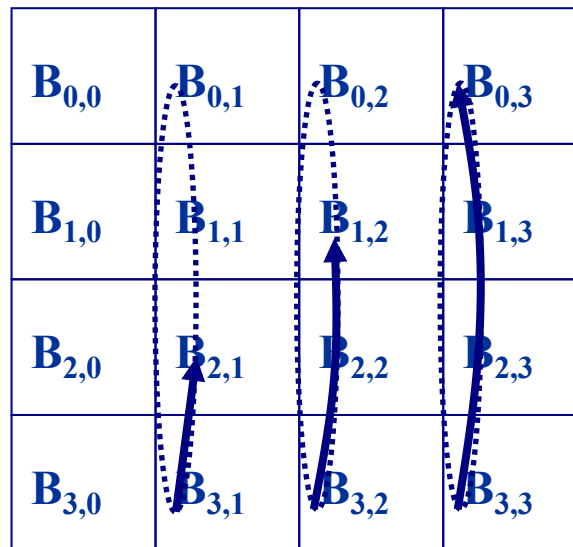
■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ Cannon算法

(a) A矩阵初始对齐



(b) B矩阵初始对齐操作

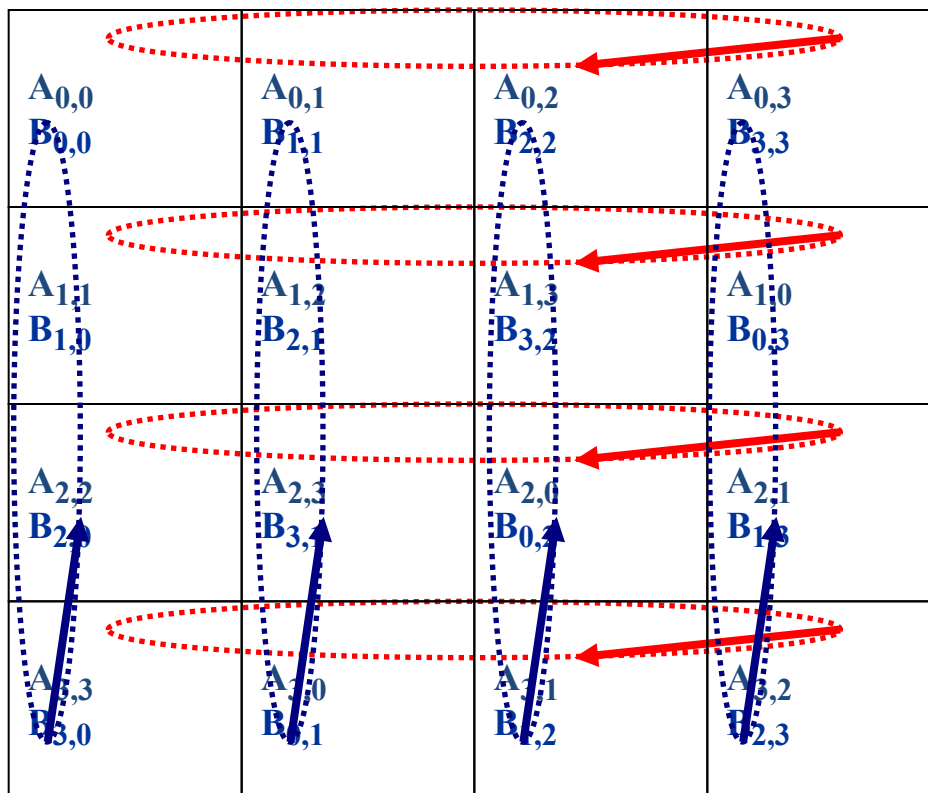


矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ Cannon算法

(c) 初始对齐操作之后的A、B矩阵

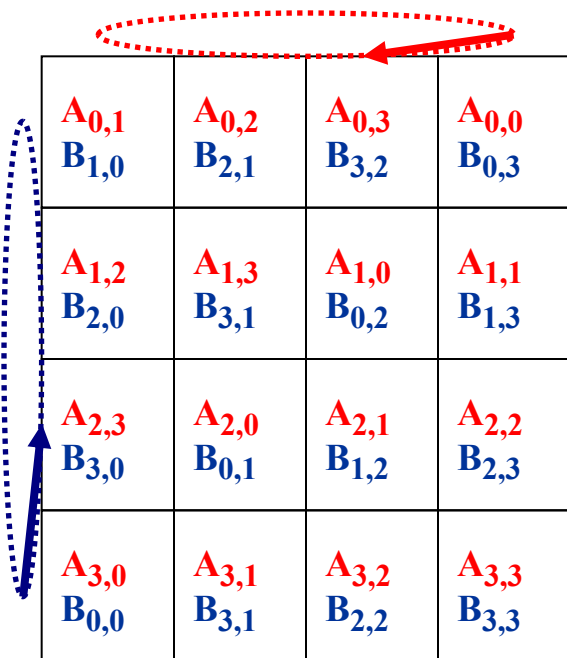


矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

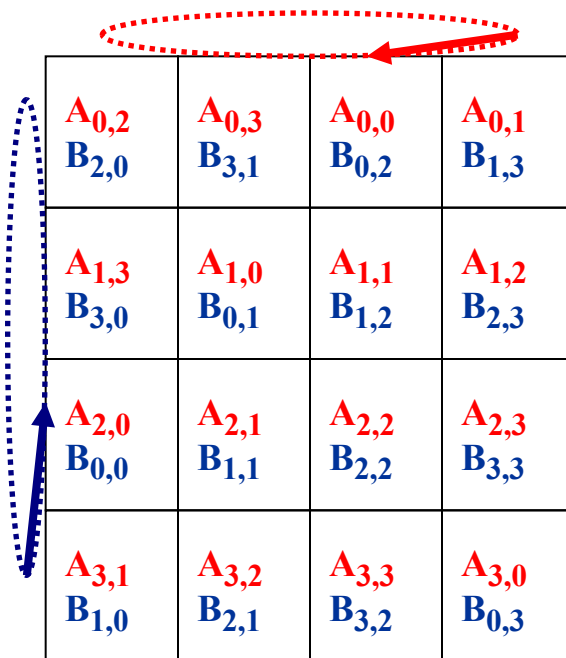
➤ Cannon算法

第一次循环移位后



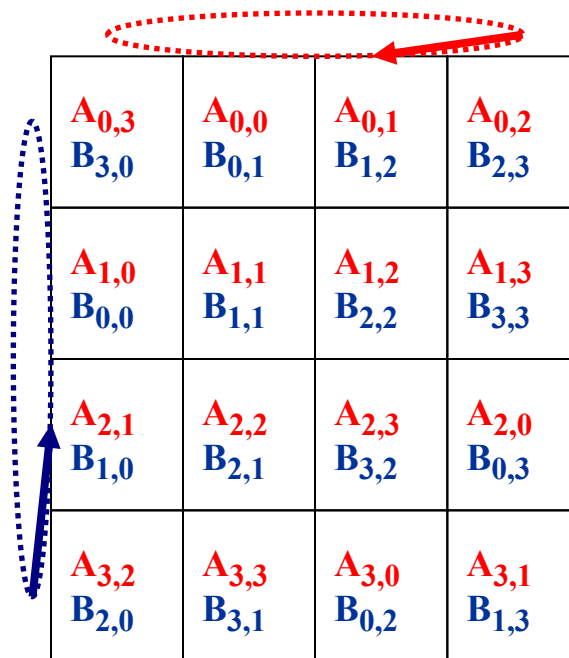
$A_{0,1}$ $B_{1,0}$	$A_{0,2}$ $B_{2,1}$	$A_{0,3}$ $B_{3,2}$	$A_{0,0}$ $B_{0,3}$
$A_{1,2}$ $B_{2,0}$	$A_{1,3}$ $B_{3,1}$	$A_{1,0}$ $B_{0,2}$	$A_{1,1}$ $B_{1,3}$
$A_{2,3}$ $B_{3,0}$	$A_{2,0}$ $B_{0,1}$	$A_{2,1}$ $B_{1,2}$	$A_{2,2}$ $B_{2,3}$
$A_{3,0}$ $B_{0,0}$	$A_{3,1}$ $B_{3,1}$	$A_{3,2}$ $B_{2,2}$	$A_{3,3}$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

第二次循环移位后



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

第三次循环移位后



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

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ Cannon算法

- 执行局部分块子矩阵乘法
- A 的分块向左循环移位, B 的分块向上循环移位
- 执行下一次局部分块子矩阵乘法, 并累加到部分结果; 重复上述步骤, 直到所有 \sqrt{p} 个分块都被计算

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ Cannon算法

- 每个循环移位操作所需时间是 $t_s + t_w n^2 / p$, 一共有 $2\sqrt{p}$ 次循环移位
- 计算 \sqrt{p} 个 $(n/\sqrt{p}) \times (n/\sqrt{p})$ 维矩阵乘法所需时间为 n^3 / p
- 并行运行时间约为:

$$T_P = \frac{n^3}{p} + 2\sqrt{p}t_s + 2t_w \frac{n^2}{\sqrt{p}}.$$

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ DNS算法

- 使用三维划分 (3-D partitioning)
- 将矩阵乘法视作立方体, 矩阵 A 和 B 来自两个正交面, 结果 C 来自另一个正交面
- 立方体中的每个内部节点代表一个加乘 (Add-Multiply) 运算
- DNS算法使用三维分块方案来划分立方体

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

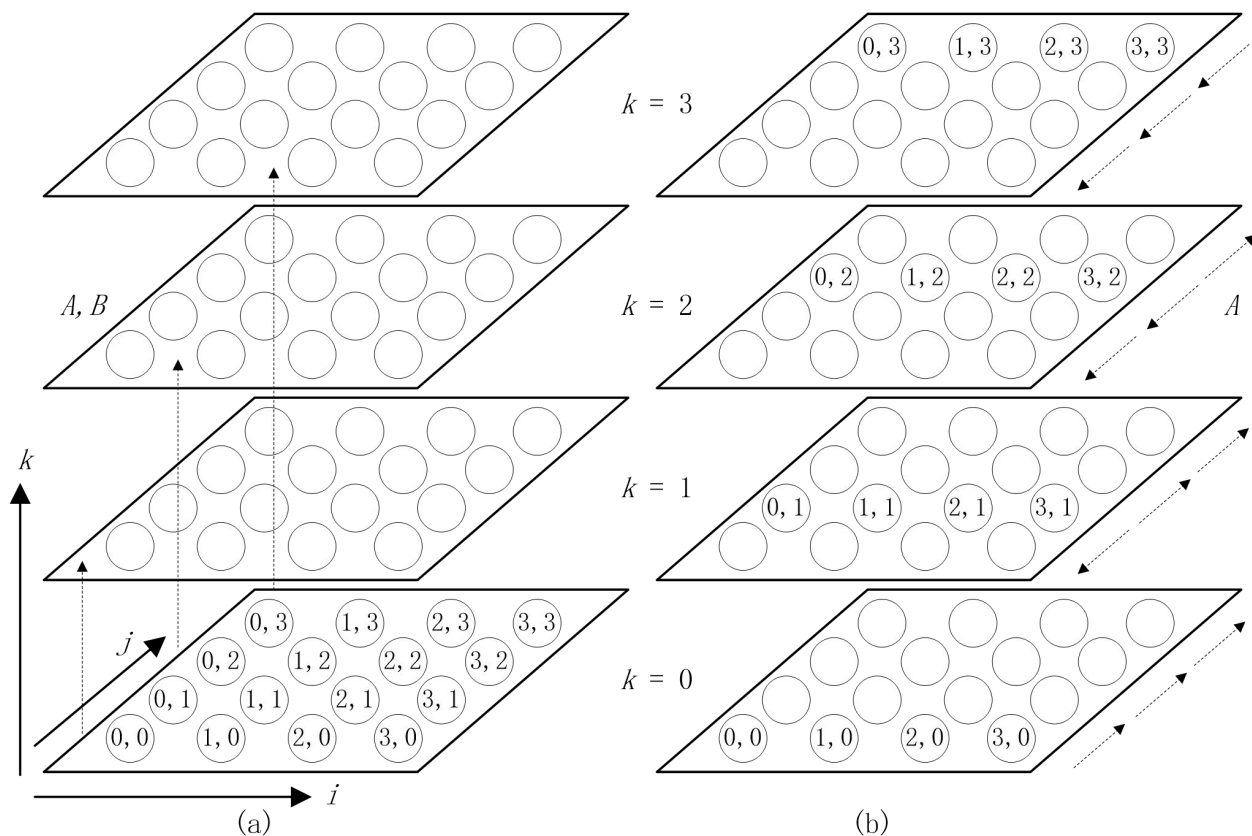
➤ DNS算法

- 假设处理器组成 $n \times n \times n$ 网格
- 移动 A 的列和 B 的行, 并执行广播操作
- 每个处理器计算一个加乘运算
- 之后沿 C 方向累加加乘的结果
- 由于每个加乘操作花费常数时间, 累加和广播操作花费 $\log n$ 时间, 因此总的时间复杂度是 $\log n$
- 上述不是开销最优的, 在累加方向使用 $n / \log n$ 个处理器可使开销最优

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ DNS算法



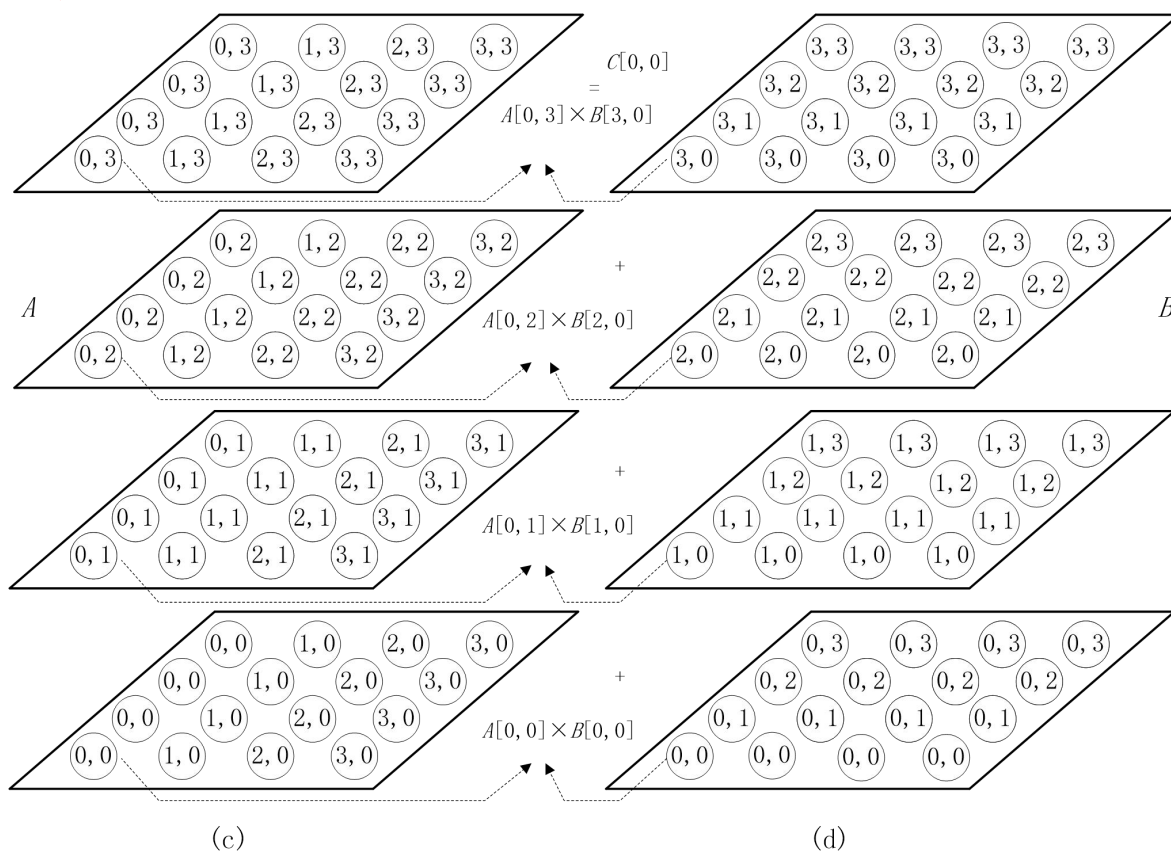
A、B的初始分布

将 $A[i, j]$ 从 $P_{i,j,0}$ 移动到 $P_{i,j,k}$

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ DNS算法



沿j方向广播A[i, j], 得到A的最终分布

B的最终分布情况

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ DNS算法

- 当处理器数量小于 n^3 时, 即假设进程数量 $p = q^3$, $q < n$
- 则两矩阵分块的子矩阵维度是 $(n/q) \times (n/q)$
- 每个矩阵可以被视作 $q \times q$ 的二维方阵, 其中每个元素是一个子矩阵
- 该算法与前一个算法相同, 只是在这种情况下, 操作的基本单位是子块而不是单个元素

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ DNS算法

- 当处理器数量小于 n^3 时
- 第一个One-to-One通讯被应用与A和B，每个矩阵上花费的时间是

$$t_s + t_w(n/q)^2$$

- 两个One-to-All广播通讯，每个矩阵上花费的时间为

$$2(t_s \log q + t_w(n/q)^2 \log q)$$

- 规约操作花费的时间为

$$t_s \log q + t_w(n/q)^2 \log q$$

- 计算 $(n/q) \times (n/q)$ 个子矩阵乘法花费的时间为 $(n/q)^3$

- 并行运行时间约为

$$T_P = \frac{n^3}{p} + t_s \log p + t_w \frac{n^2}{p^{2/3}} \log p.$$

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ 基本定义

- 内存 (M) — 对于给定问题, 所需的存储空间 (例如: 字节数)
- 工作负载 (W) — 对于给定问题, 所需的操作数 (例如: 浮点操作数), 包括数据加载和存储操作
- 速度 (V) — 单个处理器上, 单位时间操作数 (例如: 浮点数/秒)
- 时间 (T) — 经过的墙上时钟时间, 从计算开始到结束 (例如: 秒)
- 开销 (C) — 处理器数量与执行时间的乘积 (例如: 处理器-秒)

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ 基本定义

- 下标表示使用的处理器数量 (例如: T_1 表示串行时间, W_p 表示 p 个处理器的工作负载)
- 一般假设 $M_p \geq M_1$, 如果没有数据复制, 假设 $M_p = M_1$ ($p \geq 1$) 是合理的, 此时不再使用下标只记作 M
- 如果串行算法是最优的, 且忽略偶然情况, 那么 $W_p \geq W_1$; 通常情况下 $W_p > W_1$ ($p > 1$)
- 并行开销: $O_p = W_p - W_1$

矩阵-矩阵相乘

■ 矩阵-矩阵相乘 (Matrix-Matrix Multiplication)

➤ 基本定义

- 数据量通常决定计算量，在这种情况下，我们可以用 $W(M)$ 来表示计算复杂度对存储复杂度的依赖
- 例如：计算两个满秩矩阵 (n 维) 乘法
 - $M = (n^2)$, $W = (n^3)$, 可以推导出 $W(M) = (M^{3/2})$
 - 由于每个数据项都可能用于至少一个操作，因此假设工作 W 至少随内存 M 线性增长是合理的