

第7章 互连网络

李峰

fli@sdu.edu.cn

<https://funglee.github.io>

7.1 互连网络的基本概念

7.2 互连网络的种类

7.3 静态互连网络

7.4 动态互连网络

7.5 阵列处理机

7.6 脉动处理机

7.7 消息传递机制

7.1 互连网络的基本概念

7.1.1 互连网络的作用

7.1.2 互连函数

7.1.3 互连网络的特性

7.1.4 互连网络的传输性能参数

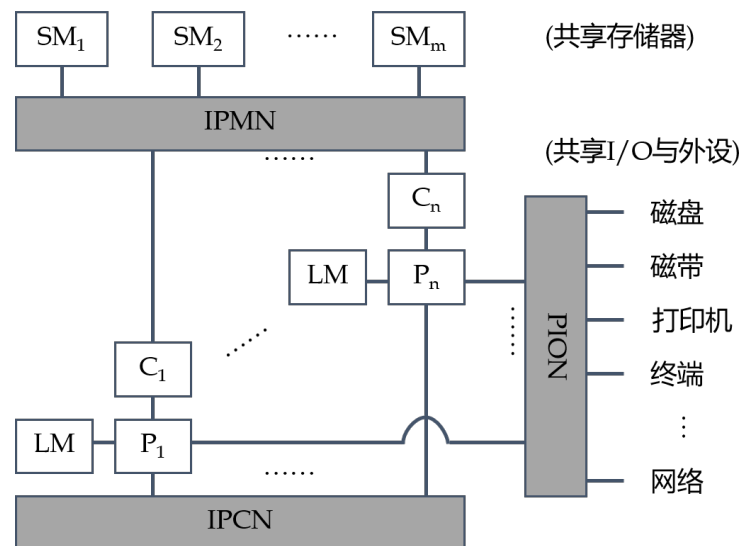
7.1.5 互连网络的种类

7.1.1 互连网络的作用

- 互连网络是一种由开关元件按照一定的拓扑结构和控制方式构成的网络，用来实现计算机系统中节点之间的相互连接。
- 互连网络已成为并行处理系统的核心组成部分
- 互连网络对整个计算机系统的性能价格比有着决定性的影响

7.1.1 互连网络的作用

- 每台处理机 P_i 与自己的本地存储器（LM）和私有高速缓存（ C_i ）相连
- 多处理机-存储器互联网络IPMN与共享存储器模块（SM）相连
- 处理机通过处理机-I/O网络IPON访问共享的I/O与外围设备
- 处理机之间通过处理机间通信网络IPCN进行通信



具有本地存储器、私有高速缓存、共享存储器和共享外围设备的一般处理机系统的互连结构

- IPMN（处理机-存储器网络）
- PION（处理机-I/O网络）
- IPCN（处理机之间通信网络）
- P（处理机），C（高速缓冲存储器）
- SM（共享存储器），LM（本地存储器）

7.1.1 互连网络的作用

- 互连网络是一种由开关元件按照一定的拓扑结构和控制方式构成的网络，用来实现计算机系统中节点之间的相互连接。
- 节点：处理器、存储模块或其他设备。
 - 在拓扑上，互连网络为输入节点到输出节点之间的一组互连或映射。
 - SIMD计算机和MIMD计算机的关键组成部分。
 - 3大要素：互连结构，开关元件，控制方式。

7.1.2 互连函数

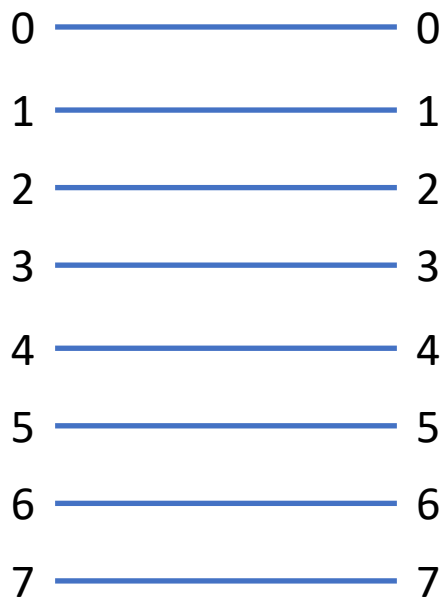
- 为反映互连网络的连接特性，每种互连网络可用一组互连函数来描述
- 如果将互连网络的 N 个输入端和 N 个输出端分别用整数 $0, 1, \dots, N-1$ 来表示，则互连函数表示相互连接的输出端号和输入端号之间的一一对应关系，即在互连函数 f 的作用下，输入端 x 连接到输出端 $f(x)$ 。
- 当互连网络用来实现处理器与处理器之间的数据变换时，互连函数反映了网络输入数组和输出数组之间对应的置换关系或排列关系，有时也称为置换函数或排列函数
- 设 $n = \log_2 N$ ，则可以用 n 位二进制来表示 N 个输入端和输出端的二进制地址，互连函数表示为：

$$f(x_{n-1}x_{n-2} \cdots x_1x_0)$$

7.1.2 互连函数

- 恒等函数：相同编号的输入端与输出端一一对应互连所实现的置换

$$I(x_{n-1}x_{n-2} \cdots x_0) = x_{n-1}x_{n-2} \cdots x_0$$



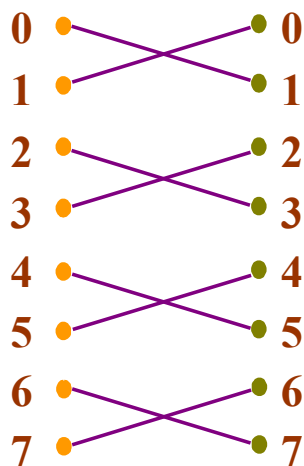
$N = 8$ 的恒等置换

7.1.2 互连函数

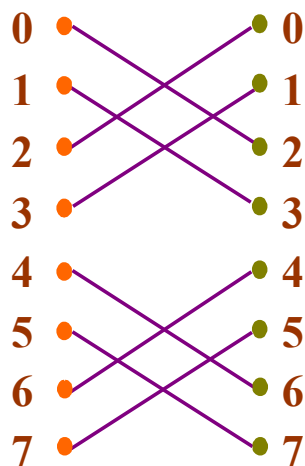
- 交换函数：实现二进制地址编码中第 k 位互反的输入端与输出端之间的连接

$$C_k(x_{n-1}x_{n-2} \cdots x_1x_0) = x_{n-1}x_{n-2} \cdots \overline{x_k} \cdots x_0$$

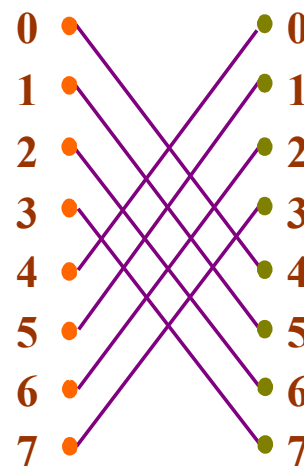
- 它共有 $n = \log_2 N$ 种互连函数（ N 为节点个数）



(a) Cube_0 交换函数



(b) Cube_1 交换函数



(c) Cube_2 交换函数

7.1.2 互连函数

- 交换函数：实现二进制地址编码中第0位互反的输入端与输出端之间的连接

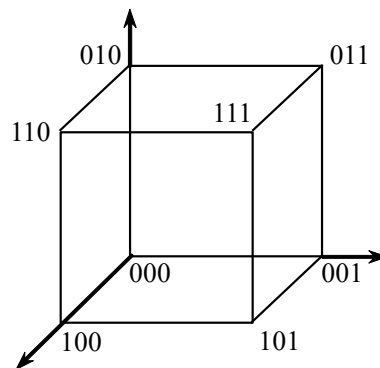
$$C_k(x_{n-1}x_{n-2} \cdots x_1x_0) = x_{n-1}x_{n-2} \cdots \bar{x}_k \cdots x_0$$

- 它共有 $n = \log_2 N$ 种互连函数（ N 为节点个数）
- 主要用于构造立方体互连网络和各种超立方体互连网络。
- 当 $N=8$ 时， $n=3$ ，可得到常用的立方体互连函数：

$$Cube_0(x_2x_1x_0) = x_2x_1\bar{x}_0$$

$$Cube_1(x_2x_1x_0) = x_2\bar{x}_1x_0$$

$$Cube_2(x_2x_1x_0) = \bar{x}_2x_1x_0$$



7.1.2 互连函数

- 均匀洗牌函数 (shuffle) : 将输入端分成数目相等的两半, 前一半和后一半按类似均匀混洗扑克牌的方式交叉地连接到输出端 (输出端相当于混洗的结果, 也称为混洗函数或混洗置换)

$$\sigma(x_{n-1}x_{n-2} \cdots x_1x_0) = x_{n-2}x_{n-3} \cdots x_1x_0x_{n-1}$$

- 子洗牌 (子混洗, subshuffle) : 即把输入端的二进制编号中的低k位循环左移一位

$$\begin{aligned}\sigma_{(k)}(x_{n-1}x_{n-2} \cdots x_k \textcolor{red}{x_{k-1}x_{k-2} \cdots x_1x_0}) \\ = x_{n-1}x_{n-2} \cdots x_k \textcolor{red}{x_{k-2} \cdots x_1x_0x_{k-1}}\end{aligned}$$

- 超洗牌 (超混洗, supershuffle) : 即把输入端的二进制编号中的高k位循环左移一位

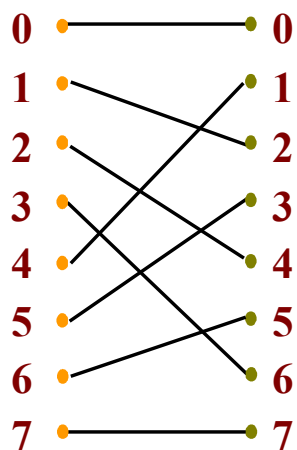
$$\begin{aligned}\sigma^{(k)}(\textcolor{red}{x_{n-1}x_{n-2} \cdots x_{n-k}}x_{n-k-1} \cdots x_1x_0) \\ = \textcolor{red}{x_{n-2} \cdots x_{n-k}x_{n-1}}x_{n-k-1} \cdots x_1x_0\end{aligned}$$

7.1.2 互连函数

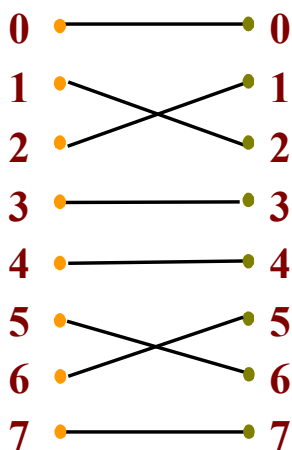
- 均匀洗牌函数: $\sigma(x_{n-1}x_{n-2} \cdots x_1x_0) = x_{n-2}x_{n-3} \cdots x_1x_0x_{n-1}$
- 子洗牌: $\sigma_{(k)}(x_{n-1}x_{n-2} \cdots x_k \textcolor{red}{x_{k-1}x_{k-2}} \cdots \textcolor{red}{x_1x_0}) = x_{n-1}x_{n-2} \cdots x_k \textcolor{red}{x_{k-2}} \cdots \textcolor{red}{x_1x_0} \textcolor{red}{x_{k-1}}$
- 超洗牌: $\sigma^{(k)}(\textcolor{red}{x_{n-1}x_{n-2}} \cdots \textcolor{red}{x_{n-k}}x_{n-k-1} \cdots x_1x_0) = \textcolor{red}{x_{n-2}} \cdots \textcolor{red}{x_{n-k}} \textcolor{red}{x_{n-1}}x_{n-k-1} \cdots x_1x_0$
$$\sigma^n(x) = \sigma_n(x) = \sigma(x)$$
$$\sigma^1(x) = \sigma_1(x) = x$$
- 逆混洗: $\sigma^{-1}(x_{n-1}x_{n-2} \cdots x_1x_0) = x_0x_{n-1} \cdots x_1$

7.1.2 互连函数

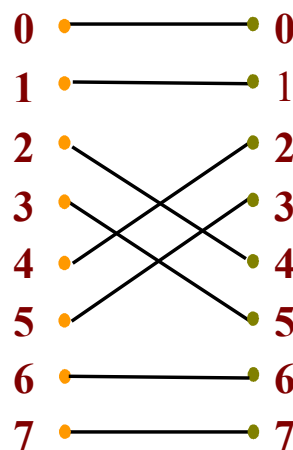
- 均匀洗牌函数: $\sigma(x_{n-1}x_{n-2} \cdots x_1x_0) = x_{n-2}x_{n-3} \cdots x_1x_0x_{n-1}$
- 子洗牌: $\sigma_{(k)}(x_{n-1}x_{n-2} \cdots x_kx_{k-1}x_{k-2} \cdots x_1x_0) = x_{n-1}x_{n-2} \cdots x_kx_{k-2} \cdots x_1x_0x_{k-1}$
- 超洗牌: $\sigma^{(k)}(x_{n-1}x_{n-2} \cdots x_{n-k}x_{n-k-1} \cdots x_1x_0) = x_{n-2} \cdots x_{n-k}x_{n-1}x_{n-k-1} \cdots x_1x_0$
- 逆混洗: $\sigma^{-1}(x_{n-1}x_{n-2} \cdots x_1x_0) = x_0x_{n-1} \cdots x_1$



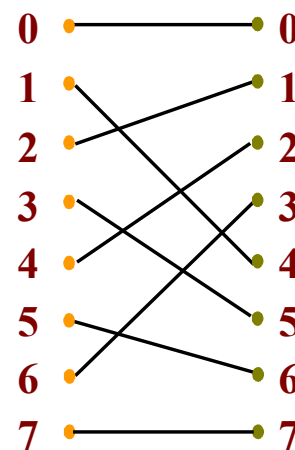
(a) 均匀洗牌函数 σ



(b) 子洗牌函数 $\sigma_{(2)}$



(c) 超洗牌函数 $\sigma^{(2)}$



(d) 逆均匀洗牌函数 σ^{-1}

7.1.2 互连函数

- 蝶式互连函数 (Butterfly)：把输入端的二进制编号的最高位与最低位互换位置，便得到了输出端的编号。

$$\beta(x_{n-1}x_{n-2} \cdots x_1x_0) = x_0x_{n-2} \cdots x_1x_{n-1}$$

- 子蝶式 (subbutterfly)：把输入端的二进制编号的低 k 位中的最高位与最低位互换。

$$\beta_{(k)}(x_{n-1} \cdots x_k \textcolor{red}{x_{k-1}x_{k-2} \cdots x_1x_0}) = x_{n-1} \cdots x_k \textcolor{red}{x_0x_{k-2} \cdots x_1x_{k-1}}$$

- 超蝶式 (superbutterfly)：把输入端的二进制编号的高 k 位中的最高位与最低位互换。

$$\begin{aligned} & \beta^{(k)}(\textcolor{red}{x_{n-1}x_{n-2} \cdots x_{n-k+1}x_{n-k}}x_{n-k-1} \cdots x_1x_0) \\ &= \textcolor{red}{x_{n-k}x_{n-2} \cdots x_{n-k+1}x_{n-1}}x_{n-k-1} \cdots x_1x_0 \end{aligned}$$

7.1.2 互连函数

- 下列等式成立

$$\beta^{(n)}(X) = \beta_{(n)}(X) = \beta(X)$$

$$\beta^{(1)}(X) = \beta_{(1)}(X) = X$$

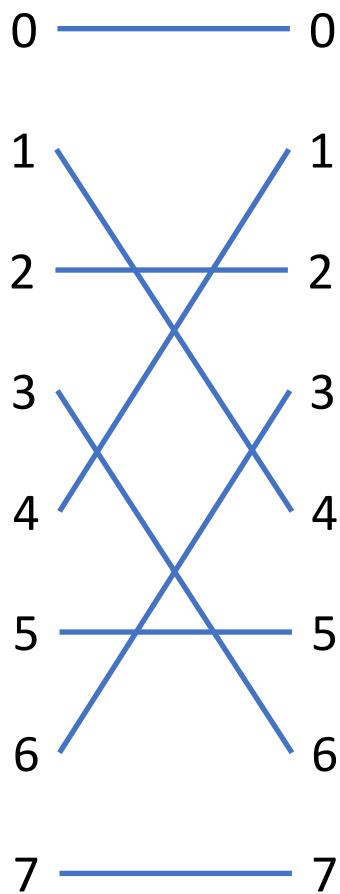
- 当N=8时，有：

$$\beta(x_2x_1x_0) = x_0x_1x_2$$

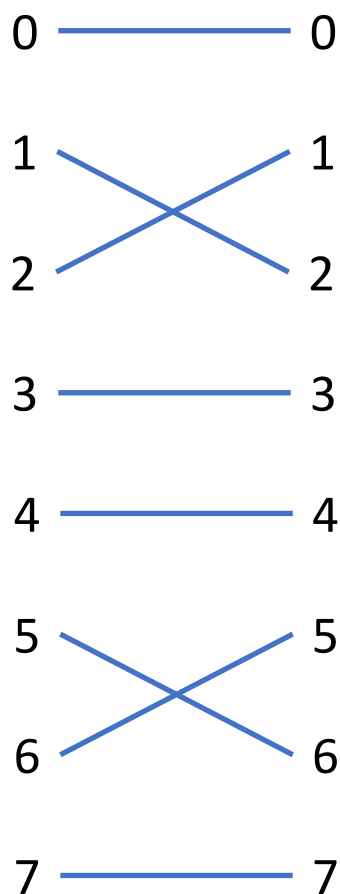
$$\beta_{(2)}(x_2x_1x_0) = x_2x_0x_1$$

$$\beta^{(2)}(x_2x_1x_0) = x_1x_2x_0$$

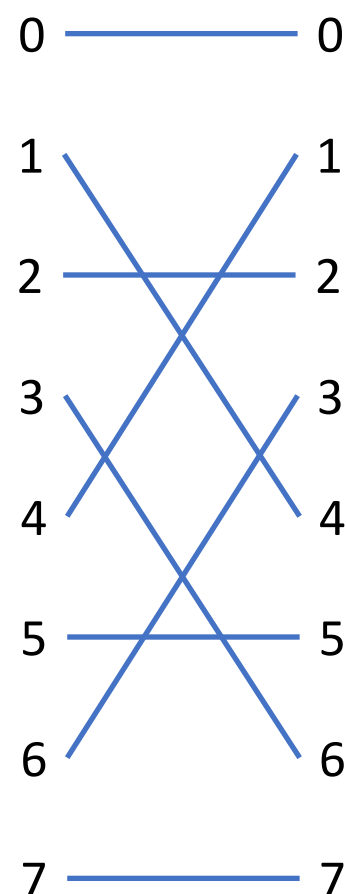
- 蝶式变换与交换变换的多级组合可作为构成方体多级网络的基础



β



$\beta_{(2)}$



$\beta^{(3)} = \beta$

7.1.2 互连函数

- 反位序函数 (Bit Reversal) : 将输入端二进制编号的位序颠倒过来求得相应输出端的编号。

$$\rho(x_{n-1}x_{n-2} \cdots x_1x_0) = x_0x_1 \cdots x_{n-2}x_{n-1}$$

- 子反位序: 即把输入端的二进制编号的低 k 位中各位的次序颠倒过来

$$\rho_{(k)}(x_{n-1} \cdots x_k \textcolor{red}{x_{k-1}x_{k-2} \cdots x_1x_0}) = x_{n-1} \cdots x_k \textcolor{red}{x_0x_1 \cdots x_{k-2}x_{k-1}}$$

- 超反位序: 即把输入端的二进制编号的高 k 位中各位的次序颠倒过来。

$$\begin{aligned} & \rho^{(k)}(\textcolor{red}{x_{n-1}x_{n-2} \cdots x_{n-k+1}x_{n-k}}x_{n-k-1} \cdots x_1x_0) \\ &= \textcolor{red}{x_{n-k}x_{n-k+1} \cdots x_{n-2}x_{n-1}}x_{n-k-1} \cdots x_1x_0 \end{aligned}$$

➤ 下列等式成立

$$\rho^{(n)}(X) = \rho_{(n)}(X) = \rho(X)$$

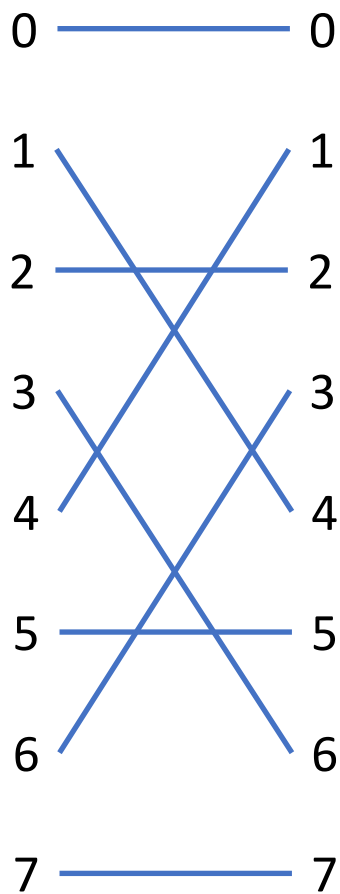
$$\rho^{(1)}(X) = \rho_{(1)}(X) = \rho(X)$$

➤ 当N=8时，有：

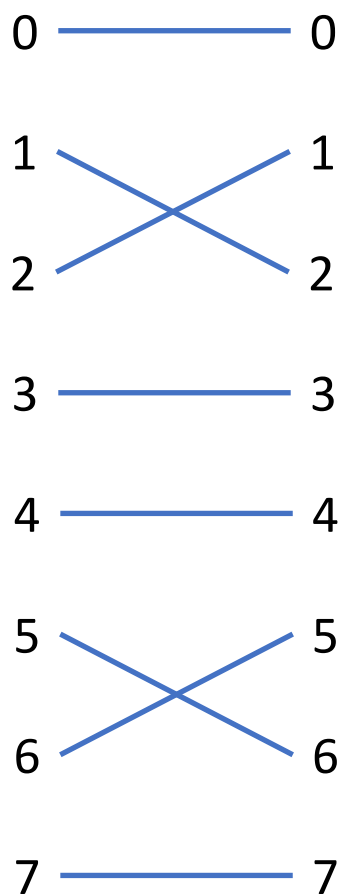
$$\rho(x_2x_1x_0) = x_0x_1x_2$$

$$\rho_{(2)}(x_2x_1x_0) = x_2x_0x_1$$

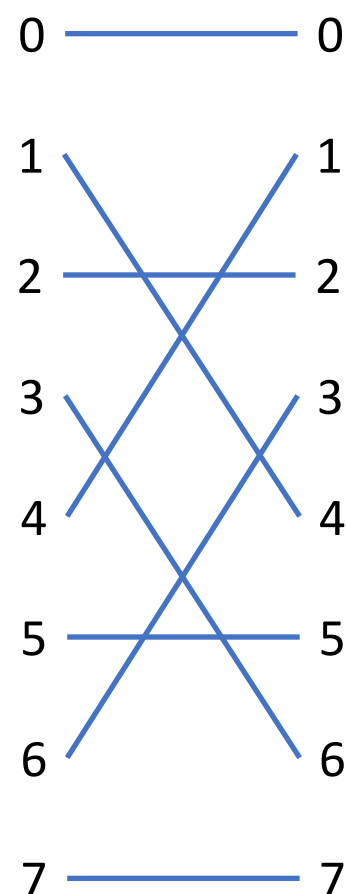
$$\rho^{(2)}(x_2x_1x_0) = x_1x_2x_0$$



$$\rho = \beta$$



$$\rho = \beta_{(2)}$$

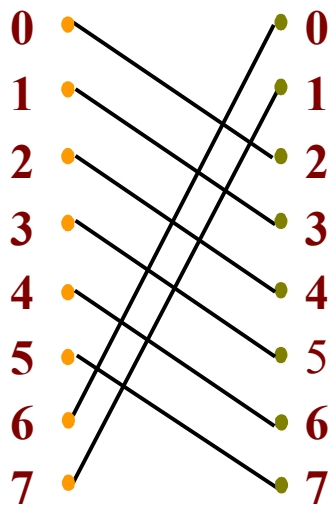


$$\beta^{(3)} = \rho_{(3)}$$

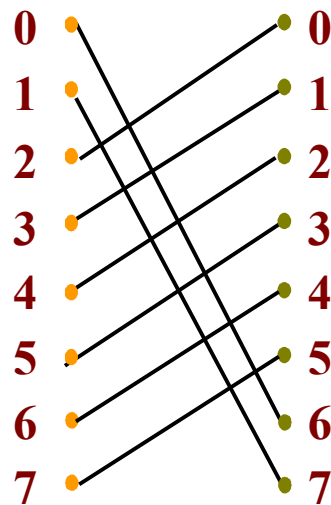
7.1.2 互连函数

- 移数函数：将各输入端都错开一定的位置（模 N ）后连到输出端。

$$\alpha(x) = (x \pm k) \bmod N \quad 1 \leq x \leq N-1, \quad 1 \leq k \leq N-1$$



(a) 左移移数函数 $k=2$



(b) 右移移数函数 $k=2$

7.1.2 互连函数

- PM2I函数： P 和 M 分别表示加和减， $2I$ 表示 2^i 。
 - 该函数又称为“加减 2^i ”函数。
 - 一种移数函数，将各输入端都错开一定的位置（模 N ）后连到输出端。
 - 互连函数

$$PM2_{+i}(x) = (x + 2^i) \bmod N$$

$$PM2_{-i}(x) = (x - 2^i) \bmod N$$

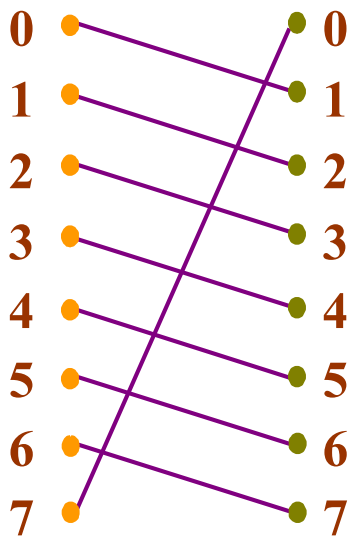
其中： $0 \leq x \leq N-1$ ， $0 \leq i \leq n-1$ ， $n=\log_2 N$ ， N 为节点数。

- PM2I互连网络共有 $2n$ 个互连函数。

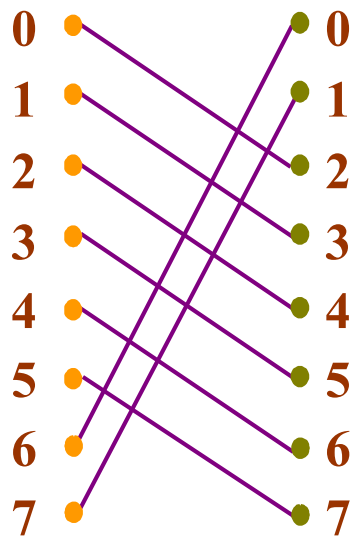
7.1.2 互连函数

➤ PM2I函数：当 $N=8$ 时，有6个PM2I函数：

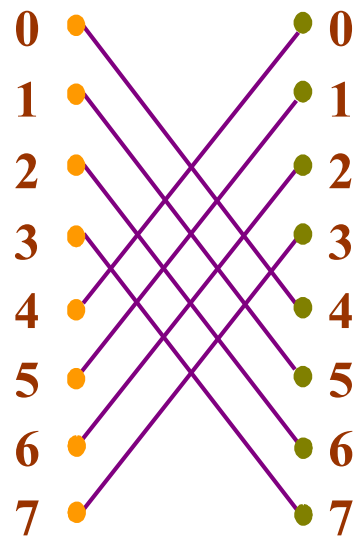
- $PM2_{+0}$: (0 1 2 3 4 5 6 7)
- $PM2_{-0}$: (7 6 5 4 3 2 1 0)
- $PM2_{+1}$: (0 2 4 6) (1 3 5 7)
- $PM2_{-1}$: (6 4 2 0) (7 5 3 1)
- $PM2_{+2}$: (0 4) (1 5) (2 6) (3 7)
- $PM2_{-2}$: (4 0) (5 1) (6 2) (7 3)



(a) $PM2_{+0}$



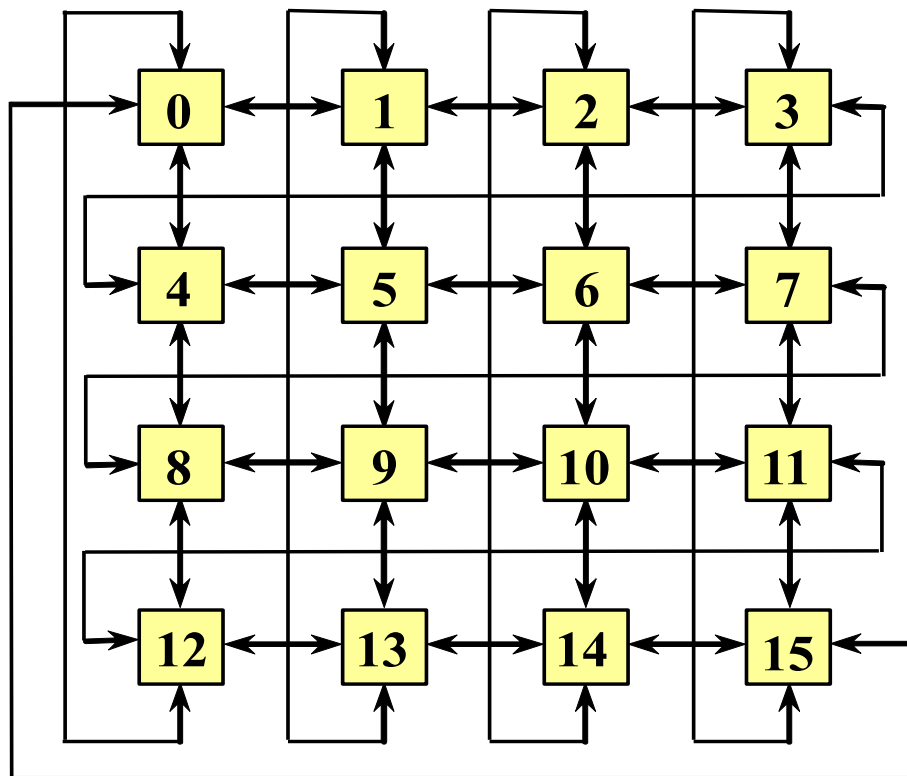
(b) $PM2_{+1}$



(c) $PM2_{+2}$

N=8 的PM2I函数

- 阵列计算机ILLIAC IV采用 $PM2_{\pm 0}$ 和 $PM2_{\pm n/2}$ 构成其互连网络，实现各处理单元之间的上下左右互连。



用移数函数构成ILLIAC IV 阵列机的互连网络

现有16个处理器，编号分别为0, 1, ..., 15，用一个 $N=16$ 的互连网络互连。处理器 i 的输出通道连接互连网络的输入端 i ，处理器 i 的输入通道连接互连网络的输出端 i 。当该互连网络实现的互连函数分别为：

(1) Cube_3

(2) PM2_{+3}

(3) PM2_{-0}

(4) σ

(5) $\sigma(\sigma)$

分别给出与第13号处理器所连接的处理器号。

7.1.3 互连网络的特性

互连网络通常是用有向边或无向边连接有限个结点的组成，主要特性有：

- 1) **网络规模**：网络中结点的个数
- 2) **结点度**：与结点相连接的边数称为结点度, 进入结点的边数叫入度, 从结点出来的边数则叫出度
- 3) **距离**：两个结点之间相连的最少边数
- 4) **网络直径**：网络中任意两个结点间距离的最大值。用结点间的连接边数表示
- 5) **等分宽度**：把由 N 个节点构成的网络切成节点数相同 ($N/2$) 的两半，在各种切法中，沿切口边数的最小值
- 6) **结点间的线长**：两个节点间线的长度（影响信号的时延、时钟扭斜和对功率的需要）
- 7) **对称性**：从任何节点看到的拓扑结构都是相同的网络称为对称网络

7.1.4 互连网络的传输性能参数

➤ 发送方的步骤如下：

- (1) 用户程序把要发送的数据拷贝到系统缓冲区。
- (2) 缓冲区中的数据打包并发送到网络接口部件。
- (3) 网络接口硬件开始发送消息。

➤ 数据包的接收步骤如下：

- (1) 把数据包从网络接口部件拷贝到系统缓冲区。
- (2) 检查收到的数据包，如果正确，发回答信号。
- (3) 把接收到的数据拷贝到用户地址空间。

➤ 发送方接收到回答信号后释放系统缓冲区（若发送方的超时计数器已超时，则重新发送消息）

7.1.4 互连网络的传输性能参数

➤互连网络的主要性能参数:

- 1) 频带宽度 (Bandwidth): 传输信息的最大速率
- 2) 传输时间 (Transmission time): 等于消息长度除以频宽。
- 3) 飞行时间 (Time of flight): 第一位信息到达接收方所花费的时间。
- 4) 传输时延 (Transport latency): 等于飞行时间与传输时间之和。
- 5) 发送方开销 (Sender overhead): 处理器把消息放到互连网络的时间。
- 6) 接收方开销 (Receiver overhead): 处理器把消息从网络取出来的时间。

7.1.4 互连网络的传输性能参数

➤ 一个消息的总时延可以用下面公式表示

总时延 = 发送方开销 + 飞行时间 + 消息长度/频宽 + 接收方开销

➤ 例：假设一个网络的频宽为10Mb/S，发送方开销为230us，接收方开销分别为270us。如果两台机器相距100米，现在要发送一个1000字节的消息给另一台机器，试计算总时延。如果两台机器相距1000公里，那么总时延为多大？

解：光的速度为299792.5KM/S，信号在导体中传递速度大约是光速的50%。

相距100米时总时延为：

$$\begin{aligned} T &= \text{发送方开销} + \text{飞行时间} + \frac{\text{消息长度}}{\text{频宽}} + \text{接收方开销} \\ &= 230\mu s + \frac{0.1Km}{0.5 \times 299792.5Km/s} + \frac{1000 \times 8\text{位}}{10\text{兆位/秒}} + 270\mu s \\ &= 230\mu s + 0.67\mu s + 800\mu s + 270\mu s = 1301\mu s \end{aligned}$$

相距1000公里时的总时延为：

$$\begin{aligned} T &= 230\mu s + \frac{1000 \times 10^6}{0.5 \times 299792.5} \mu s + \frac{1000 \times 8}{10} \mu s + 270\mu s \\ &= 230\mu s + 6671\mu s + 800\mu s + 270\mu s = 7971\mu s \end{aligned}$$

7.2 互连网络的种类

➤ 互连网络通常可以分为两大类：

- 静态互连网络：各节点之间有固定的连接通路、且在运行中不能改变的网络。
- 动态互连网络：由交换开关构成、可按运行程序的要求动态地改变连接状态的网络。

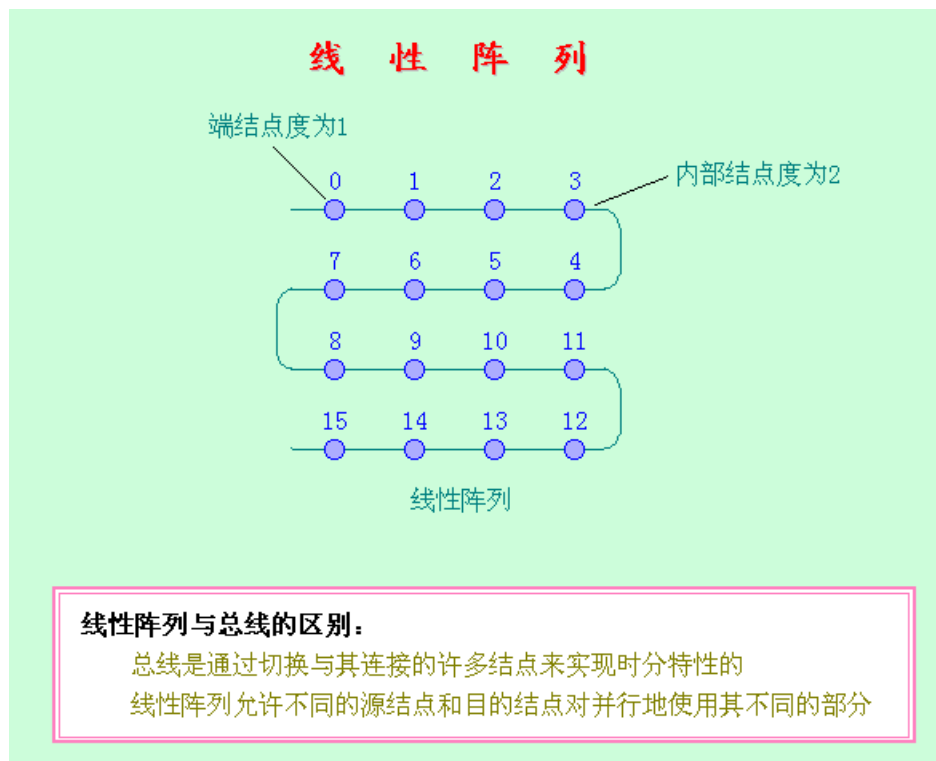
7.3 静态互连网络

- 在各结点之间有固定的连接通路，在运行过程中不能改变的网络结构。
- 一般静态互连网络不能实现任意结点到结点之间的互连。
- 一维的有线性阵列结构；二维的有环形、星形、树形、网格形等；三维的有立方体等；三维以上的有超立方体等。

7.3 静态互连网络

1. 线性阵列：一种一维的线性网络，其中 N 个节点用 $N-1$ 个链路连成一行

- 端节点的度：1
- 其余节点的度：2
- 直径： $N-1$
- 等分宽度 $b=1$

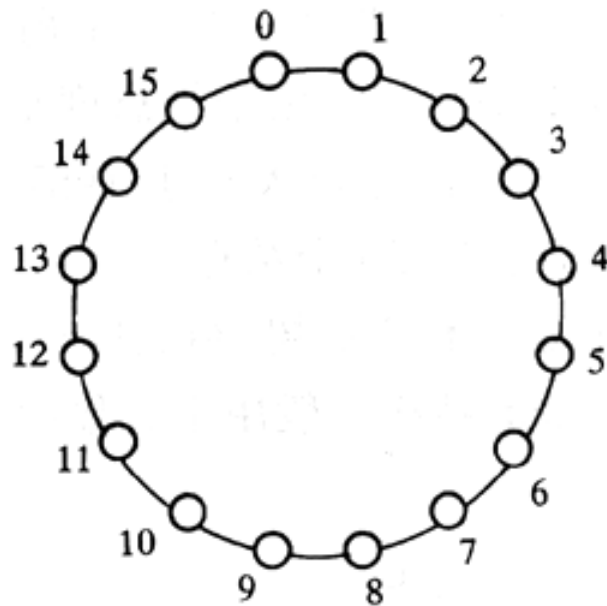


7.3 静态互连网络

2. 环和带弦环

➤ 环：用一条附加链路将线性阵列的两个端点连接起来而构成。可以单向工作，也可以双向工作

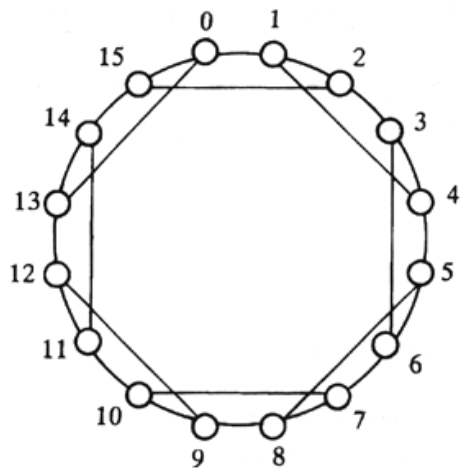
- 对称
- 节点的度：2
- 双向环的直径： $N/2$
- 单向环的直径： N
- 环的等分宽度 $b=2$



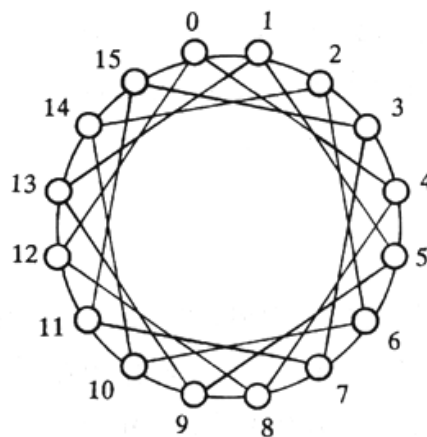
7.3 静态互连网络

2. 环和带弦环

➤ 带弦环：分别增加一条或两条附加链路可以得到如下的带弦环，增加的链路愈多，节点度愈高，网络直径就愈小



节点度3，直径5



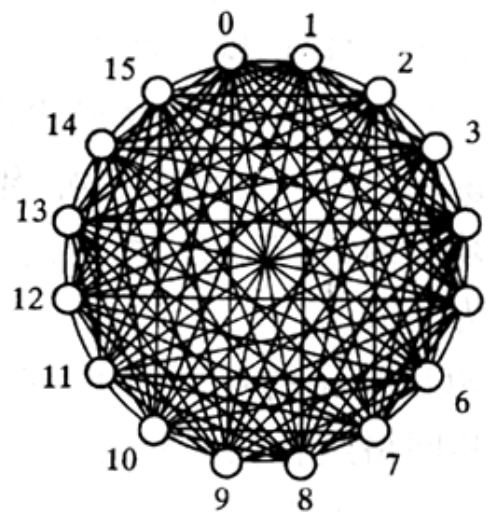
节点度4，直径3

7.3 静态互连网络

2. 环和带弦环

➤ 带弦环：分别增加一条或两条附加链路可以得到如下的带弦环，增加的链路愈多，节点度愈高，网络直径就愈小

- 节点度：15
- 直径最短，为1。

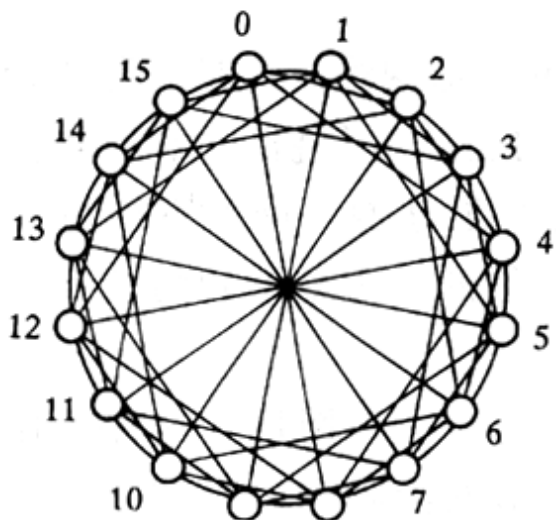


全连接

7.3 静态互连网络

3. 循环移数网络

➤ 通过在环上每个节点到所有与其距离为2的整数幂的节点之间都增加一条附加链而构成。



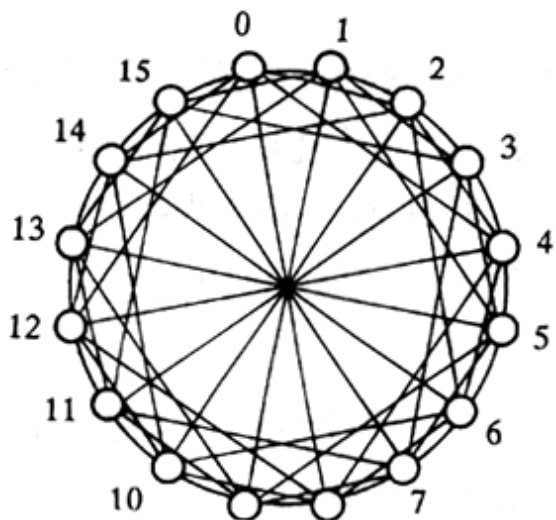
(e) 循环移数网络

- 循环移数网络的连接特性与结点度较低的任何带弦环相比有了改进
- 对 $N = 16$ 的情况，循环移数网络的结点度为7，直径为2
- 复杂性比全连接网络要低得多

7.3 静态互连网络

3. 循环移数网络

➤ 通过在环上每个节点到所有与其距离为2的整数幂的节点之间都增加一条附加链而构成。



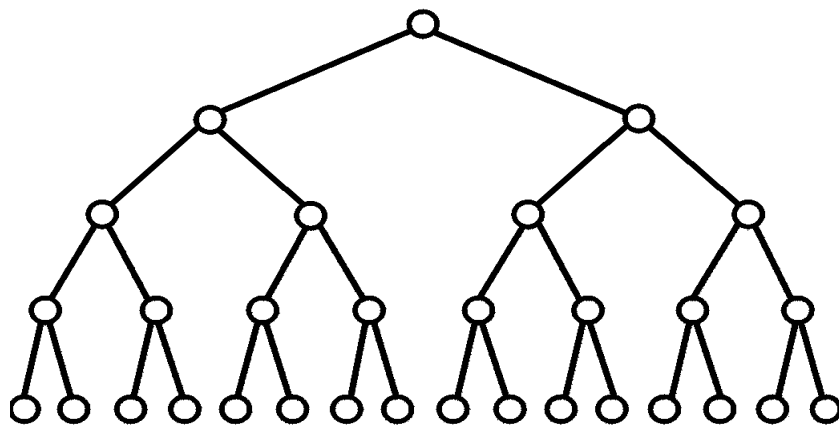
(e) 循环移数网络

- 如果 $|j - i| = 2^r$ ($r = 0, 1, \dots, n - 1$)，则节点 i 与节点 j 连接。
- 节点度: $2n - 1$
- 直径: $\frac{n}{2}$
- 网络规模 $N = 2^n$

7.3 静态互连网络

4. 树形和星形

- 树形：一颗 k 层完全平衡的二叉树应有 $N = 2^{k-1}$ 个结点，最大结点度是3，直径是 $2(k - 1)$ ，等分宽度 $b = 1$
- 由于结点度是常数，因此二叉树是一种可扩展的系统结构，但其直径相当长



(a) 二叉树

7.3 静态互连网络

4. 树形和星形

- 星形：是一种2层树，结点度较高，为 $d = N - 1$ ；直径较小，为常数2；
等分宽度为 $b = \lfloor N/2 \rfloor$
- 可靠性较低，只要中心结点出故障，整个系统就会瘫痪

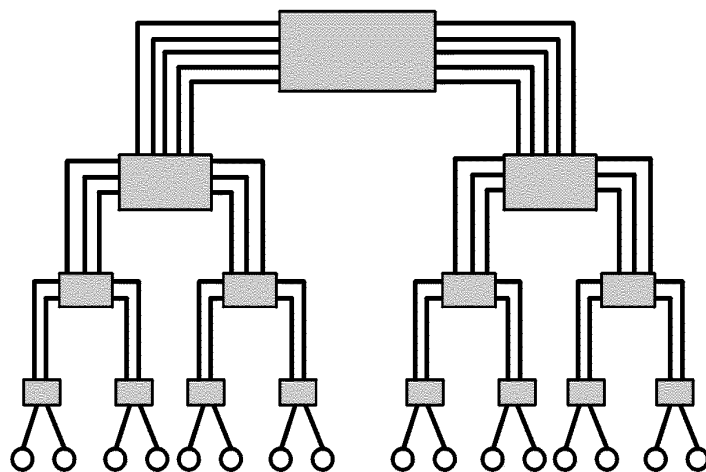
星

(b) 星形

7.3 静态互连网络

5. 胖形树

- 胖树的通道宽度从叶结点往根结点上行方向逐渐增宽
- 传统二叉树的主要问题之一就是通向根结点的瓶颈问题，这是因为根部的通信最忙，而胖树的提出使问题得到了缓解



(c) 二叉胖树

7.3 静态互连网络

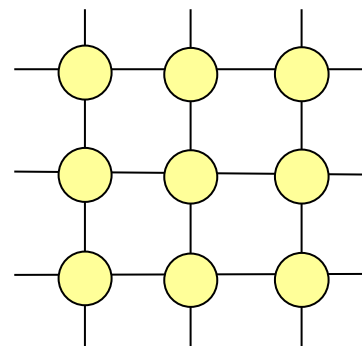
6. 网格形与环网形

➤ 网格形

○ 一个规模为 $N = n \times n$ 的2维网格形网络

- 内部节点的度 $d = 4$
- 边节点的度 $d = 3$
- 角节点的度 $d = 2$
- 网络直径 $D = 2(n - 1)$
- 等分宽度 $b = n$

○ 一个由 $N = n^k$ 个节点构成的 k 维网格形网络（每维 n 个节点）的内部节点度 $d = 2k$ ，网络直径 $D = k(n - 1)$

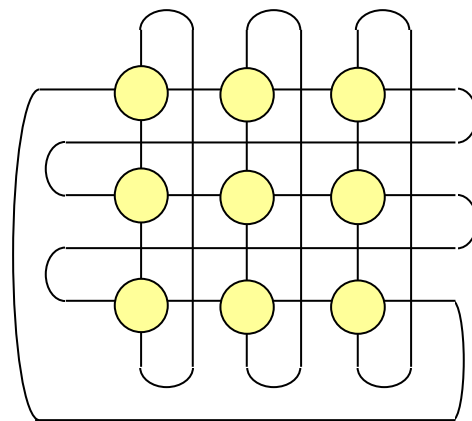


(a) 网格形

7.3 静态互连网络

6. 网格形与环网形

- Illiac网络：名称来源于采用这种网络的Illiac IV计算机
 - 把二维网格形网络的每一列的两个端节点连接起来，再把每一行的尾节点与下一行的头节点连接起来，并把最后一行的尾节点与第一行的头节点连接起来
 - 一个规模为 $n \times n$ 的Illiac网络
 - 所有结点的度 $d = 4$
 - 网络直径 $D = n - 1$
 - 等分宽度 $b = 2n$



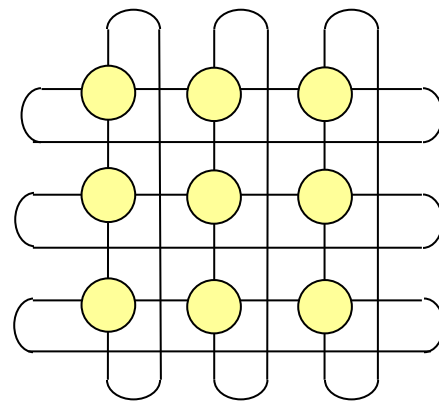
(b) Illiac网

7.3 静态互连网络

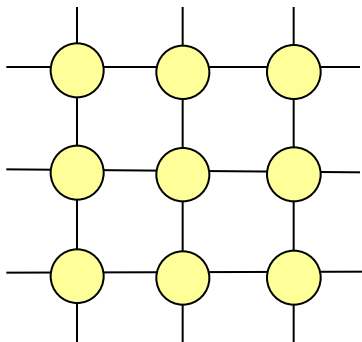
6. 网格形与环网形

➤ 环网形

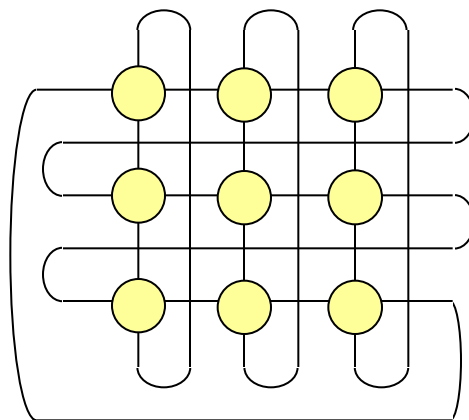
- 把2维网格形网络的每一行的两个端节点连接起来，把每一列的两个端节点也连接起来
- 将环形和网格形组合在一起，并能够向高维扩展
- 一个规模为 $n \times n$ 的环网形网络
 - 结点的度 $d = 4$
 - 网络直径 $D = 2 \times \lfloor n/2 \rfloor$
 - 等分宽度 $b = 2n$



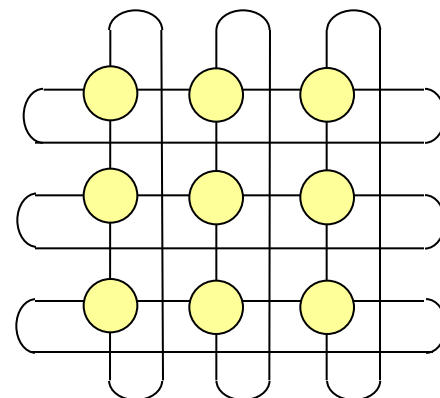
(c) 环网形



(a) 网格形



(b) Illiac网

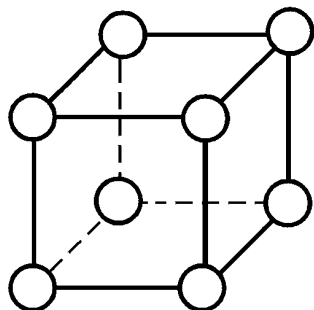


(c) 环网形

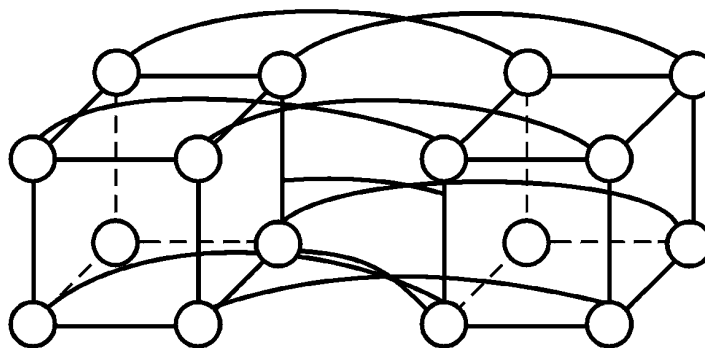
7.3 静态互连网络

7. 超立方体

- 一种二元 n -立方体结构，由 $N = 2^n$ 个节点组成，它们分布在 n 维上，每维有两个节点
- 为实现一个 n -立方体，只要把两个 $n - 1$ 立方体中相对应的节点用链路连接起来即可。共需要 2^{n-1} 条链路
- n -立方体中节点的度都是 n ，直径也是 n ，等分宽度为 $b = N/2$



(a) 3-立方体

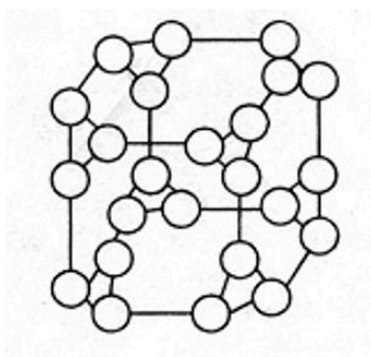


(b) 由 2 个 3-立方体组成的 4-立方体

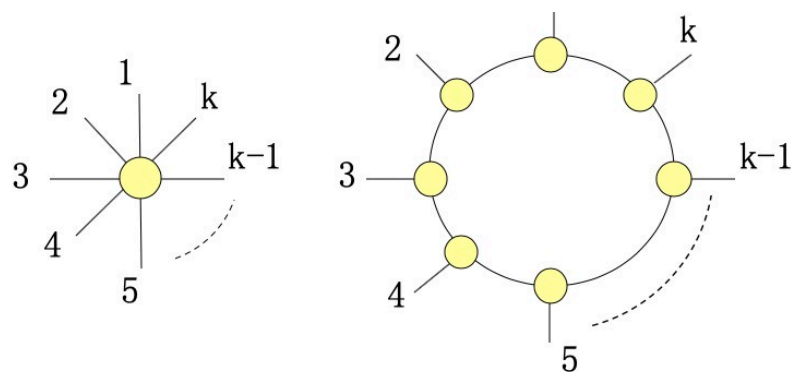
7.3 静态互连网络

8. 带环立方体

- 带环 k -立方体（简称 k -CCC），是 k -立方体的变形，通过用 k 个节点构成的环取代 k -立方体中的每个节点而形成的。
- 网络规模为 $N = k \times 2^k$ ；网络直径为 $D = 2k - 1 + \lfloor k/2 \rfloor$ ，比 k -立方体的直径大一倍；等分宽度为 $b = \frac{N}{2k}$



(c) 把3-立方体的每个节点换成一个由3个节点构成的环，形成带环3-立方体



(b) 将 k -立方体的每个节点用由 k 个节点的环来代替，组成带环 k -立方体组成

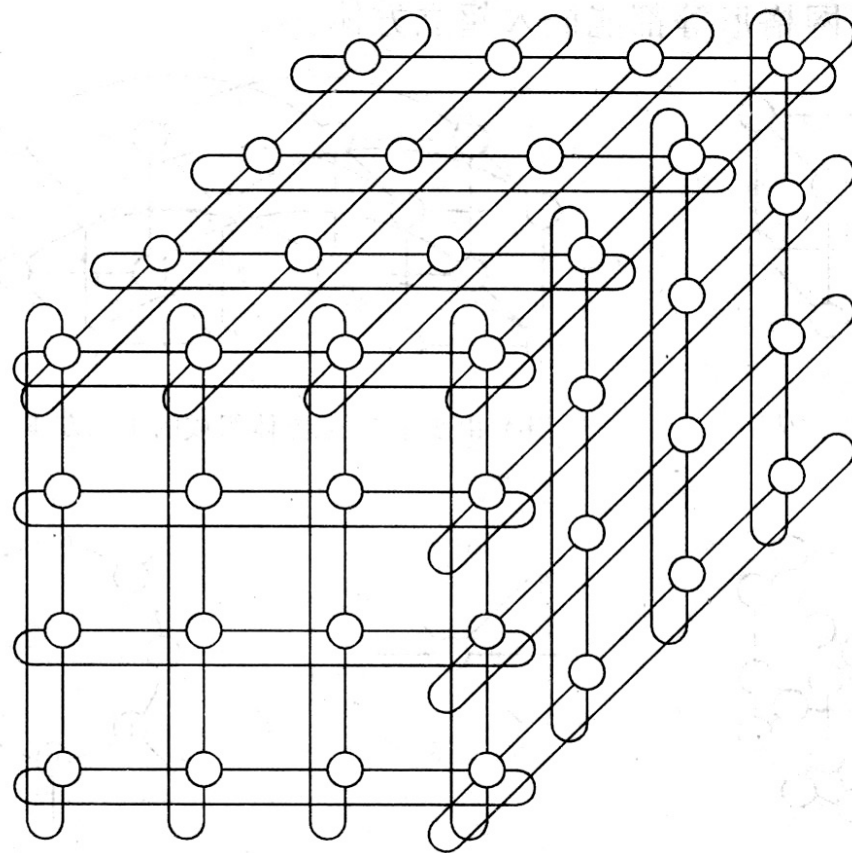
7.3 静态互连网络

9. k 元 n -立方体网络

- 环形、网格、环网形、二元 n -立方体（超立方体）和Omega网络都是 k 元 n -立方体网络系列的拓扑同构体。
- 在 k 元 n -立方体网络中，参数 n 是立方体的维数， k 是基数，即每一维上的节点个数。

$$N = k^n, (k = \sqrt[n]{N}, n = \log_k N)$$

- k 元 n -立方体的节点可以用基数为 k 的 n 位地址 $A = a_1 a_2 \dots a_n$ 来表示。
- 其中 a_i 表示该节点在第 i 维上的位置
- 通常把低维 k 元 n -立方体称为环网，而把高维 k 元 n -立方体称为超立方体。



4元3-立方体网络

静态互连网络特征一览表

网络类型	节点度d	网络直径D	链路数1	等分宽度B	对称性	网络规格说明
线线阵列	2	$N-1$	$N-1$	1	非	N个节点
环形	2	$[N/2]$	N	2	是	N个节点
全连接	$N-1$	1	$N(N-1)/2$	$(N/2)^2$	是	N个节点
二叉树	3	$2(h-1)$	$N-1$	1	非	树高 $h=[\log_2 N]$
星形	$N-1$	2	$N-1$	$[N/2]$	非	N个节点
2D网格	4	$2(r-1)$	$2N-2r$	r	非	$r \times r$ 网格, $r = \sqrt{N}$
Illiac网	4	$r-1$	$2N$	$2r$	非	与 $r = \sqrt{N}$ 的带弦环等效
2D环网	4	$2[r/2]$	$2N$	$2r$	是	$r \times r$ 环网, $r = \sqrt{N}$
超立方体	n	n	$nN/2$	$N/2$	是	N个节点, $n=[\log_2 N]$ (维数)
CCC	3	$2k-1+[k/2]$	$3N/2$	$N/(2k)$	是	$N=k \times 2^k$ 节点 环长 $k \geq 3$
k元n-立方体	$2n$	$n[k/2]$	nN	$2k^{n-1}$	是	$N=k^n$ 个节点

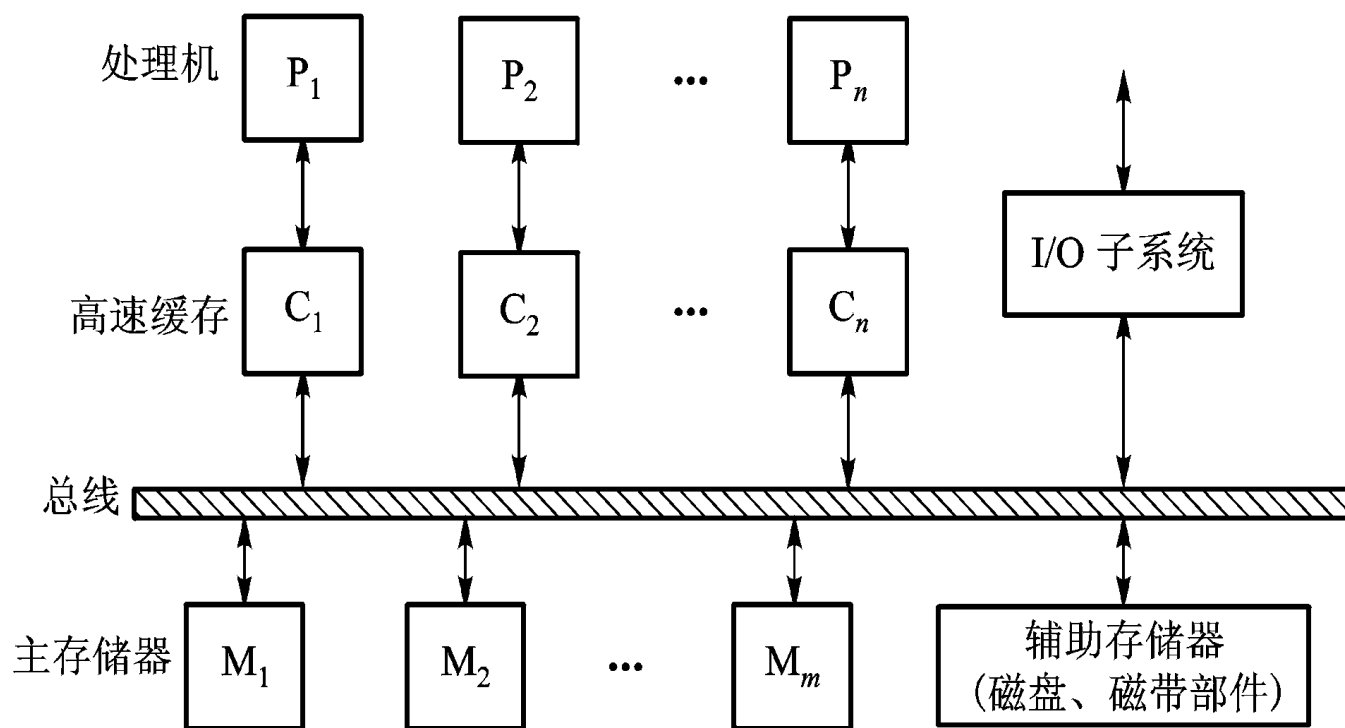
7.4 动态互连网络

7.4.1 总线网络

- 由一组导线和插座构成，经常被用来实现计算机系统中处理机模块、存储模块和外围设备等之间的互连。
 - 总线只能用于一个源（主部件）到一个或多个目的（从部件）之间一回处理一次业务
 - 多个功能模块之间的争用总线（Contention Bus）或时分总线（Time-Sharing Bus）

7.4 动态互连网络

7.4.1 总线网络



一种由总线连接的多处理机系统

7.4 动态互连网络

7.4.1 总线网络

➤ 特点

- 结构简单、实现成本低、带宽较窄
- 系统总线在处理机、I/O子系统、主存储器以及辅助存储设备（磁盘、磁带机等）之间提供了一条公用通路。
- 系统总线通常设置在印刷电路板底板上。处理器板、存储器板和设备接口板都通过插座或电缆插入底板。

7.4 动态互连网络

7.4.1 总线网络

➤ 解决总线带宽较窄问题：采用多总线或多层次的总线

- 多总线是设置多条总线，有两种做法：1) 为不同的功能设置专门的总线；2) 重复设置相同功能的总线
- 多层次的总线是按层次的架构设置速度不同的总线，使得不同速度的模块有比较适合的总线连接。

7.4 动态互连网络

7.4.2 交叉开关网络

➤ 单级开关网络

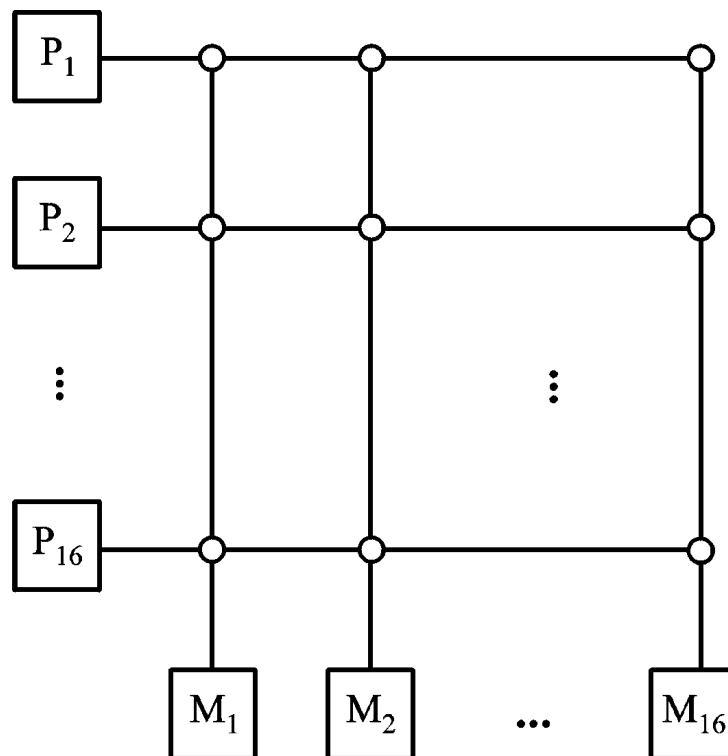
- 交叉开关能在对偶（源、目的）之间形成动态连接，同时实现多个对偶之间的无阻塞连接。
- 带宽和互连特性最好。
- 一个 $n \times n$ 的交叉开关网络，可以无阻塞地实现 $n!$ 种置换。
- 对一个 $n \times n$ 的交叉开关网络来说，需要 n^2 套交叉点开关以及大量的连线。
- 当 n 很大时，交叉开关网络所需要的硬件数量非常巨大。

7.4 动态互连网络

7.4.2 交叉开关网络

➤ C. mmp多处理机的互连结构

- 用 16×16 的交叉开关网络把16台PDP-11处理机与16个存储模块连在一起
- 最多可同时实现16台处理机对16个不同存储模块的并行访问
- 每个存储模块一次只能满足一台处理机的请求
- 当多个请求要同时访问同一存储模块时，交叉开关就必须分解所发生的冲突，每一列只能接通一个交叉点开关。
- 为了支持并行（或交叉）存储器访问，可以在同一行中接通几个交叉点开关。

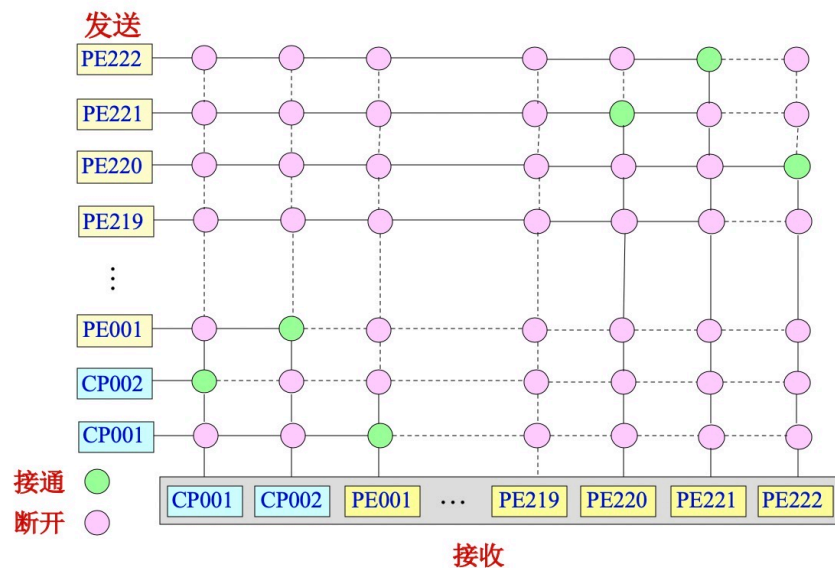


7.4 动态互连网络

7.4.2 交叉开关网络

➤ Fujitsu公司制造的向量并行处理机VPP500所采用的大型交换开关网络(224×224)

- PE: 带存储器的处理机
- CP: 控制处理机
- 每一行和每一列只能接通一个交叉点开关



7.4 动态互连网络

7.4.3 多级互连网络

➤ 多级互连网络的构成

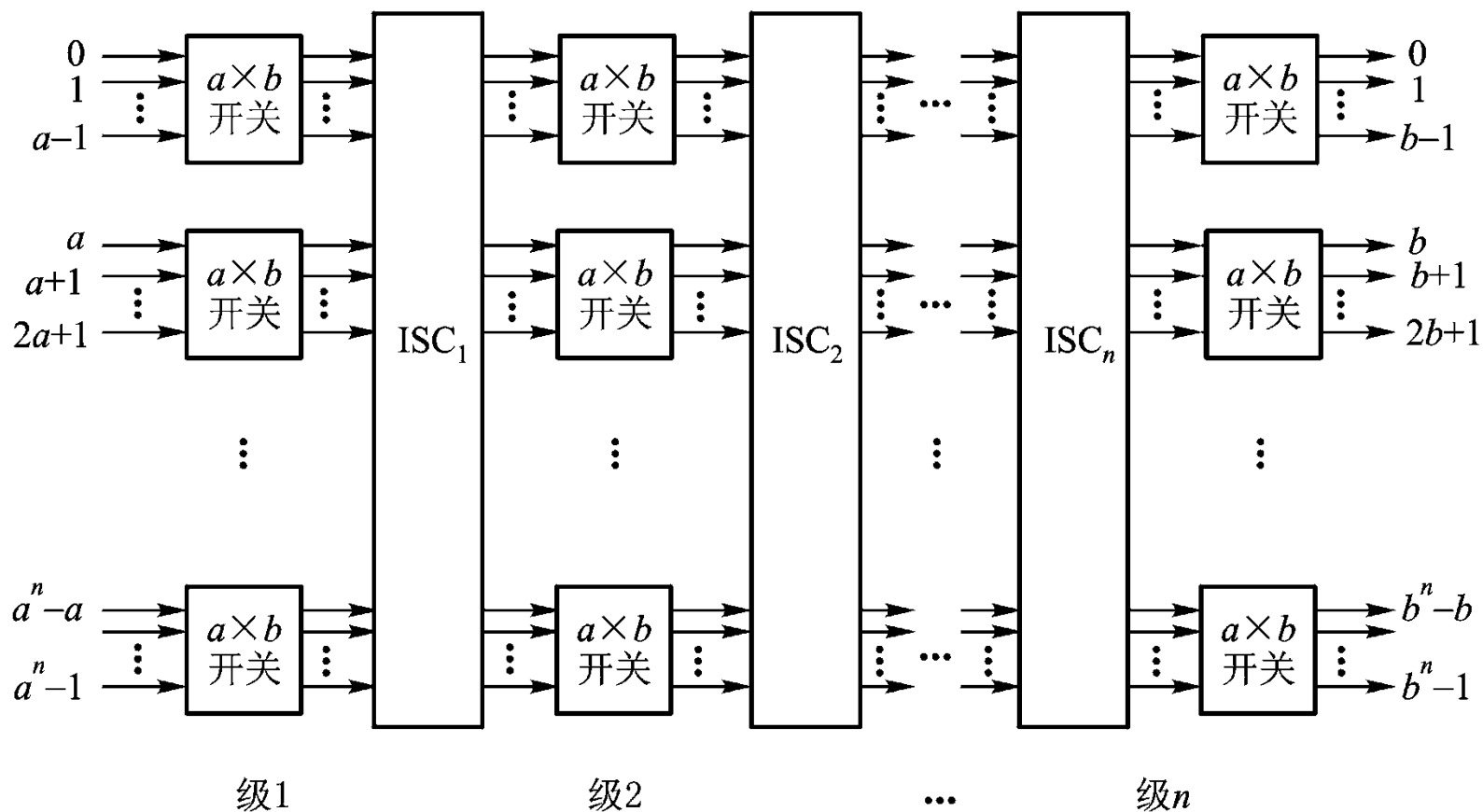
□ MIMD 和 SIMD 计算机都采用多级互连网络 MIN (Multistage Interconnection Network)

□ 一种通用的多级互连网络

- 由 $a \times b$ 开关模块和级间连接构成的通用多级互连网络结构
- 每一级都用了多个 $a \times b$ 开关
 - a 个输入和 b 个输出
 - 在理论上, a 和 b 不一定相等, 然而实际上 a 和 b 经常选为 2 的整数幂, 即 $a=b=2^k$, $k \geq 1$ 。
- 相邻各级开关之间都有固定的级间连接

7.4 动态互连网络

7.4.3 多级互连网络



7.4 动态互连网络

7.4.3 多级互连网络

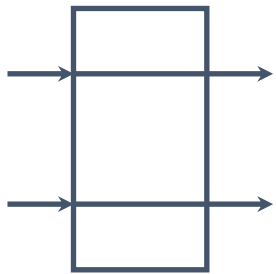
- 几种常用的开关模块
- 每个输入可与一个或多个输出相连，但是在输出端不许发生冲突

模块大小	合法状态	置换连接
2×2	4	2
4×4	256	24
8×8	16 777 216	40 320
$n \times n$	n^n	$n!$

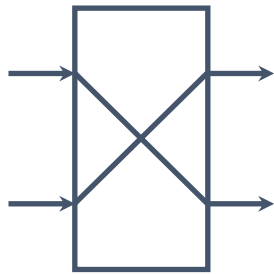
7.4 动态互连网络

7.4.3 多级互连网络

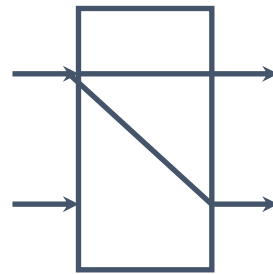
- 最简单的开关模块： 2×2 开关



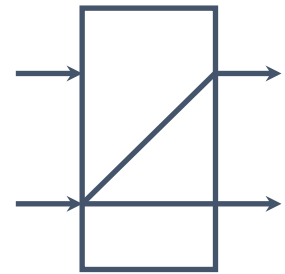
直连



交换



上播



下播

7.4 动态互连网络

7.4.3 多级互连网络

- 各种多级互连网络的区别在于所用开关模块、控制方式和级间互连模式的不同。

□ 控制方式：对各个开关模块进行控制的方式。

- 级控制：每一级的所有开关只用一个控制信号控制，只能同时处于同一种状态。
- 单元控制：每一个开关都有一个独立的控制信号，可各自处于不同的状态。
- 部分级控制：第 i 级的所有开关分别用 $i+1$ 个信号控制， $0 \leq i \leq n-1$ ， n 为级数。

□ 常用的级间互连模式：均匀洗牌、蝶式、多路洗牌、纵横交叉、立方体连接等

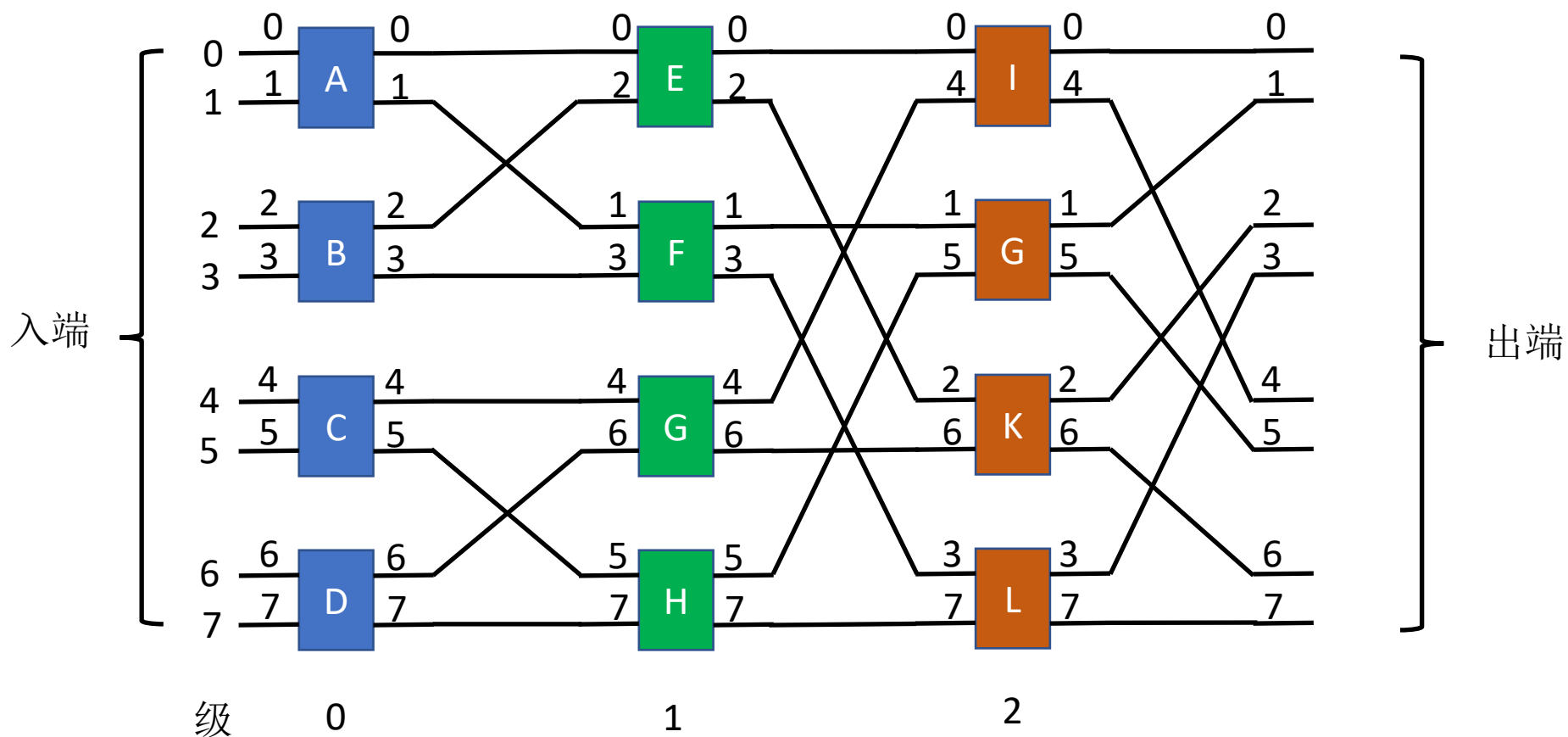
7.4.3 多级互连网络

1 多级立方体网络

- 多级立方体网络包括STARAN网络和间接二进制 n 方体网络等。
 - 两者仅在控制方式上不同：STRAN网络采用级控制（称交换网络）和部分级控制（其中可实现移数功能的称移数网络），而间接二进制 n 方体网络采用单元控制, 具有更大的灵活性。
 - 共同点：都采用二功能（直送和交换）的 2×2 开关；当第 i 级（ $0 \leq i \leq n - 1$ ）交换开关处于交换状态时，实现的是 $Cube_i$ 互连函数。
- 一个 N 输入的多级立方体网络有 $\log_2 N$ 级，每级用 $N/2$ 个 2×2 开关模块，共需要 $\log_2 N \times N/2$ 个开关。

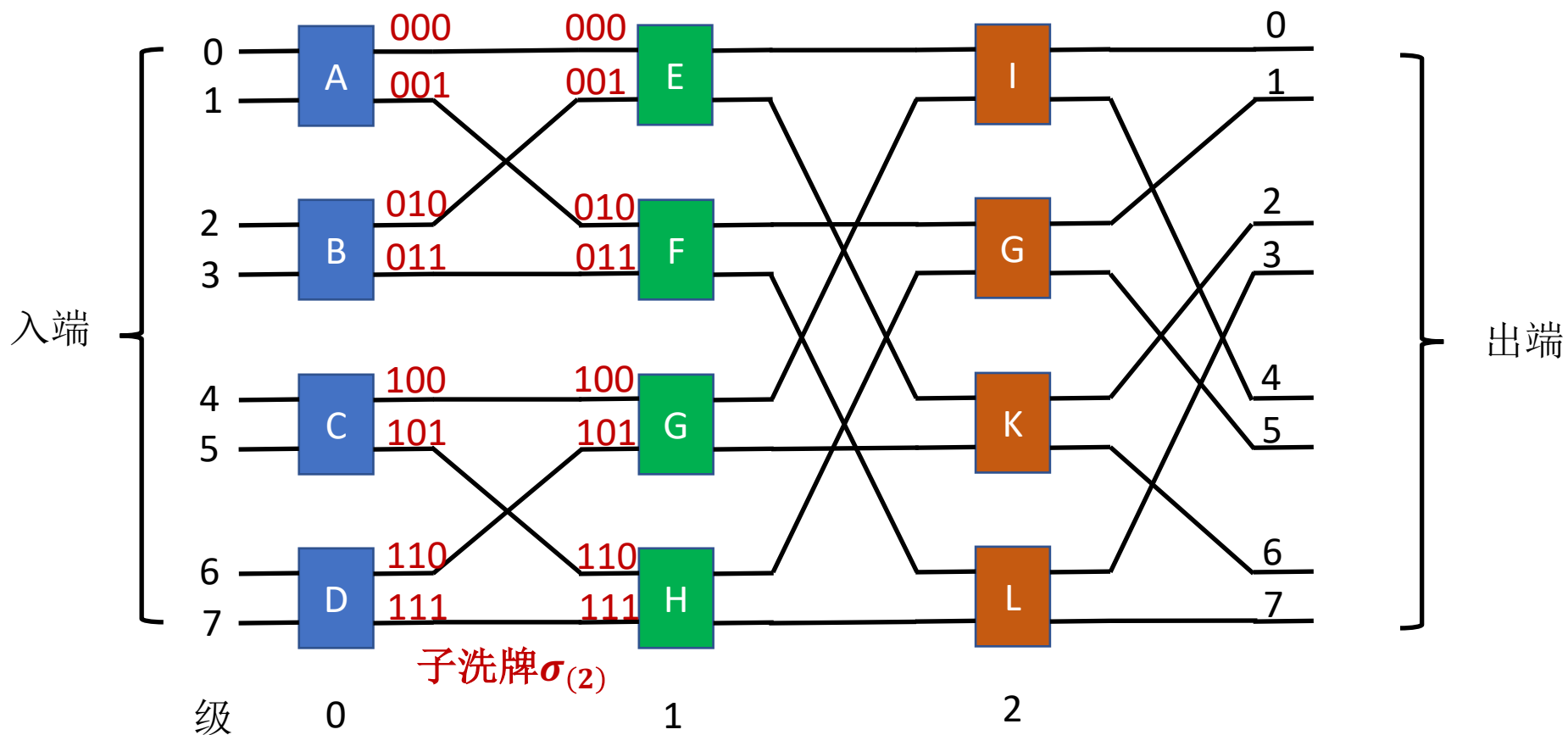
7.4.3 多级互连网络

1 多级立方体网络



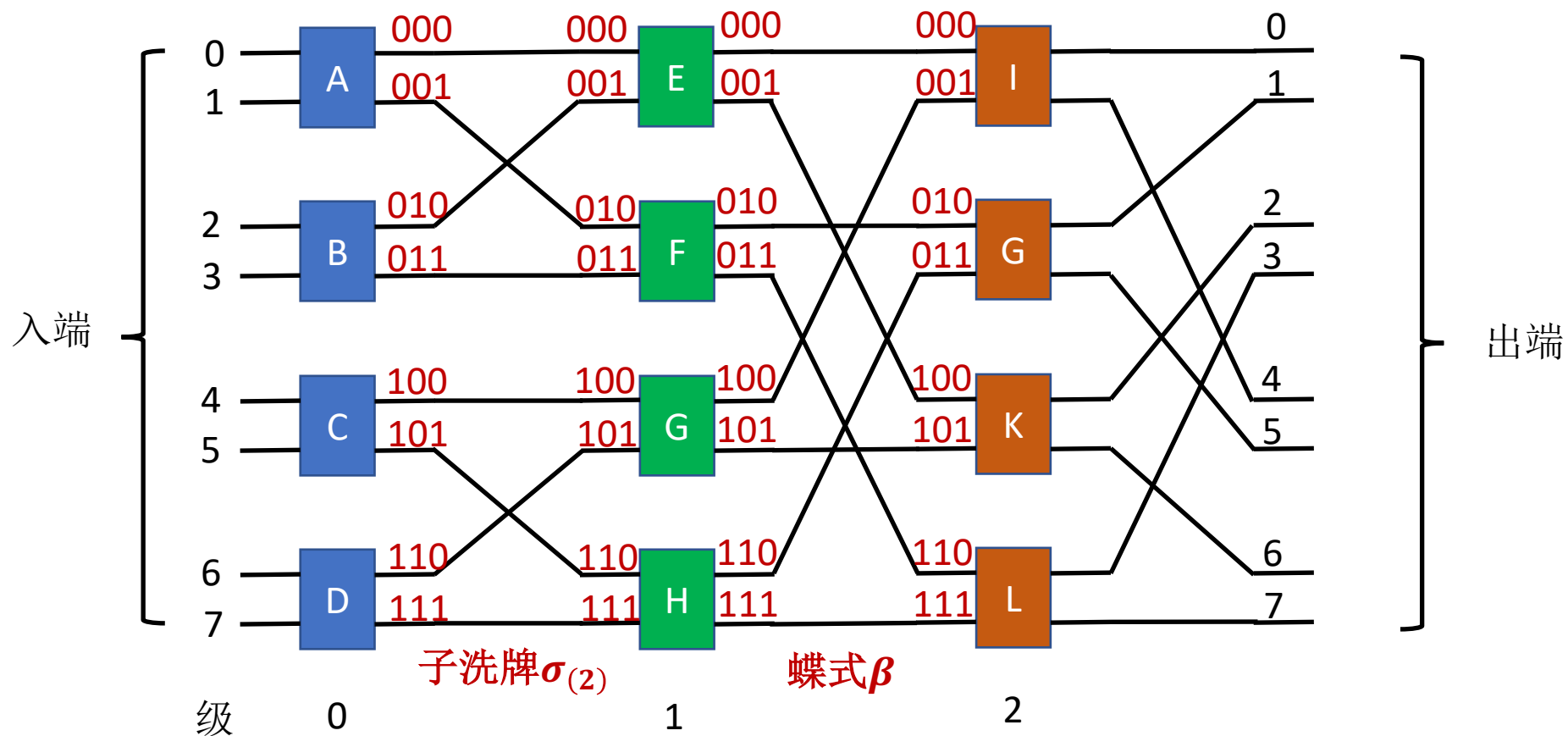
7.4.3 多级互连网络

1 多级立方体网络



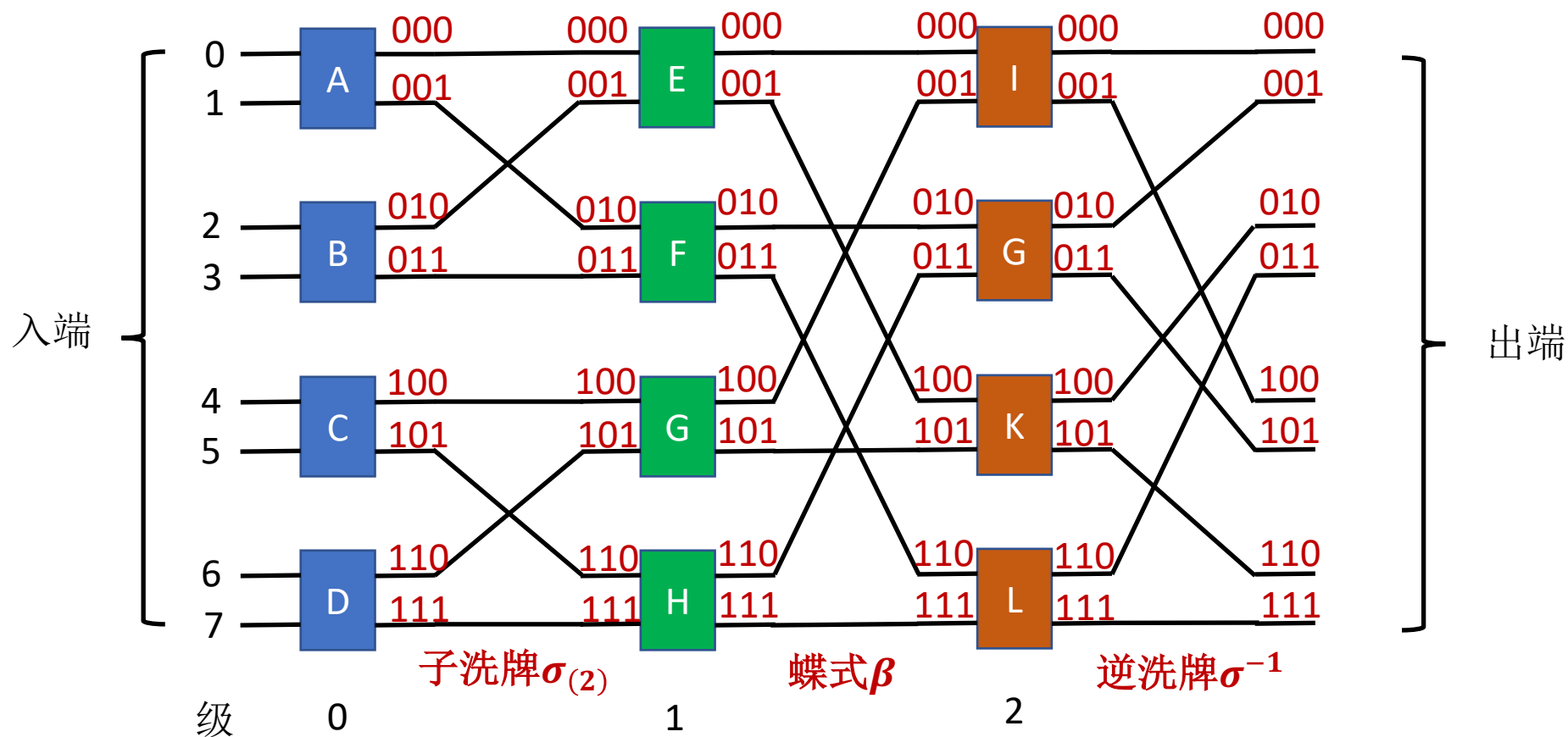
7.4.3 多级互连网络

1 多级立方体网络



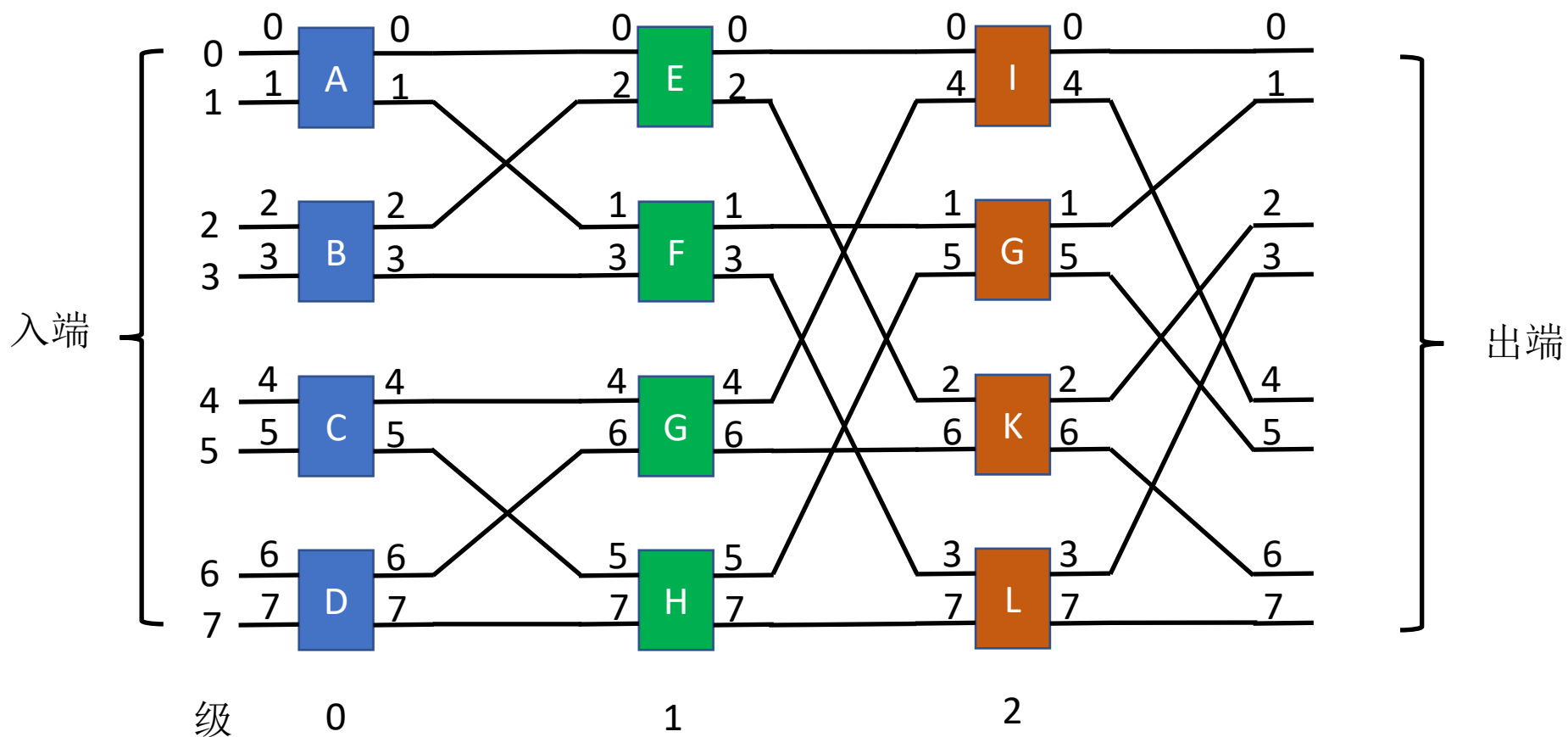
7.4.3 多级互连网络

1 多级立方体网络



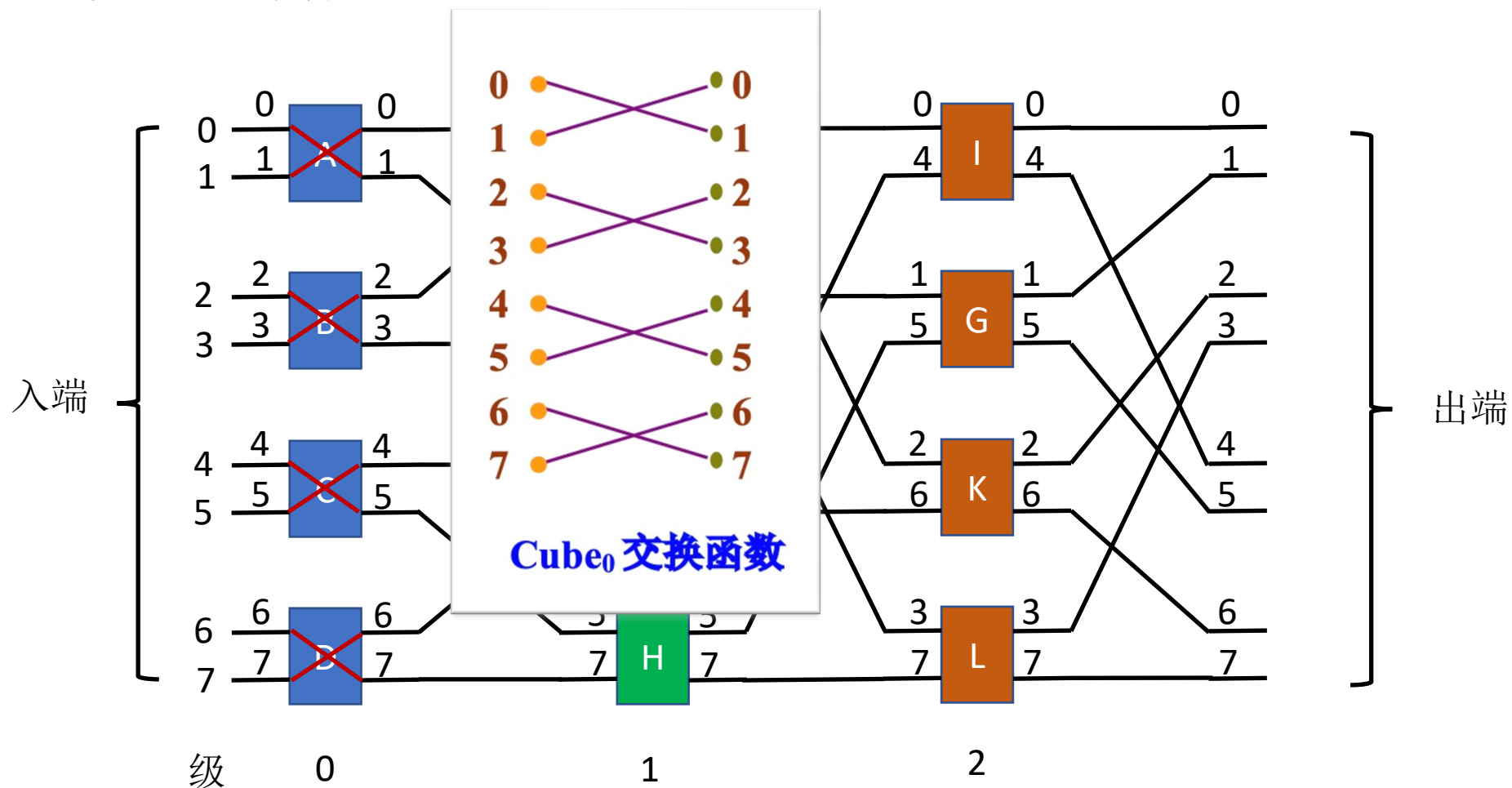
7.4.3 多级互连网络

1 多级立方体网络



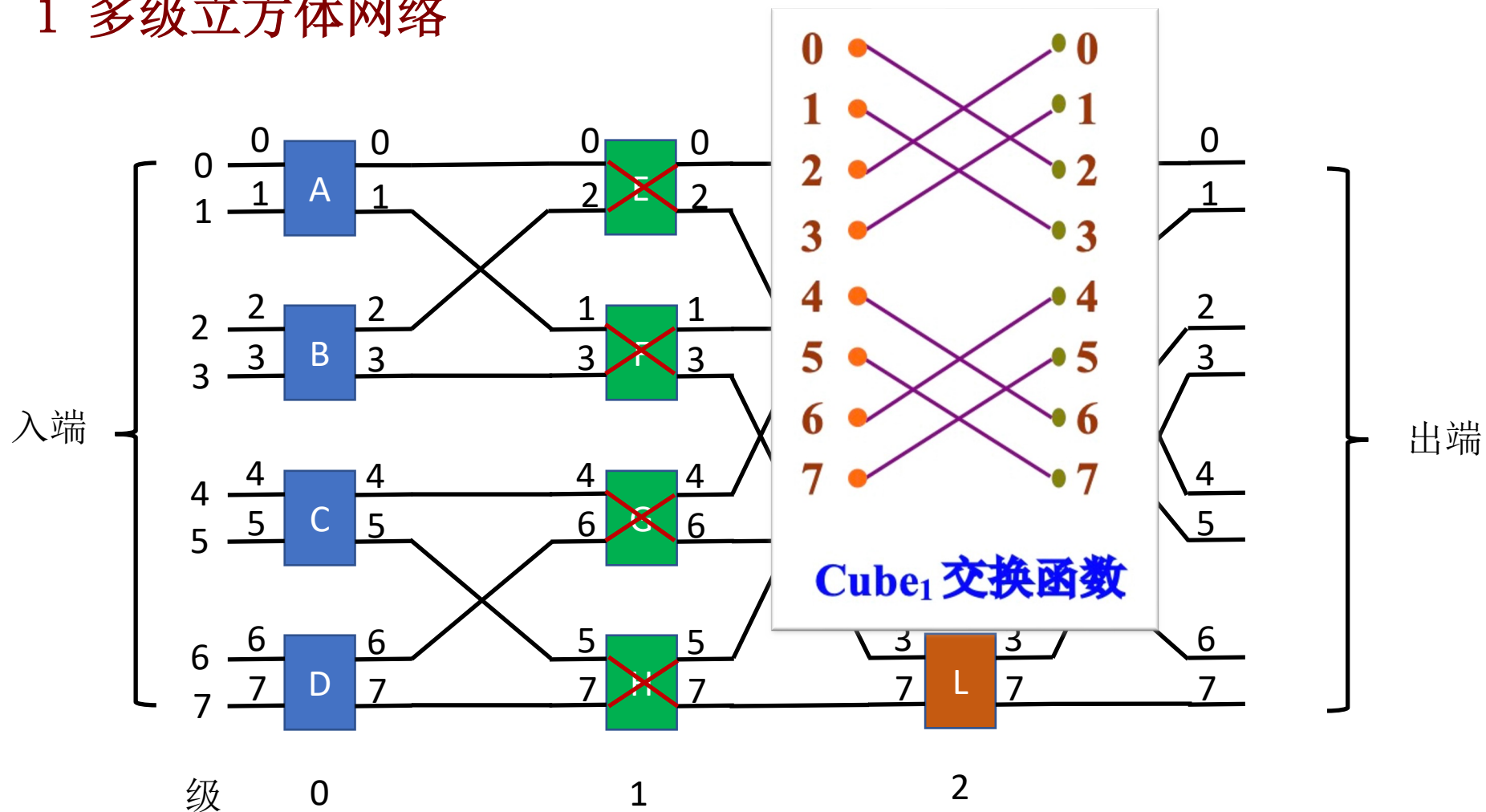
7.4.3 多级互连网络

1 多级立方体网络



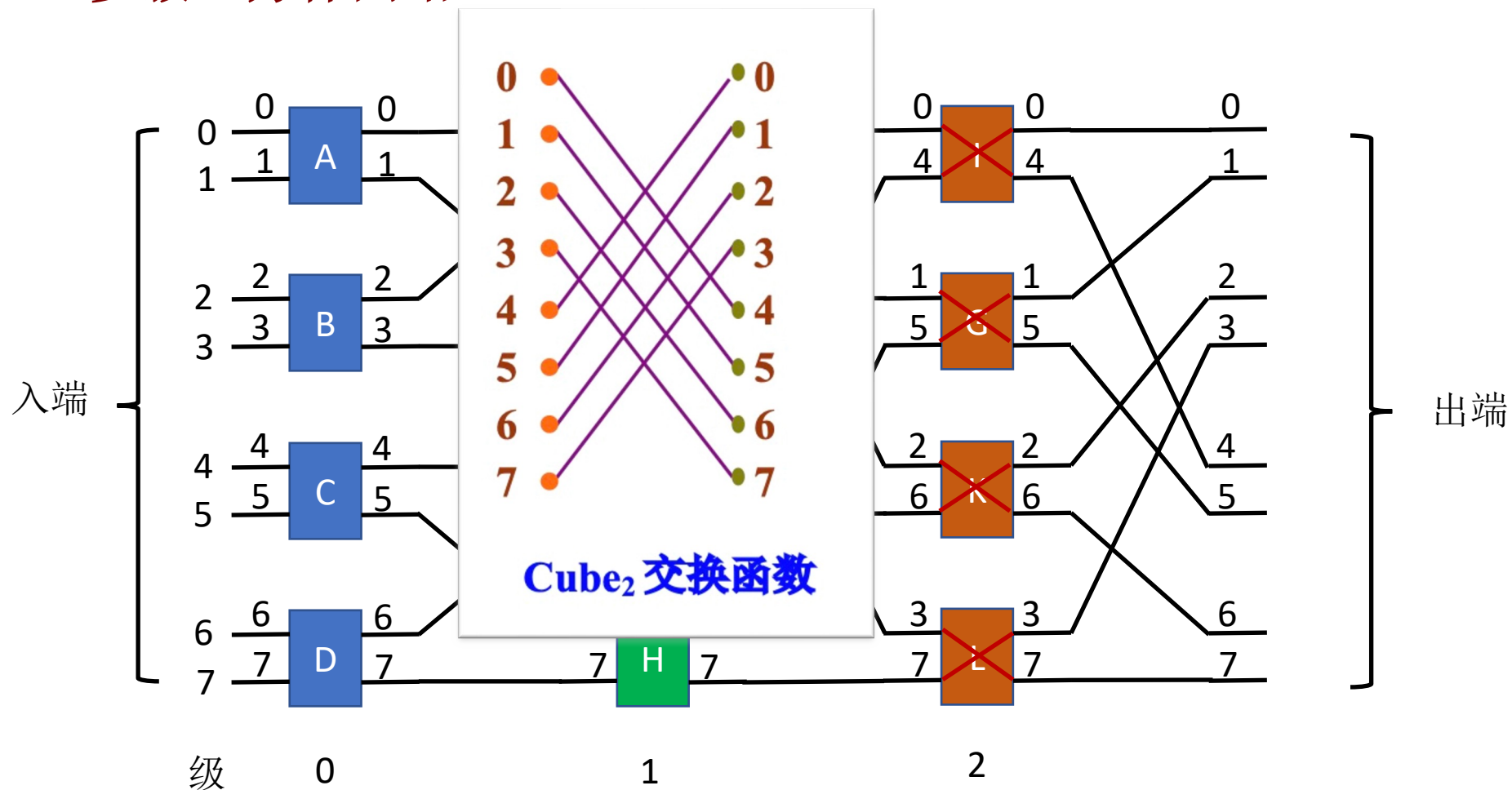
7.4.3 多级互连网络

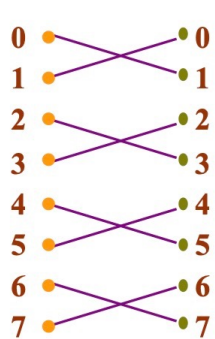
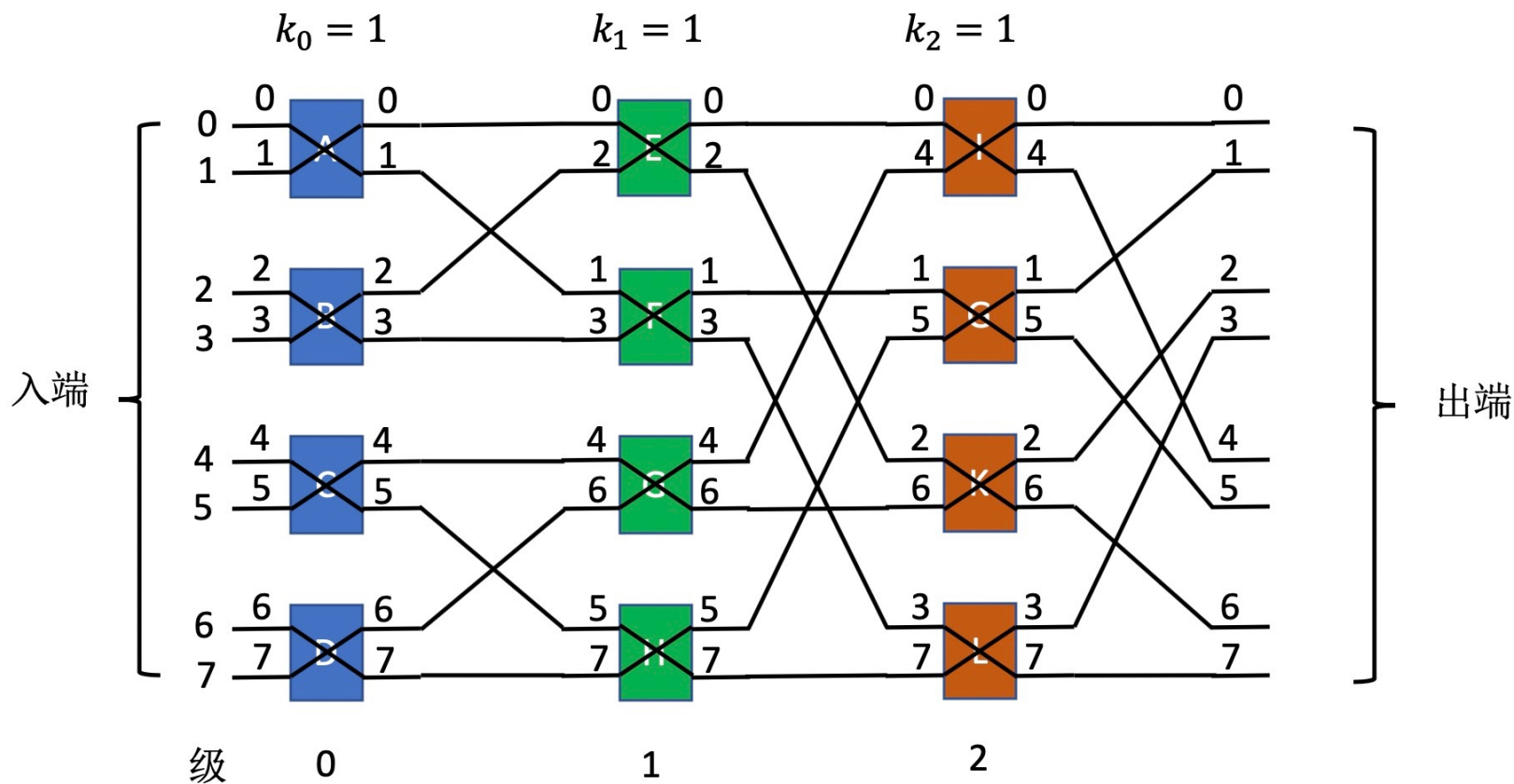
1 多级立方体网络



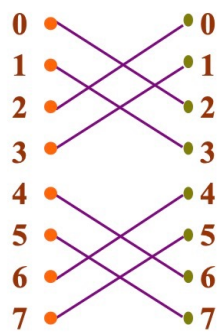
7.4.3 多级互连网络

1 多级立方体网络

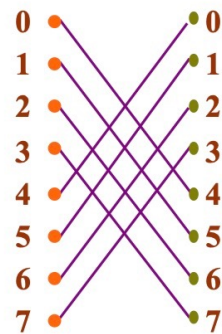




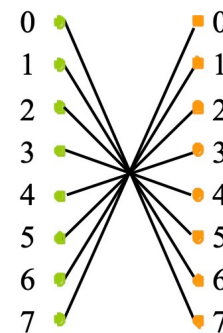
$Cube_0$ 交换函数



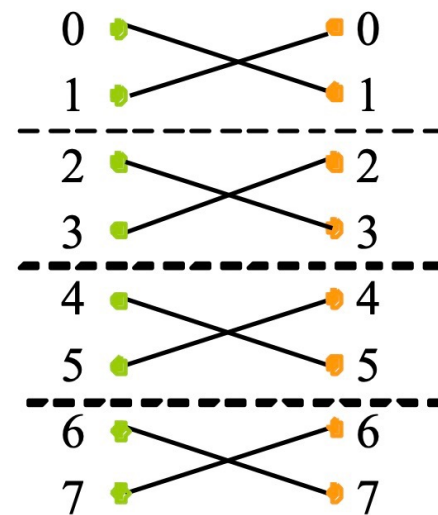
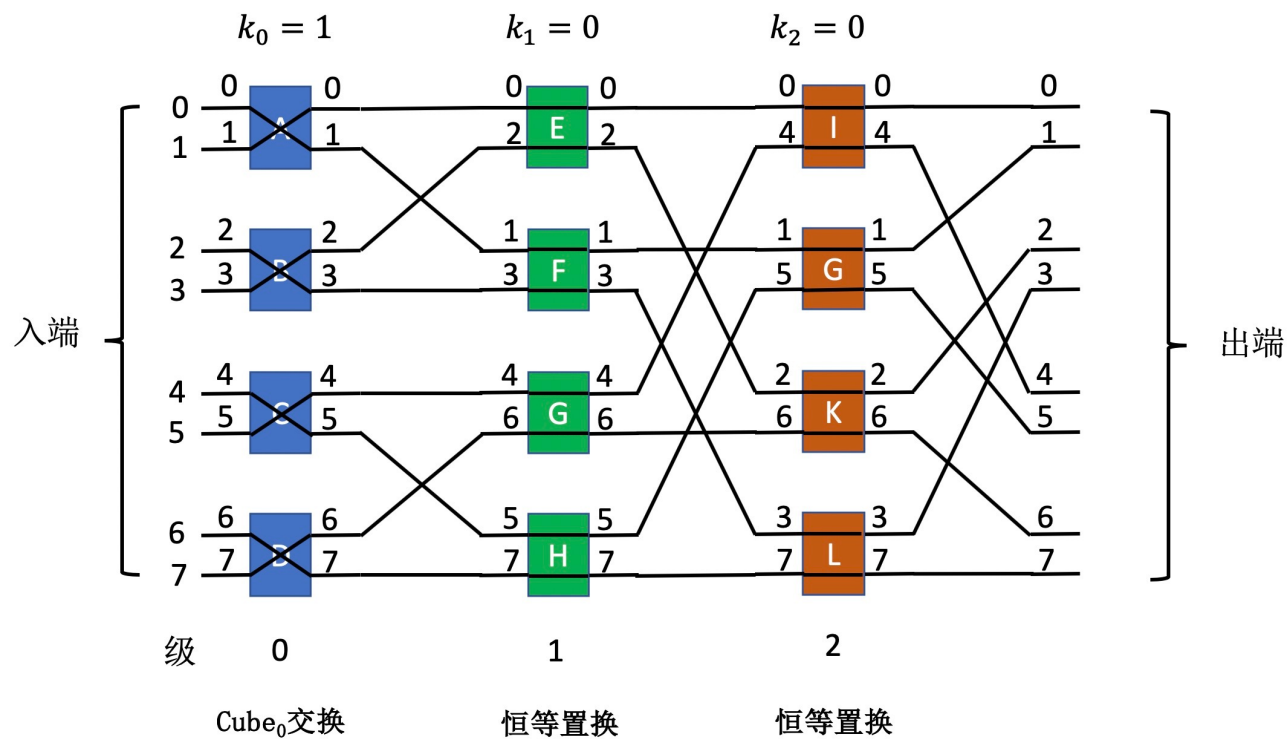
$Cube_1$ 交换函数

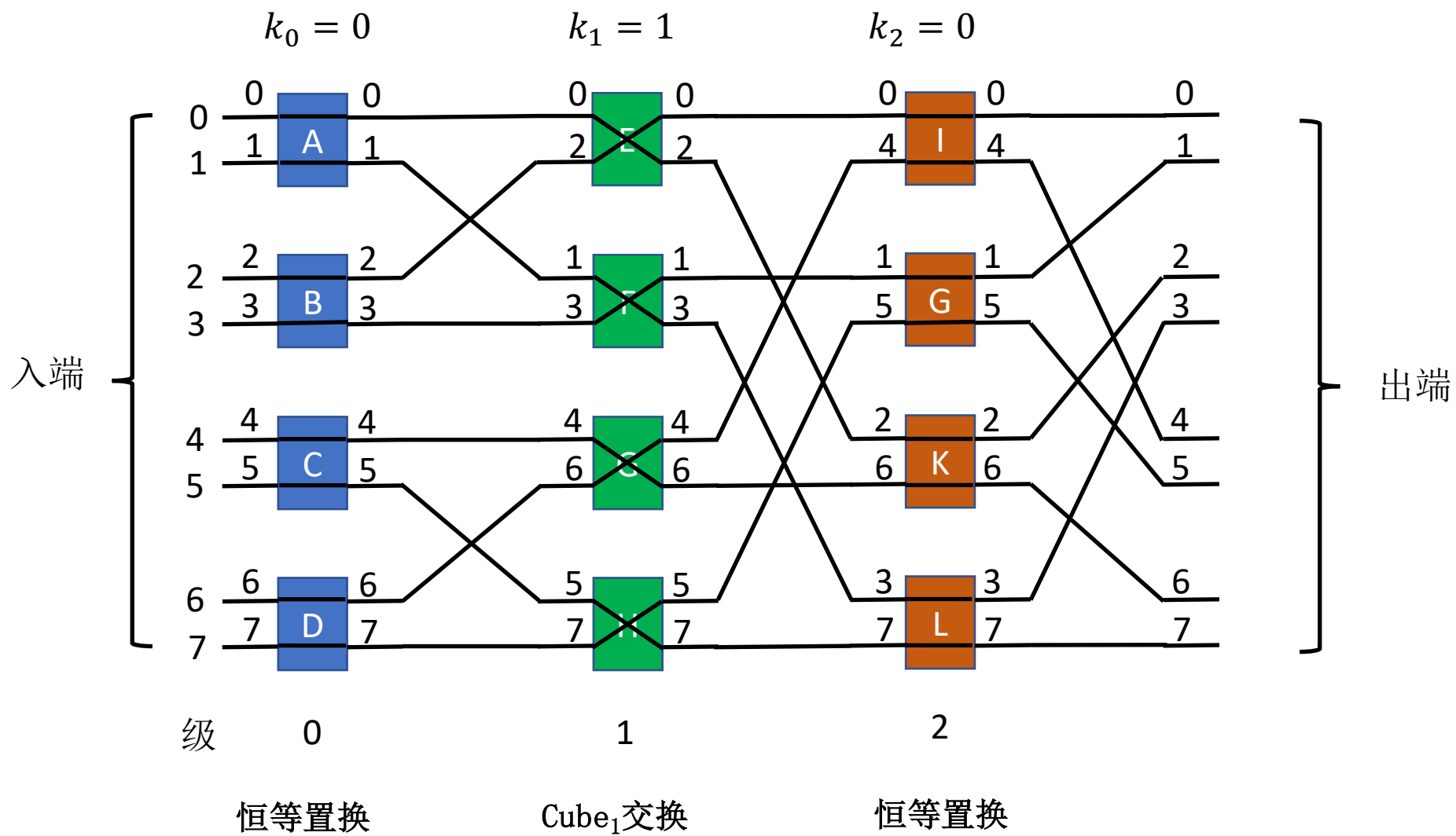


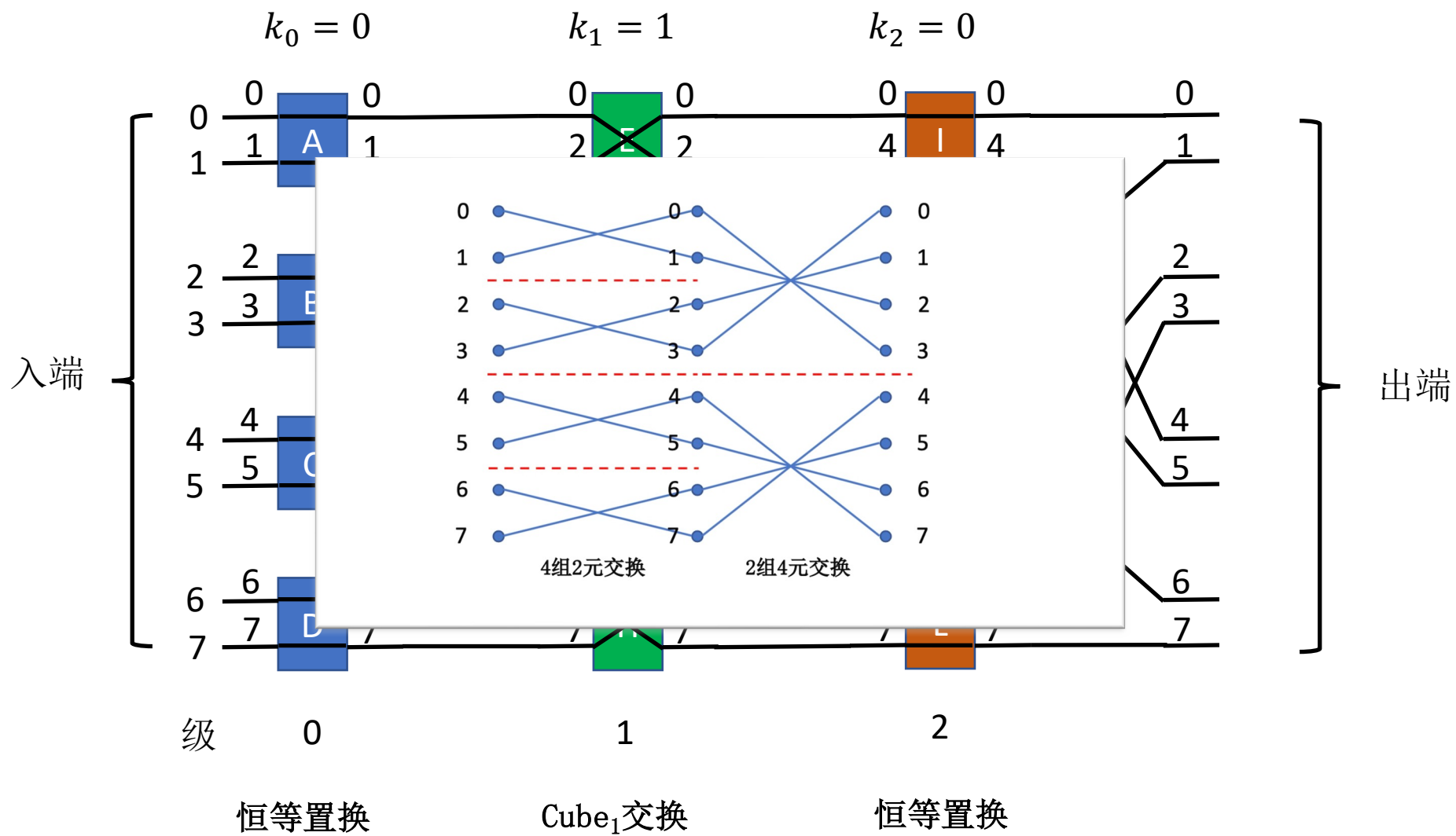
$Cube_2$ 交换函数



1组8元交换







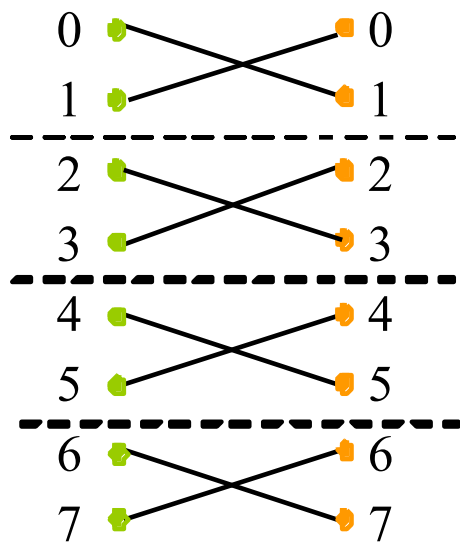
7.4.3 多级互连网络

1 多级立方体网络

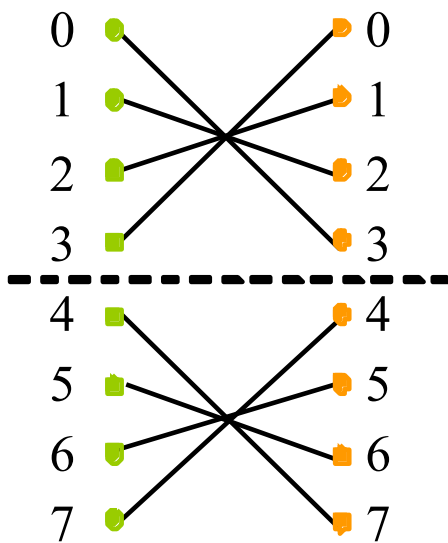
- STARAN网络采用级控制和部分级控制。
 - 采用级控制时，所实现的是交换功能；
 - 采用部分级控制时，则能实现移数功能。
- 间接二进制n方体网络则采用单元控制。
 - 具有更大的灵活性。
- 交换
 - 将有序的一组元素头尾对称地进行交换。

7.4.3 多级互连网络

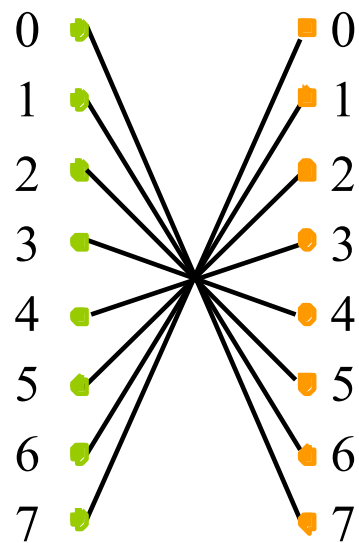
1 多级立方体网络



(a) 4 组 2 元交换



(b) 2 组 4 元交换



(c) 1 组 8 元交换

8个元素的基本交换图形

7.4.3 多级互连网络

1 多级立方体网络

- 3级STARAN网络在各种级控制信号的情况下所实现的入出端连接以及所实现的交换函数和功能。
 - $K_2k_1k_0$: 控制信号, k_i ($i=0, 1, 2$) 为第 i 级的级控制信号。
 - 从表中可以看出: 下面的4行中每一行所实现的功能可以从级控制信号为其反码的一行中所实现的功能加上1组8元变换来获得。
 - 例如: 级控制信号为110所实现的功能是其反码001所实现的4组2元交换再加上1组8元交换来获得。

级控制信号 $k_2k_1k_0$	连接的输出端号序列 (入端号序列: 01234567)	实现的分组交换	实现的互连函数
000	0 1 2 3 4 5 6 7	恒等	I
001	1 0 3 2 5 4 7 6	4组2元交换	Cube ₀
010	2 3 0 1 6 7 4 5	4组2元交换+ 2组4元交换	Cube ₁
011	3 2 1 0 7 6 5 4	2组4元交换	Cube ₀ +Cube ₁
100	4 5 6 7 0 1 2 3	2组4元交换+ 1组8元交换	Cube ₂
101	5 4 7 6 1 0 3 2	4组2元交换+ 2组4元交换+ 1组8元交换	Cube ₀ +Cube ₂
110	6 7 4 5 2 3 0 1	4组2元交换+ 1组8元交换	Cube ₁ +Cube ₂
111	7 6 5 4 3 2 1 0	1组8元交换	Cube ₀ +Cube ₁ + Cube ₂

➤ 当STARAN网络用作移数网络时，采用部分级控制

控制信号的分组和控制结果。

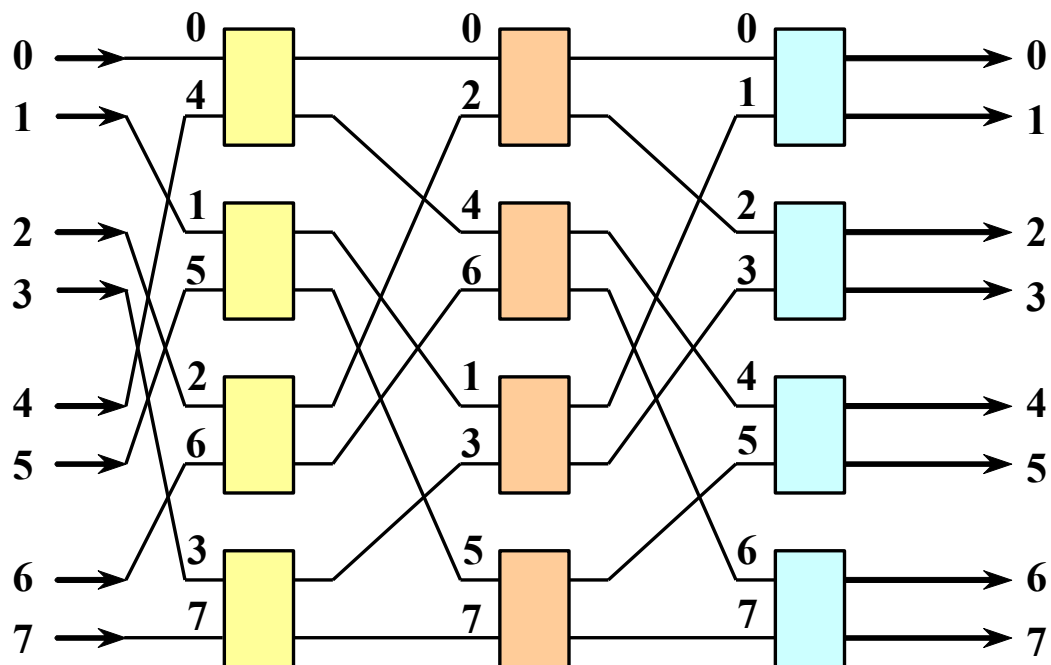
部分级控制信号						连接的输出端号序列 (入端号序列：01234567)	所实现的移 数功能
第0 级	第1 级		第2级				
A B C D	E G	F H	I	J	K L		
1	1	0	1	0	0	1 2 3 4 5 6 7 0	移1 mod 8
0	1	1	1	1	0	2 3 4 5 6 7 0 1	移2 mod 8
0	0	0	1	1	1	4 5 6 7 0 1 2 3	移4 mod 8
1	1	0	0	0	0	1 2 3 0 5 6 7 4	移1 mod 4
0	1	1	0	0	0	2 3 0 1 6 7 4 5	移2 mod 4
1	0	0	0	0	0	1 0 3 2 5 4 7 6	移1 mod 2
0	0	0	0	0	0	0 1 2 3 4 5 6 7	不移 全等

部分级控制：第*i*级的所有开关分别用*i*+1个信号控制， $0 \leq i \leq n-1$ ，*n*为级数。

7.4.3 多级互连网络

2 Omega网络（多级混洗交换网络）

- 一个 8×8 的Omega网络
- 每级由4个4功能的 2×2 开关构成
- 级间互连采用均匀洗牌连接方式



7.4.3 多级互连网络

2 Omega网络（多级混洗交换网络）

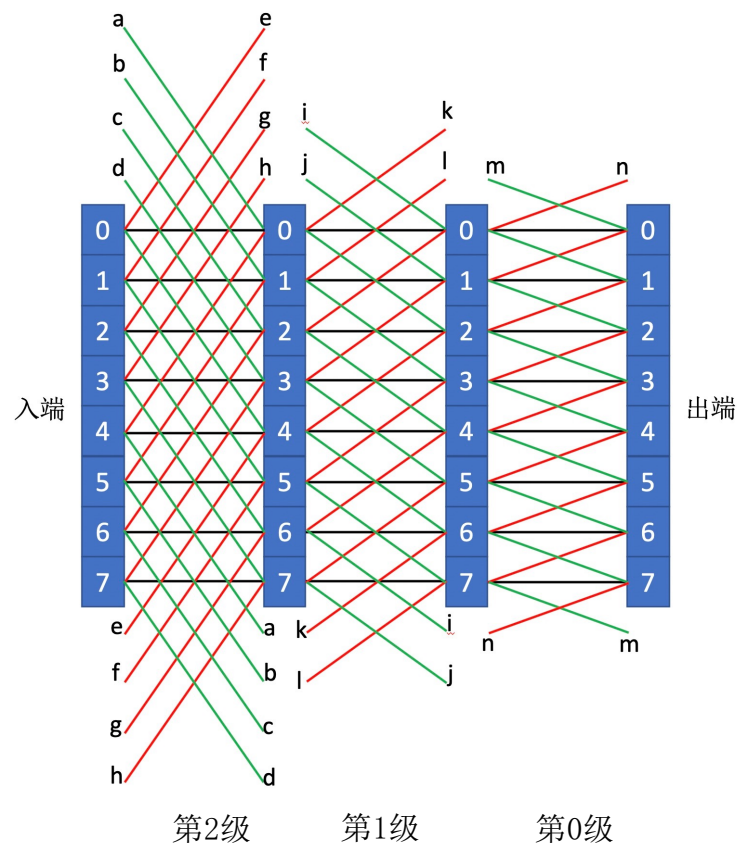
➤ 一个N输入的Omega网络

- 有 $\log_2 N$ 级，每级用 $N/2$ 个 2×2 开关模块，共需要 $N \log_2 N / 2$ 个开关。
- 每个开关模块均采用单元控制方式。
- 不同的开关状态组合可实现各种置换、广播或从输入到输出的其他连接。

7.4.3 多级互连网络

3 多级PM2I网络

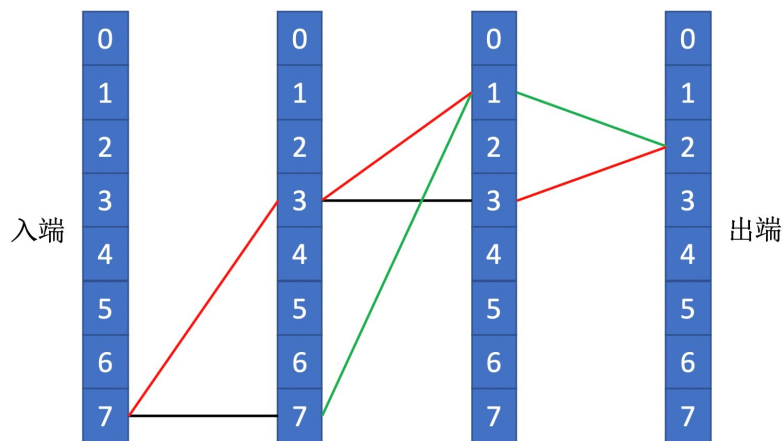
- 包含 n 级单元连接，每一级都把前后两列各 $N = 2^n$ 个单元按PM2I拓扑相互连接
- 第 i 级（ $0 \leq i \leq n-1$ ）中的每一个入单元 j 都有三根连接线分别通往出单元 j 、 $j + 2^i \bmod N$ 、和 $j - 2^i \bmod N$



7.4.3 多级互连网络

3 多级PM2I网络

- 包含 n 级单元连接，每一级都把前后两列各 $N = 2^n$ 个单元按PM2I拓扑相互连接
- 第 i 级（ $0 \leq i \leq n-1$ ）中的每一个入单元 j 都有三根连接线分别通往出单元 j 、 $j + 2^i \bmod N$ 、和 $j - 2^i \bmod N$
- 可以提供冗余路径，便于电路集成，提高可靠性

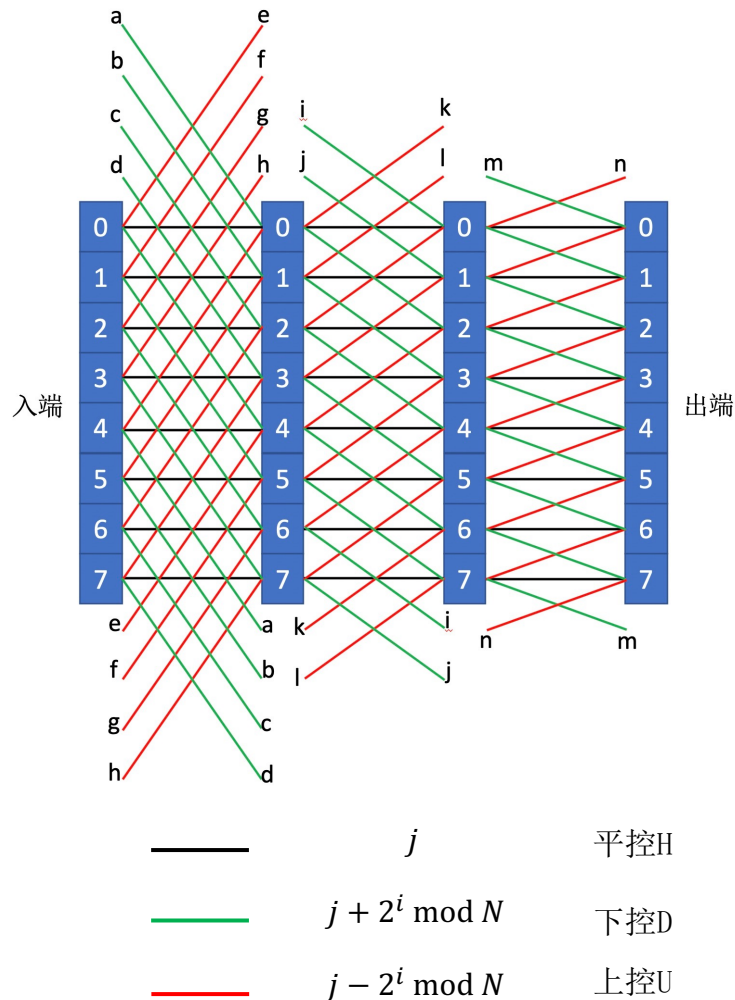


从入端口7到出端口2，有多条路径

7.4.3 多级互连网络

3 多级PM2I网络

- 包含 n 级单元连接，每一级都把前后两列各 $N = 2^n$ 个单元按PM2I拓扑相互连接
- 第 i 级（ $0 \leq i \leq n-1$ ）中的每一个入单元 j 都有三根连接线分别通往出单元 j 、 $j + 2^i \bmod N$ 、和 $j - 2^i \bmod N$
- 可以提供冗余路径，便于电路集成，提高可靠性
- 控制三种连接线的信号分别称为平控H、下控D和上控U
 - 数据变换网络（Data Manipulator, DM）：对于第 i 级，让 H_1^i, D_1^i, U_1^i 控制第 i 位为0的那些入单元，而让 H_2^i, D_2^i, U_2^i 控制第 i 位为1的那些入单元
 - 强化数据变换网络（Augmented Data Manipulator, ADM）：采用单元控制增强对各级单元控制的灵活性，让每一个单元都有自己独立的控制信号，但是由于控制线较多，成本较高



7.4.3 多级互连网络

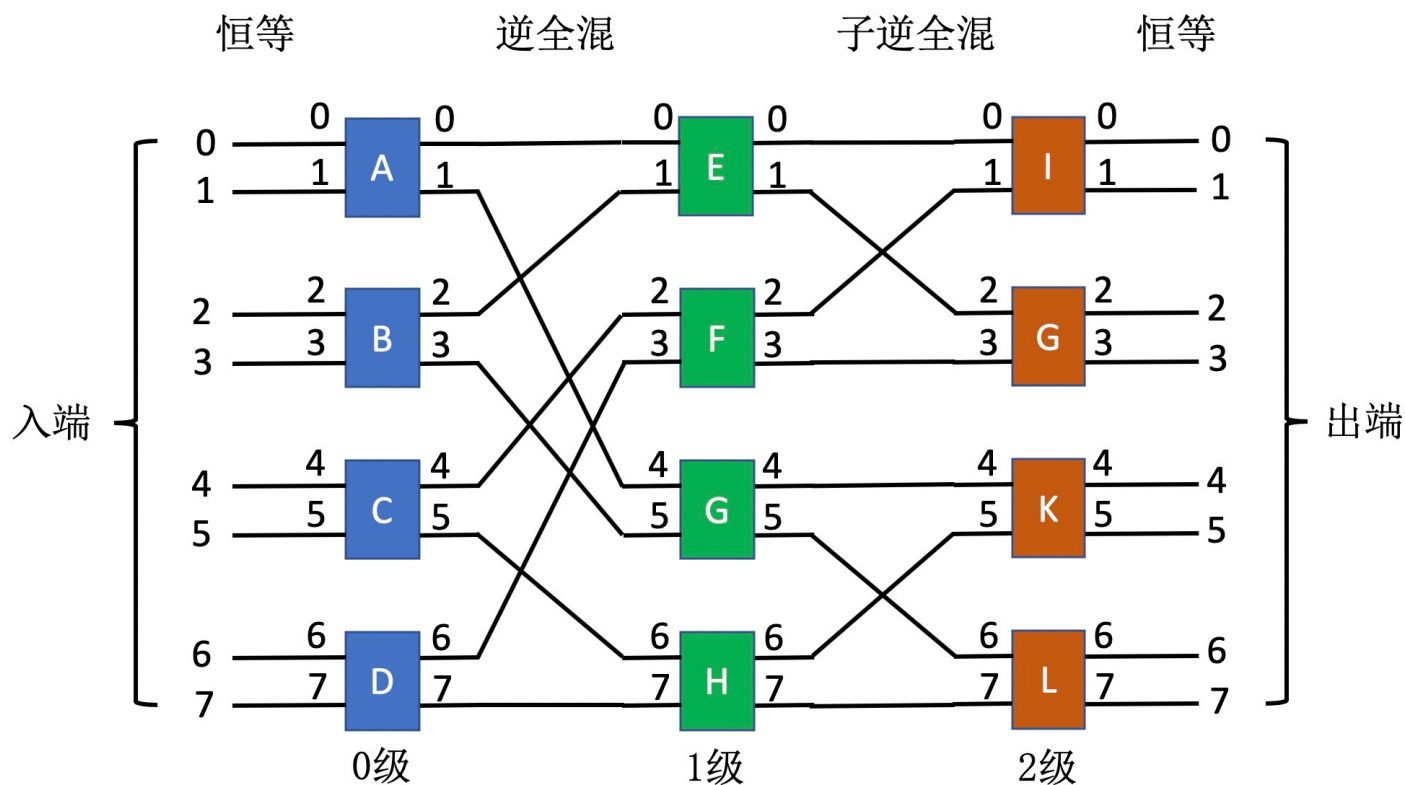
比较

- 灵活性由低到高：
 - 级控制立方体 < 部分级控制立方体 < 间接二进制立方体 < Ω < ADM
- 复杂性和成本由低到高：
 - 级控制立方体 < 部分级控制立方体 < 间接二进制立方体 < Ω < ADM
- 不同用途
 - STARAN网络和 Ω 网络都是为了进行存储器和处理单元之间的数据变换
 - 间接二进制n方体网络是为了连接成微处理器阵列

7.4.3 多级互连网络

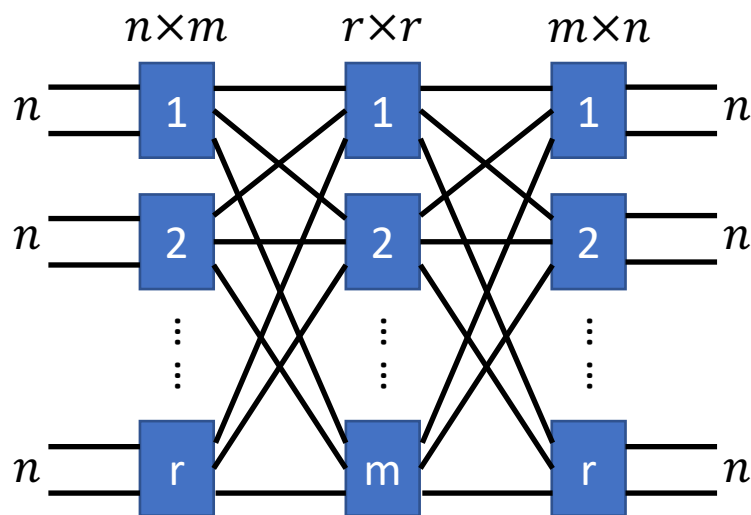
4 基准网络

➤ 所用交换单元均为二功能，采取单元控制



7.4.3 多级互连网络

5 多级交叉开关网络



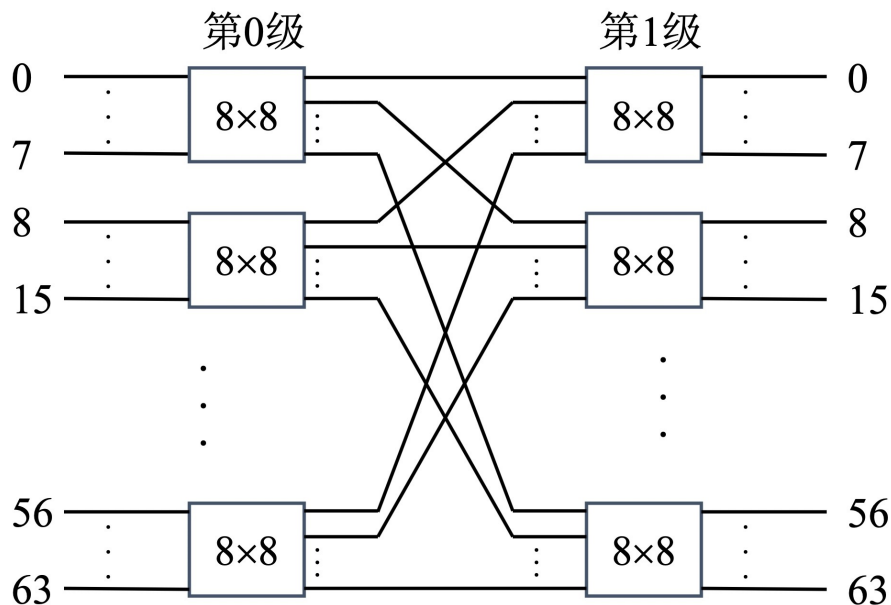
三级交叉开关网络的结构

- 其网络的入、出端口数均为 $n \times r$, 输入级有 r 个 $n \times m$ 的交叉开关, 中间级有 m 个 $r \times r$ 的交叉开关, 输出级有 r 个 $m \times n$ 的交叉开关
- 当 $m \geq 2n - 1$ 时, 该网络就成了非阻塞式网络, 可同时实现两对或多对入、出端间的连接, 均不会发生传送路径上的冲突

7.4.3 多级互连网络

6 多级蝶式网络

- 蝶式网络的开关不允许广播功能，它实际上是Omega网的一个子集
- 两级64×64的蝶式网络如下图所示：它采用16个8×8交叉开关构成，两级间采用8路洗牌连接。



7.4.3 多级互连网络

7 全排列网络

- 非阻塞型网络：所有入端、出端的连接均不发生冲突的网络，又称非阻塞型网络(全排列网络)
 - $N_{\text{入}} \rightarrow N_{\text{出}}$ 有 $N!$ 种排列
 - 灵活性好，连线多，控制复杂，成本高
- 阻塞型网络（Blocking Network）：同时实现两对或多对入端与出端之间连接时，都有可能因争用数据传送路径而发生冲突。
 - STARAN等网络属于阻塞型网络

7.4.4 动态互连网络的比较

网络特性	总线系统	多级网络	交叉开关
单位数据传送的最小时延	恒定	$O(\log_k n)$	恒定
每台处理机的带宽	$O(w/n)$ 至 $O(w)$	$O(w)$ 至 $O(nw)$	$O(w)$ 至 $O(nw)$
连线复杂性	$O(w)$	$O(nw \log_k n)$	$O(n^2 w)$
开关复杂性	$O(n)$	$O(n \log_k n)$	$O(n^2)$
连接特性和寻径性能	一次只能一对一	只要网络不阻塞， 可实现某些置换和广播	全置换， 一次一个
典型计算机	Symmetry S1, Encore Multimax	BBNTC-2000 IBM RP3	Cray Y-MP/816 Fujitsu VPP 500
说明	总线上假定有 n 台处理机；总线宽度为 w 位	$n \times n$ MIN采用 $k \times k$ 开关，其线宽为 w 位	假定 $n \times n$ 交叉开关的线宽为 w 位

7.5 阵列处理机

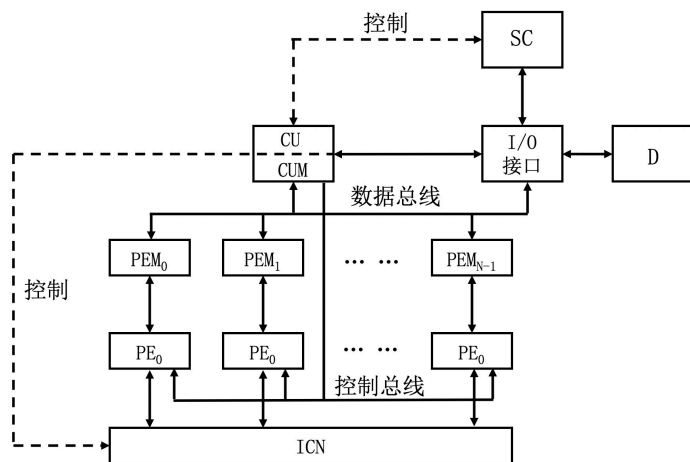
- 分布式存储器的阵列处理机构形
- 集中式共享存储器的阵列处理机构形
- 一台阵列处理机由五个部分组成
 - 多个处理单元PE
 - 多个存储器模块M
 - 一个控制器CU
 - 一个互连网络ICN
 - 一台输入输出处理机IOP

7.5 阵列处理机

- 分布式存储器的阵列处理机构形
- 集中式共享存储器的阵列处理机构形
- 一台阵列处理机由五个部分组成
 - 多个处理单元PE
 - 多个存储器模块M
 - 一个控制器CU
 - 一个互连网络ICN
 - 一台输入输出处理机IOP

7.5 阵列处理机

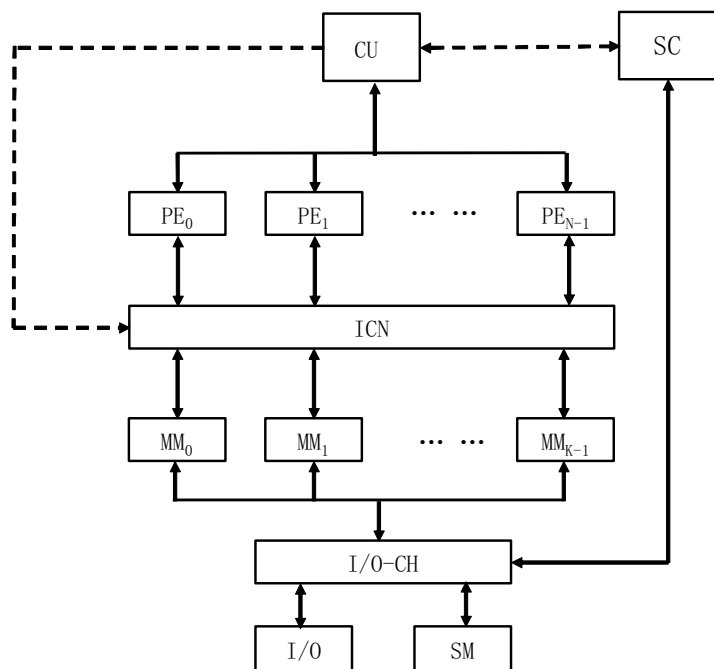
➤ 分布式存储器的阵列处理机



- 各处理单元设有局部存储器PEM（Processing Element Memory），存放被分布的数据；只能被本处理单元直接访问
- 控制部件CU设有存放程序和数据的主存储器
- 整个系统在CU控制下运行用户程序和部分系统程序
- 处理单元之间可通过互连网络ICN（Interconnection Network）
- 目前的大部分阵列处理机是基于分布式存储器模型的系统

7.5 阵列处理机

➤ 集中式共享存储器的阵列处理机构形



- 存储器由K个存储体集中组成，经互连网络ICN为全部N个处理单元所共享
- 互连网络用于在处理单元与存储体分体之间进行转接而构成数据通路

7.5 阵列处理机

➤ 阵列处理机的特点

- 背景：科学计算
 - 有限差分、矩阵、信号处理、线性规划
 - 数组、向量处理
- 资源重复，利用并行性中的同时性
- 简单而规整的互联网络——设计重点
- 专用机
- 与并行算法紧密相联

7.5 阵列处理机

➤ 阵列处理机实质上是由

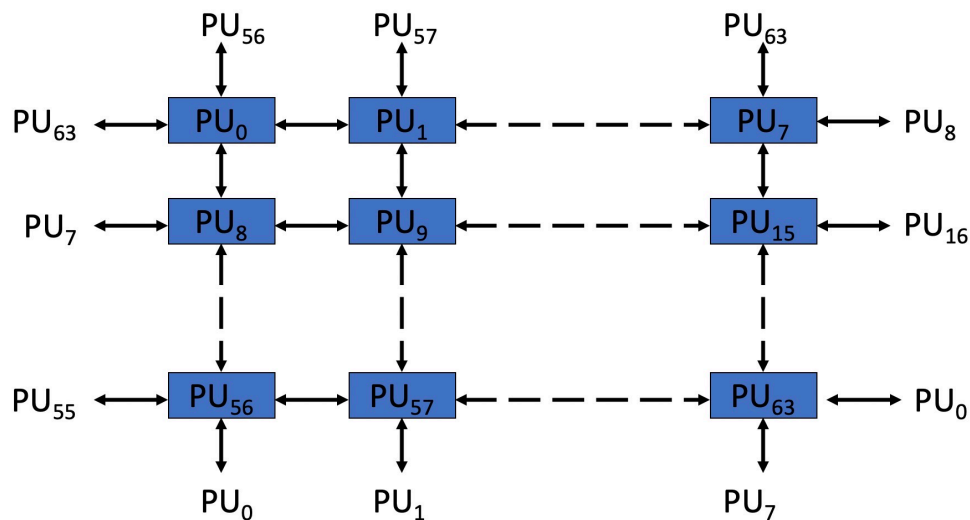
- 专门对付数组运算的处理单元阵列组成的处理机
- 专门从事处理单元阵列的控制及标量处理的处理机
- 专门从事系统输入输出及操作系统管理的处理机

➤ 组成的一个异构型多处理机系统

7.5 阵列处理机

➤ ILLIAC IV阵列机

- 分布式存储器构形
- 闭合螺线阵列，任意单元的最短距离不超过7步
- 一般来讲： $N = \sqrt{N} \times \sqrt{N}$ 个处理单元组成的阵列中，任意两个处理单元之间的最短距离不会超过 $\sqrt{N} - 1$ 步
- 处理单元为通常的累加型运算器，把累加寄存器RGA中的数据和存储器来的数据进行操作



7.5 阵列处理机

➤ ILLIAC IV阵列机矩阵加

- 两个8*8矩阵相加 ($A+B=C$)，把分量放在每一个PEM内

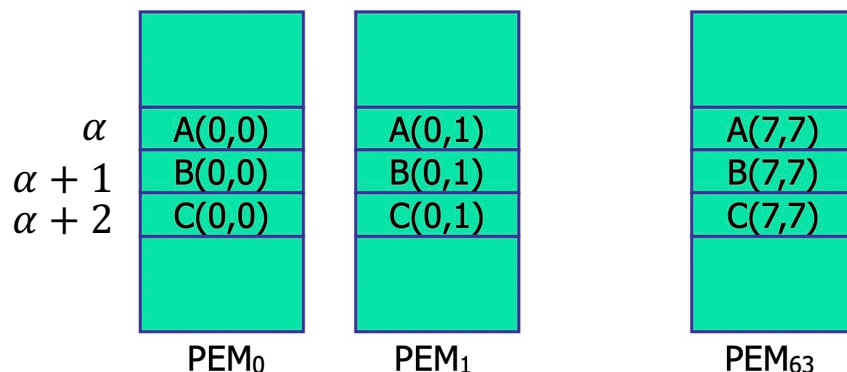
- 算法：

LDA ALPHA ; 全部(α)由PEM_i送PE_i的累加器RGA_i

ADRN ALPHA+1 ; 全部($\alpha+1$)与(RGA_i)浮点加，结果送RGA_i

STA ALPHA+2 ; 全部(RGA_i)由PE_i送PEM_i的 $\alpha+2$ 单元

- 说明：速度提高64倍；信息如何分布于局部存储器的算法与系统结构及求解问题直接相关



7.5 阵列处理机

➤ ILLIAC IV阵列机矩阵加

- 两个8*8矩阵相加 ($A+B=C$)，把分量放在每一个PEM内

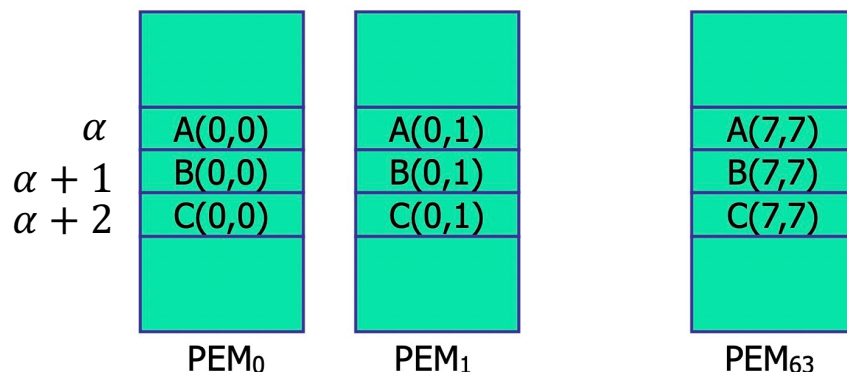
- 算法：

LDA ALPHA ; 全部(α)由PEM_i送PE_i的累加器RGA_i

ADRN ALPHA+1 ; 全部($\alpha+1$)与(RGA_i)浮点加，结果送RGA_i

STA ALPHA+2 ; 全部(RGA_i)由PE_i送PEM_i的 $\alpha+2$ 单元

- 说明：速度提高64倍；信息如何分布于局部存储器的算法与系统结构及求解问题直接相关



7.5 阵列处理机

➤ ILLIAC IV阵列机矩阵乘

- SISD 算法（需 $8*8*8=512$ 次运算）：

D0 10 I=0, 7

D0 10 J=0, 7

C(I, J)=0

D0 10 K=0, 7

10 C(I, J)=C(I, J)+A(I, K)*B(K, J)

7.5 阵列处理机

➤ ILLIAC IV阵列机矩阵乘

- SIMD算法（需 $8*8=64$ 次运算）：

DO 10 I=0, 7

C(I, J)=0

DO 10 K=0, 7

10 C(I, J)=C(I, J)+A(I, K)*B(K, J)

A(0,0)
A(1,0)
:
A(7,0)
B(0,0)
B(1,0)
:
B(7,0)
C(0,0)
C(1,0)
:
C(7,0)

PEM₀

A(0,1)
A(1,1)
:
A(7,1)
B(0,1)
B(1,1)
:
B(7,1)
C(0,1)
C(1,1)
:
C(7,1)

PEM₁

... ..

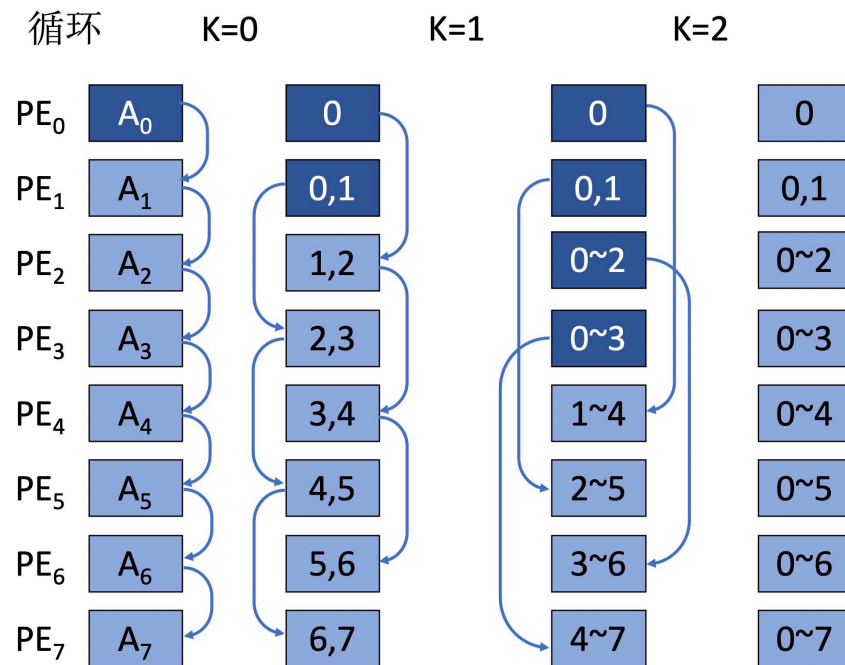
A(0,7)
A(1,7)
:
A(7,7)
B(0,7)
B(1,7)
:
B(7,7)
C(0,0)
C(1,7)
:
C(7,7)

PEM₇

7.5 阵列处理机

➤ ILLIAC IV阵列机累加和

1. 置全部 PE_i 为活跃状态, $0 \leq i \leq 7$
2. 全部 $A(I)$ 从 PEM_i 的 α 单元读到相应 PE_i 的累加寄存器 RGA_i 中, $0 \leq i \leq 7$;
3. 令 $k=0$;
4. 将全部 PE_i 的 (RGA_i) 转送到传送寄存器 RGR_i , $0 \leq i \leq 7$;
5. 将全部 PE_i 的 (RGR_i) 经过互连网络向右传送 2^k 步距, $0 \leq i \leq 7$;
6. 令 $j=2^k-1$
7. 置 PE_0 至 PE_j 为不活跃状态;
8. 处于活跃状态的所有 PE_i 执行 $(RGA_i) := (RGA_i) + (RGR_i)$, $j < i \leq 7$;
9. $k:=k+1$;
10. 如 $k < 3$, 则转回第4步, 否则往下继续执行;
11. 置全部 PE_i 为活跃状态, $0 \leq i \leq 7$;
12. 将全部 PE_i 的累加寄存器内容 (RGA_i) 存入相应 PEM_i 的 $\alpha+1$ 单元中, $0 \leq i \leq 7$ 。



7.5 阵列处理机

➤ 共享主存构形的阵列处理机中并行存储器的无冲突访问

0	1	2	3
a0	a1	a2	a3
a4	a5	a6	a7
a8	a9	a10	a11
a12	.	.	.

一维数组

- 连续访问 $m=4$ 个元素访问，无冲突
- 按2变址，冲突，降低一半
- m 应取质数，避免冲突（只要变址跳距与 m 互质）

7.5 阵列处理机

➤ 共享主存构形的阵列处理机中并行存储器的无冲突访问

0	1	2	3
a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23
a30	a31	a32	a33

二维数组

- 同时访问某一行、主对角线或次对角线上的所有元素，无冲突
- 同时访问某一列，冲突，降低成1/4

7.5 阵列处理机

➤ 共享主存构形的阵列处理机中并行存储器的无冲突访问

0	1	2	3
a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23
a30	a31	a32	a33

- 同时访问某一行、主对角线或次对角线上的所有元素，无冲突
- 同时访问某一列，冲突，降低成1/4

0	1	2	3
a00	a01	a02	a03
a13	a10	a11	a12
a22	a23	a20	a21
a31	a32	a33	a30

- 错位存放，行或列都能够并行访问，但是主对角线和次对角线上各元素的并行访问会产生冲突

7.5 阵列处理机

➤ 共享主存构形的阵列处理机中并行存储器的无冲突访问

0	1	2	3	4
a00	a01	a02	a03	
a13		a10	a11	a12
a21	a22	a23		a20
	a30	a31	a32	a33

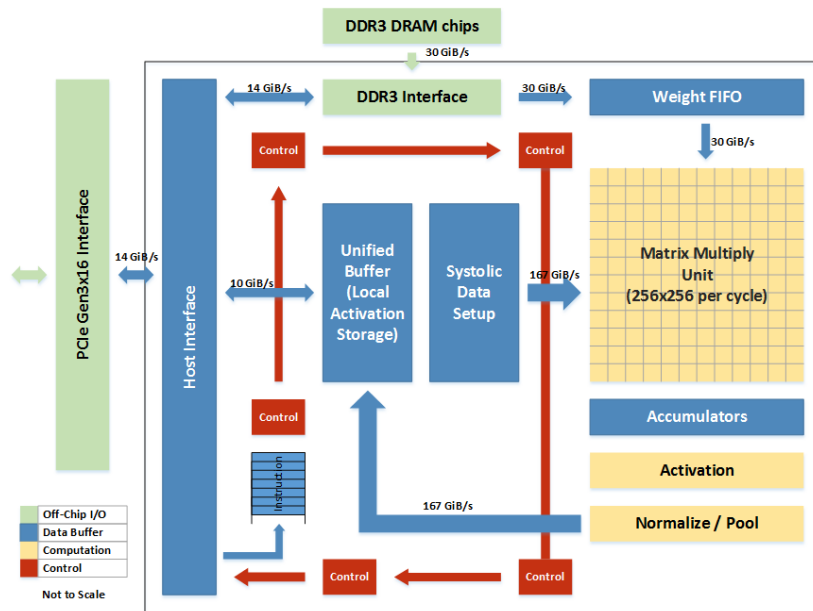
$$P = 1, m = 5, n = 4, \delta_1 = 2, \delta_2 = 1$$

- 假设 $n \times n$ 的二维数组在并行存储器中同一列两个相邻元素地址错开距离为 δ_1 ，同一行两个相邻元素错开距离为 δ_2 ，令 $m = 2^{2P} + 1$ （ P 为正整数），
- 实现无冲突访问的充分条件是 $\delta_1 = 2^P, \delta_2 = 1$
- $n \times n$ 的二维数组 A 中任意一个元素 A_{ab}

$$\begin{cases} \text{体号地址 } j = (a\delta_1 + b\delta_2 + c) \bmod m \\ \text{体内地址 } i = a \end{cases}$$

7.6 脉动阵列流水处理机

- 为要求计算量很大的信号/图像处理及科学计算的特定算法需要
- 卡内基-梅隆大学的美籍华人H. T. Kung于1978年提出脉动阵列处理（Systolic Array）机^[1]
- 最近被应用到Google TPU的设计当中^[2]



[1] H.T. Kung, "Why Systolic Architectures?", Computers, vol. 15, no. 1, pp. 37-46, 1982.

[2] N. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor processing Unit", in Proc. of the 44th ISCA, pp. 1-12, 2017.

7.6 脉动阵列流水处理机

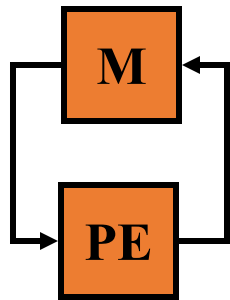
- 脉动阵列结构是由一组处理单元（PE）构成的阵列
- 每个PE内部结构相同，一般由一个加法/逻辑运算部件或加法/乘法运算部件在加上若干锁存器构成
- 阵列内所有PE的数据锁存器受一个时钟控制
- 运算时，数据在阵列结构的各个PE间沿各自的方向同步向前推进，就像血液受心脏有节奏地搏动在各条血管同步向前流动一样，因此被称为“脉动阵列结构”

7.6 脉动阵列流水处理机

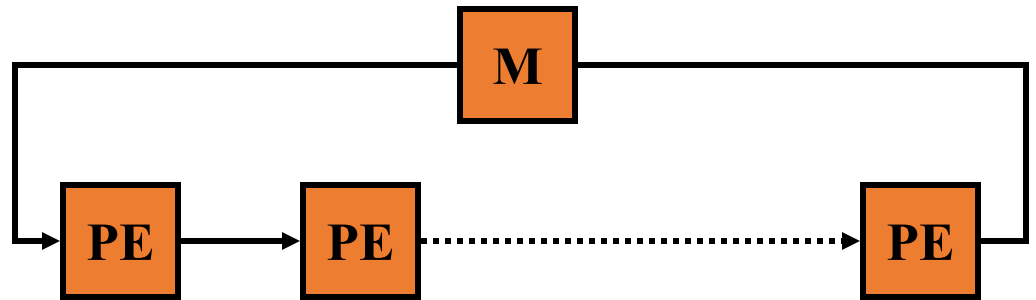
- Different from pipelining
 - Nonlinear array structure, multidirection data flow, each PE may have (small) local instruction and data memory
- Different from SIMD
 - Each PE may do something different
- Initial motivation
 - VLSI enables inexpensive special-purpose chips
 - Represent algorithms directly by chips connected in regular pattern

7.6 脉动阵列流水处理机

- Replace a PE with an array of PEs without increasing I/O bandwidth



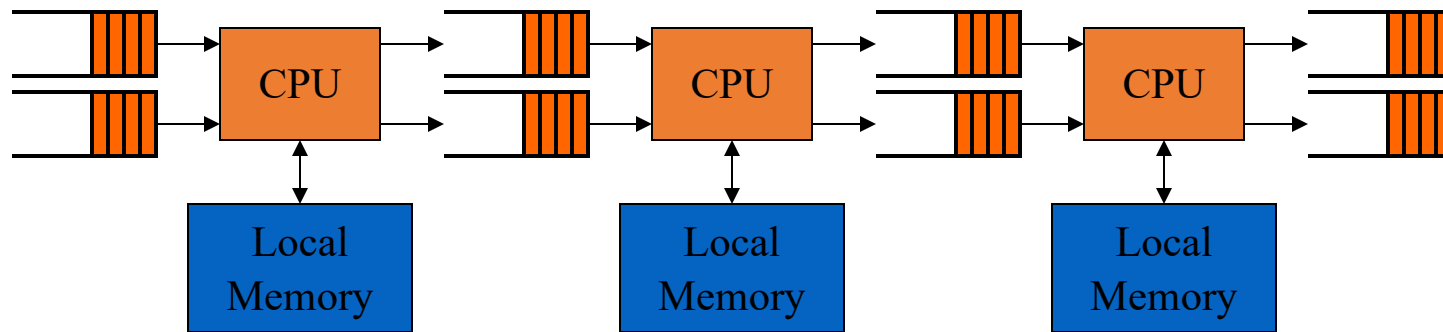
Conventional



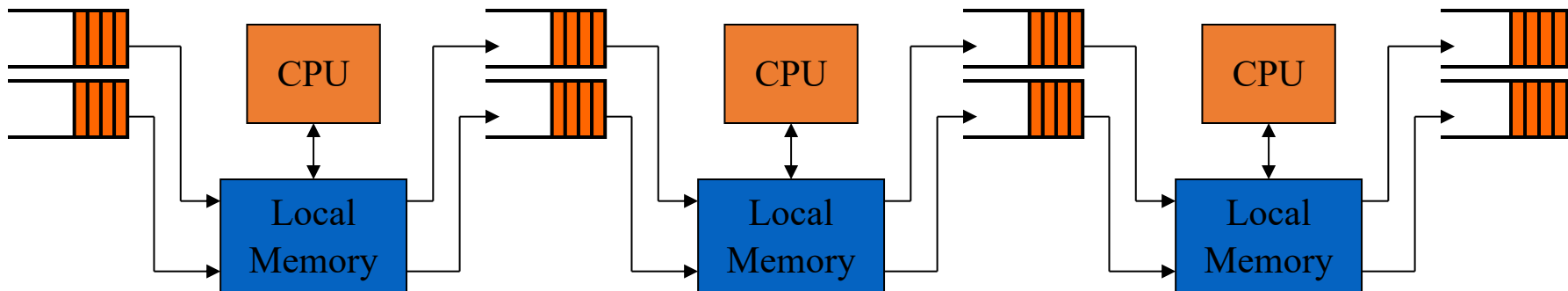
Systolic arrays

7.6 脉动阵列流水处理机

➤ Two communication styles



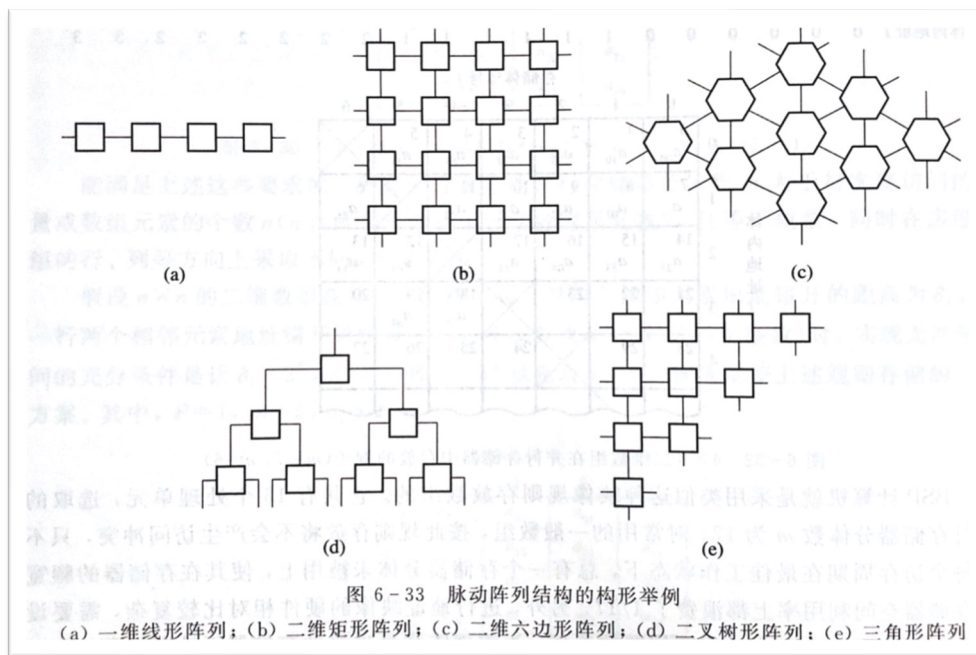
Systolic communications



Memory communications

7.6 脉动阵列流水处理机

- 阵列内部各个PE只接收前一组PE传来的数据，并向后一组PE发送数据
- 只有位于阵列边缘的处理单元，才与存储器或I/O端口进行数据通信



7.6 脉动阵列流水处理机

- 在一个脉动式二维阵列结构上进行两个3*3矩阵相乘
- 每一个处理单元PE含有一个乘法器和一个加法器，完成一个内积运算

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \quad C = A \bullet B = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

- Conventional Method: N^3

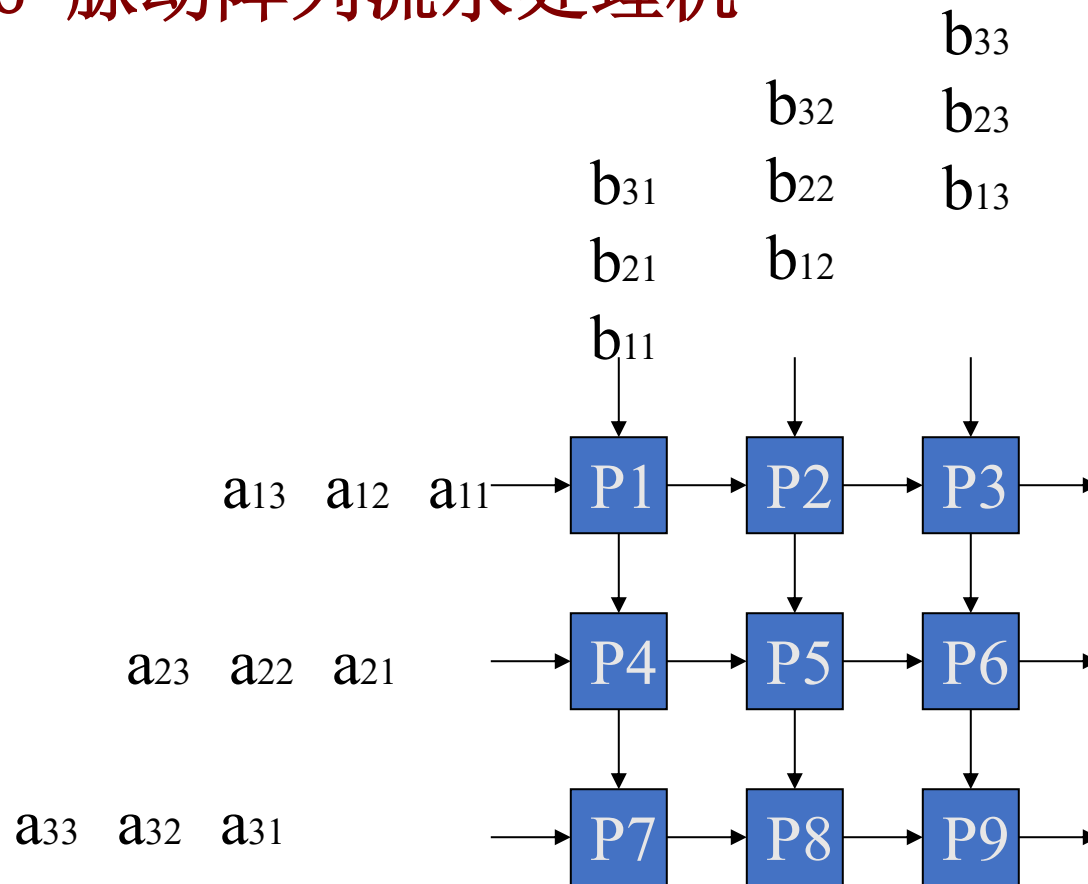
For I = 1 to N

For J = 1 to N

For K = 1 to N

$$C[I,J] = C[I,J] + A[J,K] * B[K,J];$$

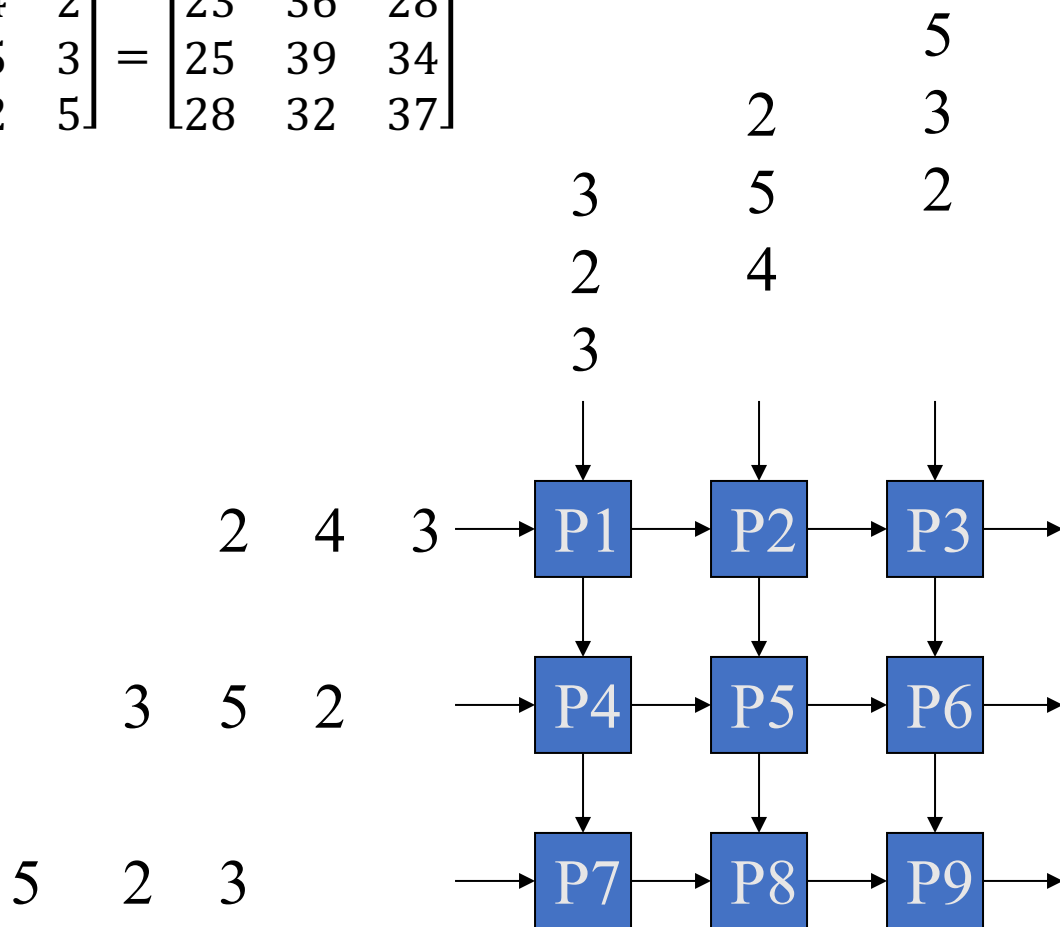
7.6 脉动阵列流水处理机



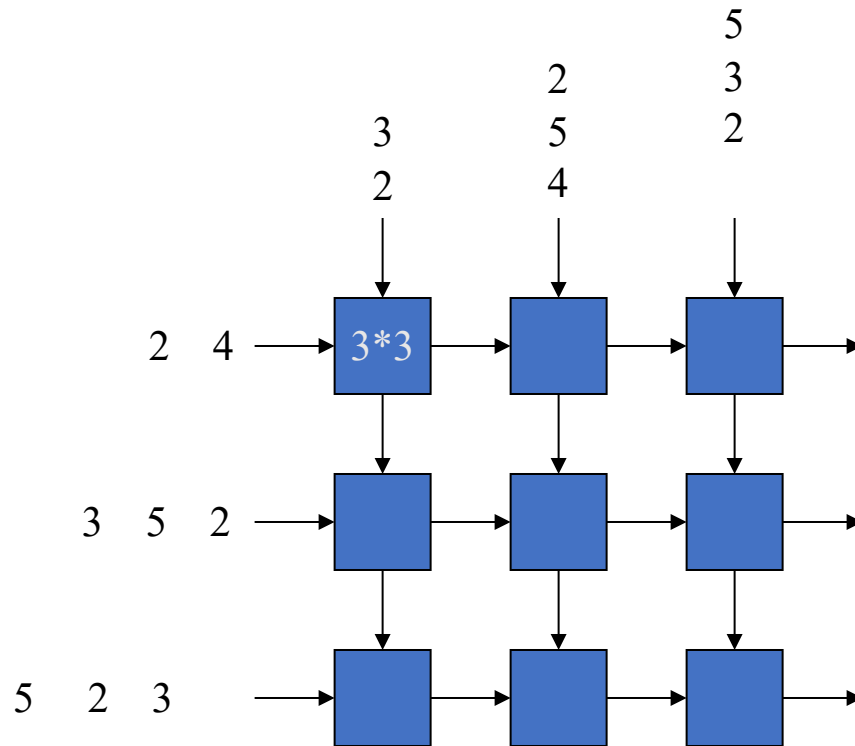
- At every tick of the global system clock data is passed to each processor from two different directions, then it is multiplied and the result is saved in a register.

7.6 脉动阵列流水处理机

$$\begin{bmatrix} 3 & 4 & 2 \\ 2 & 5 & 3 \\ 3 & 2 & 5 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 2 \\ 2 & 5 & 3 \\ 3 & 2 & 5 \end{bmatrix} = \begin{bmatrix} 23 & 36 & 28 \\ 25 & 39 & 34 \\ 28 & 32 & 37 \end{bmatrix}$$



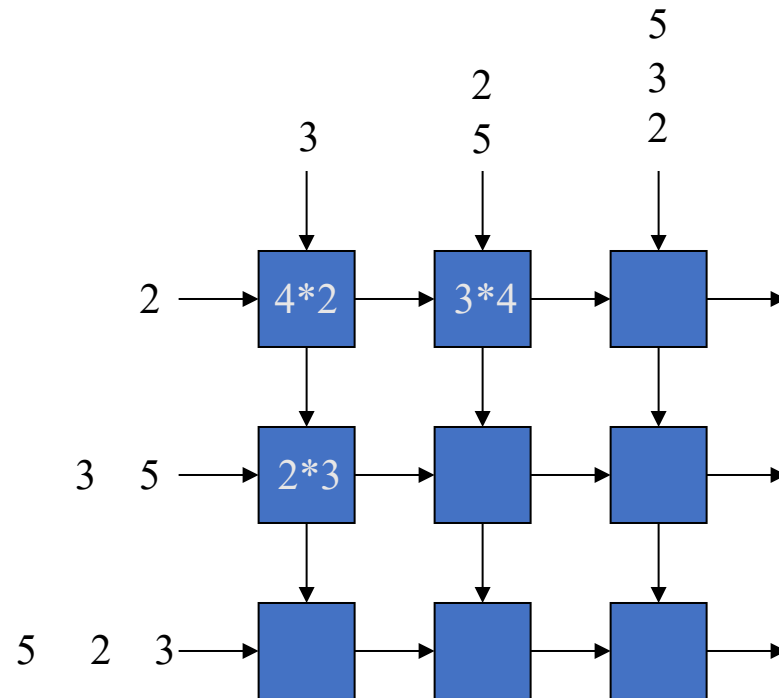
7.6 脉动阵列流水处理机



Clock tick: 1

P1	P2	P3	P4	P5	P6	P7	P8	P9
9	0	0	0	0	0	0	0	0

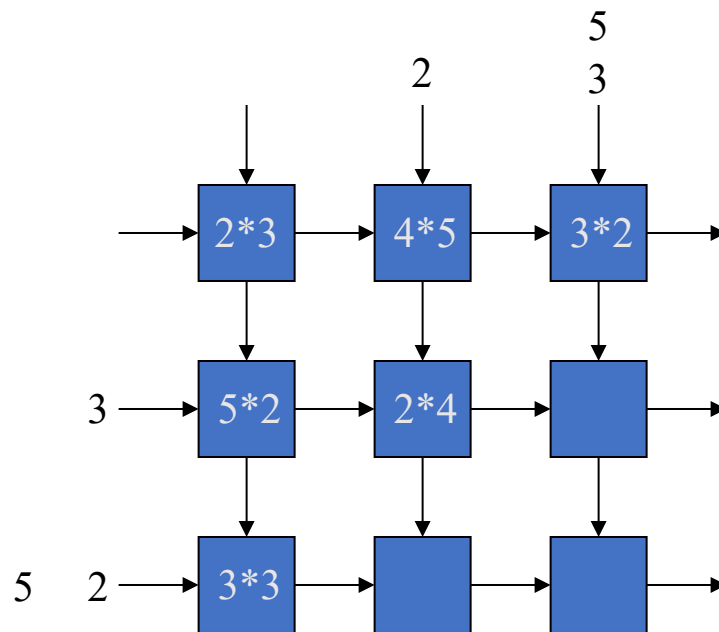
7.6 脉动阵列流水处理机



Clock tick: 2

P1	P2	P3	P4	P5	P6	P7	P8	P9
17	12	0	6	0	0	0	0	0

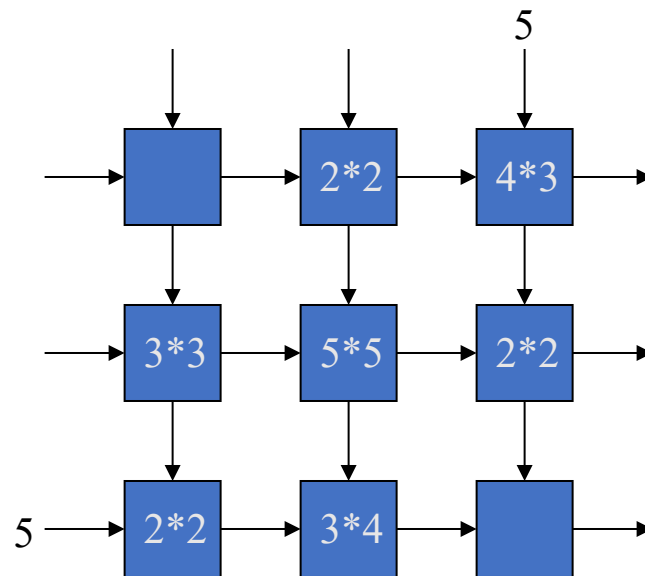
7.6 脉动阵列流水处理机



Clock tick: 3

P1	P2	P3	P4	P5	P6	P7	P8	P9
23	32	6	16	8	0	9	0	0

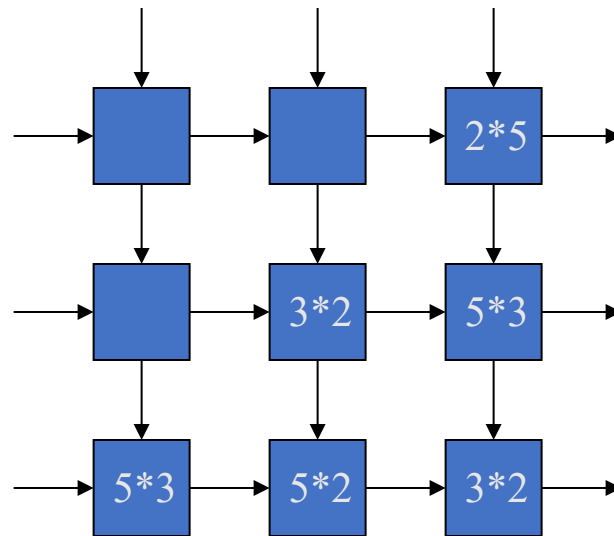
7.6 脉动阵列流水处理机



Clock tick: 4

P1	P2	P3	P4	P5	P6	P7	P8	P9
23	36	18	25	33	4	13	12	0

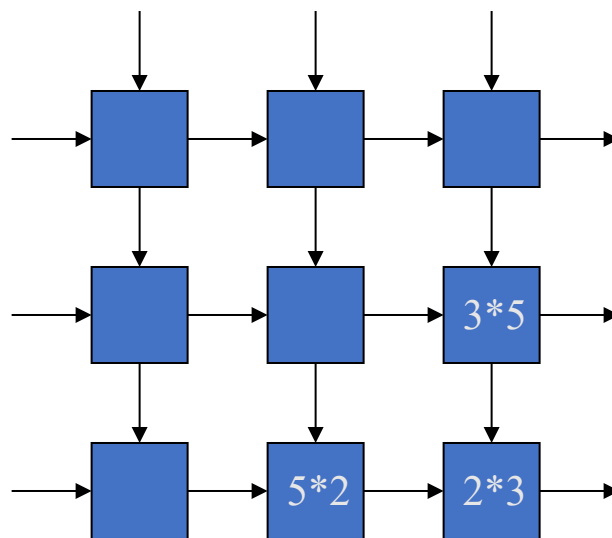
7.6 脉动阵列流水处理机



Clock tick: 5

P1	P2	P3	P4	P5	P6	P7	P8	P9
23	36	28	25	39	19	28	22	6

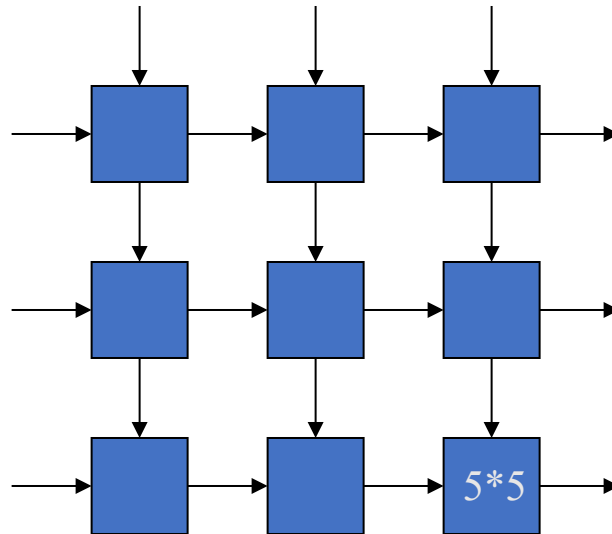
7.6 脉动阵列流水处理机



Clock tick: 6

P1	P2	P3	P4	P5	P6	P7	P8	P9
23	36	28	25	39	34	28	32	12

7.6 脉动阵列流水处理机



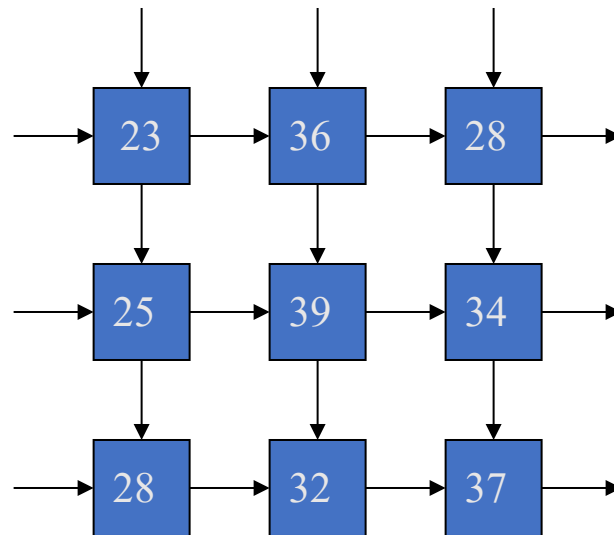
Clock tick: 7

P1	P2	P3	P4	P5	P6	P7	P8	P9
23	36	28	25	39	34	28	32	37

7.6 脉动阵列流水处理机

$$\begin{bmatrix} 3 & 4 & 2 \\ 2 & 5 & 3 \\ 3 & 2 & 5 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 2 \\ 2 & 5 & 3 \\ 3 & 2 & 5 \end{bmatrix} = \begin{bmatrix} 23 & 36 & 28 \\ 25 & 39 & 34 \\ 28 & 32 & 37 \end{bmatrix}$$

- Same answer in $2n + 1$ time!
- Can we do better?



P1	P2	P3	P4	P5	P6	P7	P8	P9
23	36	28	25	39	34	28	32	37

7.6 脉动阵列流水处理机

- 结构简单、规整，模块化强，可扩充，非常适合用超大规模集成电路实现；
- PE间数据通信距离短、规则，使数据流和控制流的设计、同步控制等均简单规整；
- 脉动阵列中所有的PE能同时运算，具有较高的计算并行性，可通过流水获得很高的运算效率和吞吐率。输入数据能被多个处理单元重复使用，大大减轻了阵列与外界的I/O通信量，降低了对系统主存和I/O系统频宽的要；
- 脉动阵列的构形于特定计算任务和算法密切相关，具有某种专用性，限制了应用范围，这对VLSI不利；

7.6 脉动阵列流水处理机

➤ 通用脉动阵列结构

- 关键因素：受阵列结构的通用性及I/O带宽约束所限制的阵列结构的规模大小
- 不同算法要求有不同的阵列结构，以及大小不同的阵列

➤ 发展通用脉动阵列结构的途径

- 通过增设附加的硬件，对阵列的拓扑结构和互连方式用可编程开关进行重构，即经程序重新配置阵列的结构
- 用软件把不同的算法映像到固定的阵列结构上
- 探寻与问题大小无关的脉动处理方法，以及VLSI运算系统的分割矩阵算法，使他们可以克服阵列只能求解固定大小题目的缺陷，同时探寻发展适合一类计算问题的通用算法和相应的设置方案

7.7 消息传递机制

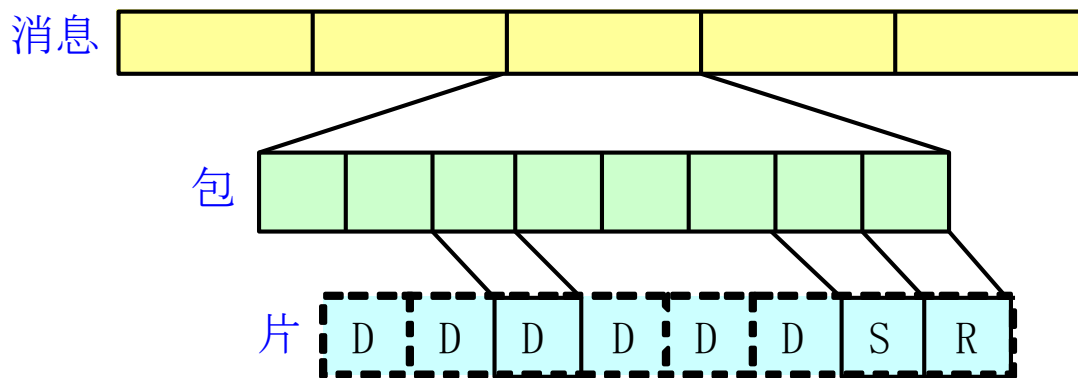
- 当源节点和目的节点之间没有直接的连接时，消息需要经过中间的节点进行传递。寻径就是用来实现这种传递的通信方法和算法。有的称之为路由。

7.7.1 消息寻径方案

1 消息的格式

➤消息：节点之间进行通信的逻辑单位

- 由若干个“包”组成
- 包的长度是固定的，一条消息中所包含的包的个数是可变的，消息的长度是不定长的
- 消息、包和片的格式



R：寻径信息

S：顺序号

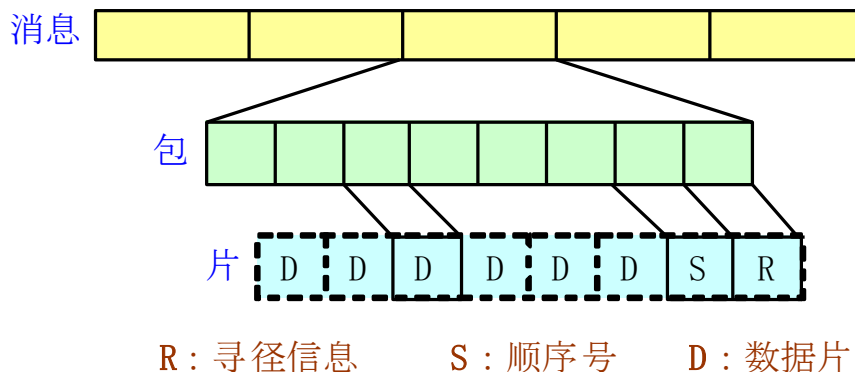
D：数据片

7.7.1 消息寻径方案

1 消息的格式

➤消息：节点之间进行通信的逻辑单位

- 包是包含寻径所需目的地址的基本单位，不同的包可能异步到达目的节点，因此一条消息中每个包都依次被分配一个序号，以便这些包到达目的节点后能重新组装出消息
- 包可以进一步分成一些更小的固定长度的“片”，寻径信息和包序列号形成头片，其余为数据片



7.7.1 消息寻径方案

2 消息寻径方式

➤ 线路交换

➤ 包交换：存储转发、虚拟直通、虫蚀

7.7.1 消息寻径方案

2 消息寻径方式：线路交换（Circuit Switch）

➤在线路交换方式下，在传递一个信息之前，需要先建立一条从源节点到目的节点的物理通路，然后再传递信息。

$$\text{传输时延 } T = \frac{L + L_t \times (D + 1)}{B}$$

- L ：信息包的长度（位数）
- L_t ：建立路径所需的小信息包的长度
- D ：经过的中间节点个数
- B ：带宽

7.7.1 消息寻径方案

2 消息寻径方式：线路交换（Circuit Switch）

- 在线路交换方式下，在传递一个信息之前，需要先建立一条从源节点到目的节点的物理通路，然后再传递信息。

$$\text{传输时延} \quad T = \frac{L + L_t \times (D + 1)}{B}$$

- 优点：传输带宽较大，平均传输时延较小，而且使用的缓冲区小，适合于具有动态和突发性的大规模并行处理数据的传送。
- 缺点：需要频繁地建立源节点到目的节点的物理通路，时间开销会很大。

7.7.1 消息寻径方案

2 消息寻径方式：存储转发 (Store and Forward)

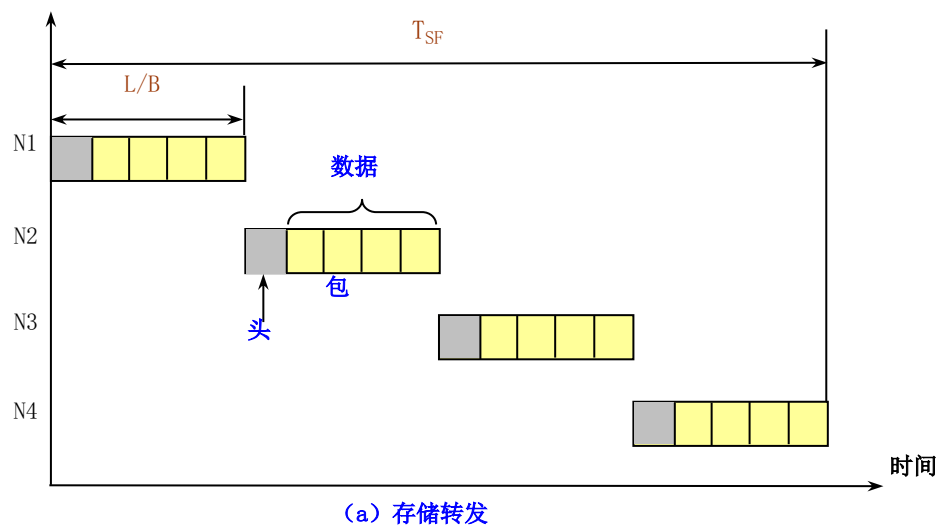
- 包是信息传递的基本单位，从源节点经过一系列中间节点到达目的节点
- 当一个包到达一个中间结点时，它首先被存放到缓冲区，当所要求的的输出通道和接收结点的包缓冲区可用时，再将它传送给下一个结点

7.7.1 消息寻径方案

2 消息寻径方式：存储转发（Store and Forward）

➤ 存储转发网络的时延与源和目的地之间的距离成正比。

$$\text{传输时延 } T = \frac{(D + 1)L}{B}$$



7.7.1 消息寻径方案

2 消息寻径方式：存储转发 (Store and Forward)

- 包是信息传递的基本单位，从源节点经过一系列中间节点到达目的节点
- 当一个包到达一个中间结点时，它首先被存放到缓冲区，当所要求的的输出通道和接收结点的包缓冲区可用时，再将它传送给下一个结点
- 存储转发网络的时延与源和目的地之间的距离成正比。

$$\text{传输时延 } T = (D + 1)L/B$$

- 优点：占用物理通路的时间比较短。
- 缺点：包缓冲区大，时延大（与结点距离成正比）。

7.7.1 消息寻径方案

2 消息寻径方式：虚拟直通 (Virtual Cut Through)

- 为减少时延，没有必要等到整个消息全部缓冲后再进行路由选择，只接收到用作寻径的消息头部即可判断
- 当接收到用作寻径的消息头部时，即开始路由选择。
- 通信时延公式：

$$T = \frac{L + L_h(D + 1)}{B} \approx L/B$$

其中： L_h 是寻径头部的长度，一般 $L \gg L_h \times (D + 1)$

- 当出现寻径阻塞时，只能将整个消息存储在寻径结点中。
- 优点：通信延迟与结点数无关。
- 缺点：每个结点需要有足够大的缓冲区。在最坏的情况下与存储转发方式的通信时延相同，经过的每个结点都阻塞，都需要缓冲。

7.7.1 消息寻径方案

2 消息寻径方式：虫蚀寻径（Wormhole）

- 把包分成更小的片。每个结点的寻径器中设置有片缓冲区。
- 用头片直接开辟一条从输入结点到输出结点的路径，每个消息中的片以流水方式在网络中向前“蠕动”。
- 当消息的头片到达一个结点的寻径器后，寻径器根据头片的寻径消息立即做出路由选择
- 一个结点一旦开始传送一个包中的头片后，该结点就必须等待这个包的所有片都传送出去之后，才能传送其它包；不同包的片不能混合传送
- 如果所选择的通道或结点的片缓冲区不可用时，头片必须在该结点的片缓冲区中等待，其它数据片也在原来的结点上等待

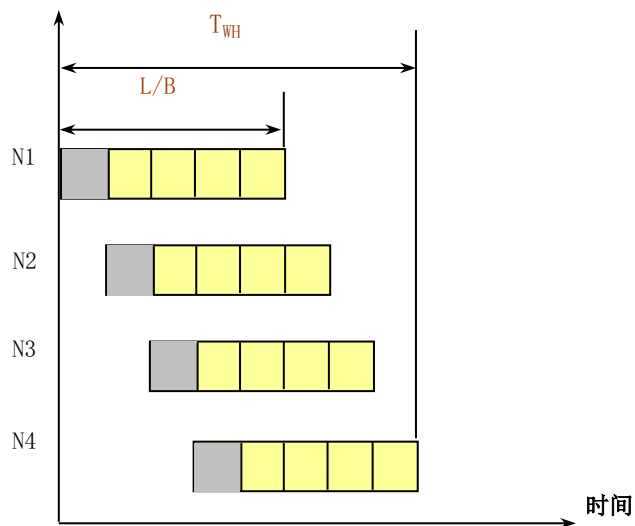
7.7.1 消息寻径方案

2 消息寻径方式：虫蚀寻径 (Wormhole)

➤ 时延公式：

$$T = T_f \times D + \frac{L}{B} = \frac{L + L_f \times D}{B} \approx \frac{L}{B}$$

其中： L_f 是片的长度， T_f 是片经过一个结点所需时间。一般有 $L \gg L_f \times D$ ，
时延近似为： $T = L/B$ ，与结点数无关



(b) 虫蚀方式

7.7.1 消息寻径方案

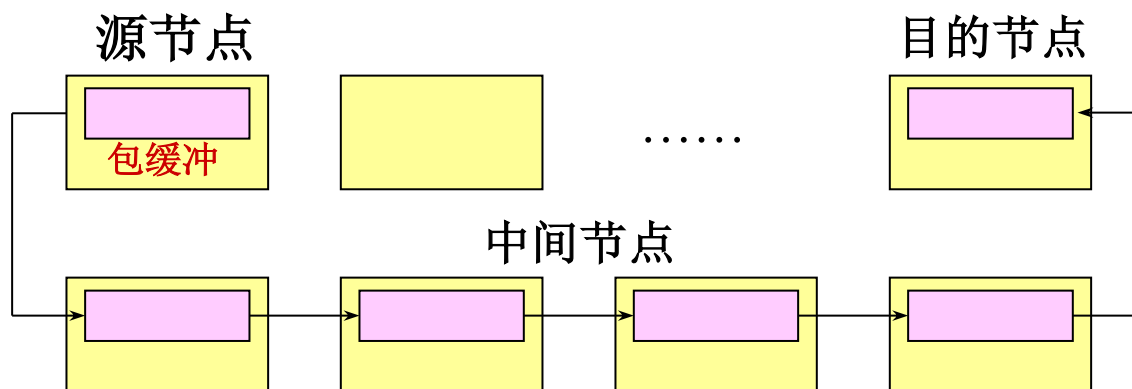
2 消息寻径方式：虫蚀寻径（Wormhole）

➤时延公式：

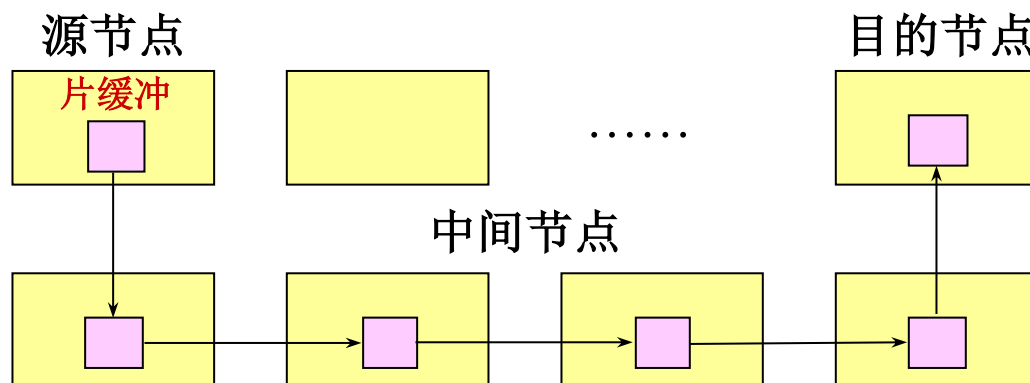
$$T = T_f \times D + \frac{L}{B} = \frac{L + L_f \times D}{B} \approx \frac{L}{B}$$

其中： L_f 是片的长度， T_f 是片经过一个结点所需时间。一般有 $L \gg L_f \times D$ ，时延近似为： $T=L/B$ ，与结点数无关。

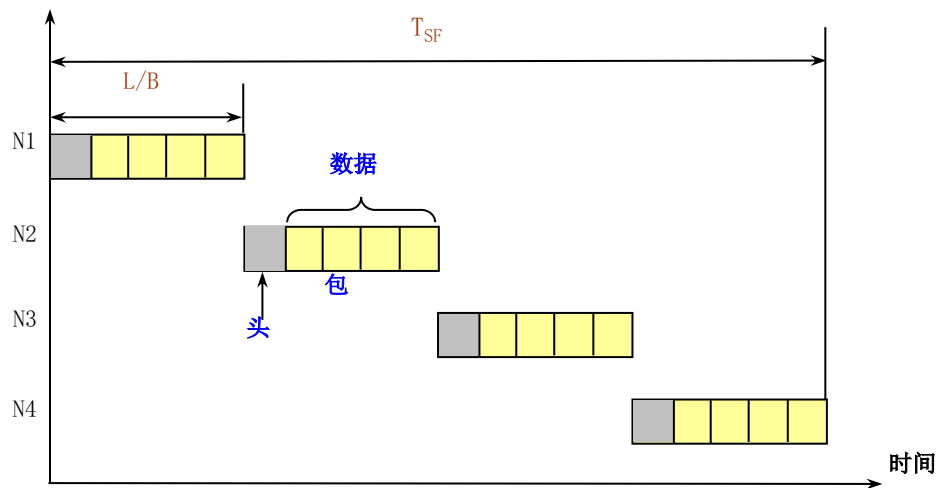
- 优点：每个结点的缓冲区较小；较低的网络传输时延；通道共享性好，利用率高；易于实现选播和广播通信方式。
- 缺点：当消息的一个片被阻塞时，整个消息都被阻塞。



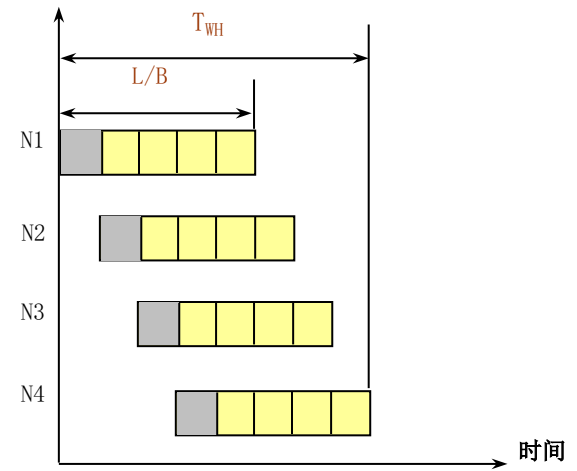
(a) 存储转发



(b) 虫蚀方式



(a) 存储转发



(b) 虫蚀方式

存储转发与虫蚀方式的时间比较

7.7.2 流控制策略

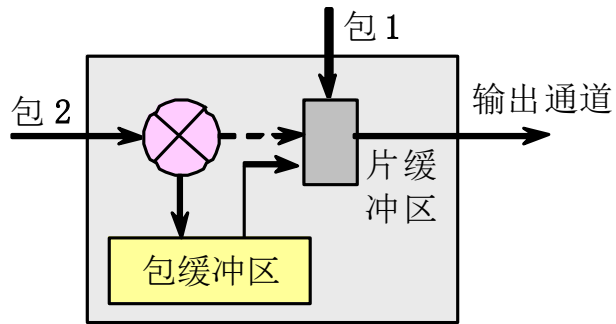
1. 包冲突的解决

- 为了通过通道在两个相邻节点之间传送一个片，要同时具备3个条件：
 - 源缓冲区已存有该片；
 - 通道已分配好；
 - 接收缓冲区准备接收该片。
- 当两个包到达同一个节点时，它们可能都在请求同一个接收缓冲器或者同一个输出通道，这时必须对两个问题进行仲裁。
 - 把通道分配给哪个包？
 - 如何处理被通道拒绝的包？

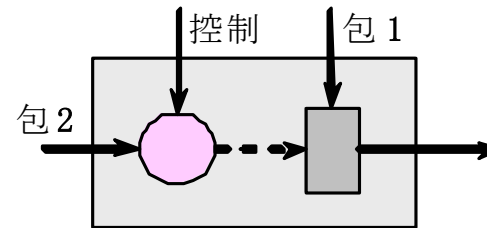
7.7.2 流控制策略

1. 包冲突的解决

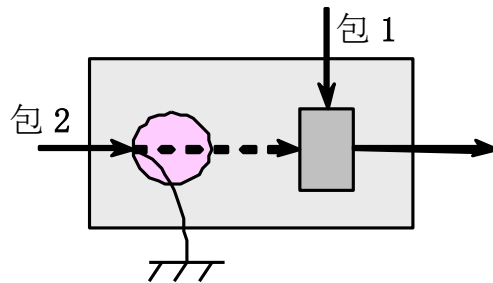
➤ 4种解决方案



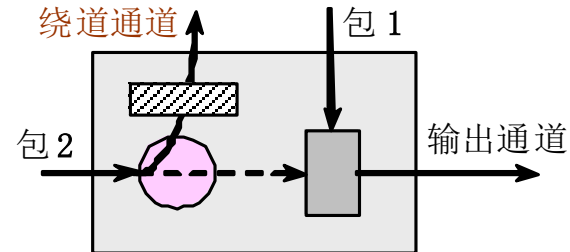
(a) 用缓冲实现虚拟通道



(b) 阻塞流控制



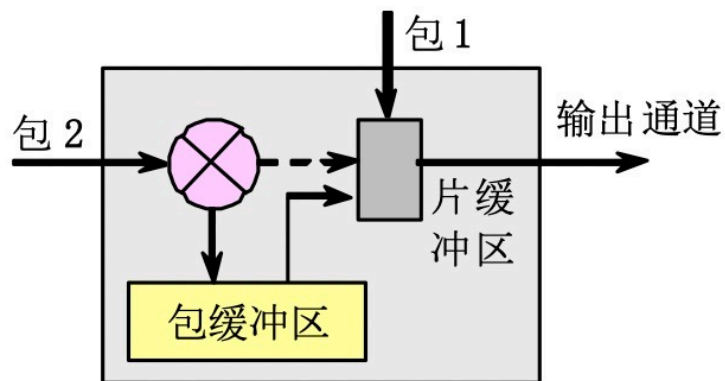
(c) 扬弃并重发



(d) 阻塞后绕道

7.7.2 流控制策略

1. 包冲突的解决

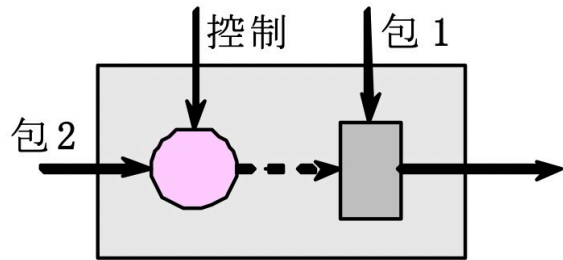


(a) 用缓冲实现虚拟通道

- 把第二个包暂存在缓冲区
- 优点: 不会浪费已经分配了的资源, 但它要求节点中有一个足够大的缓冲器来存放整个信息包。

7.7.2 流控制策略

1. 包冲突的解决

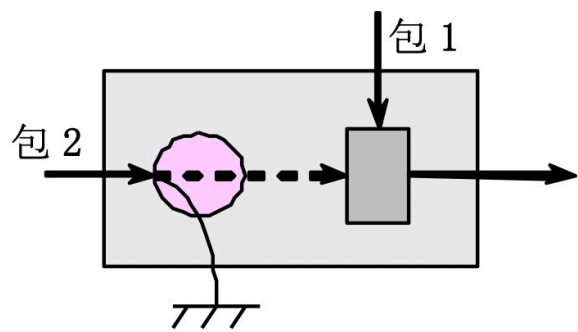


(b) 阻塞流控制

➤ 阻塞第二个包

7.7.2 流控制策略

1. 包冲突的解决

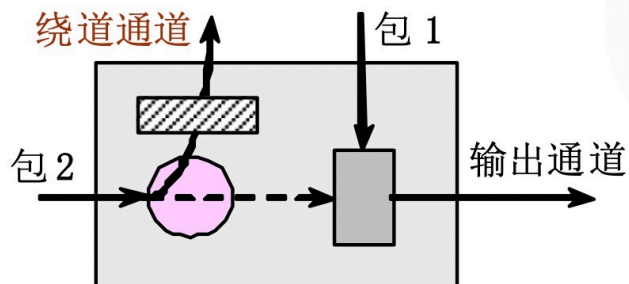


(c) 丢弃并重发

- 丢弃第二个包
- 有可能会造成严重的资源浪费，而且要求重新进行被丢弃包的传输与确认。

7.7.2 流控制策略

1. 包冲突的解决



(d) 阻塞后绕道

- 绕道
- 在包寻径方面提供了更多的灵活性，但为了到达目的节点，可能要花费超过实际需要的通道资源，造成浪费。

7.7.2 流控制策略

2. 确定性寻径和自适应寻径

- 确定性寻径：通信路径完全由源节点地址和目的地址来决定，即寻径路径是预先唯一地确定好了的，而与网络的状况无关。
- 自适应寻径：通信的通路每一次都要根据资源或者网络的情况来选择。
 - 可以避开拥挤的或者有故障的节点，使网络的利用率得到改进。

7.7.2 流控制策略

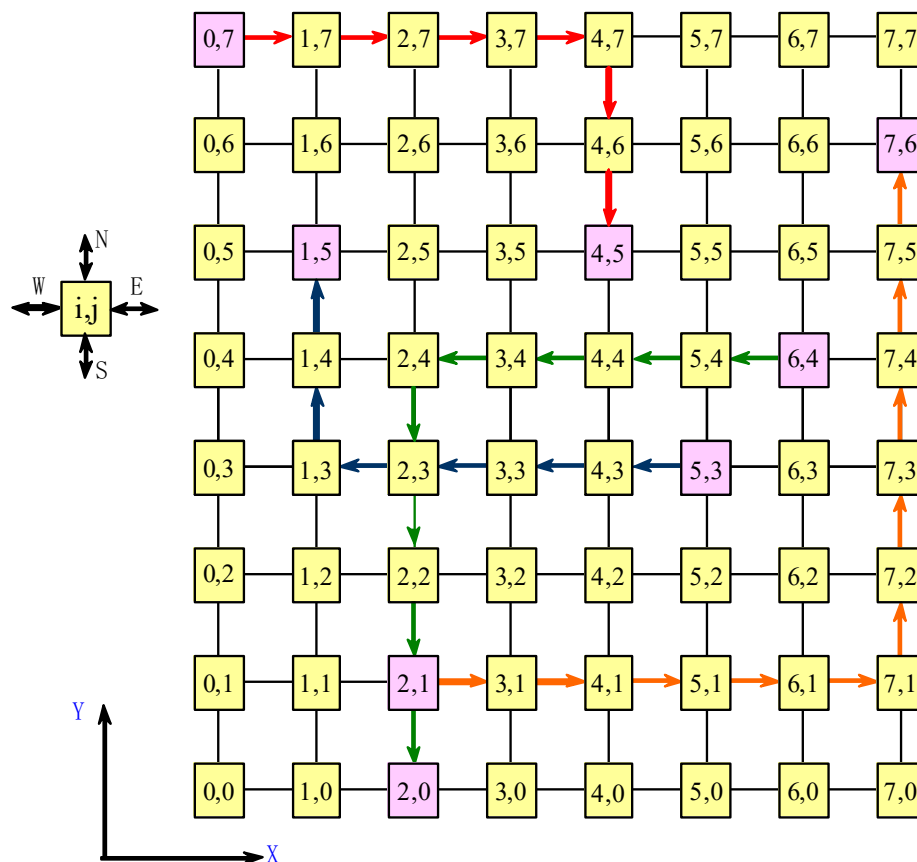
2. 确定性寻径和自适应寻径

➤ 两种确定性寻径算法

- 都是建立在维序概念之上的，即按照特定顺序选择后继通道
- 在二维网格网络中称为X-Y寻径，在超立方体中称为E立方体寻径
 - 二维网格网络的X-Y寻径：任意一个源节点 $s = (x_1, y_1)$ ，任意一个目的节点 $d = (x_2, y_2)$ ，从 s 出发，先沿 X 轴方向前进，直到找到 d 所在的列 x_2 ，然后再沿 Y 轴方向前进，直到找到目标节点 (x_2, y_2)

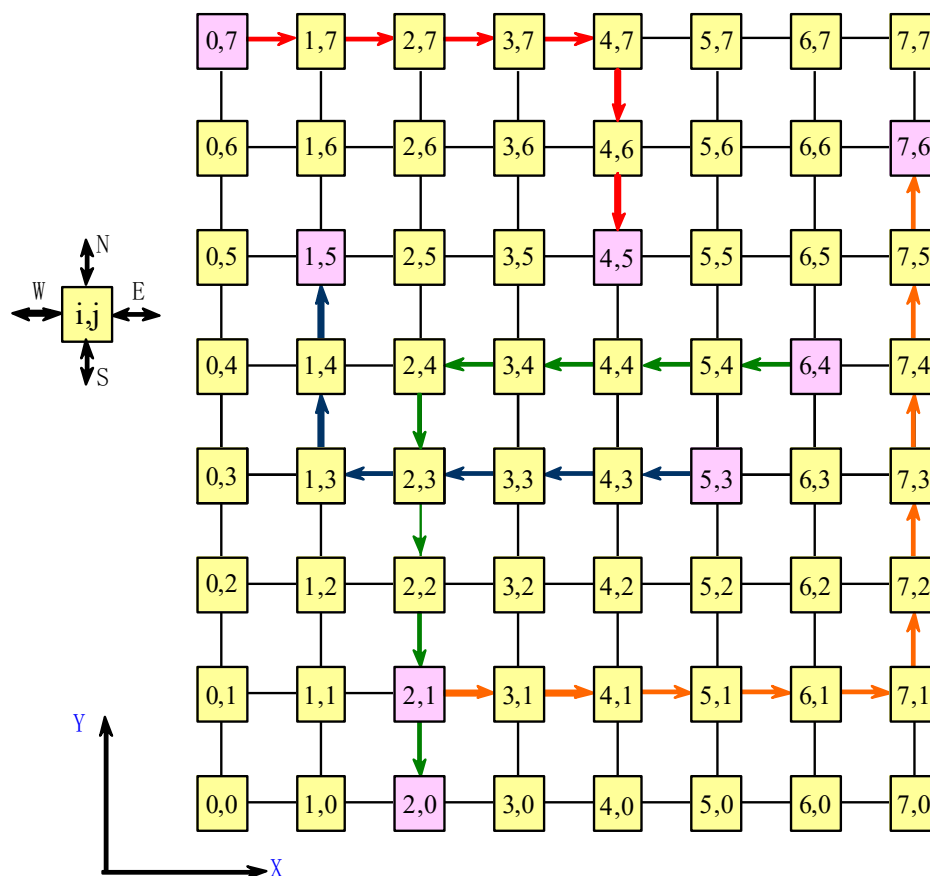
对于图所示的二维网格，确定以下4组“源节点-目的结点”所需要的路径。

- (2, 1) 到 (7, 6)
- (0, 7) 到 (4, 5)
- (6, 4) 到 (2, 0)
- (5, 3) 到 (1, 5)



所需要的路径如图所示。其中：

- (2, 1) 到 (7, 6) 需要用到的是一条东-北路径；
- (0, 7) 到 (4, 5) 需要用到的是一条东-南路径；
- (6, 4) 到 (2, 0) 需要用到的是一条西-南路径；
- (5, 3) 到 (1, 5) 需要用到的是一条西-北路径。



- 考虑一个由 $N=2^n$ 个节点构成的 n 方体，每个节点的编号为二进制编码，假设源节点 $s = s_{n-1}s_{n-2} \cdots s_1s_0$ ，目的节点 $d = d_{n-1}d_{n-2} \cdots d_1d_0$ ，现在要确定一条从 s 到 d 的步数最少的路径。
- 将这个 n 方体的各维表示成 $i=1,2,\cdots,n$ ，其中第 i 维对应于节点地址中的第 $i-1$ 位。
- 设 $v = v_{n-1}v_{n-2} \cdots v_1v_0$ 是路径中的任一节点。路径可以根据以下算法唯一地确定：
 - ① 计算方向位 $r_i = s_{i-1} \oplus d_{i-1}$ ，其中 $i=1,2,\cdots,n$ 。
令 $v = s$ ， $i = 1$ ，反复执行以下步骤：
 - ② 如果 $r_i=1$ ，则从当前节点 v 寻径到下一节点 $v \oplus 2^{i-1}$ ；否则，就跳过这一步。
 - ③ $i \leftarrow i + 1$ 。如果 $i \leq n$ ，则转第②步，否则退出。

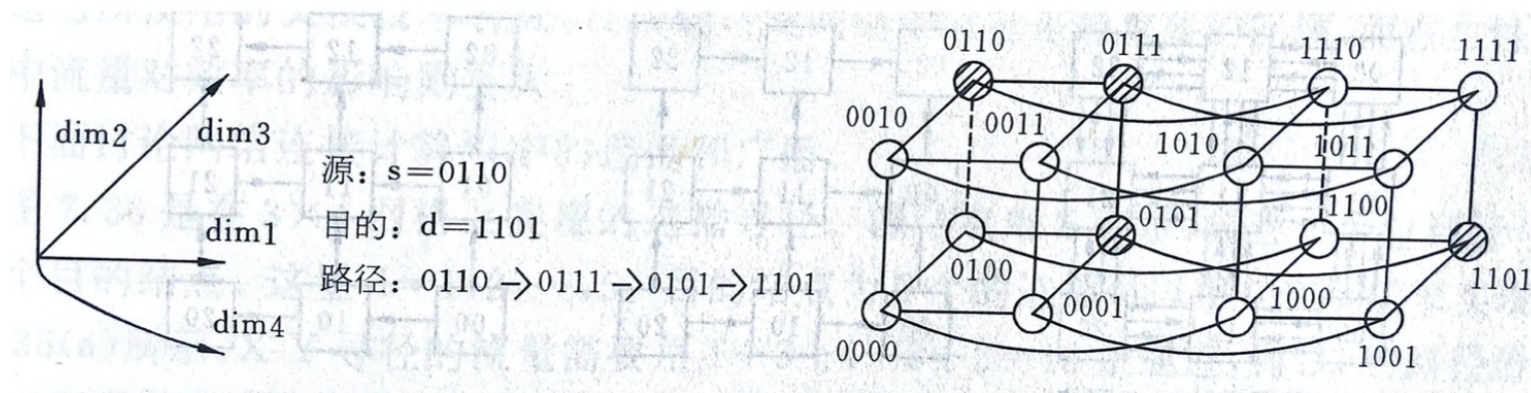
- 设 $v = v_{n-1}v_{n-2} \cdots v_1v_0$ 是路径中的任一节点。路径可以根据以下算法唯一地确定：

① 计算方向位 $r_i = s_{i-1} \oplus d_{i-1}$ ，其中 $i=1,2,\cdots,n$ 。

令 $v = s$ ， $i = 1$ ，反复执行以下步骤：

② 如果 $r_i = 1$ ，则从当前节点 v 寻径到下一节点 $v \oplus 2^{i-1}$ ；否则，就跳过这一步。

③ $i \leftarrow i + 1$ 。如果 $i \leq n$ ，则转第②步，否则退出。



7.7.4 选播和广播寻径算法

1. 多计算机网络中会出现以下4种通信模式：

- 单播：对应于一对一的通信情况，即一个源节点发送消息到一个目的节点。
- 选播：对应于一到多的通信情况，即一个源节点发送同一消息到多个目的节点。
- 广播：对应于一到全部的通信情况，即一个源节点发送同一消息到全部节点。
- 会议：对应于多到多的通信情况。

7.7.4 选播和广播寻径算法

1. 多计算机网络中会出现以下4种通信模式：

➤ 通道流量和通信时延是常用的两个参数。

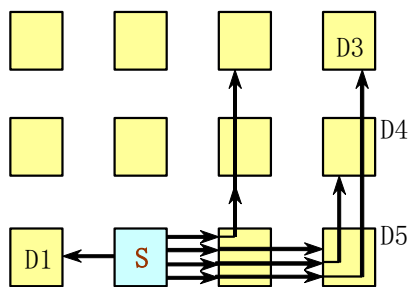
- 通道流量可用传输有关消息所使用的通道数来表示。
- 通信时延则用包的最大传输时间来表示。

➤ 优化的寻径网络应能以最小流量和最小时延实现相关的通信模式。

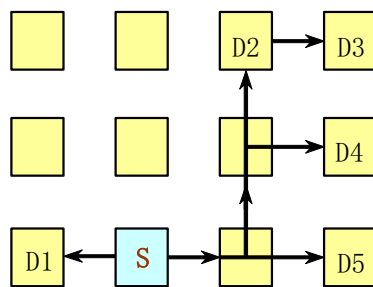
7.7.4 选播和广播寻径算法

2. 以网格网络为例，讨论选播与广播：

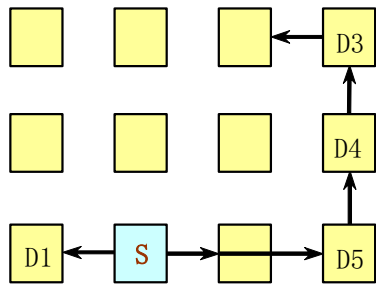
- 在 3×4 网格上实现的选播寻径，源节点S传送一个包到标号为 D_i 的5个目的节点（ $i=1, 2, \dots, 5$ ）



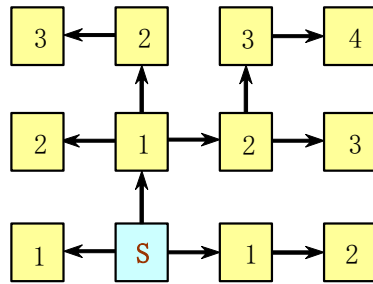
(a) 5次单播，流量为13，距离为4



(b) 流量为7，距离为4的选播方式



(c) 流量为6，距离5的选播方式



(d) 通过树结构广播给所有结点

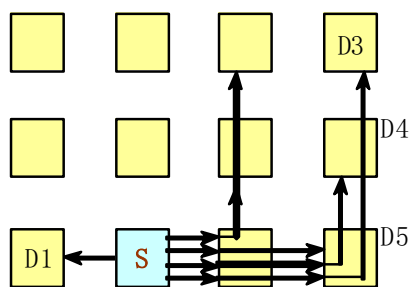
➤ 图（a）：目的节点为5个的选播可以用5次单播来实现

- X-Y寻径的流量需要用 $1+3+4+3+2=13$ 条通道。
- 到D3的路径最长，所以时延是4。

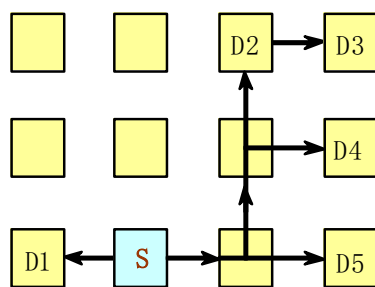
7.7.4 选播和广播寻径算法

2. 以网格网络为例，讨论选播与广播：

- 在 3×4 网格上实现的选播寻径，源节点S传送一个包到标号为 D_i 的5个目的节点（ $i=1, 2, \dots, 5$ ）



(a) 5次单播，流量为13，距离为4

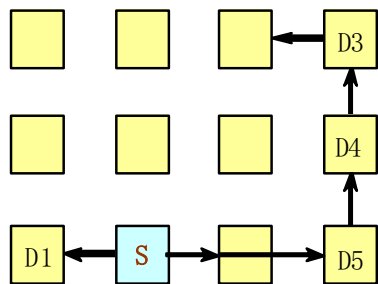


(b) 流量为7，距离为4的选播方式

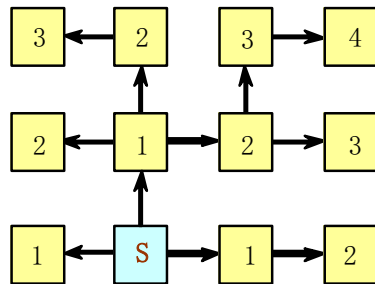
➤ 图 (b) 和 (c) 给出了两种选播寻径模式，流量分别为7和6。

➤ 在虫蚀网络中，用图 (c) 的选播寻径模式比较好。

➤ 在存储转发网络中，则用图 (b) 的寻径模式比较好，时延较短。



(c) 流量为6，距离5的选播方式

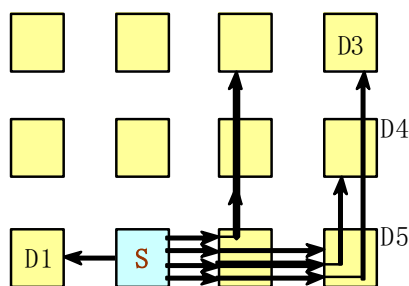


(d) 通过树结构广播给所有结点

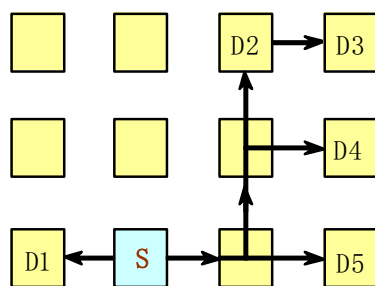
7.7.4 选播和广播寻径算法

2. 以网格网络为例，讨论选播与广播：

- 在 3×4 网格上实现的选播寻径，源节点S传送一个包到标号为 D_i 的5个目的节点（ $i=1, 2, \dots, 5$ ）

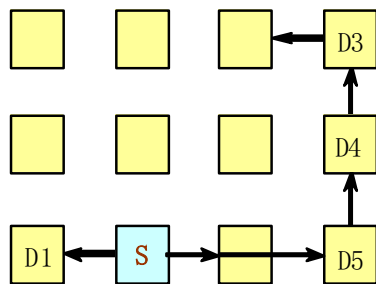


(a) 5次单播，流量为13，距离为4

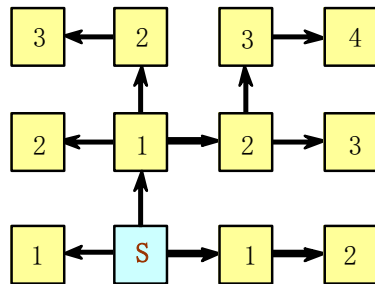


(b) 流量为7，距离为4的选播方式

- 图（d）：使用一棵4层的生成树可以把一个包从节点S广播到所有的网络节点
- 产生的时延和流量最小



(c) 流量为6，距离5的选播方式



(d) 通过树结构广播给所有结点