智能计算系统

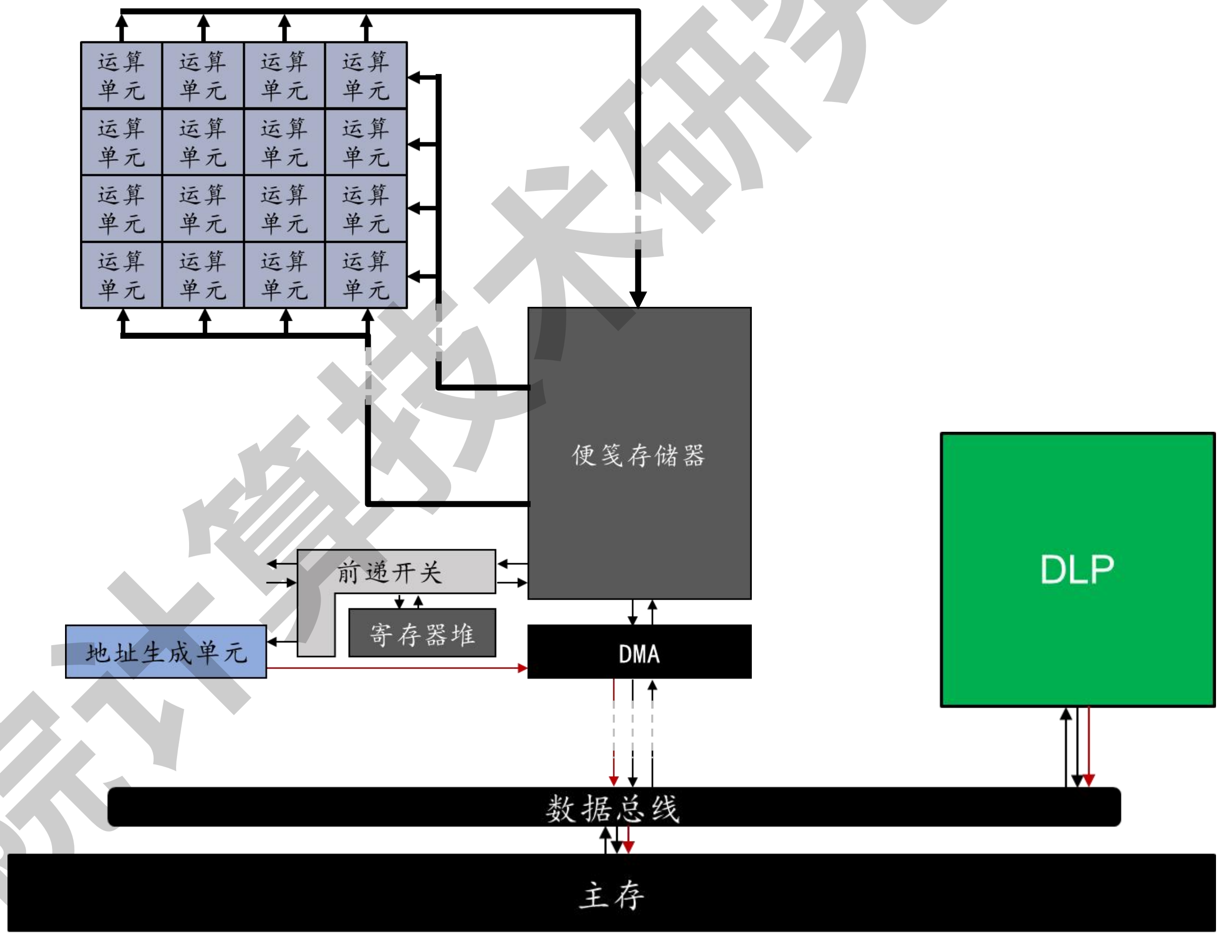
第七章

深度学习处理器架构

中国科学院计算技术研究所
李威 副研究员
liwei2017@ict.ac.cn

总体架构

- 计算
- 访存
- 通信



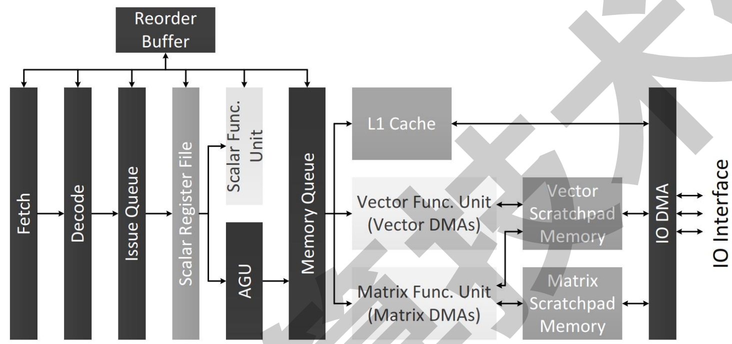
计算

- 三种计算单元
  - 矩阵
  - 向量
  - 标量

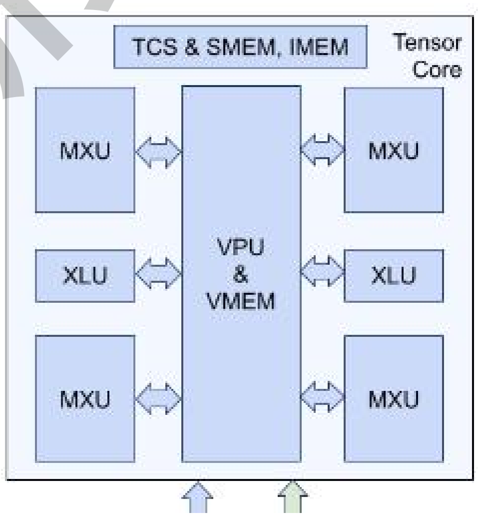
计算

- 三种计算单元
  - 矩阵
  - 向量
  - 标量

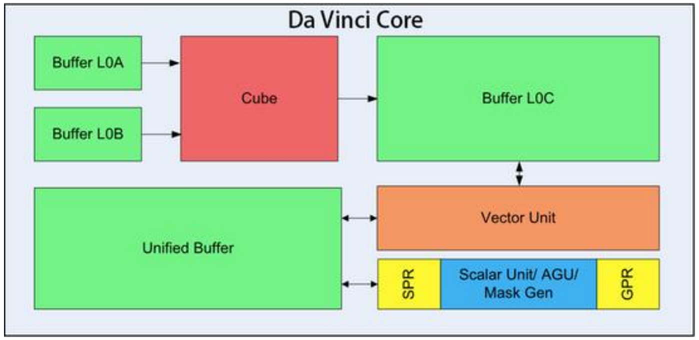
Cambricon 架构



TPUv4i 计算单元



Volta 架构

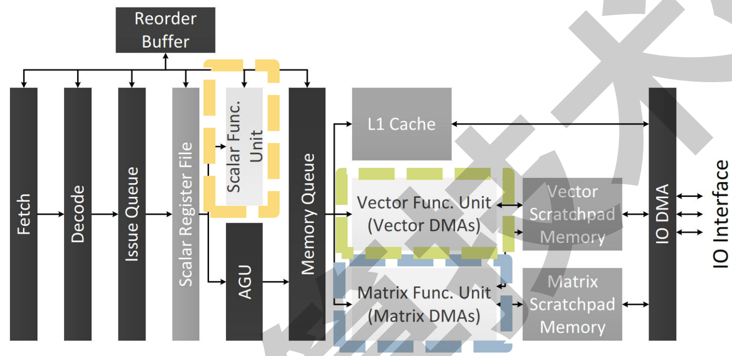


“达芬奇”架构

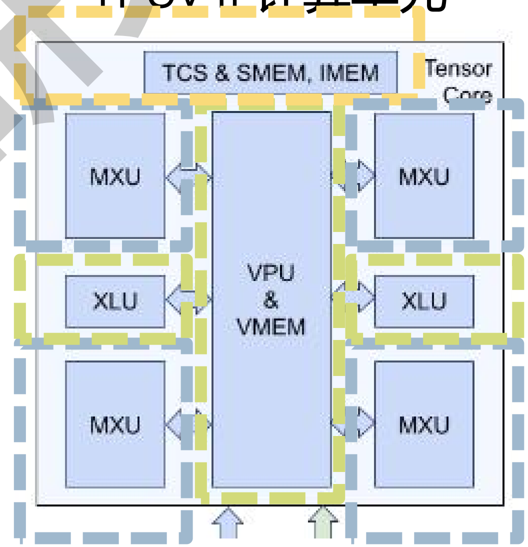
计算

- 三种计算单元
  - 矩阵
  - 向量
  - 标量

Cambricon 架构



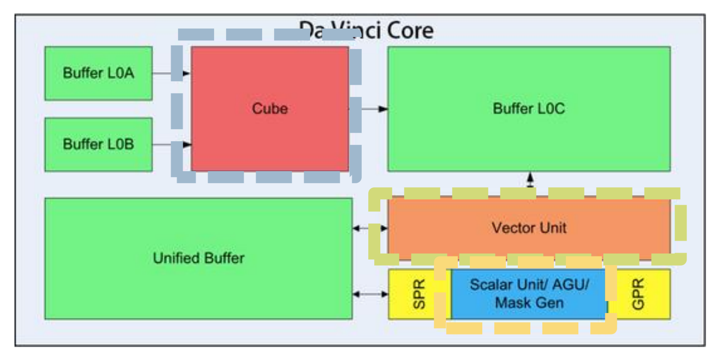
TPUv4i 计算单元



常见三种共存

各司其职

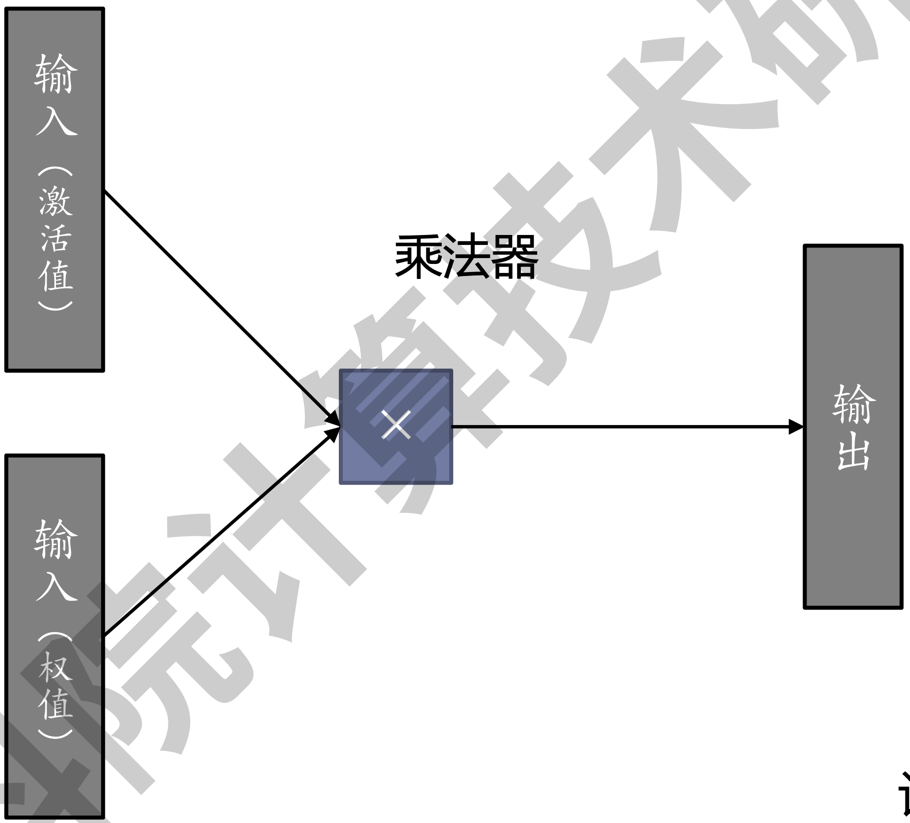
Volta 架构



“达芬奇” 架构

矩阵运算单元

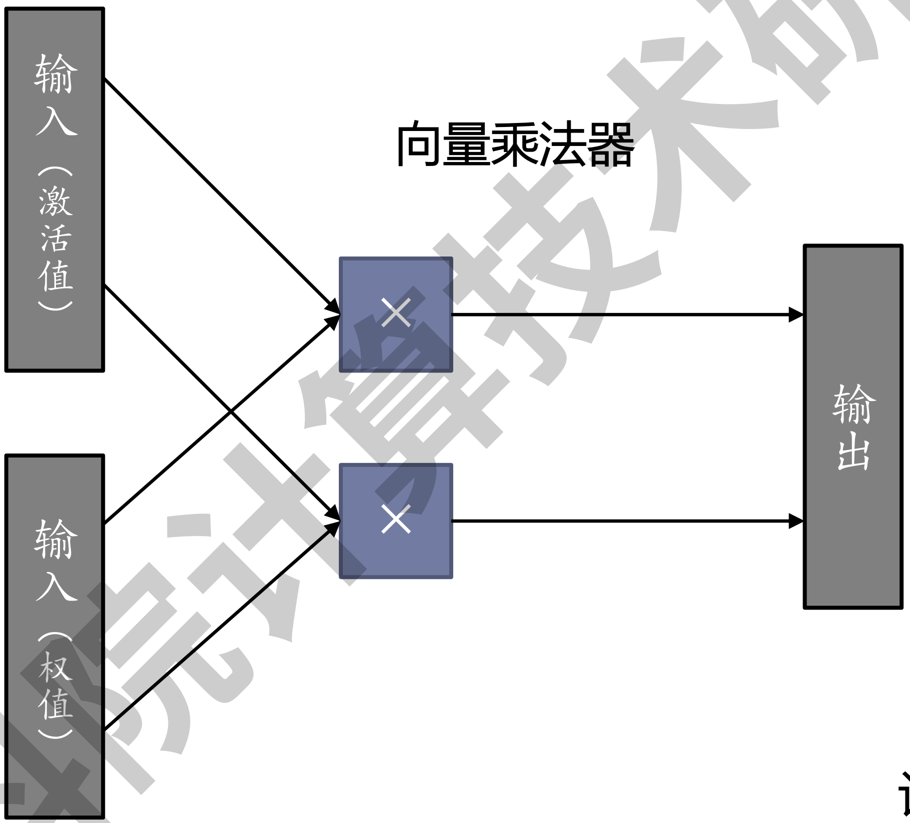
- 一种实现：由内积单元堆叠而成



计算-I/O比例 = 1:3

矩阵运算单元

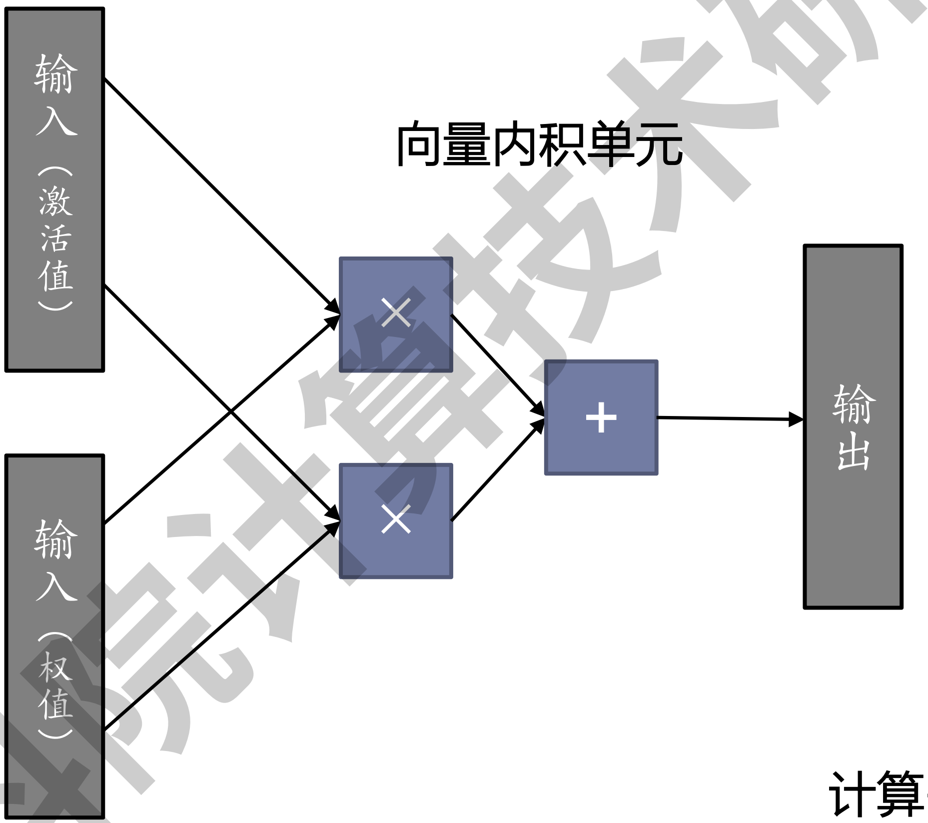
- 一种实现：由内积单元堆叠而成



计算-I/O比例 = 1:3

矩阵运算单元

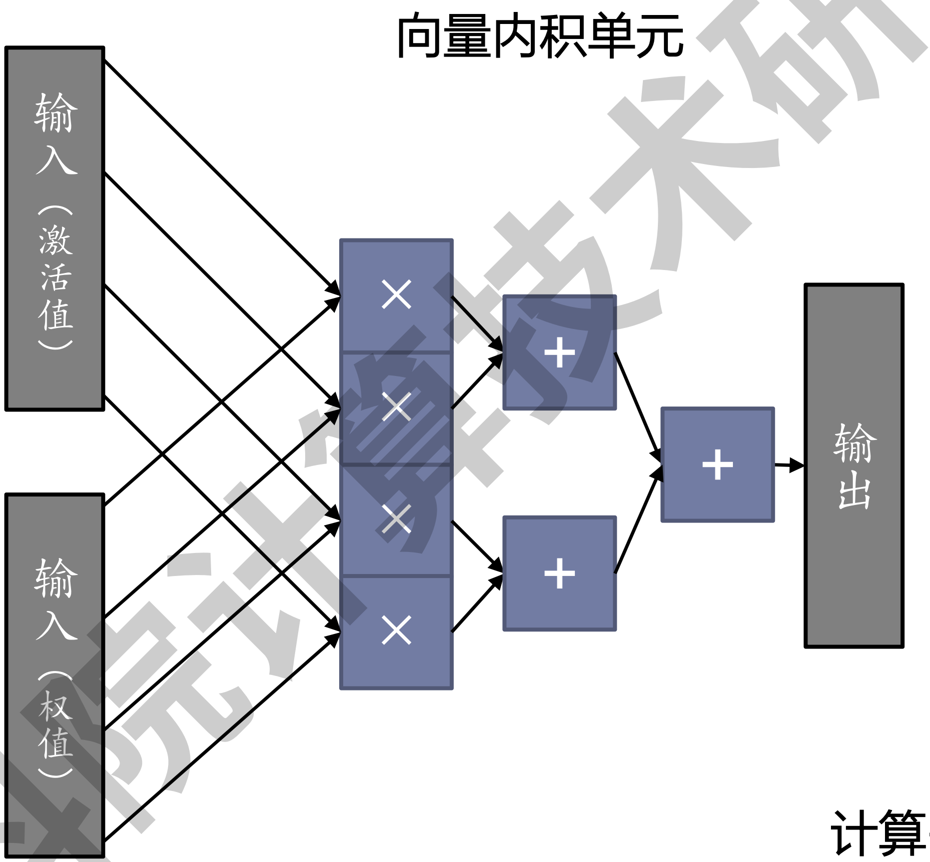
- 一种实现：由内积单元堆叠而成



计算-I/O比例 = 3:5 = 1:1.6

矩阵运算单元

- 一种实现：由内积单元堆叠而成

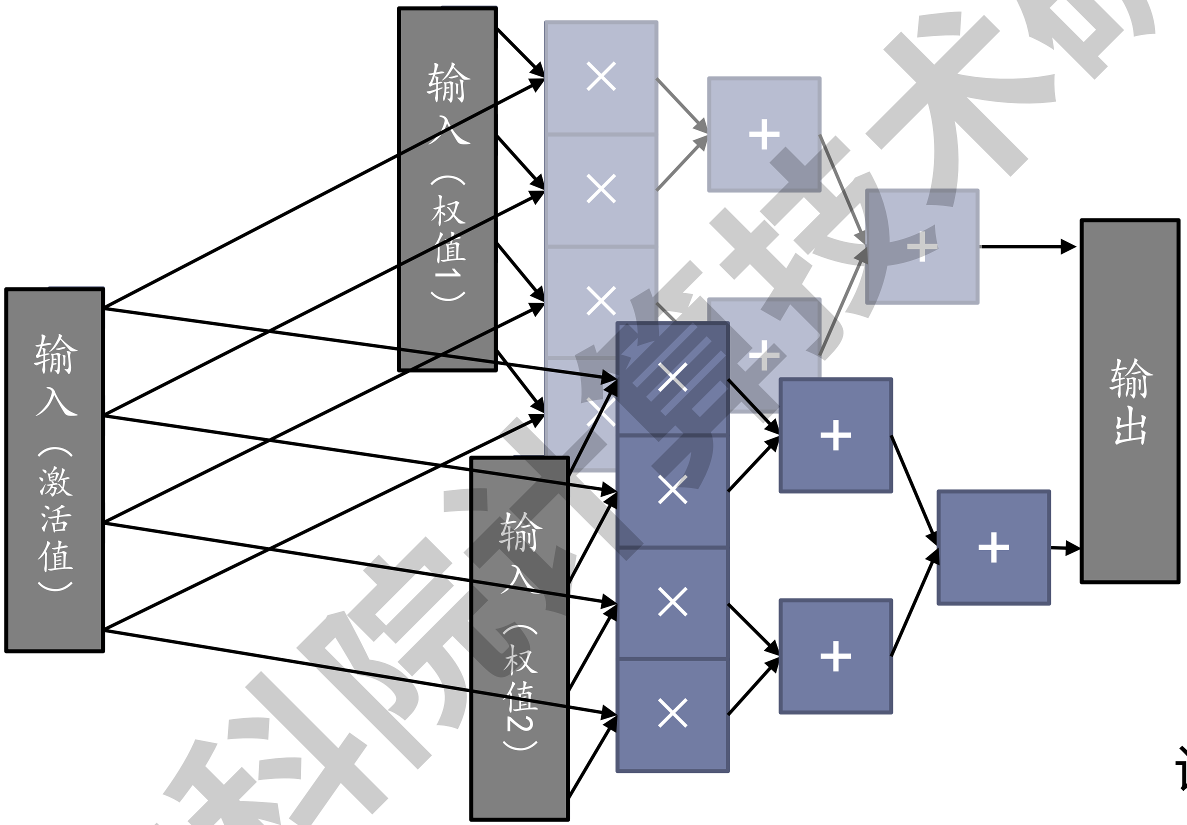


计算-I/O比例 = 7:9 = 1:1.3

矩阵运算单元

- 多个内积单元组成矩阵乘向量单元

矩阵乘向量单元

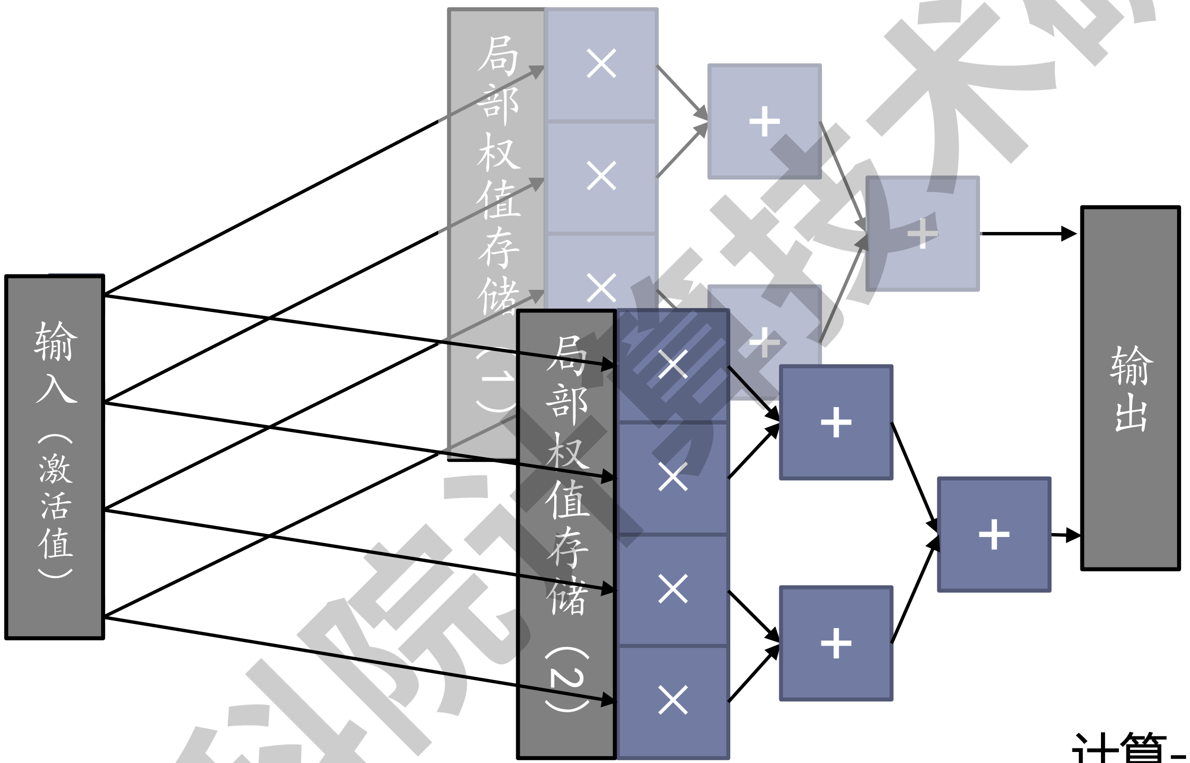


计算-I/O比例 = 1:1.3

矩阵运算单元

- 近端数据（权值）存储在内积单元附近的电路中
- 采用小而快的存储器

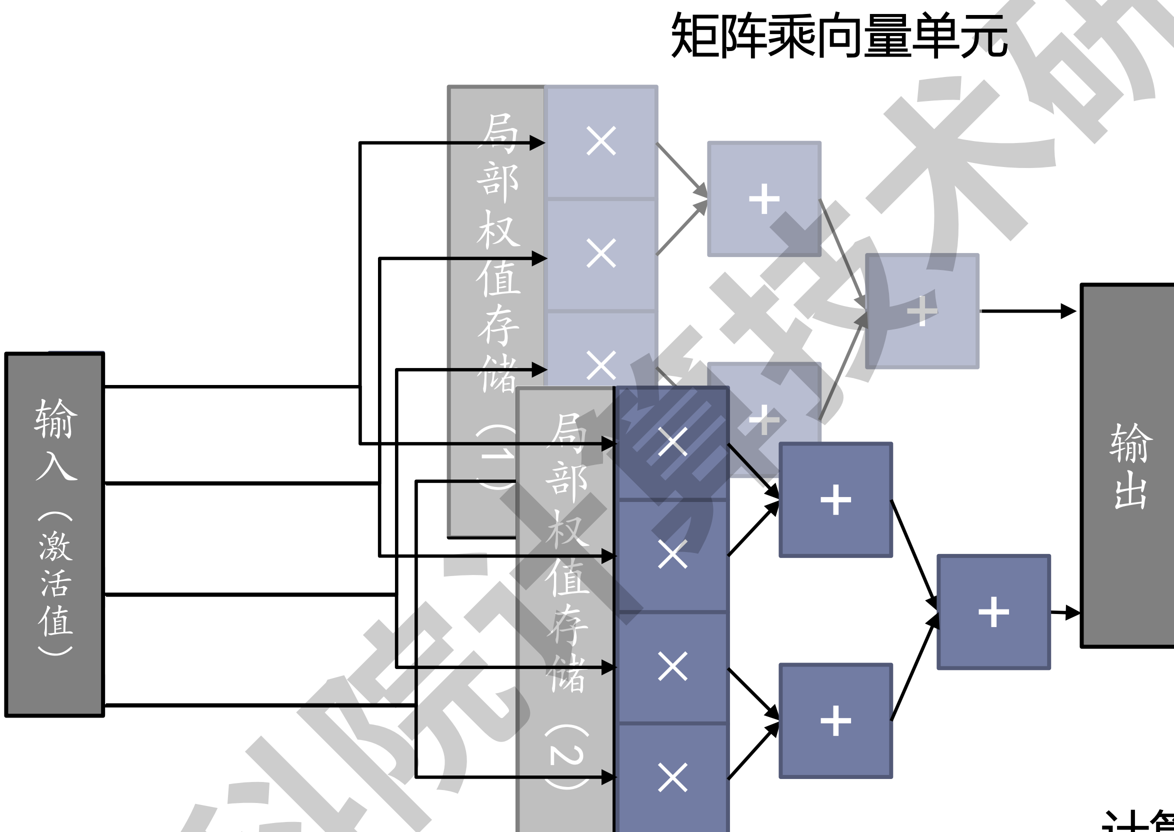
矩阵乘向量单元



计算-I/O比例 = 7:5 = 1:0.7

矩阵运算单元

- 所有内积单元共享激活值，采用广播

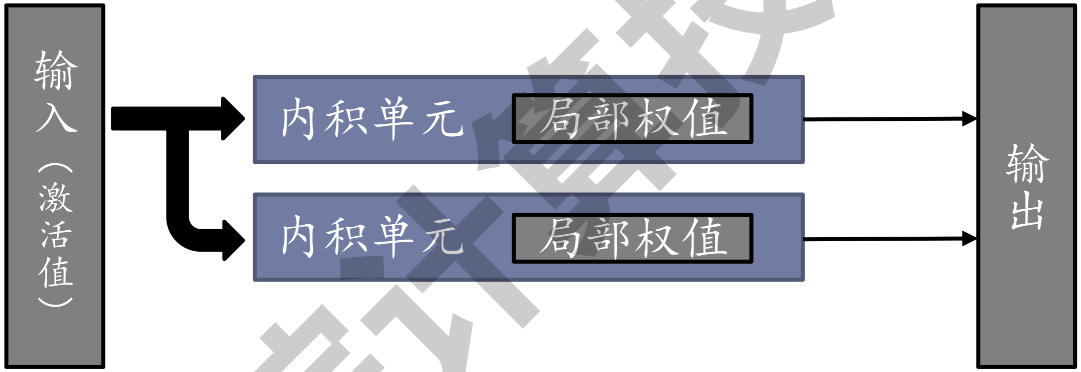


计算-I/O比例 = 7:3 = 1:0.4

矩阵运算单元

- 整理示意图

矩阵乘向量单元

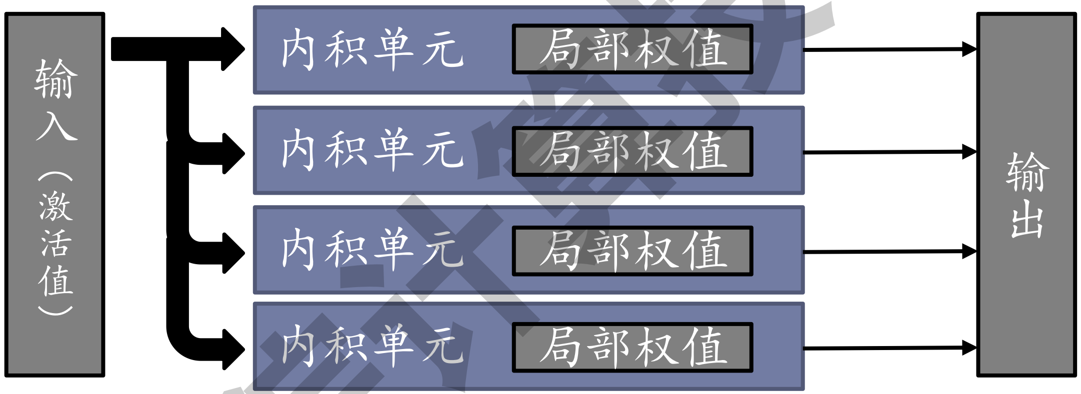


计算-I/O比例 = 1:0.4

矩阵运算单元

- 增加内积单元数量

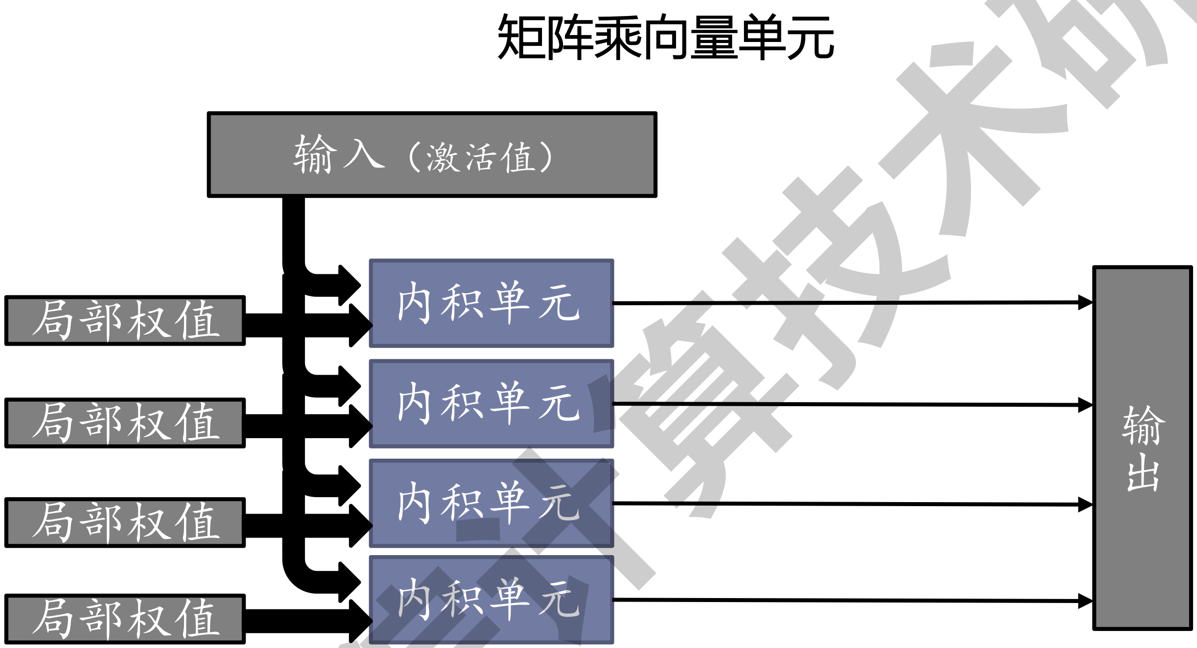
矩阵乘向量单元



计算-I/O比例 = 7:2 = 1:0.3

矩阵运算单元

- 提出权值

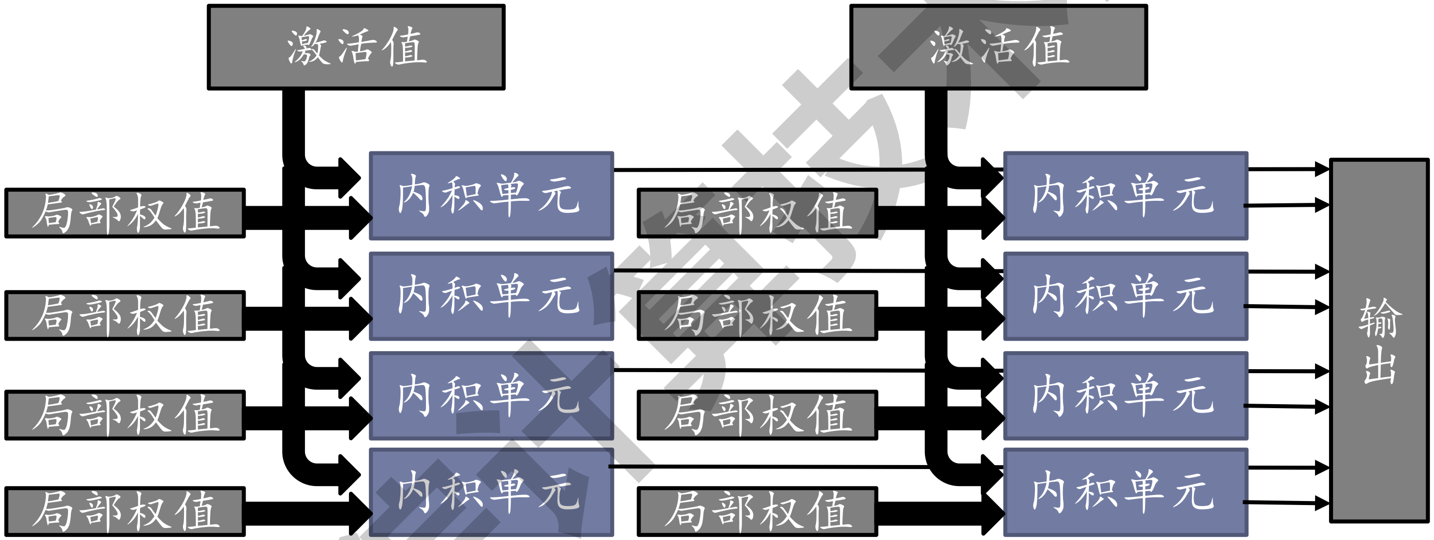


计算-I/O比例 = 28:24 = 1:0.9

矩阵运算单元

- 增加一组矩阵乘向量单元

多个矩阵乘向量单元

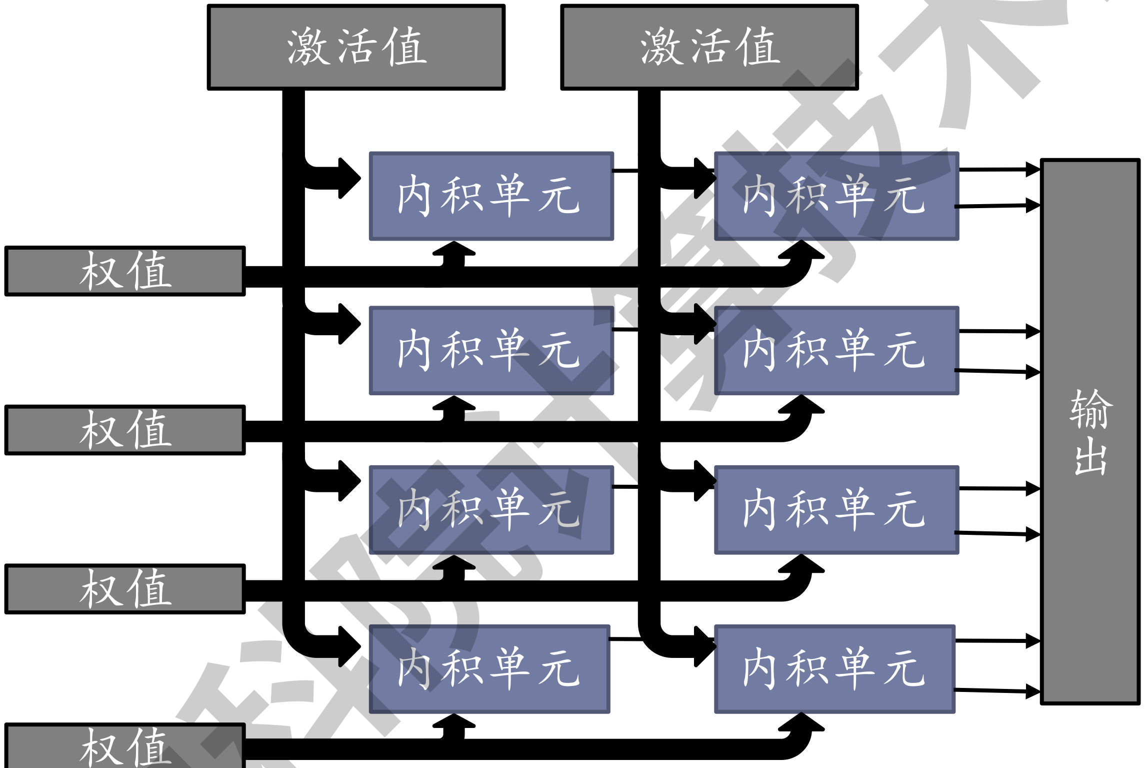


计算-I/O比例 = 1:0.9

矩阵运算单元

- 采用广播共享权值

矩阵乘矩阵单元

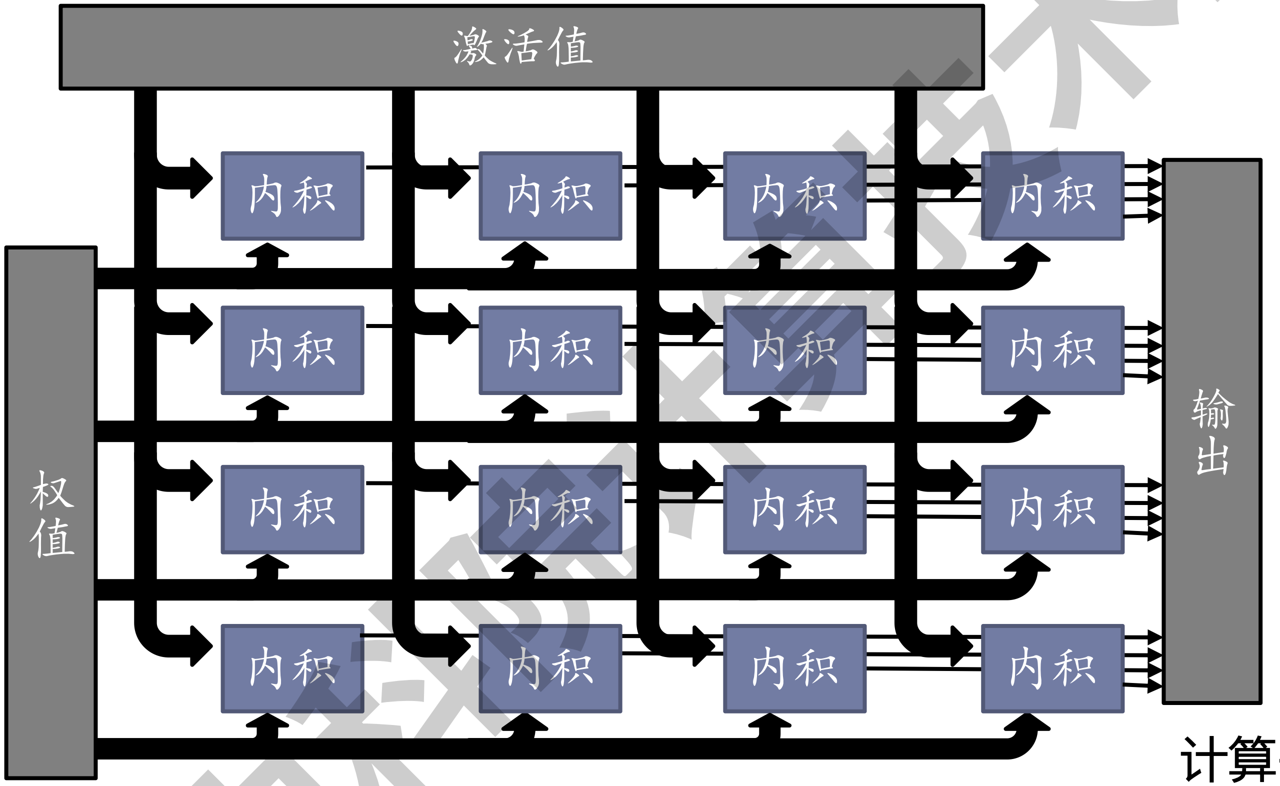


计算-I/O比例 = 56:32 = 1:0.6

矩阵运算单元

- 扩大规模

矩阵乘矩阵单元



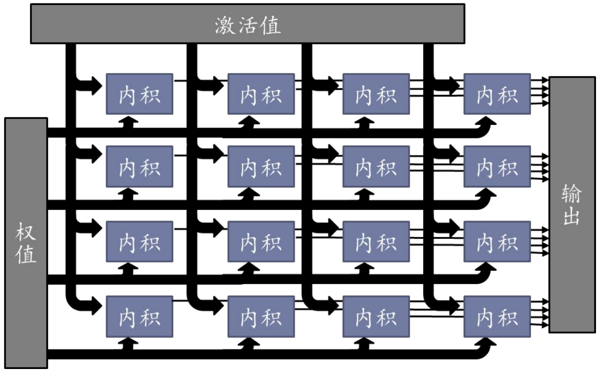
计算-I/O比例 = 112:48 = 1:0.4

矩阵运算单元

- **矩阵乘向量单元**
  - 计算密度已经较好

计算-I/O比例 = 1:0.3

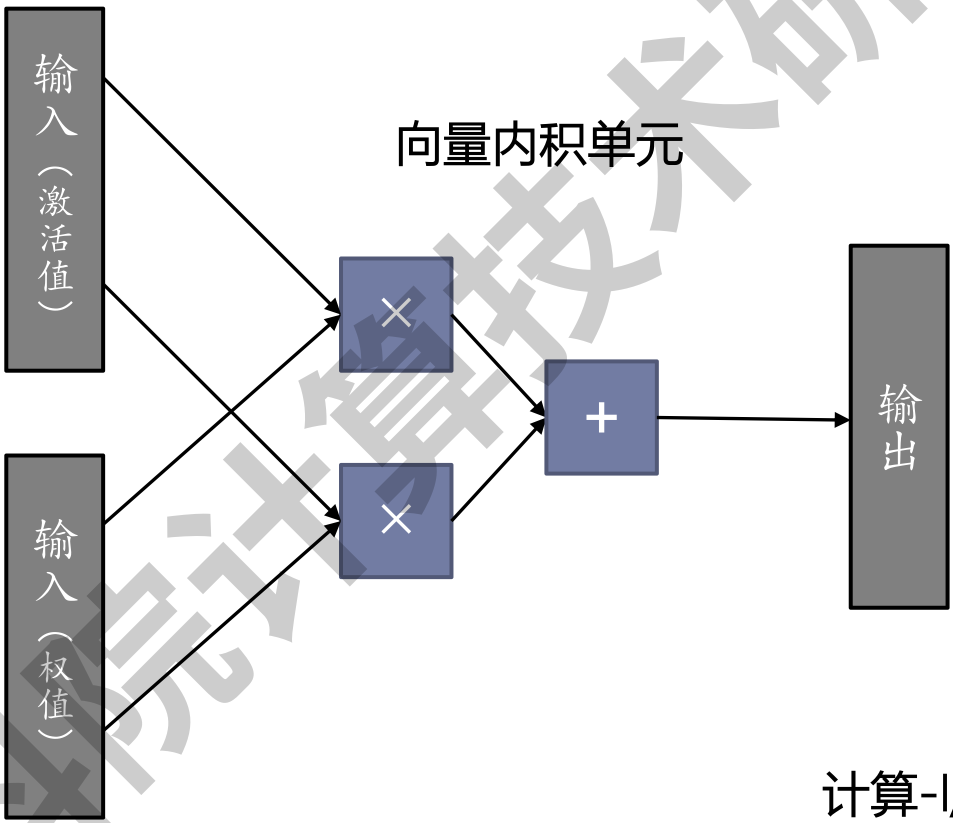
- **矩阵乘矩阵单元**
  - 优势：规模大时，理论上较好
    - （第六章）
  - 困难：连线复杂，距离远、扇出多
  - 规模不大时，未取得实际优势



计算-I/O比例 = 1:0.4

矩阵运算单元

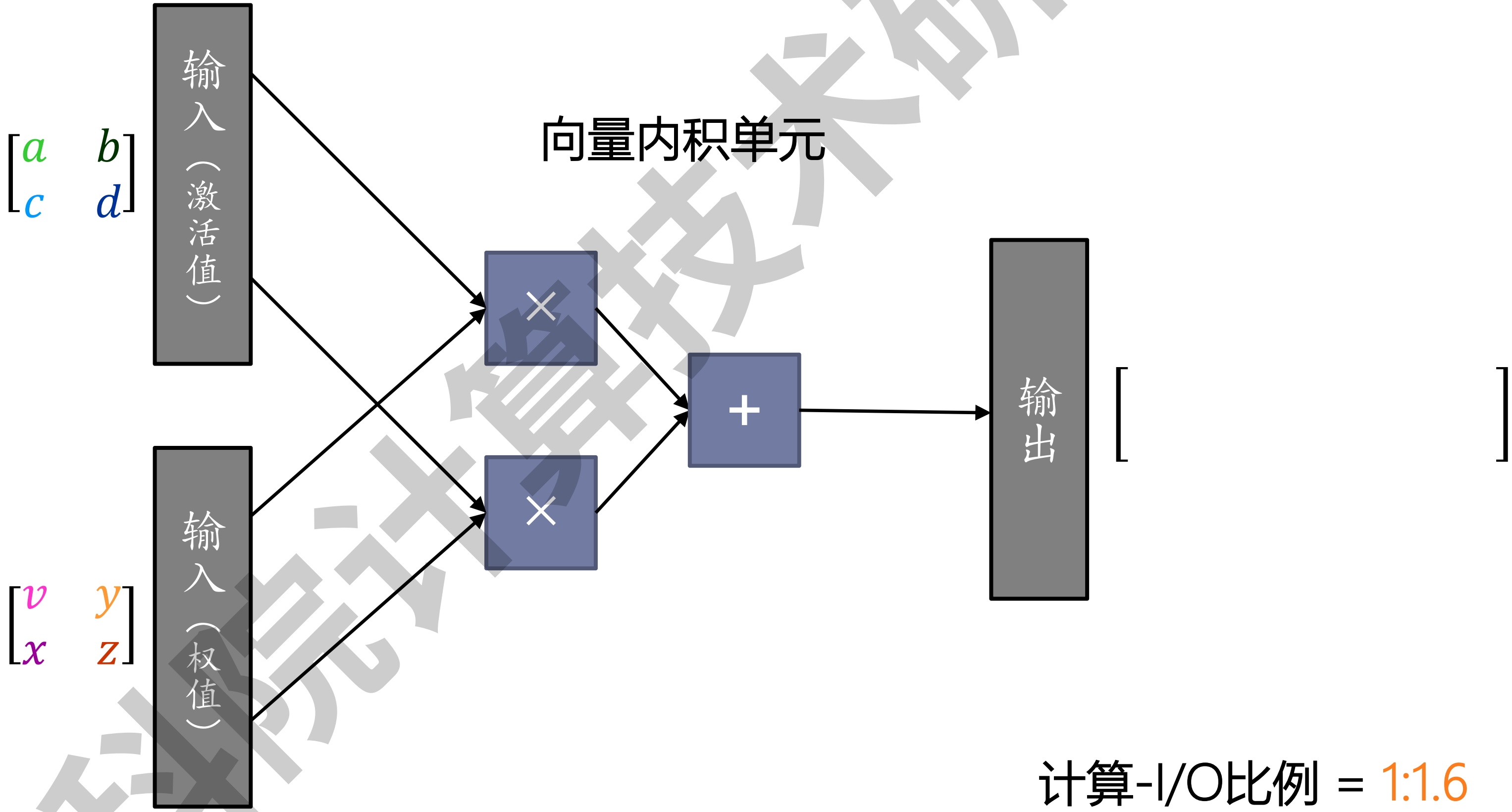
如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$



计算-I/O比例 = 3:5 = 1:1.6

矩阵运算单元

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$

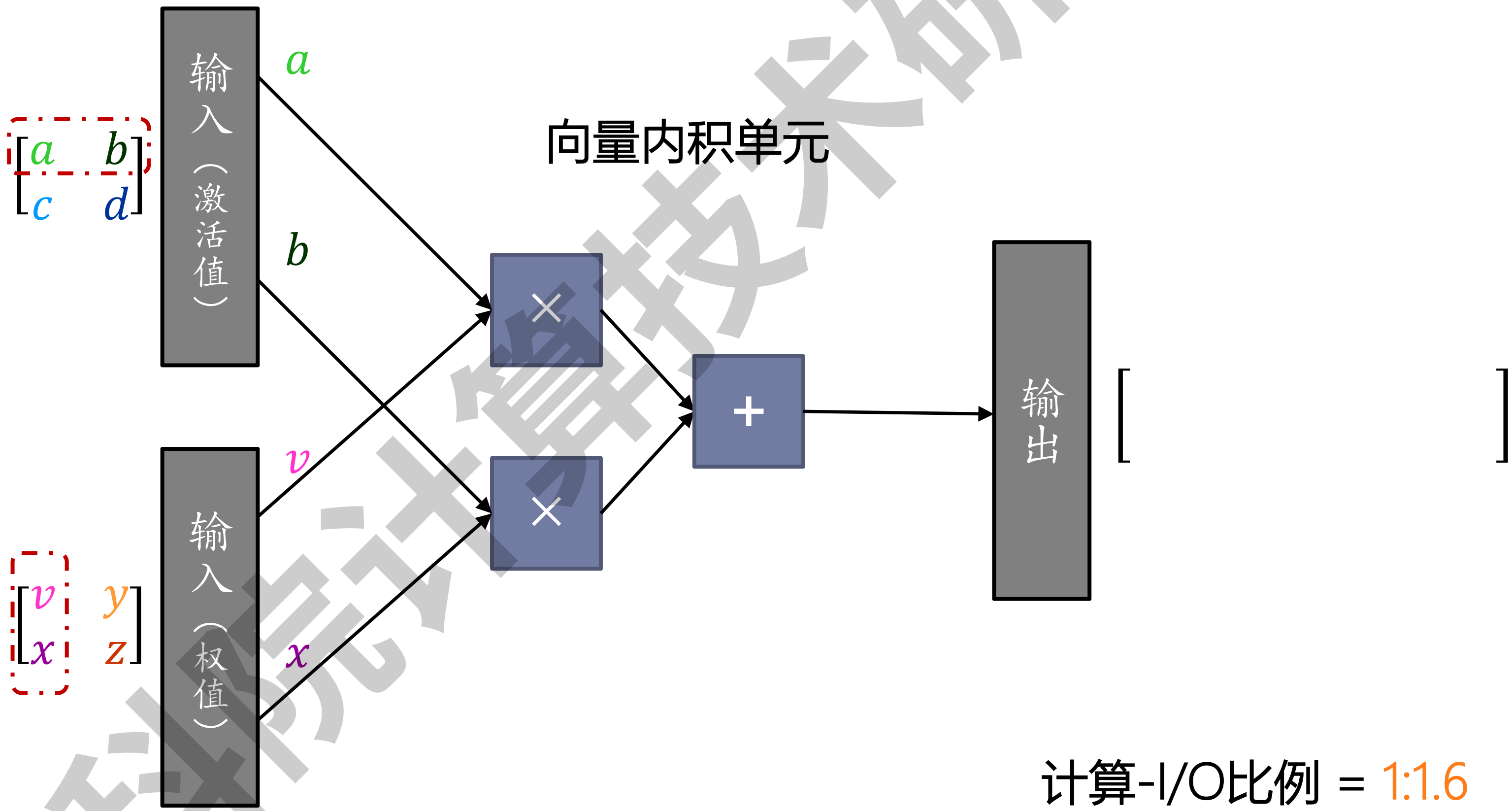


计算-I/O比例 = 1:1.6

矩阵运算单元

如何完成矩阵运算？

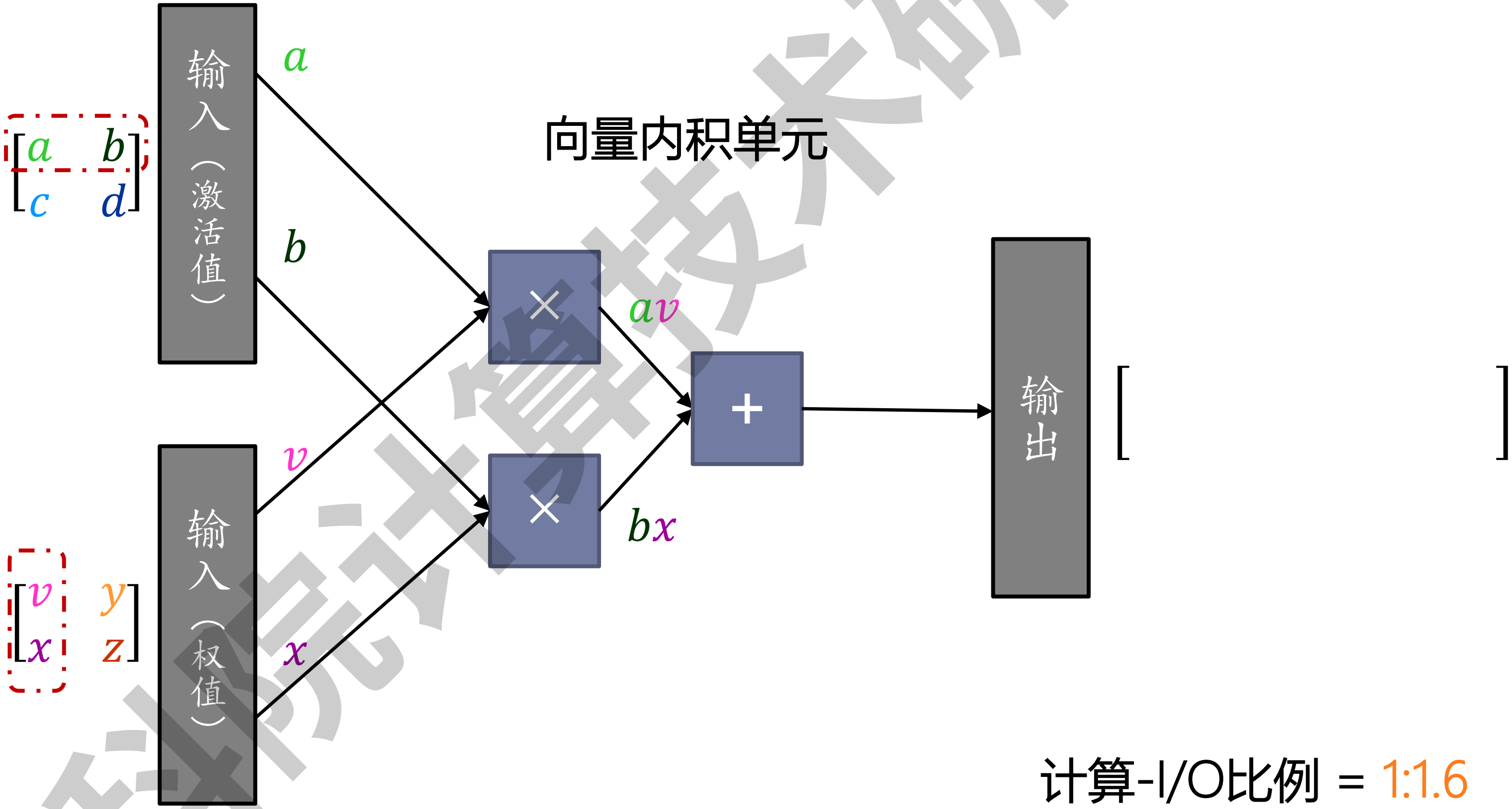
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$$



计算-I/O比例 = 1:1.6

矩阵运算单元

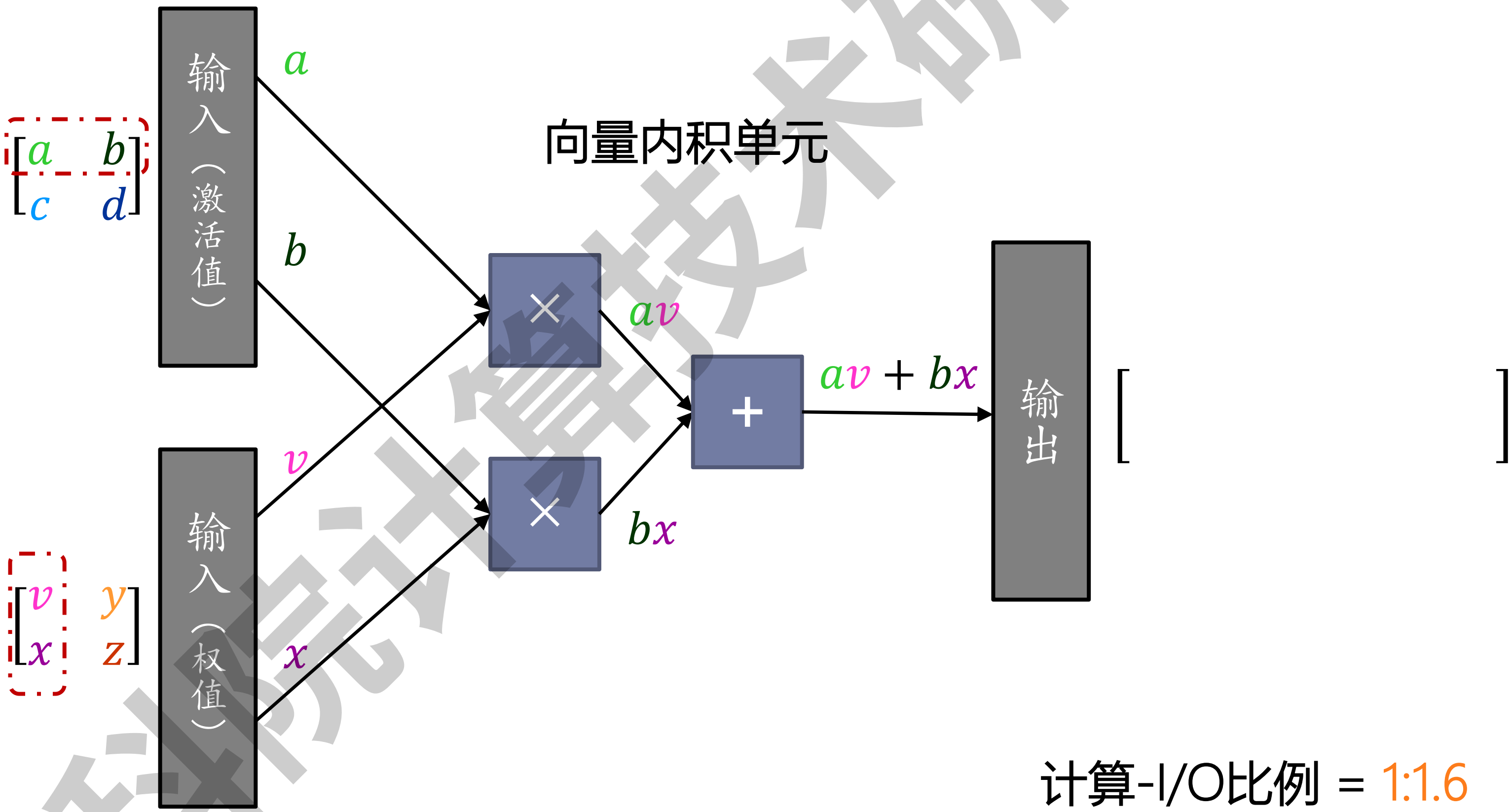
如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$



计算-I/O比例 = 1:1.6

矩阵运算单元

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$

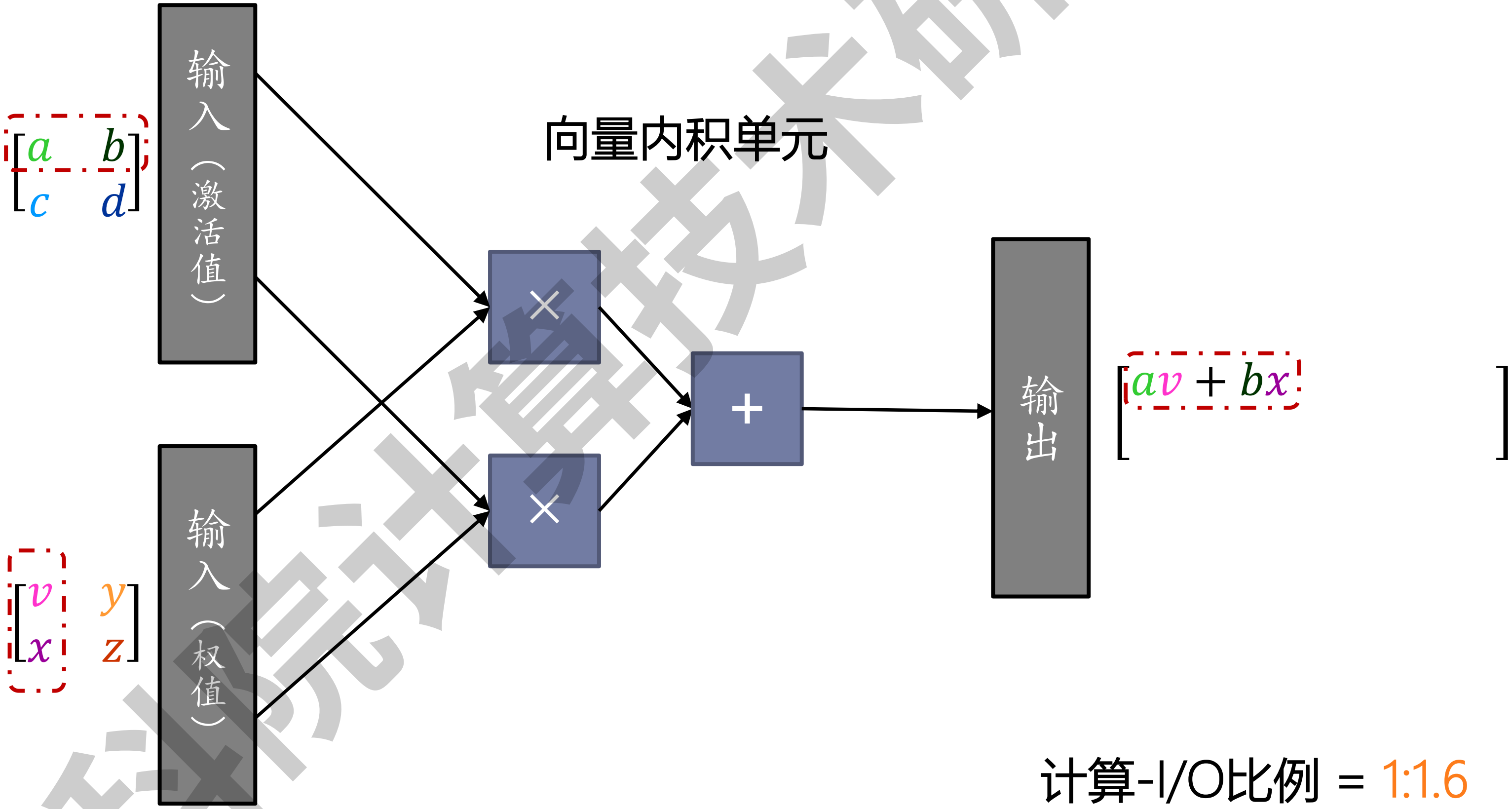


计算-I/O比例 = 1:1.6

矩阵运算单元

如何完成矩阵运算？

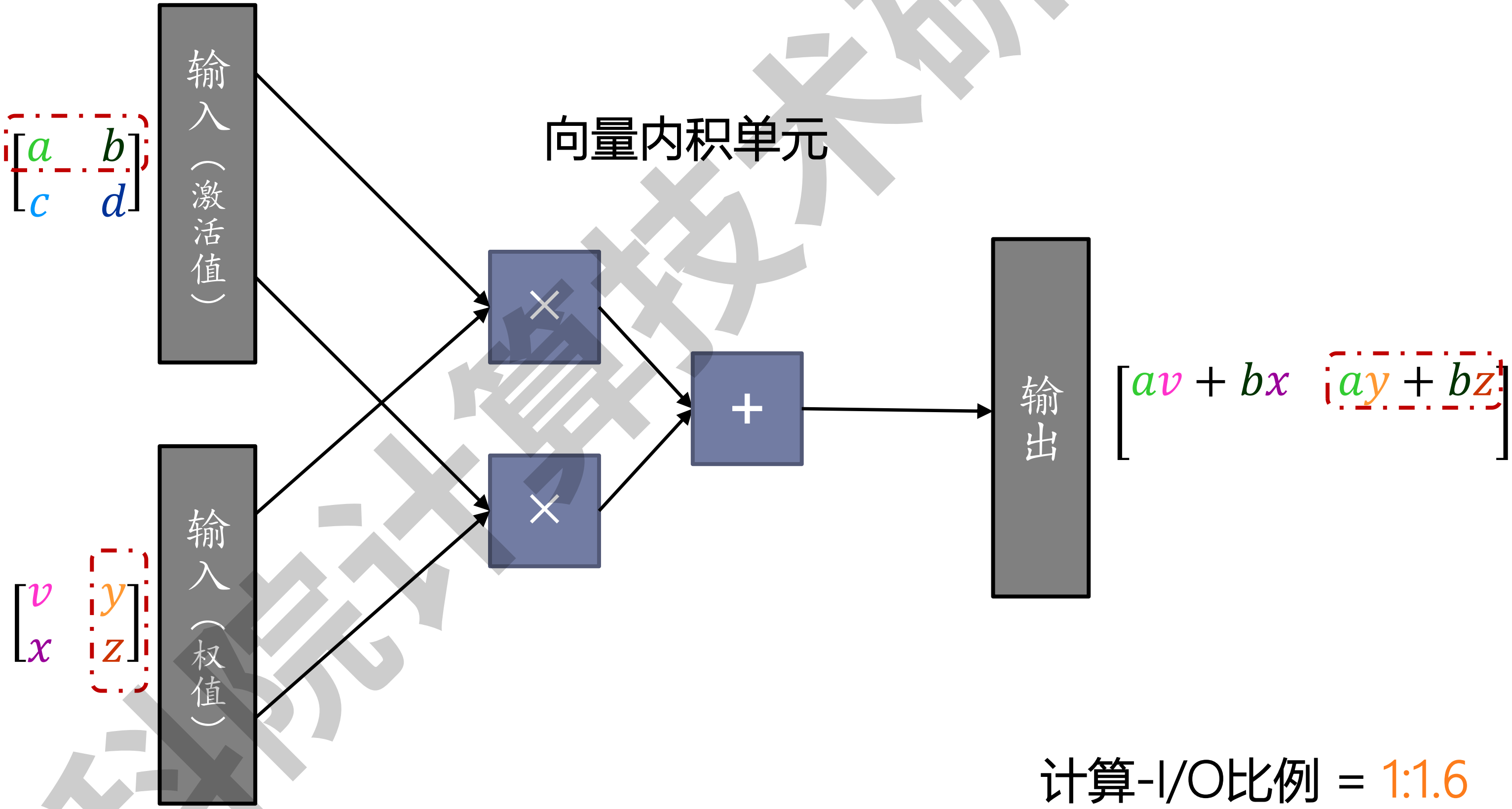
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$



计算-I/O比例 = 1:1.6

矩阵运算单元

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$

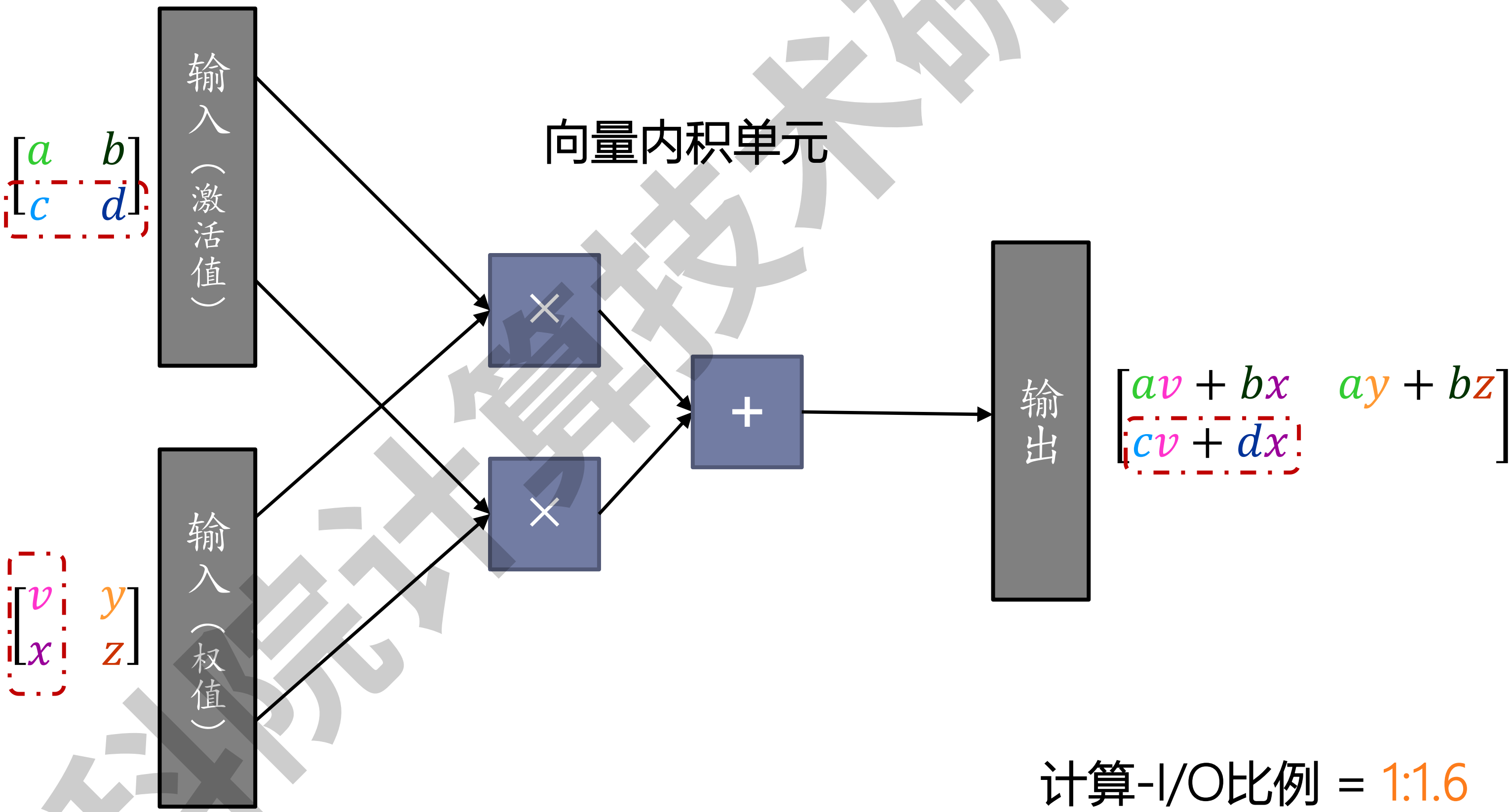


计算-I/O比例 = 1:1.6

矩阵运算单元

如何完成矩阵运算?

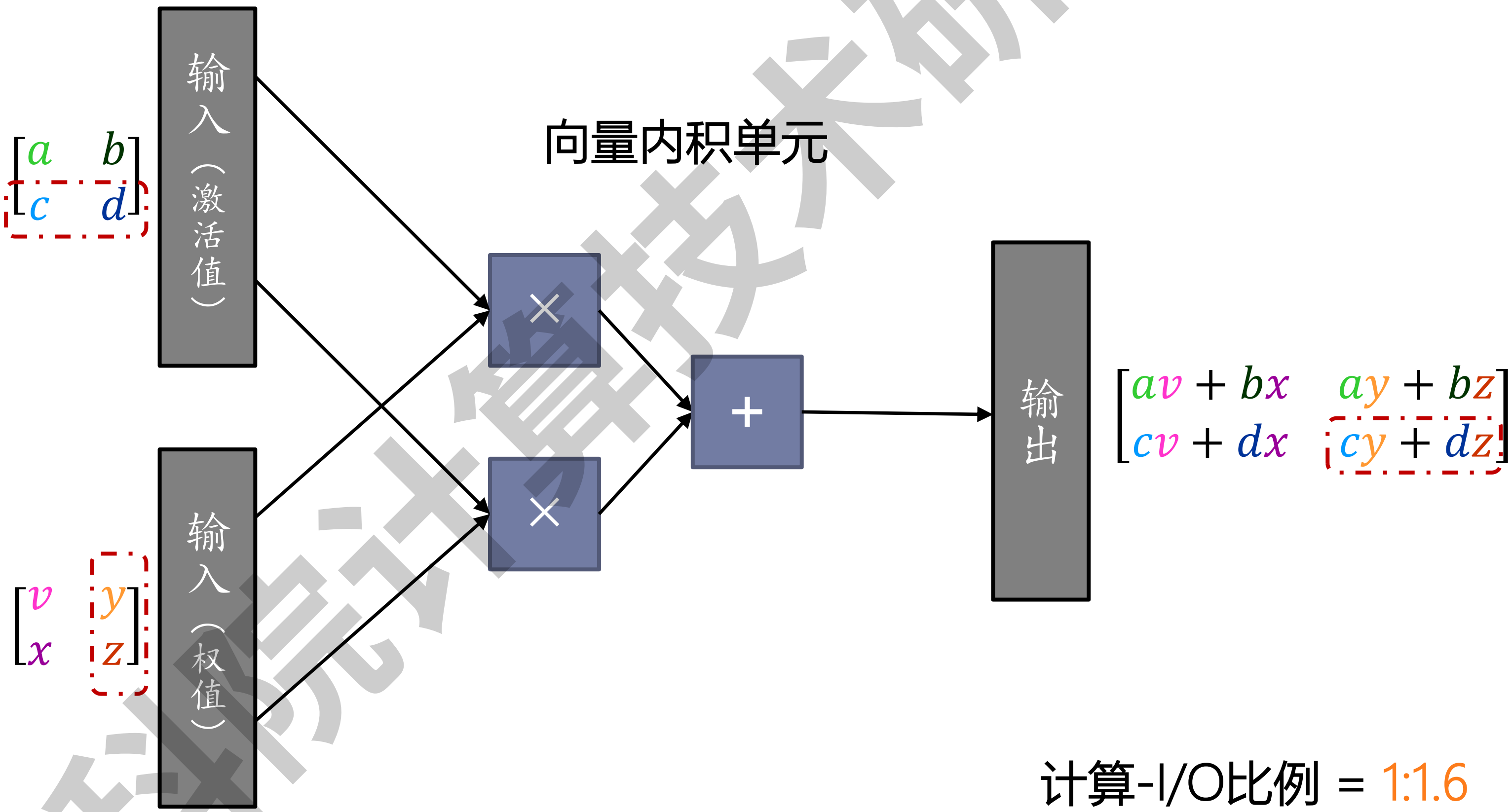
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$



计算-I/O比例 = 1:1.6

矩阵运算单元

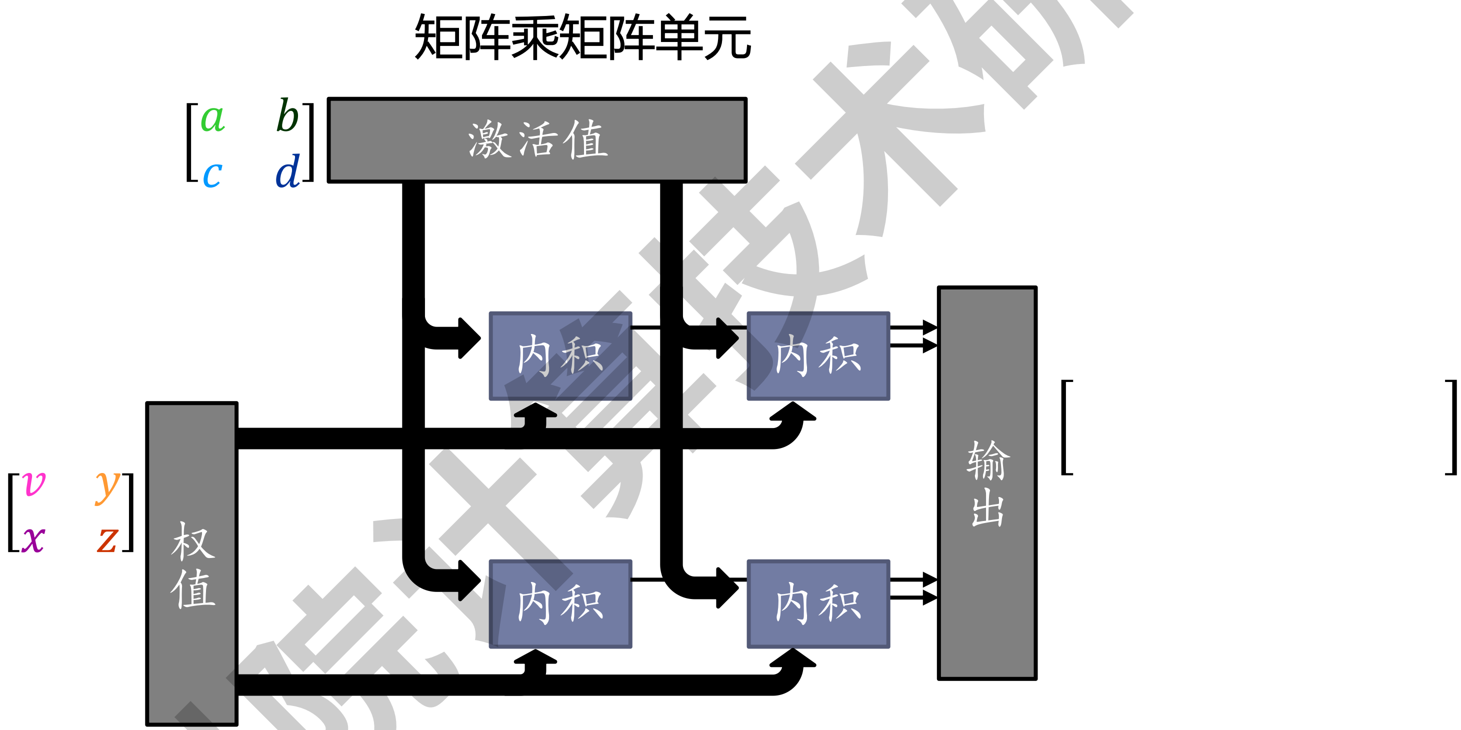
如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$



计算-I/O比例 = 1:1.6

矩阵运算单元

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$

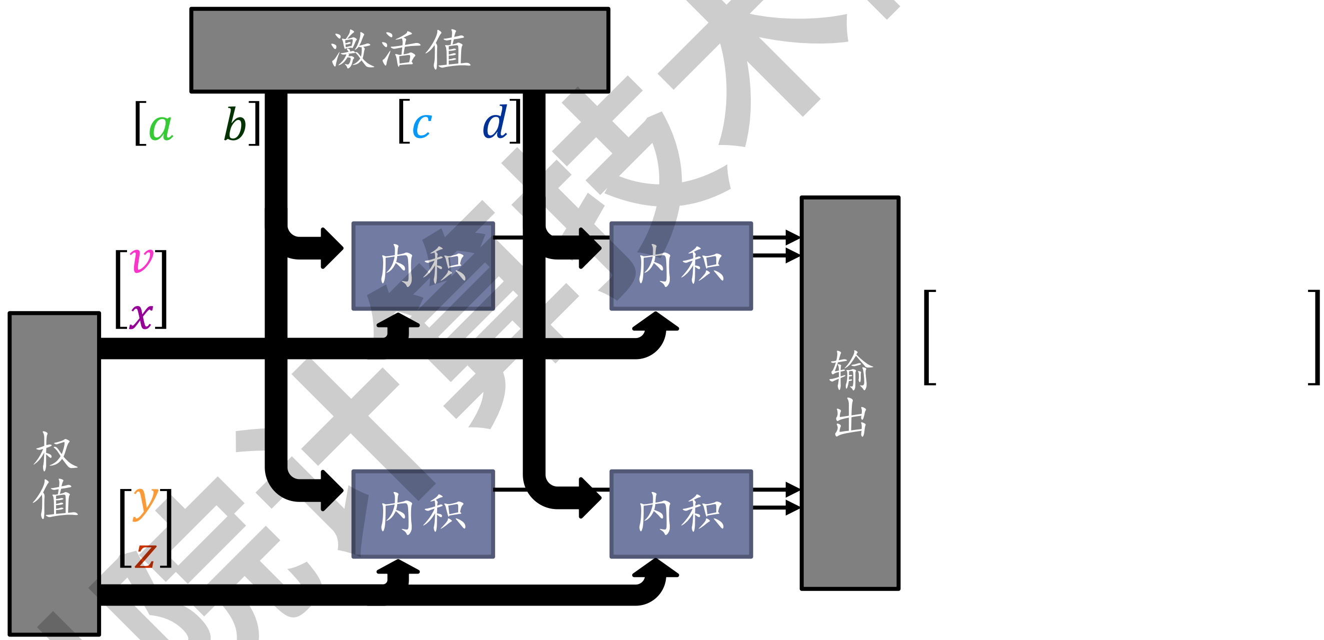


计算-I/O比例 = 1:1

矩阵运算单元

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$

矩阵乘矩阵单元



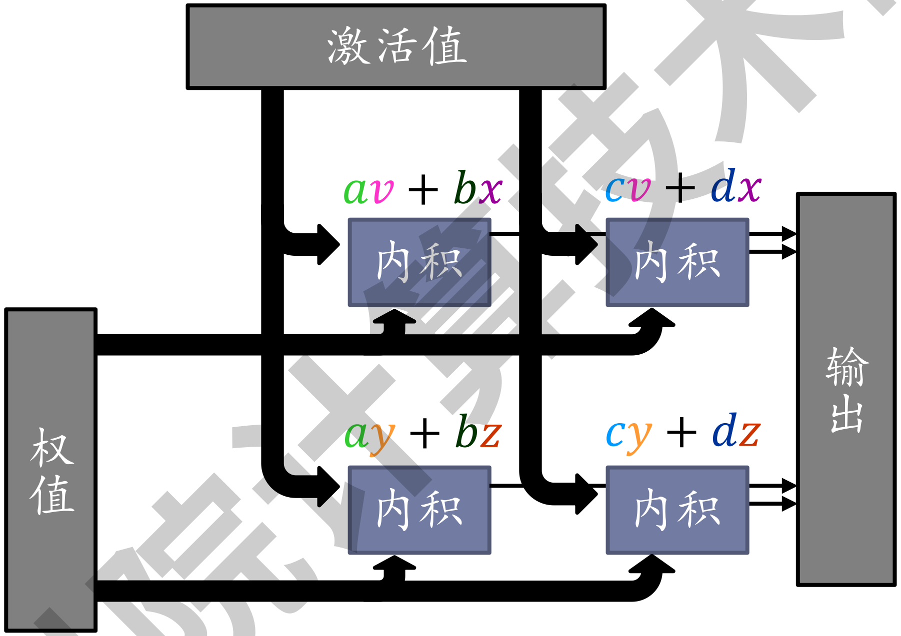
计算-I/O比例 = 1:1

矩阵运算单元

如何完成矩阵运算？

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$

矩阵乘矩阵单元

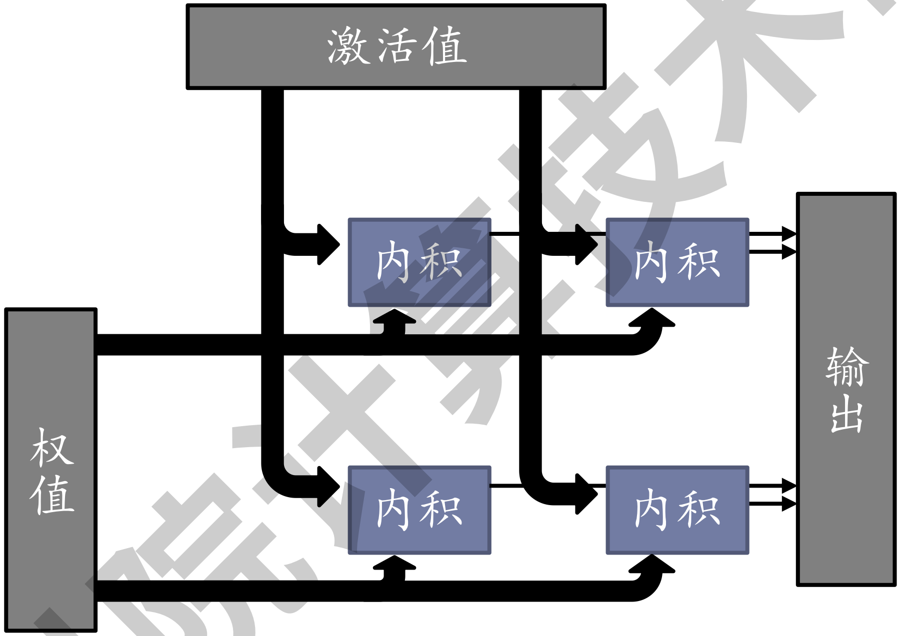


计算-I/O比例 = 1:1

矩阵运算单元

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$

矩阵乘矩阵单元



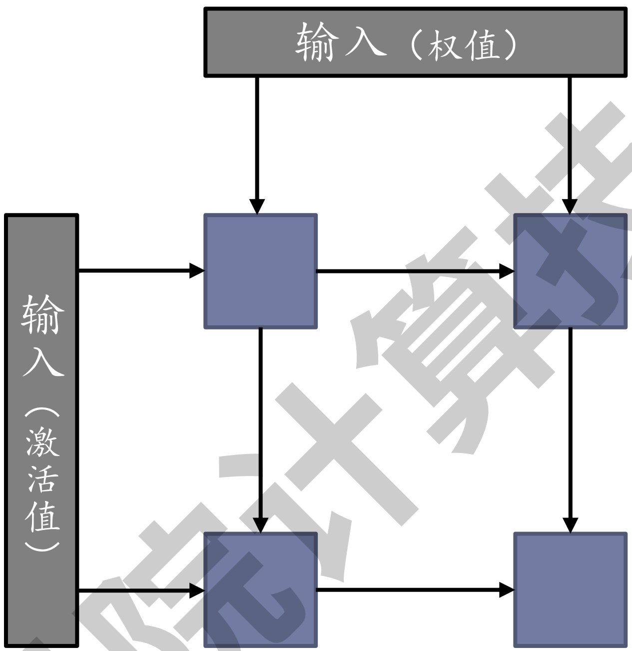
$\begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$

计算-I/O比例 = 1:1

矩阵运算单元

- **问题**：连线距离远、扇出多
- 还有其他方式吗?

脉动阵列机

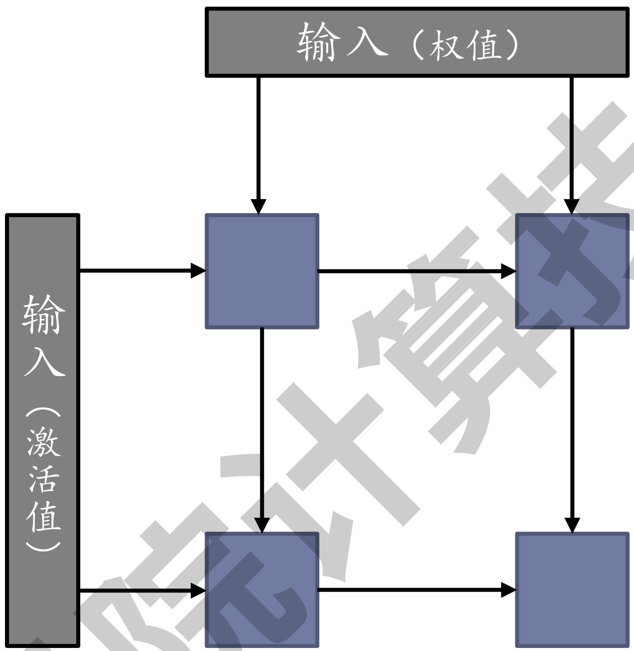


计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算?

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$$

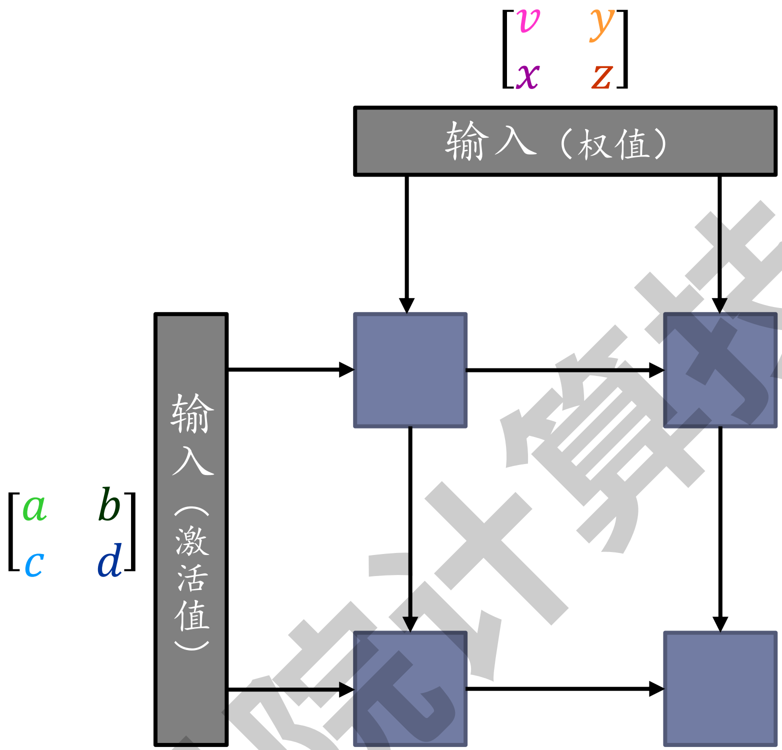


计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算？

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$

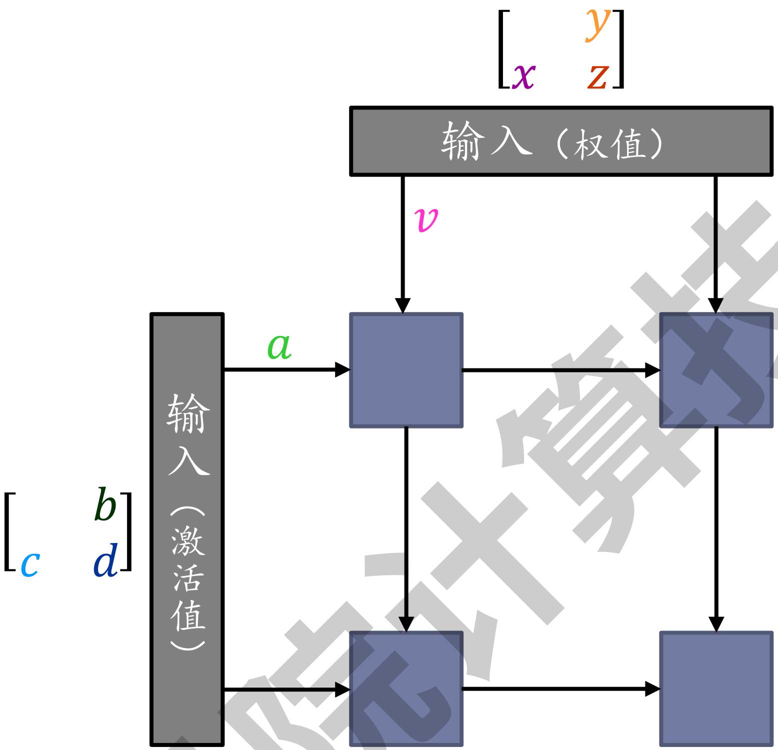


计算-I/O比例 = ？

脉动阵列机

如何完成矩阵运算？

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$$

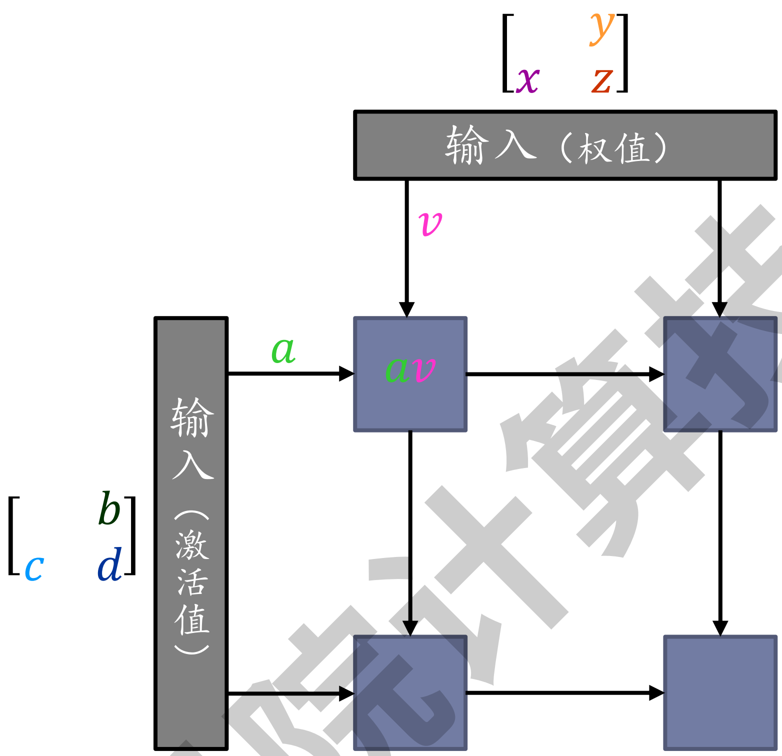


计算-I/O比例 = ？

脉动阵列机

如何完成矩阵运算?

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$$

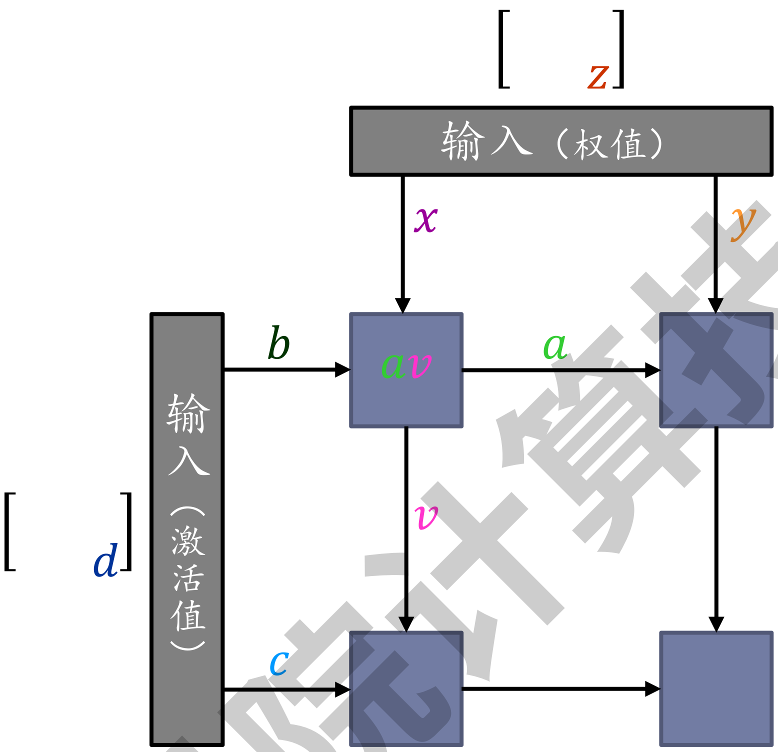


计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算?

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$$

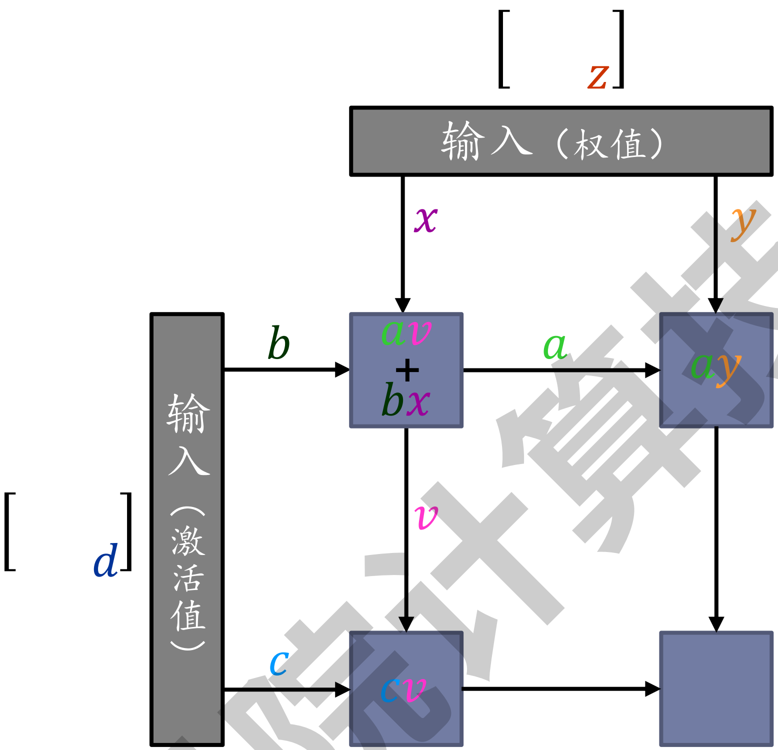


计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算？

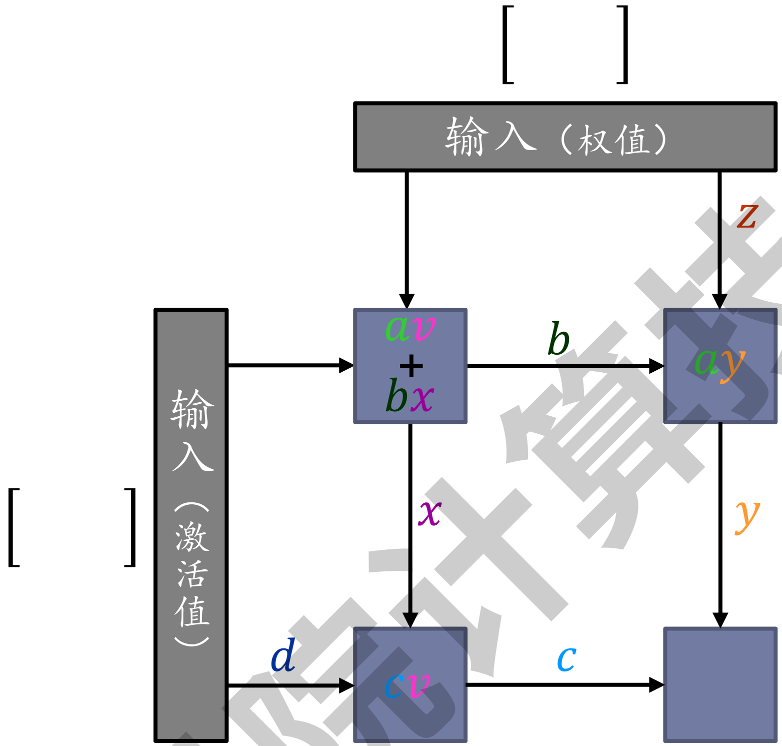
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$



计算-I/O比例 = ?

脉动阵列机

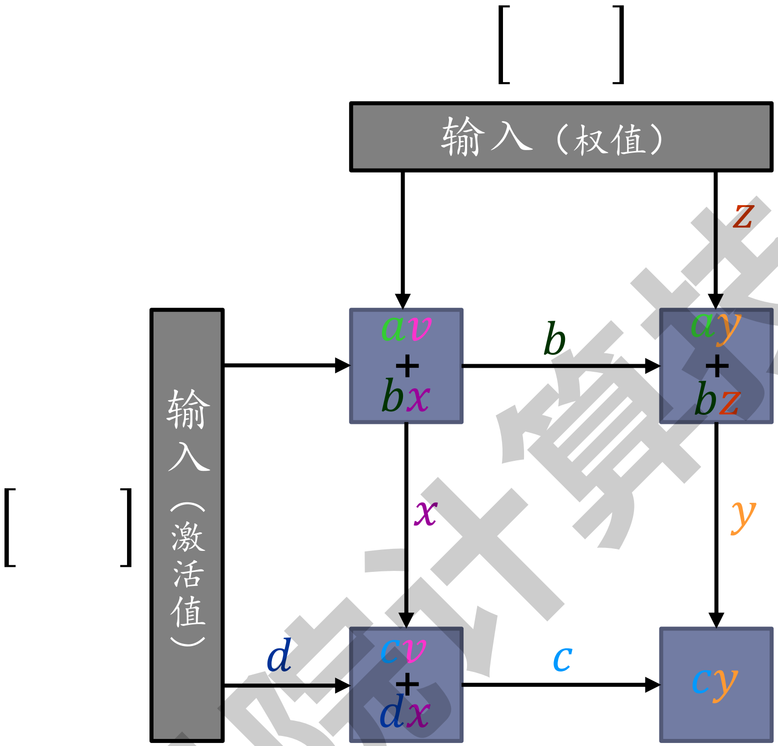
如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$



计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$

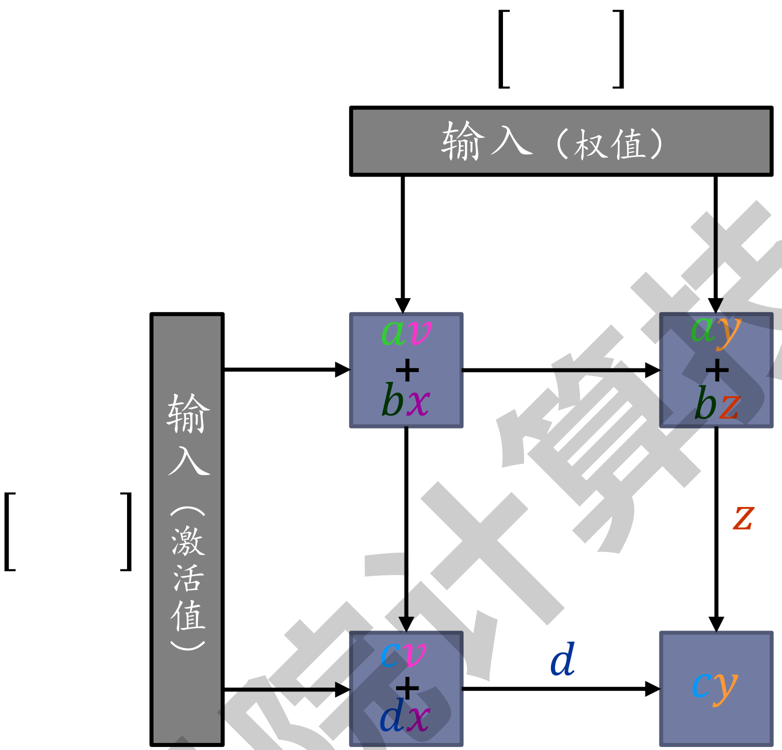


计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算？

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$

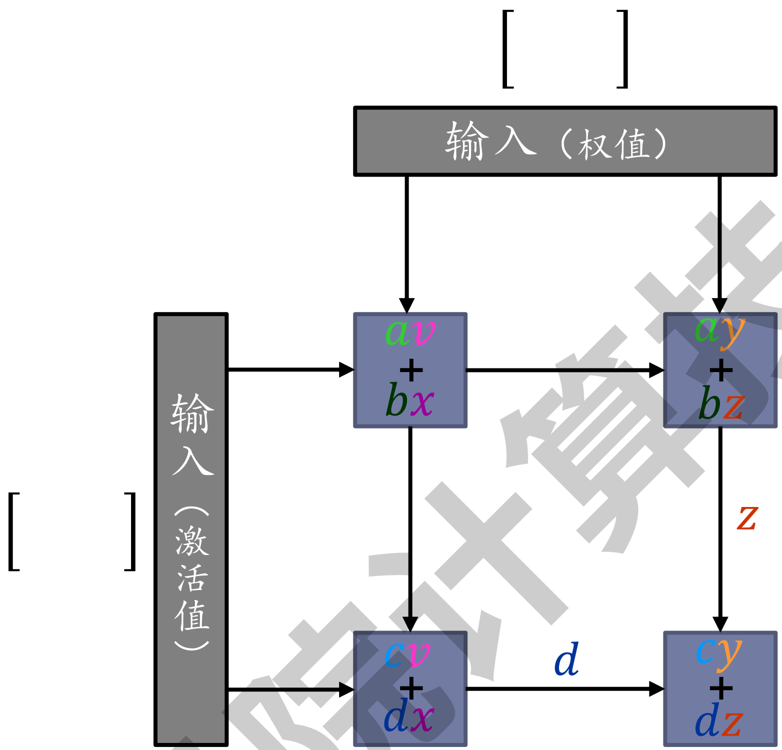


计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算?

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$

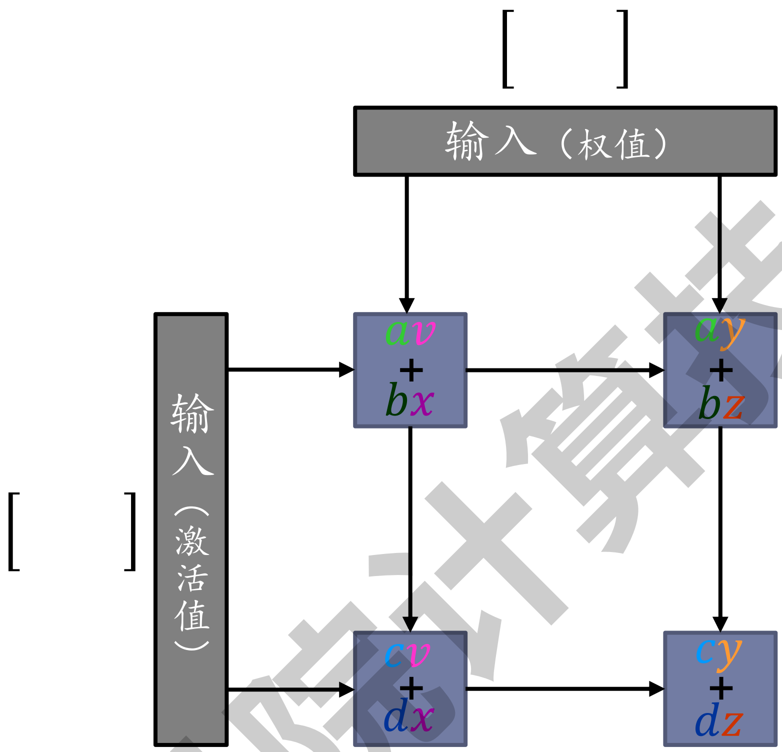


计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算?

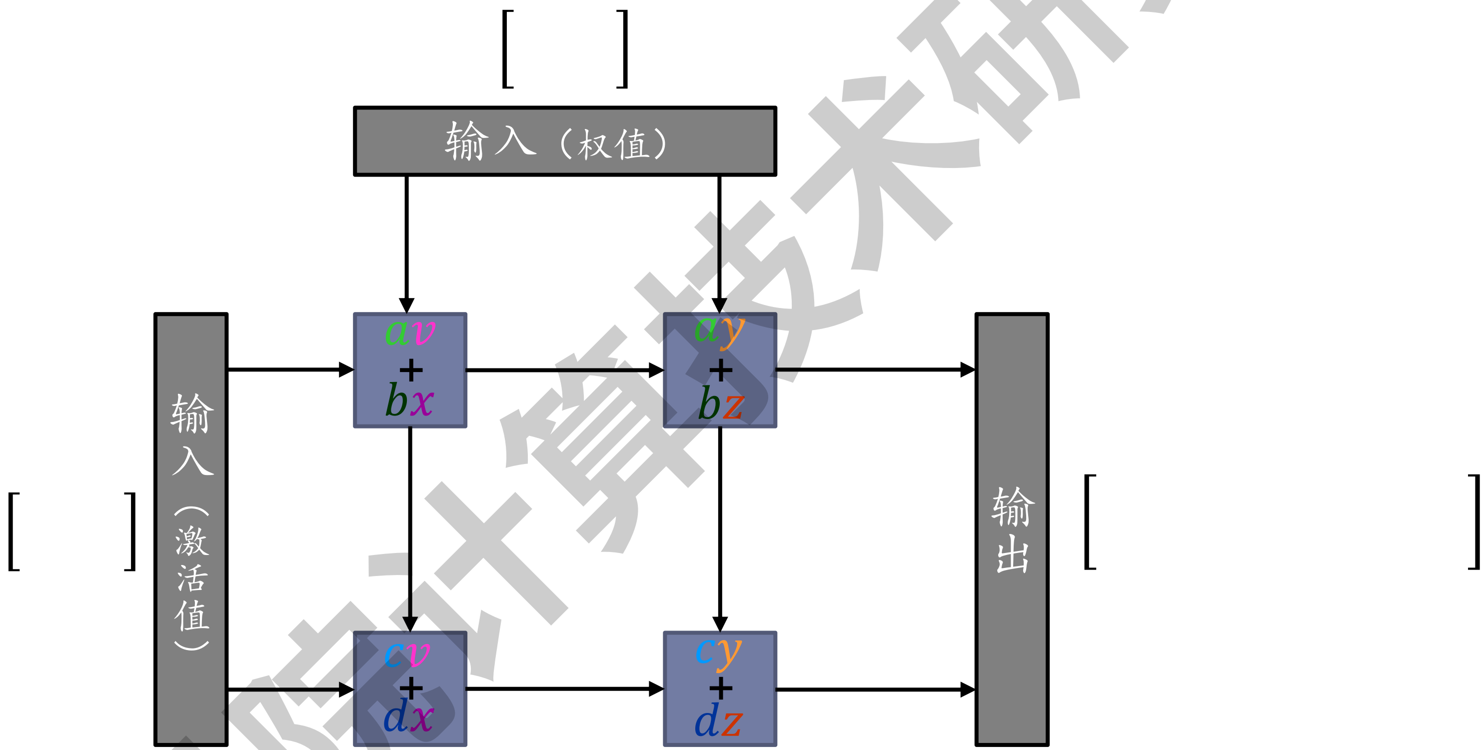
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$$



计算-I/O比例 = ?

脉动阵列机

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av + bx & ay + bz \\ cv + dx & cy + dz \end{bmatrix}$

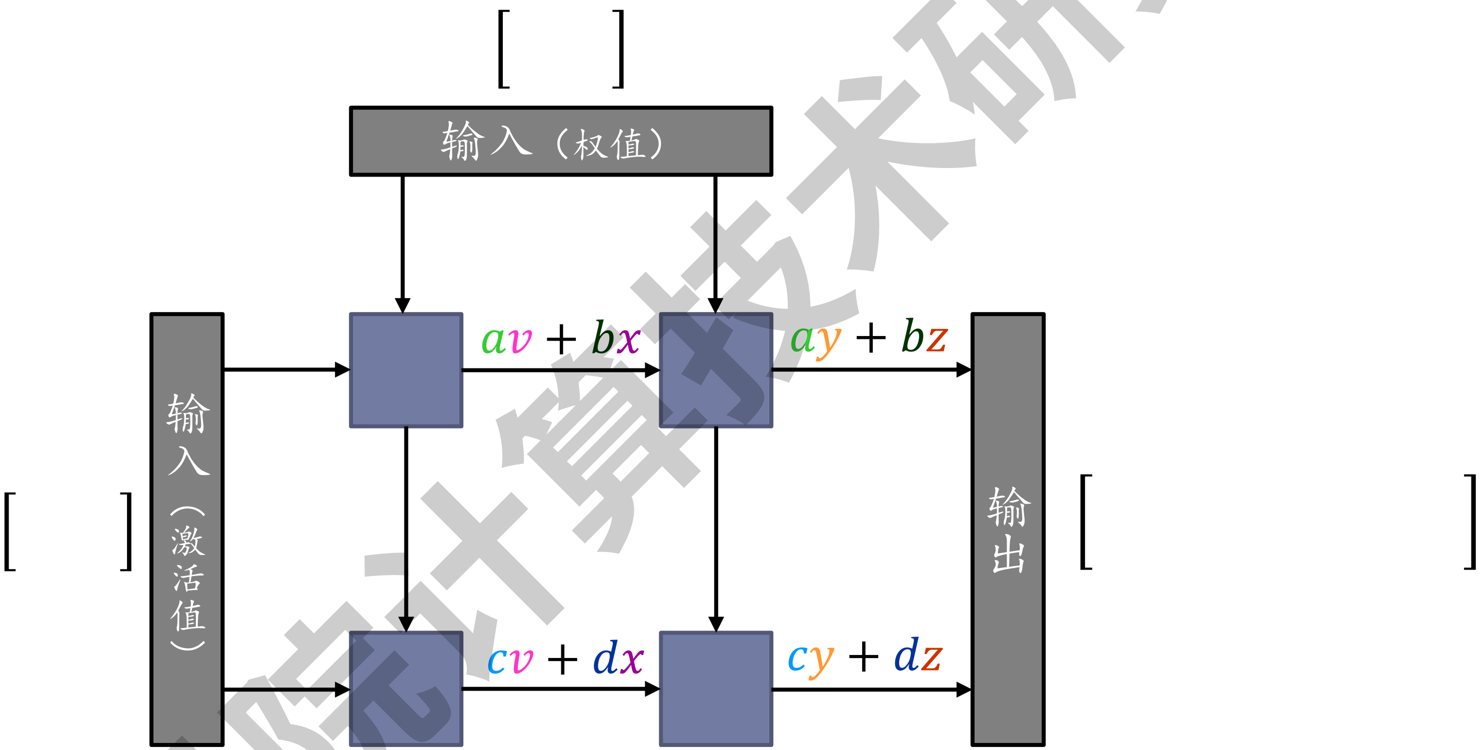


计算-I/O比例 = 1:0.75

脉动阵列机

如何完成矩阵运算?

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$

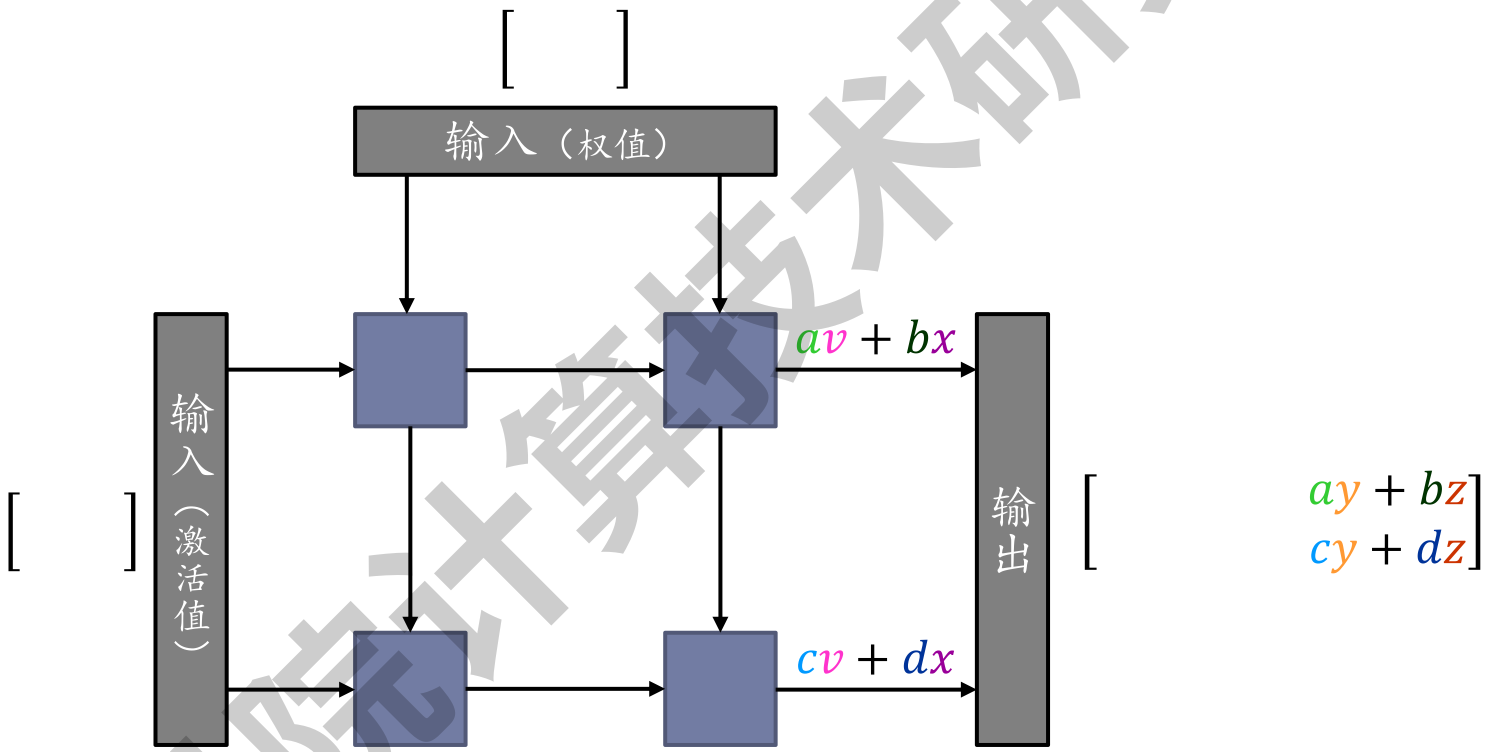


计算-I/O比例 = 1:0.75

脉动阵列机

如何完成矩阵运算?

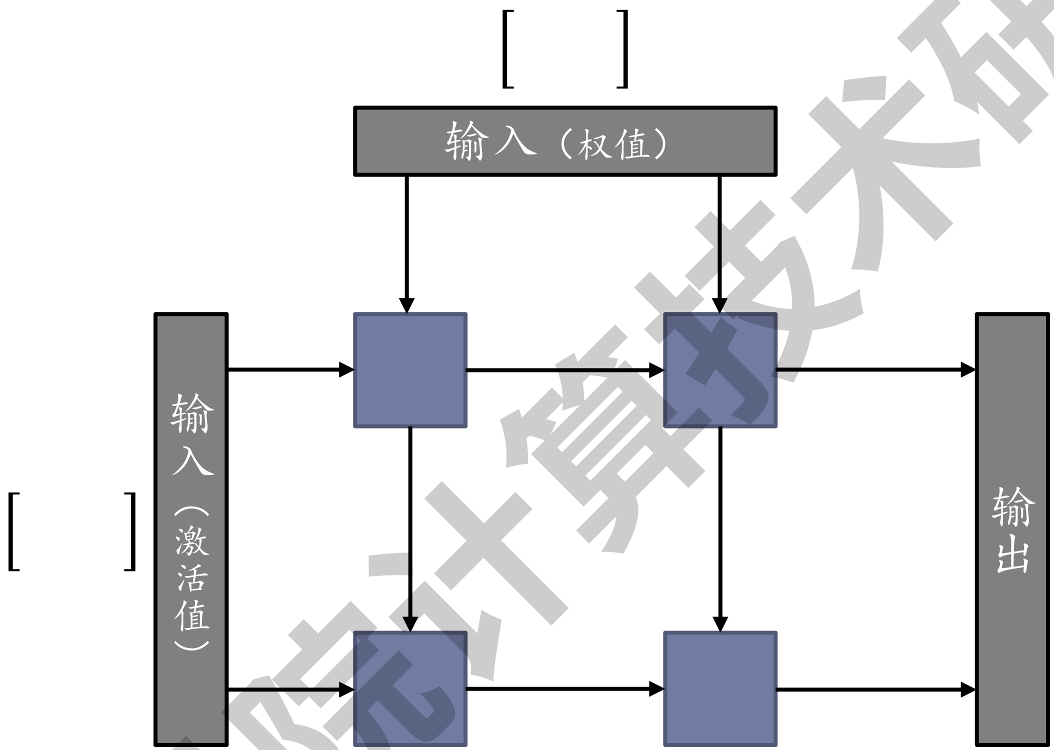
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$$



计算-I/O比例 = 1:0.75

脉动阵列机

如何完成矩阵运算？ $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} v & y \\ x & z \end{bmatrix} = \begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$

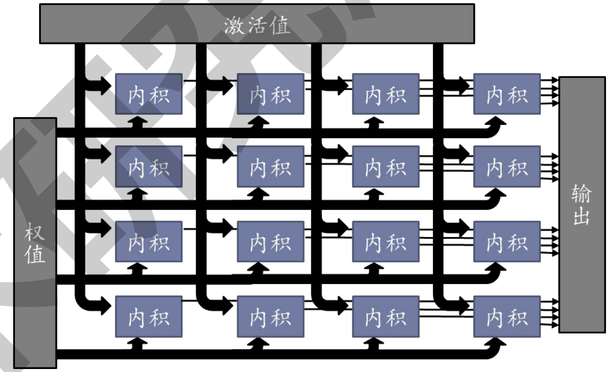


$\begin{bmatrix} av+bx & ay+bz \\ cv+dx & cy+dz \end{bmatrix}$

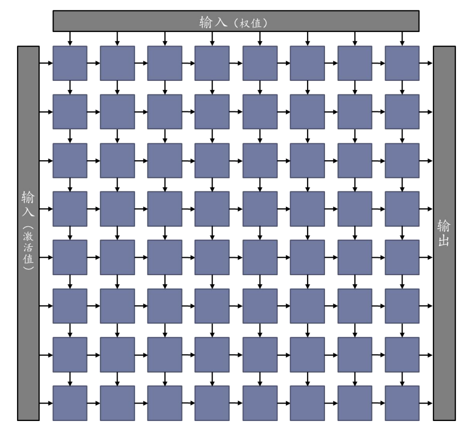
计算-I/O比例 = 1:0.75

矩阵运算单元

- 脉动阵列机 vs 矩阵乘矩阵单元
  - 优势:
    - 计算-I/O比例更高
    - 电路采用局部短连接
    - 扇出少
  - 困难:
    - 延迟高，需要等待启动/排空
    - 专用性更强，高效支持矩阵乘、卷积，但很难改造为同时支持其他功能



计算-I/O比例 = 1:0.4



计算-I/O比例 = 1:0.2

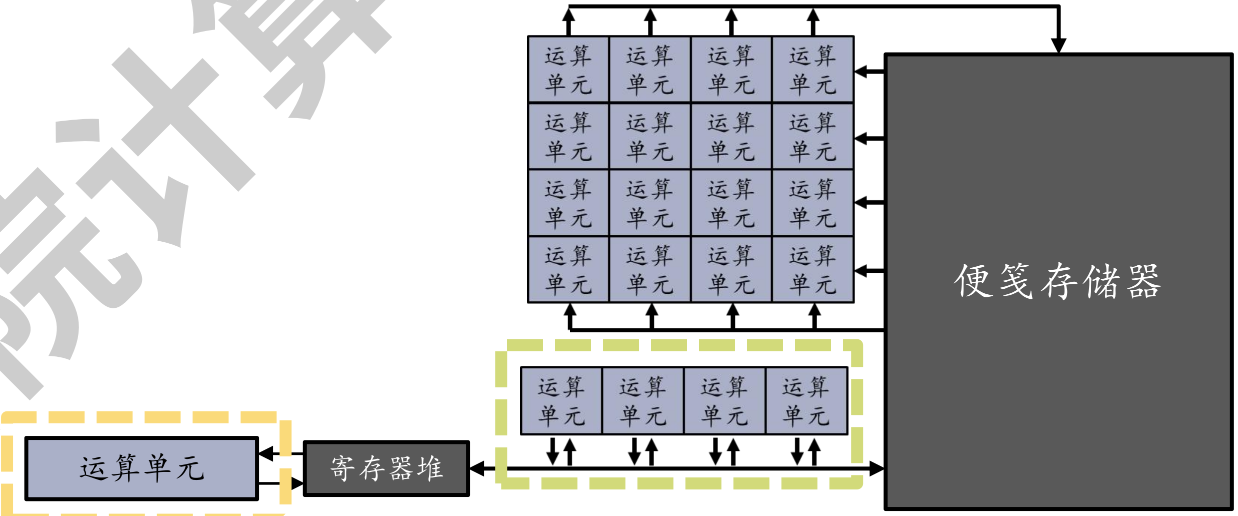
历史

- 脉动阵列机（systolic array）
- 相似概念出现于二战时期
  - 英国*巨人计算机二型*（Colossus Mark II，1944）
  - 用于破译纳粹德国军事密文，长期处于保密状态，战后被销毁
- 孔祥重、Charles E. Leiserson于1978年发明
  - 多种结构，对应多种算法
  - 分别用于矩乘、线性方程组求解、LU分解、最大公约数等

向量和标量单元

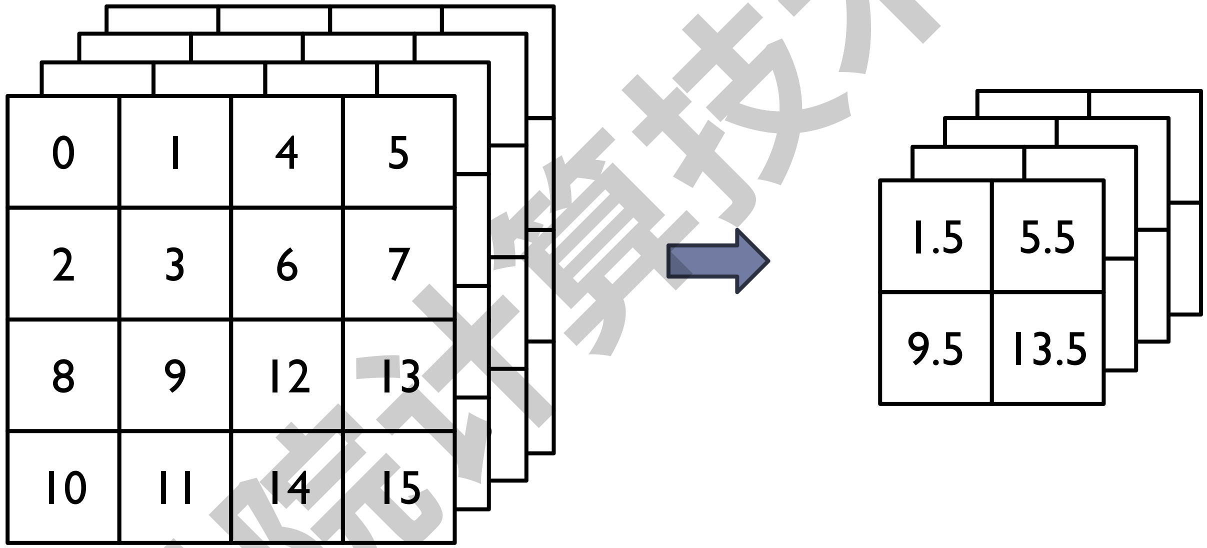
主要功能：

- 池化、归一化
- Dropout、ReLU、Sigmoid、Softmax等特殊变换
- 求最大/最小值、排序、计数、前缀求和等
- 数据重排布



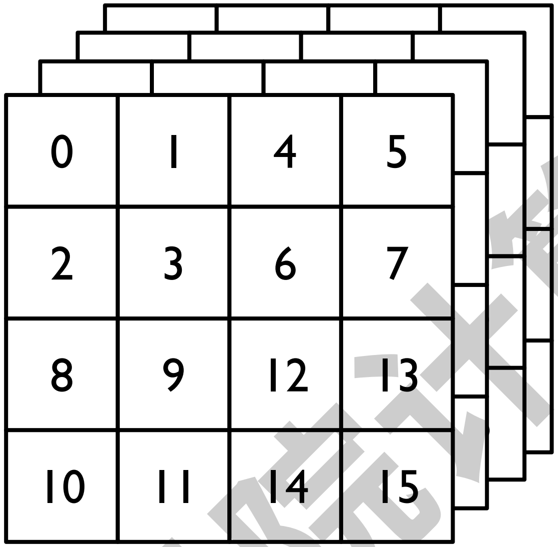
池化/均一化

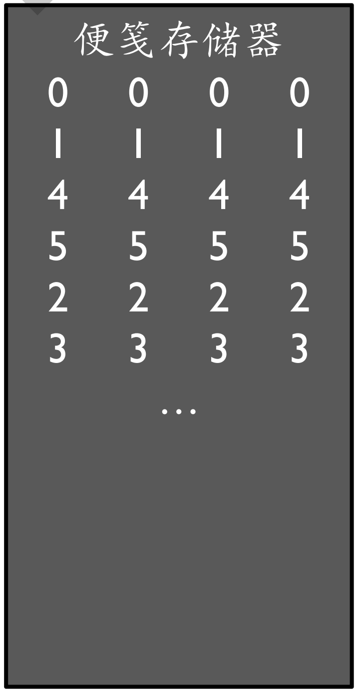
- 如何完成池化?



池化/均一化

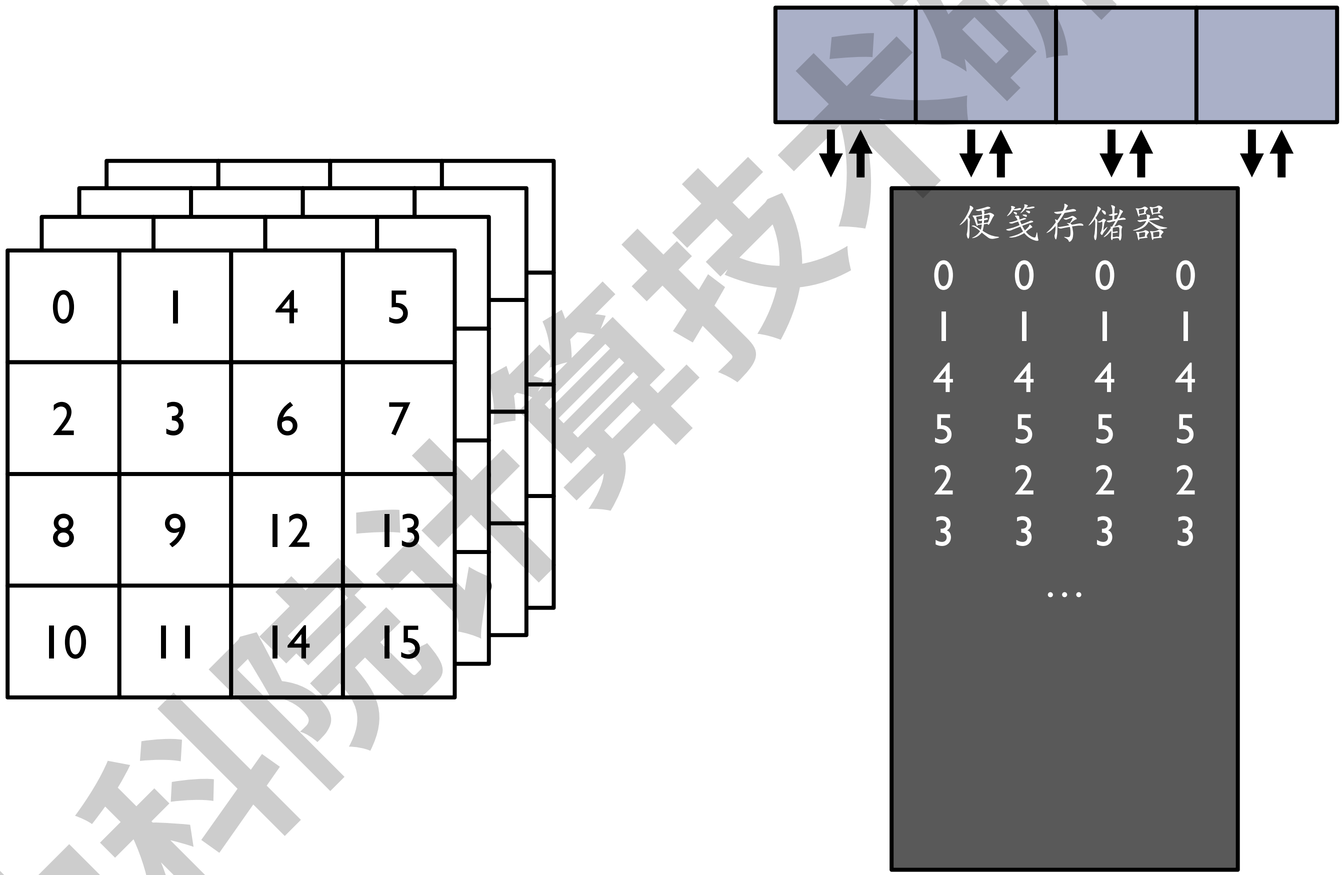
- 如何完成池化?





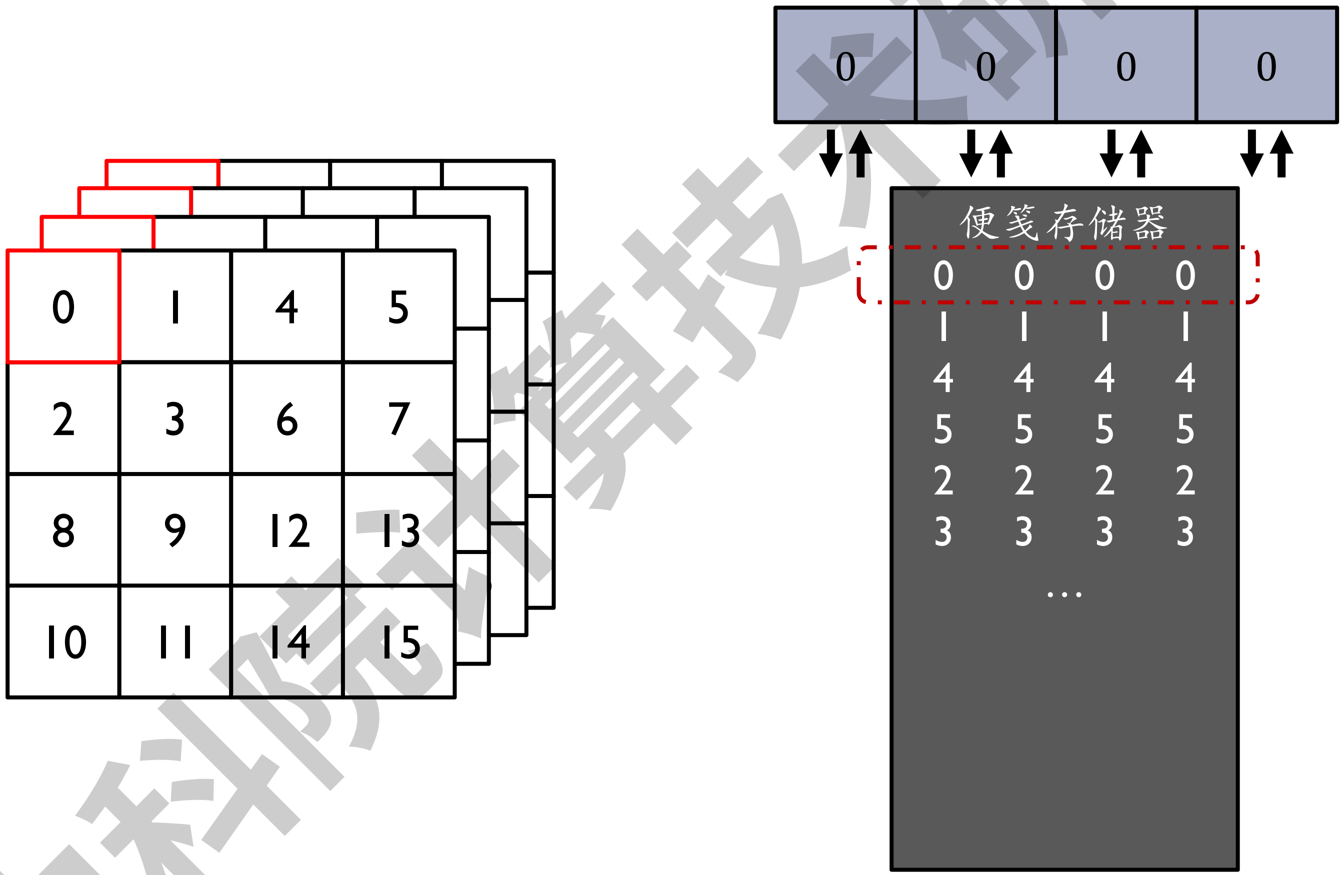
池化/均一化

- 如何完成池化?



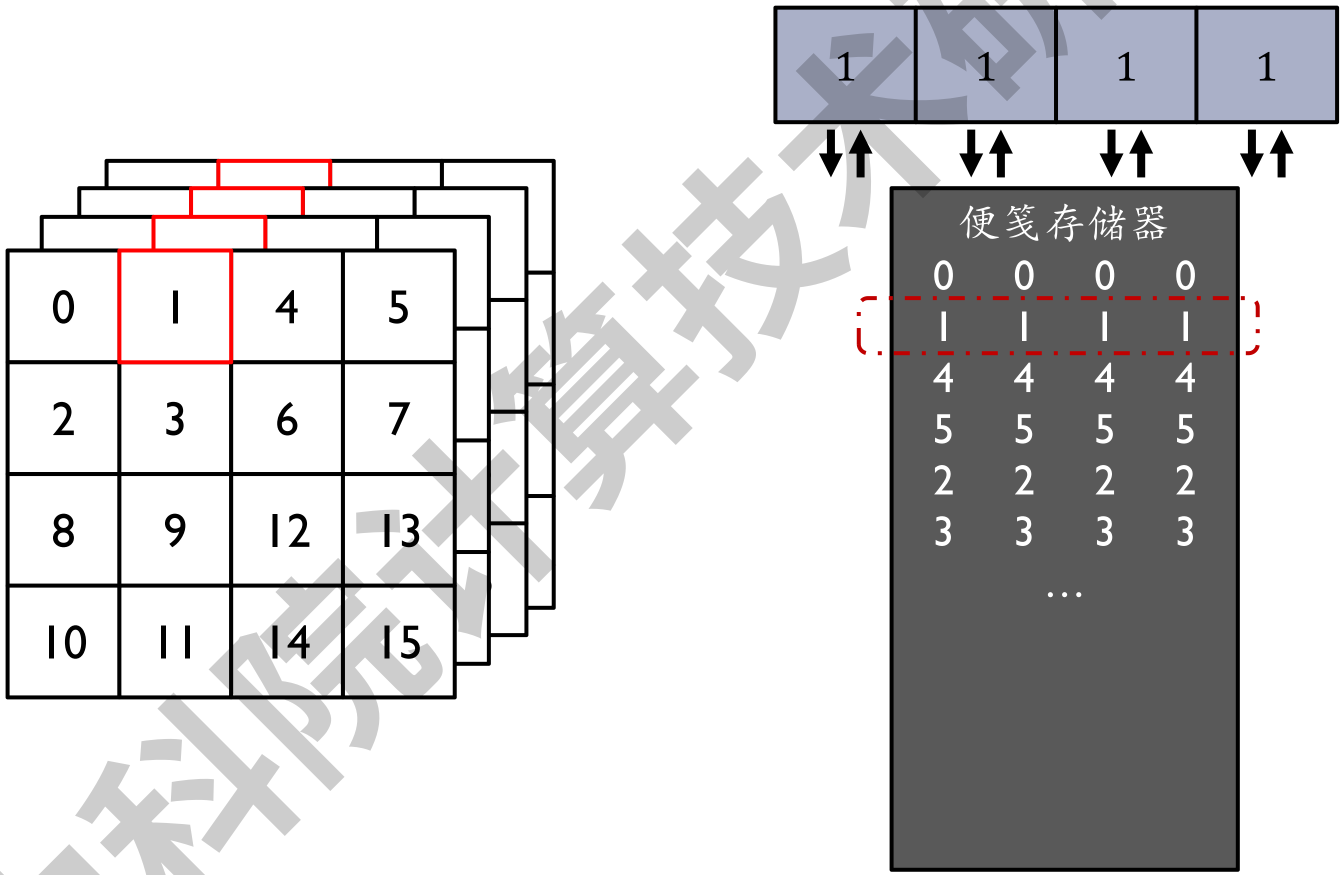
池化/均一化

- 如何完成池化?



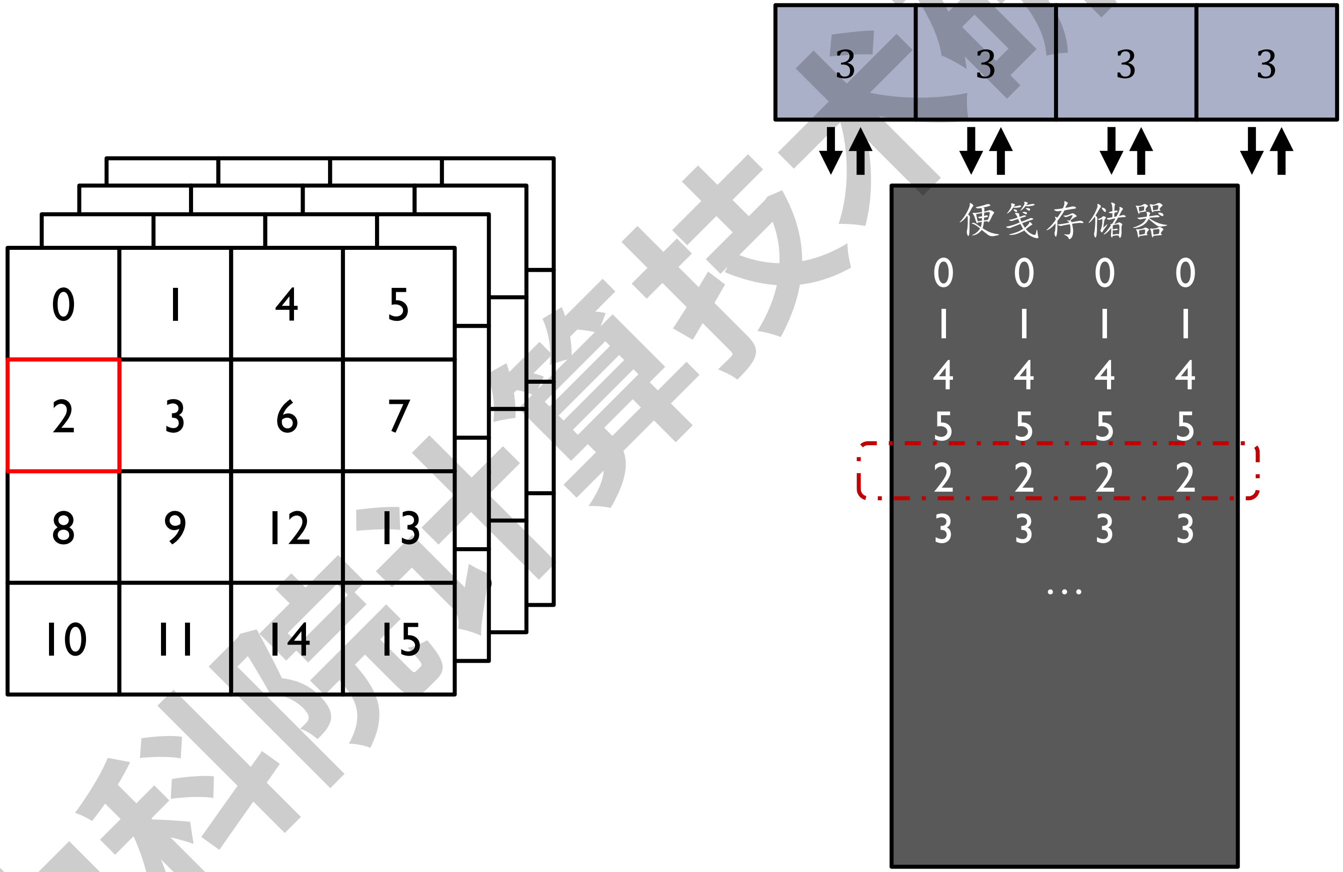
池化/均一化

- 如何完成池化?



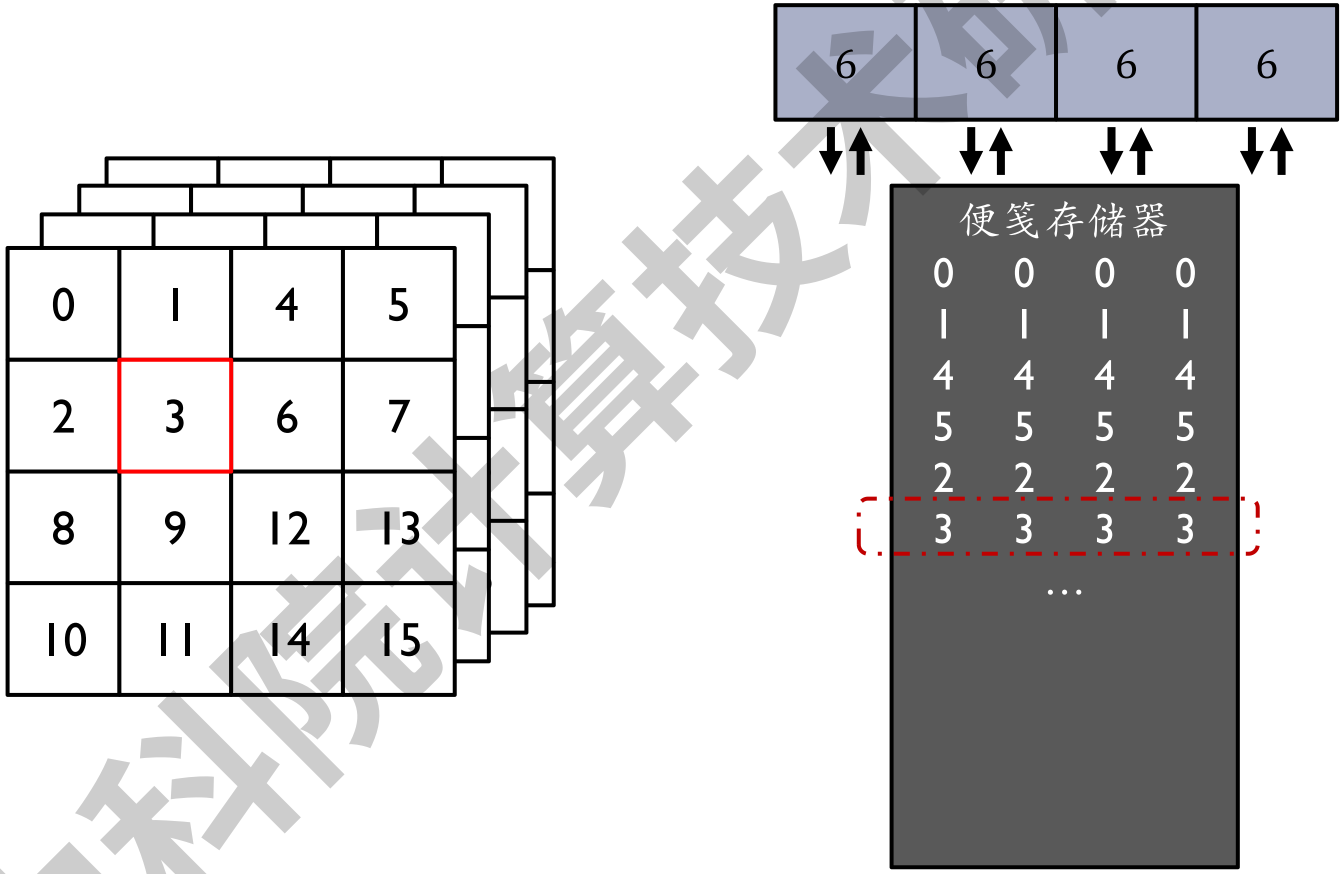
池化/均一化

- 如何完成池化?



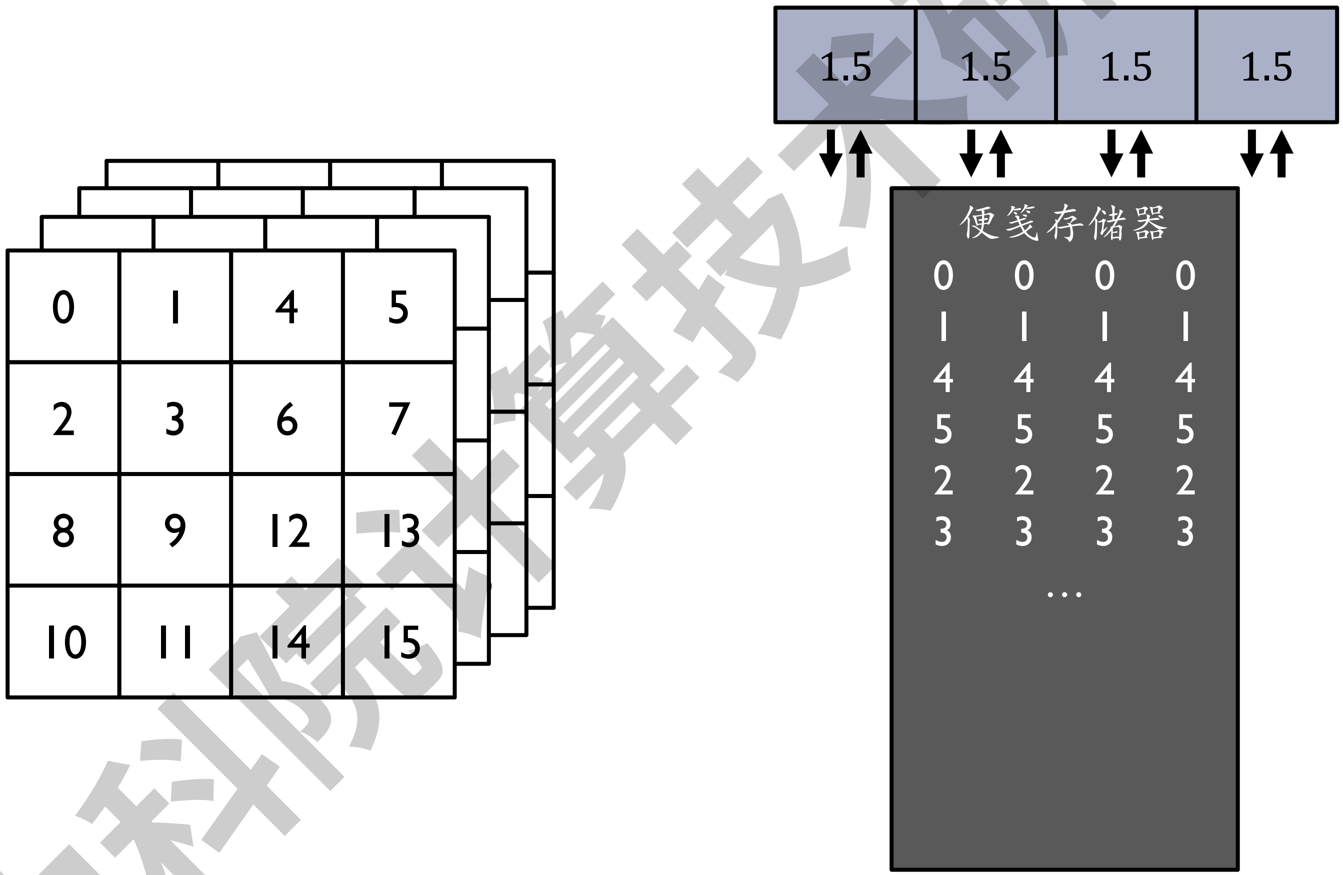
池化/均一化

- 如何完成池化?



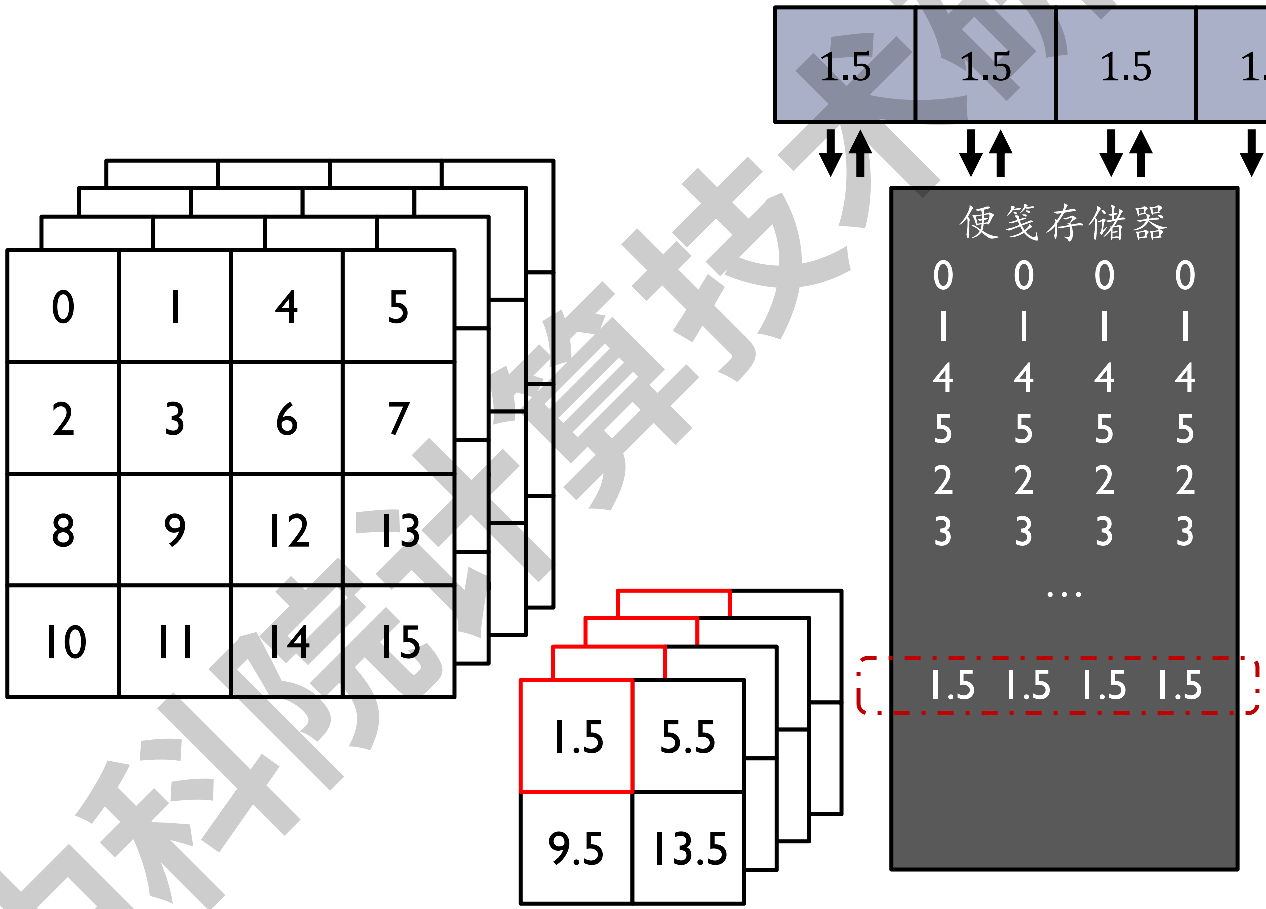
池化/均一化

- 如何完成池化?



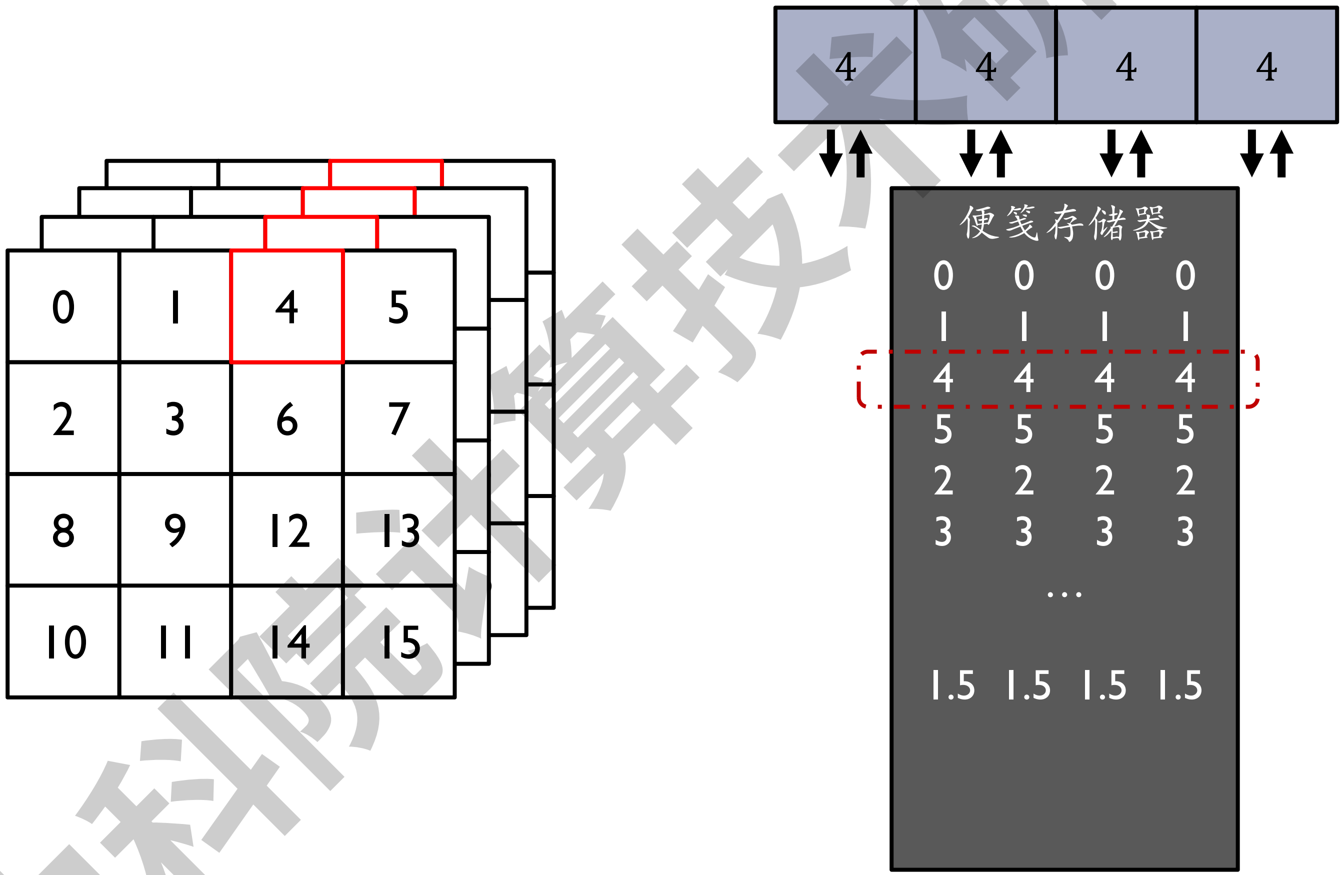
池化/均一化

- 如何完成池化?



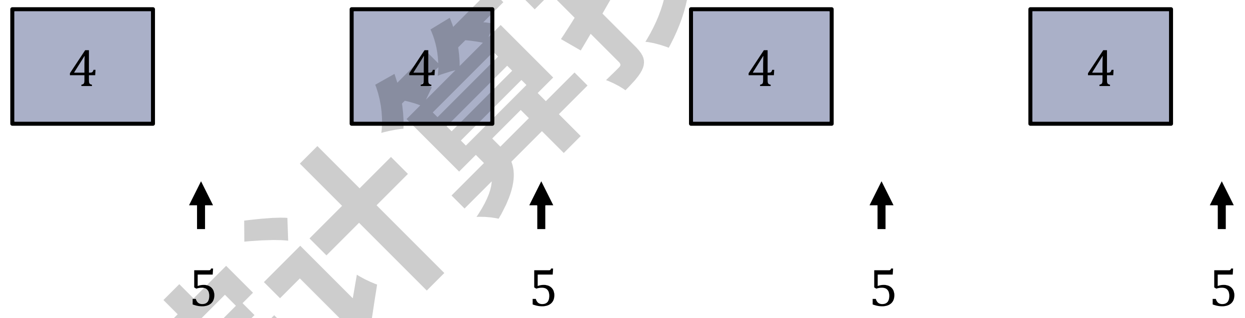
池化/均一化

- 如何完成池化?



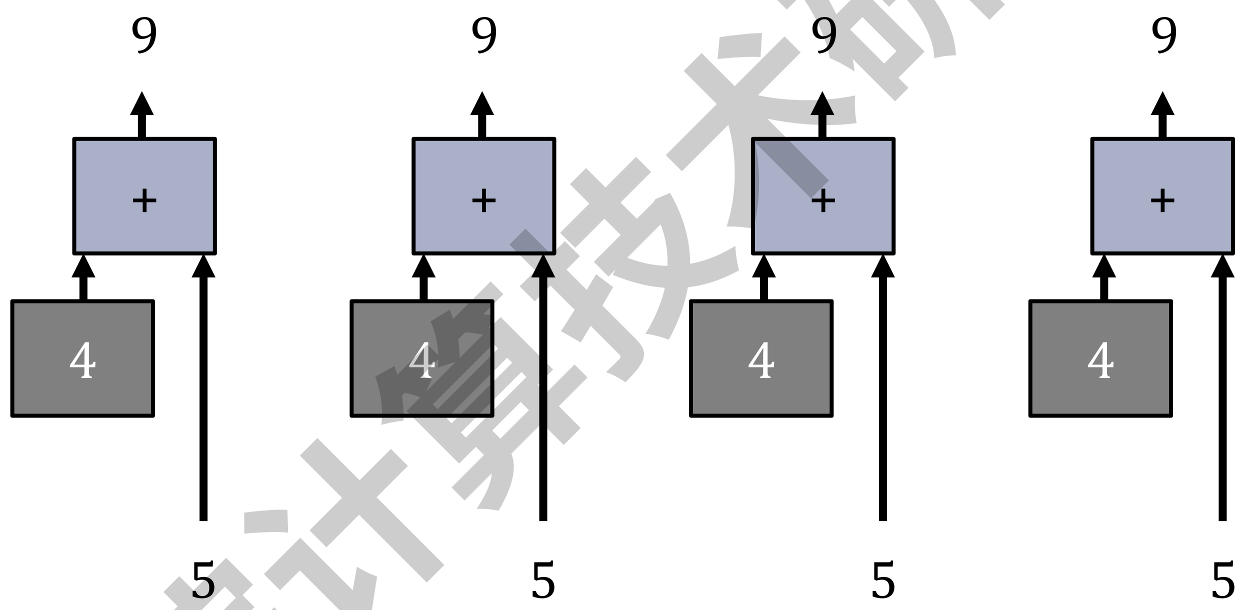
池化/均一化

- 运算单元结构



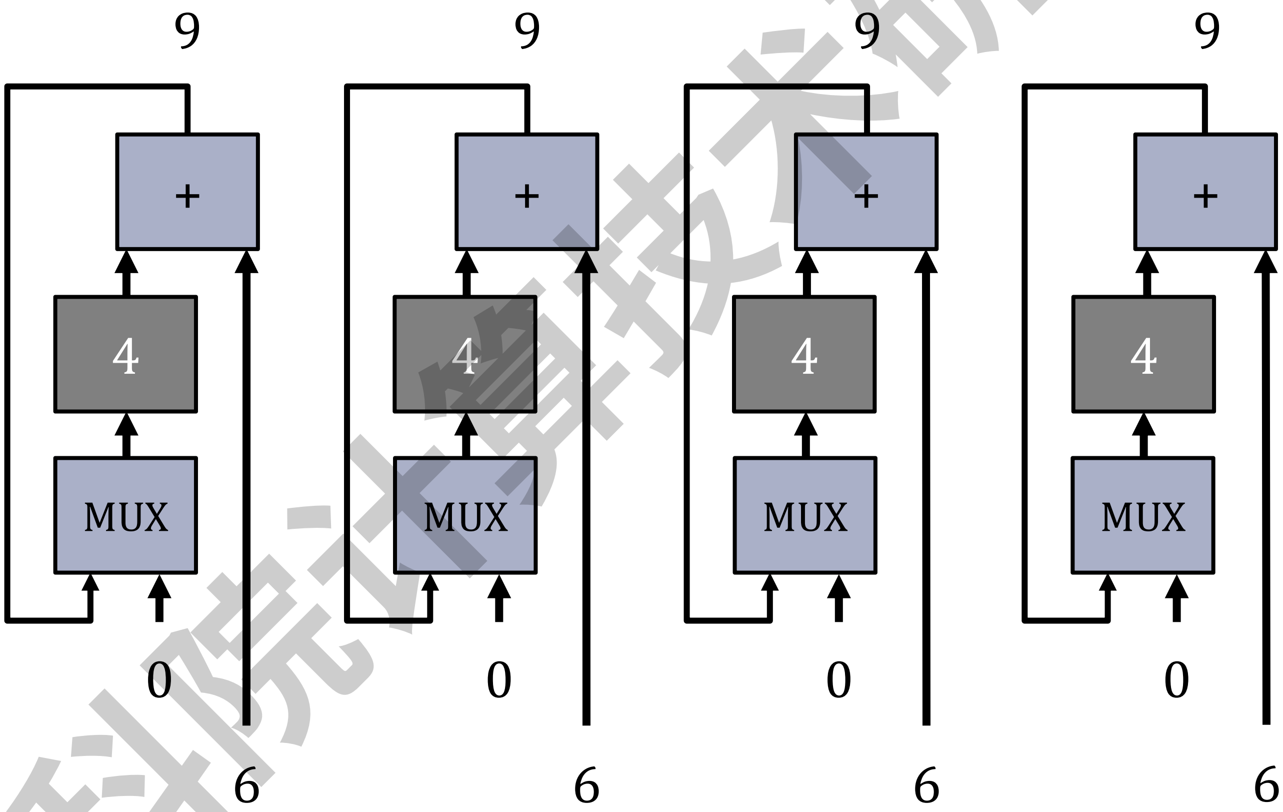
池化/均一化

- 运算单元结构



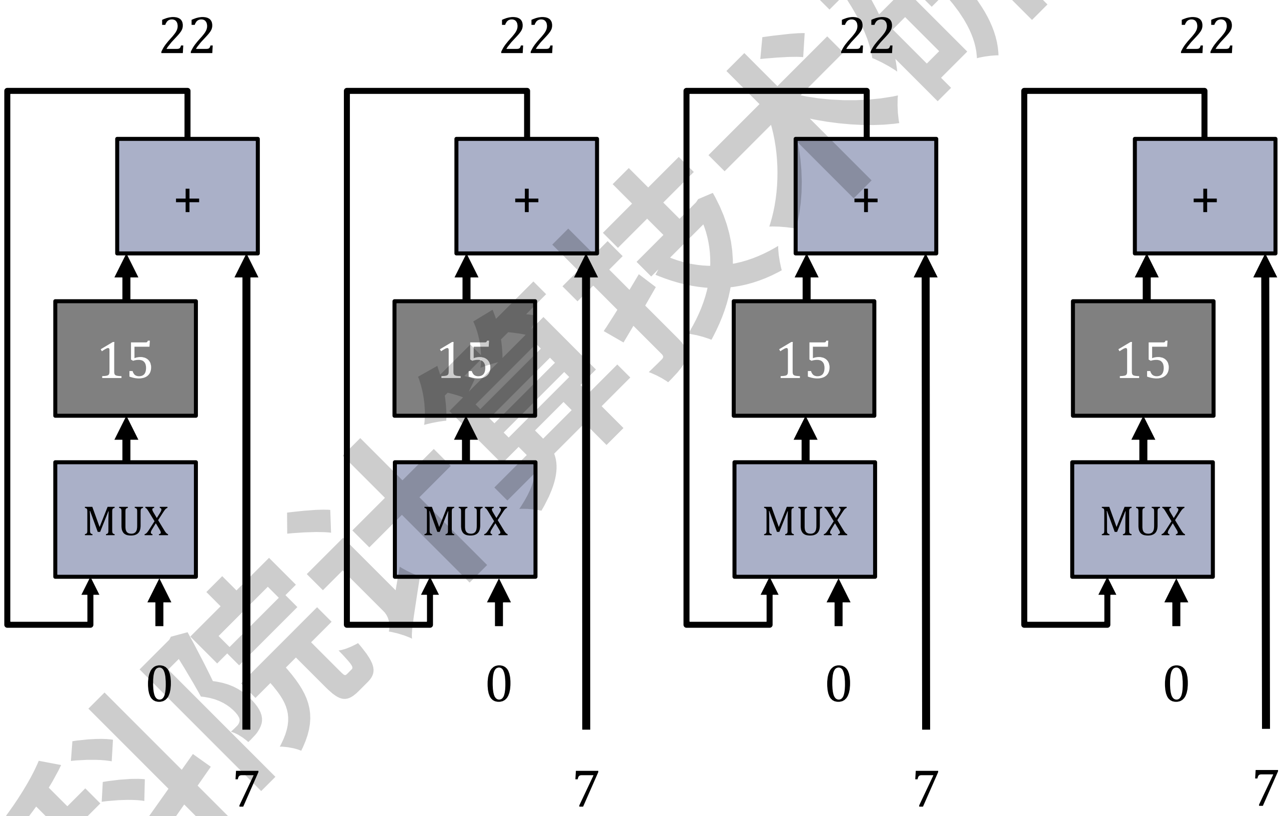
池化/均一化

- 运算单元结构



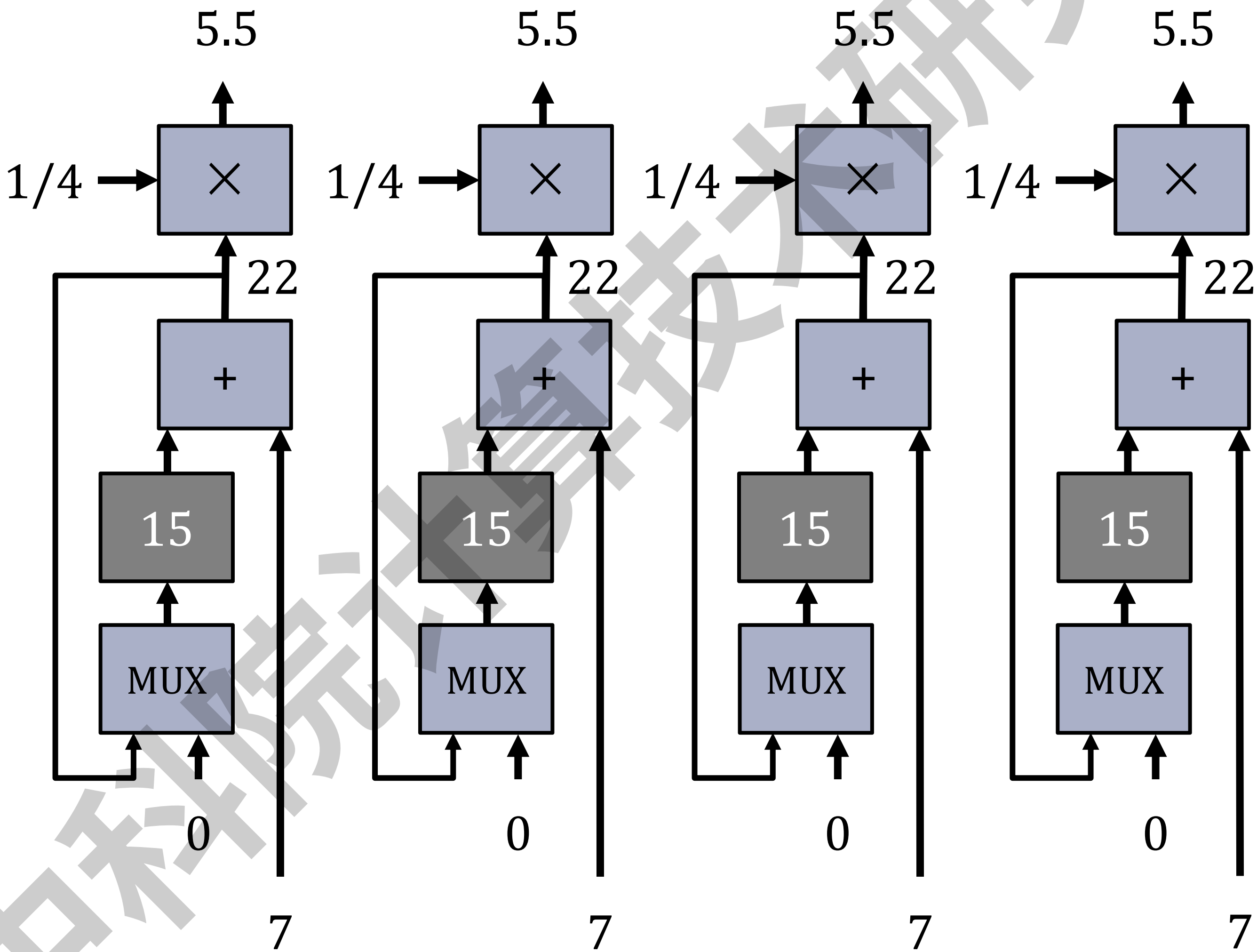
池化/均一化

- 运算单元结构



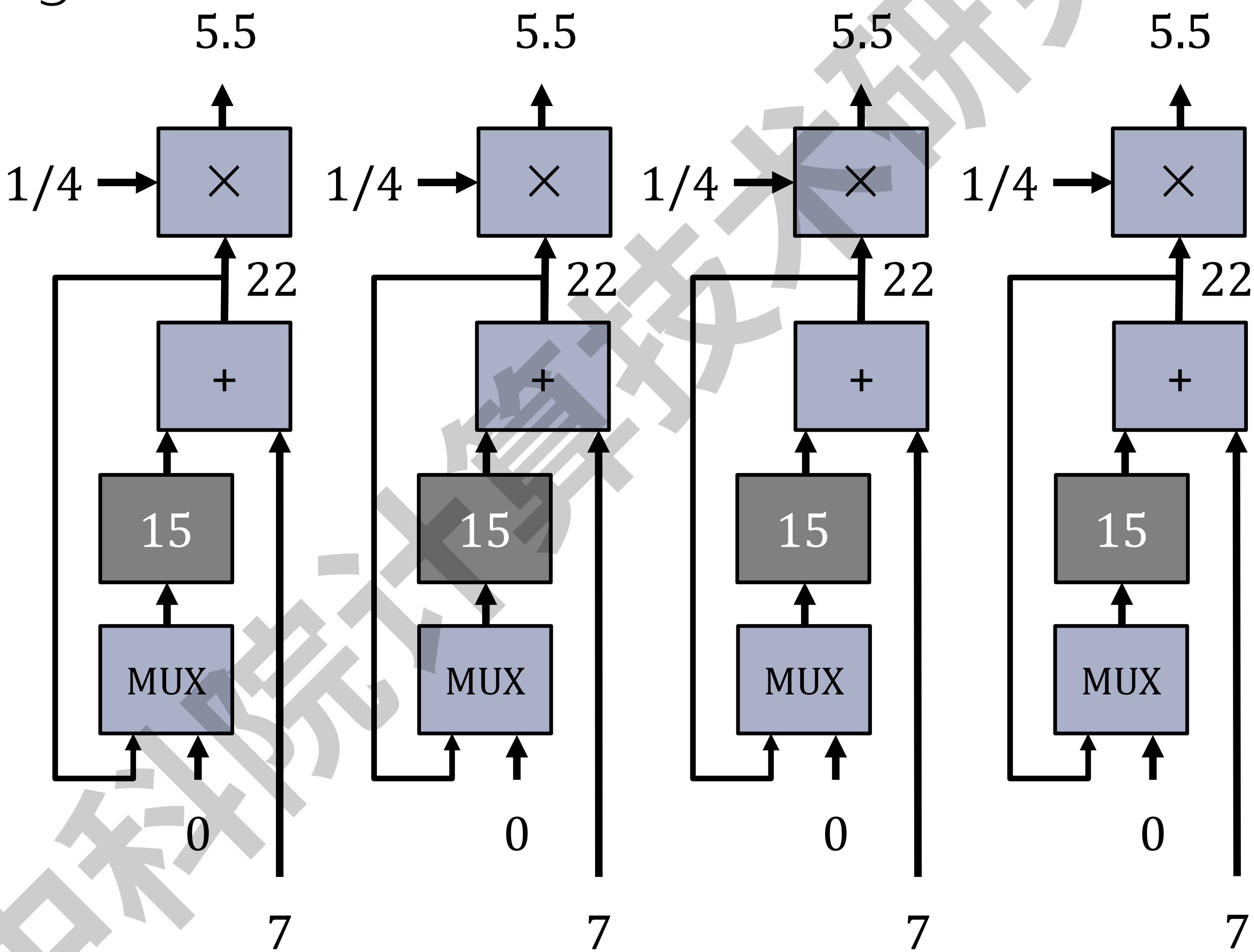
池化/均一化

- 运算单元结构



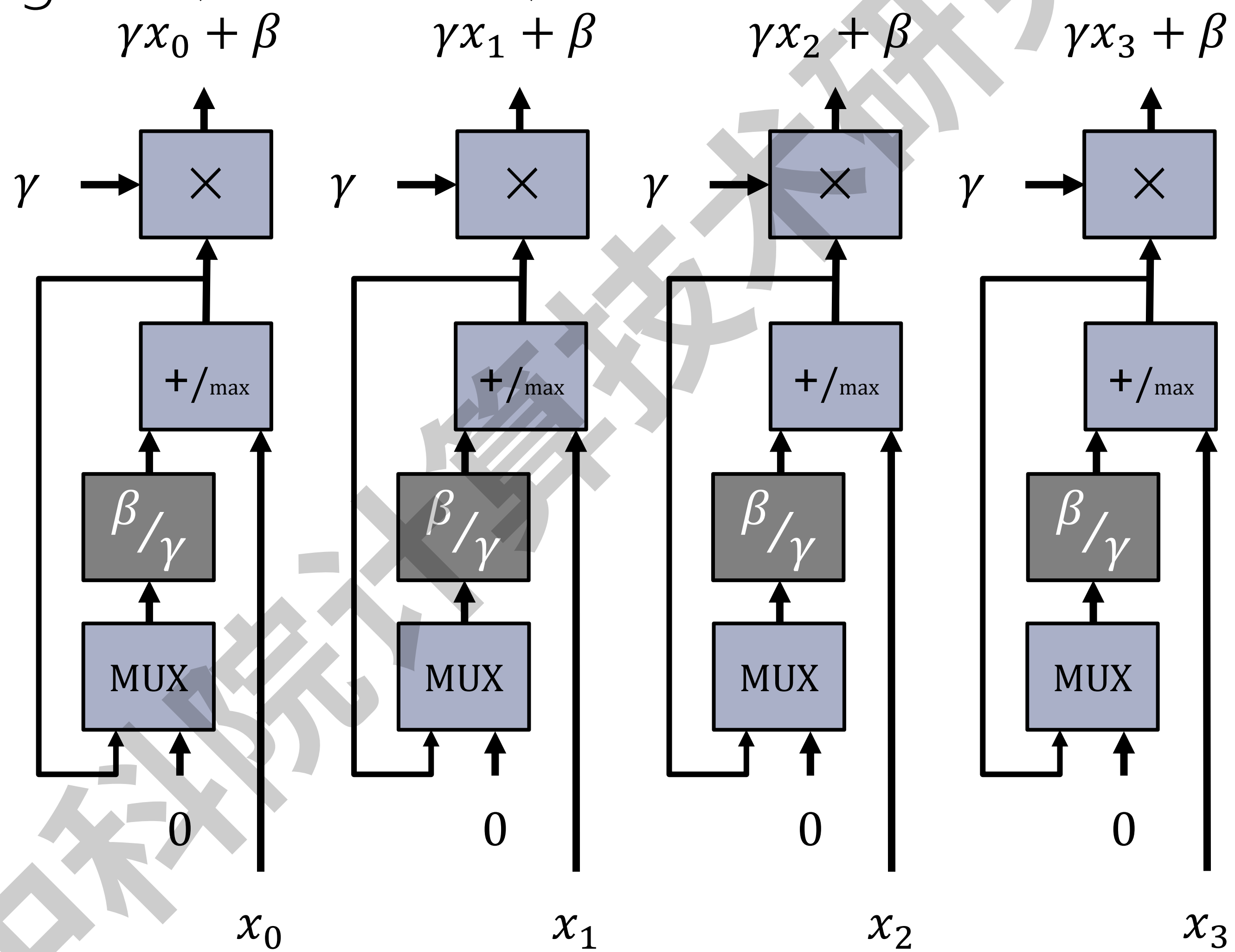
池化/均一化

- 支持AvgPool



池化/均一化

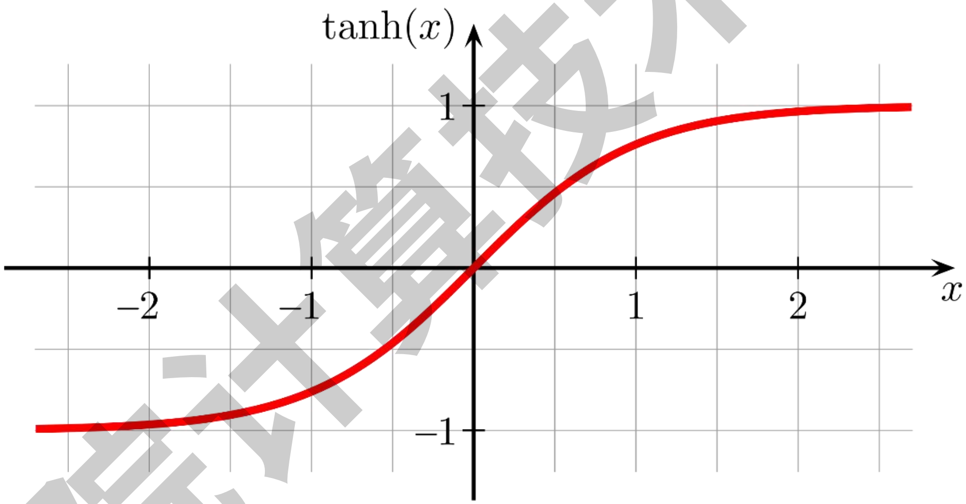
- 支持AvgPool、MaxPool 、BatchNorm



激活函数

- 如何计算双曲正切激活（tanh）？

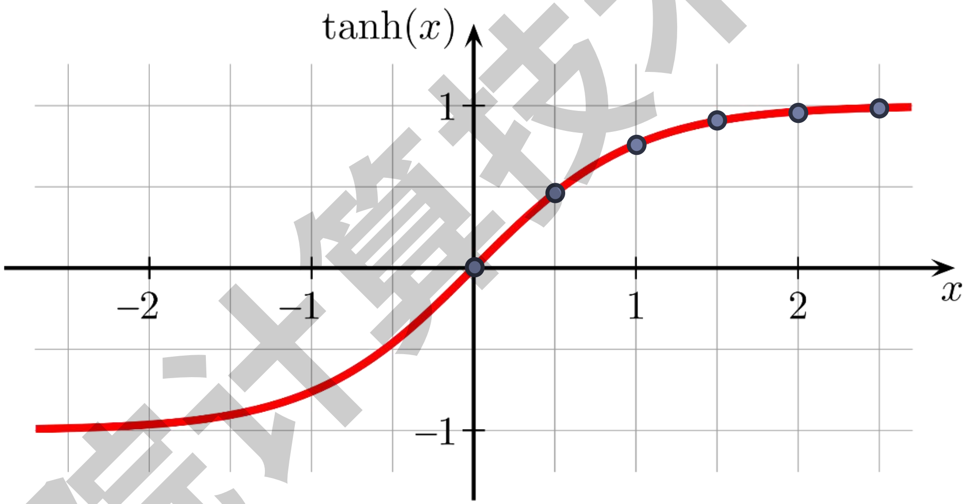
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



激活函数

- 如何计算双曲正切激活（tanh）？

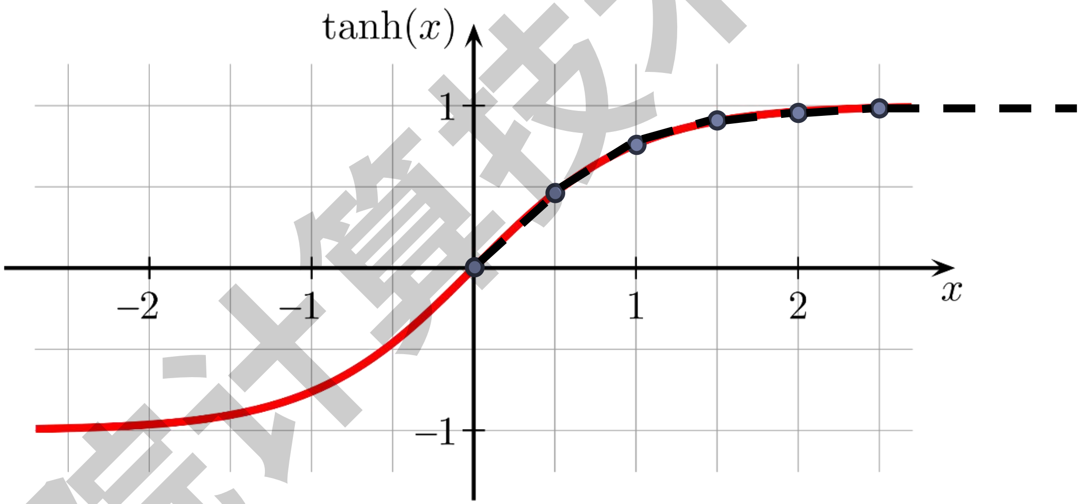
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



激活函数

- 如何计算双曲正切激活（tanh）？

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

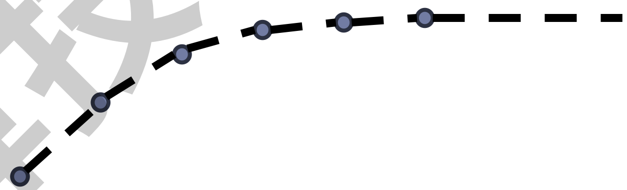


激活函数

- 如何计算双曲正切激活（tanh）？

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

最小二乘法确定分段线性系数A、B



函数 $\tanh^+(x)$ 定义为：

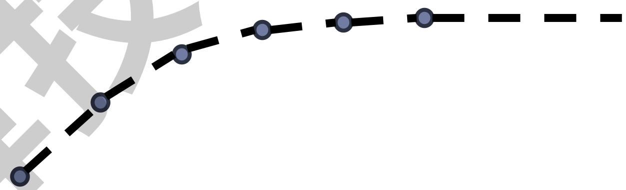
1. $i = x \div 0.5$ 取整，但不超过5
2. $a = A[i]$，$b = B[i]$
3. $\tanh^+(x) = ax + b$

激活函数

- 如何计算双曲正切激活（tanh）？

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

最小二乘法确定分段线性系数A、B



函数 $\tanh^{+}(x)$ 定义为：

1. $i = x \div 0.5$ 取整，但不超过5
2. $a = A[i]$，$b = B[i]$
3. $\tanh^{+}(x) = ax + b$

函数 $\tanh(x)$ 定义为：

$$\tanh(x) = \begin{cases} \tanh^{+}(x), & x \geq 0 \\ -\tanh^{+}(-x), & x < 0 \end{cases}$$

激活函数

- 如何计算双曲正切激活（tanh）？

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

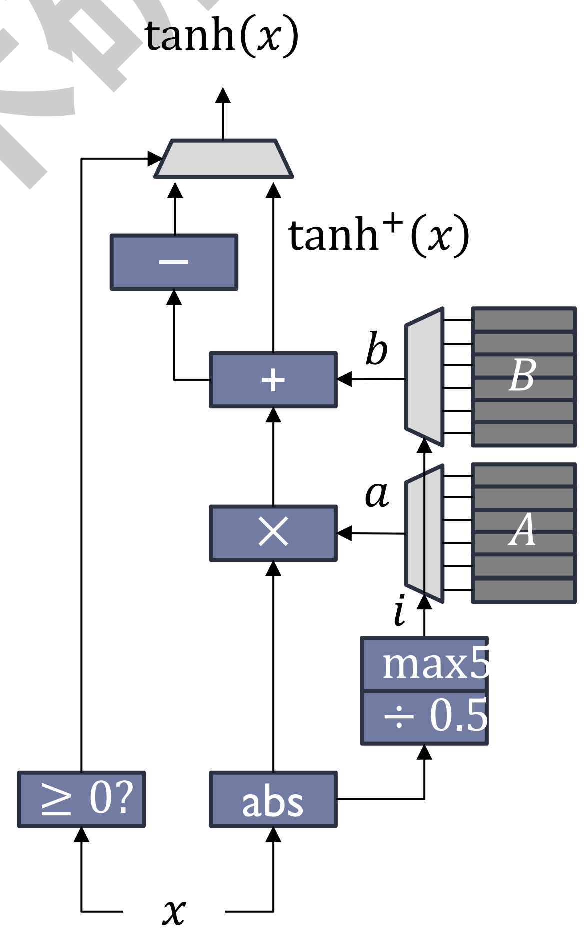
最小二乘法确定分段线性系数A、B

函数 $\tanh^+(x)$ 定义为：

1. $i = x \div 0.5$ 取整，但不超过5
2. $a = A[i]$，$b = B[i]$
3. $\tanh^+(x) = ax + b$

函数 $\tanh(x)$ 定义为：

$$\tanh(x) = \begin{cases} \tanh^+(x), & x \geq 0 \\ -\tanh^+(-x), & x < 0 \end{cases}$$



精确计算特殊函数

分段线性插值 在深度学习推理任务中，基本满足需求

- 如果需要精确计算，怎么办?

精确计算特殊函数

分段线性插值 在深度学习推理任务中，基本满足需求

- 如果需要精确计算，怎么办?
- 可以采用硬件或软件实现：
  - 各函数的快速数值算法
    - 例如：Beame-Cook-Hoover快速倒数算法
  - 数值方法
    - 例如：牛顿迭代法
  - 分段插值/快速估计+数值方法
    - 例如：0x5f3759df算法+牛顿迭代法

前缀计算

- **前缀计算** 是一类重要的并行计算模式

$$y_0 = x_0, \ y_1 = x_0 \odot x_1, \ y_2 = x_0 \odot x_1 \odot x_2, \ \ldots$$

- 例子：
  - 前缀求和 $1, 2, 3, 4, 5 \longrightarrow 1, 3, 6, 10, 15$
  - 横向求和 $1, 2, 3, 4, 5 \longrightarrow 15$
  - 横向求平均值 $1, 2, 3, 4, 5 \longrightarrow 3$
  - 横向求最大值 $1, 2, 3, 4, 5 \longrightarrow 5$
  - 非零计数 $0, 2, 0, 0, 5 \longrightarrow 2$
  - …

前缀计算

- 实现很简单。有高效的方式吗？

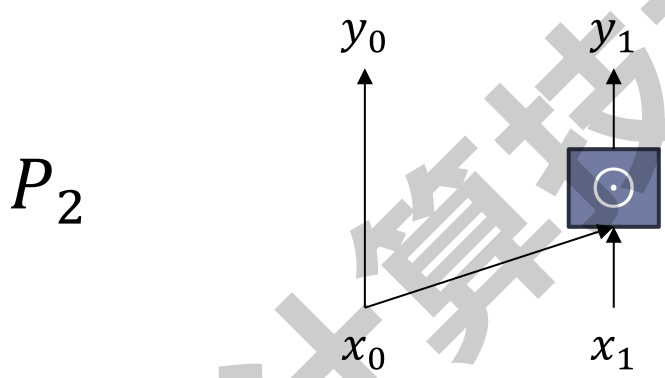
前缀计算

- 实现很简单。有高效的方式吗？

P_1

前缀计算

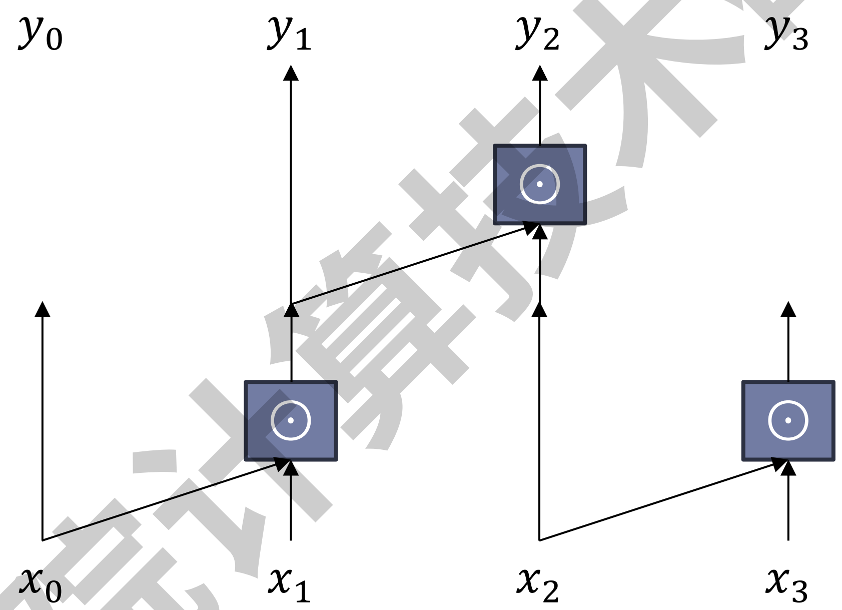
- 实现很简单。有高效的方式吗？



前缀计算

- 实现很简单。有高效的方式吗？

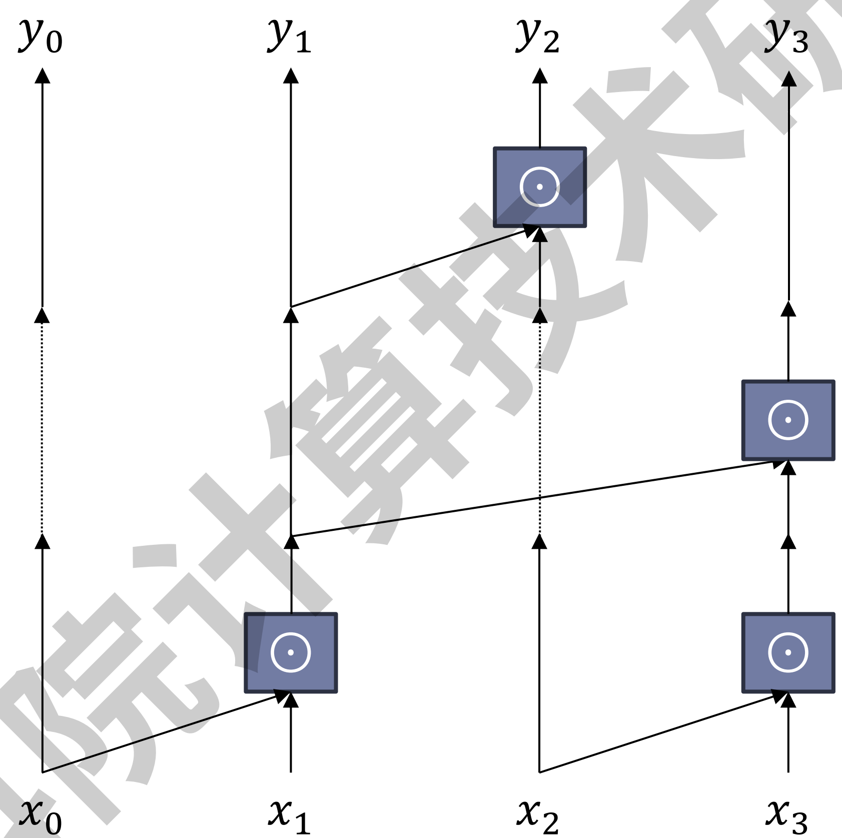
P_4



前缀计算

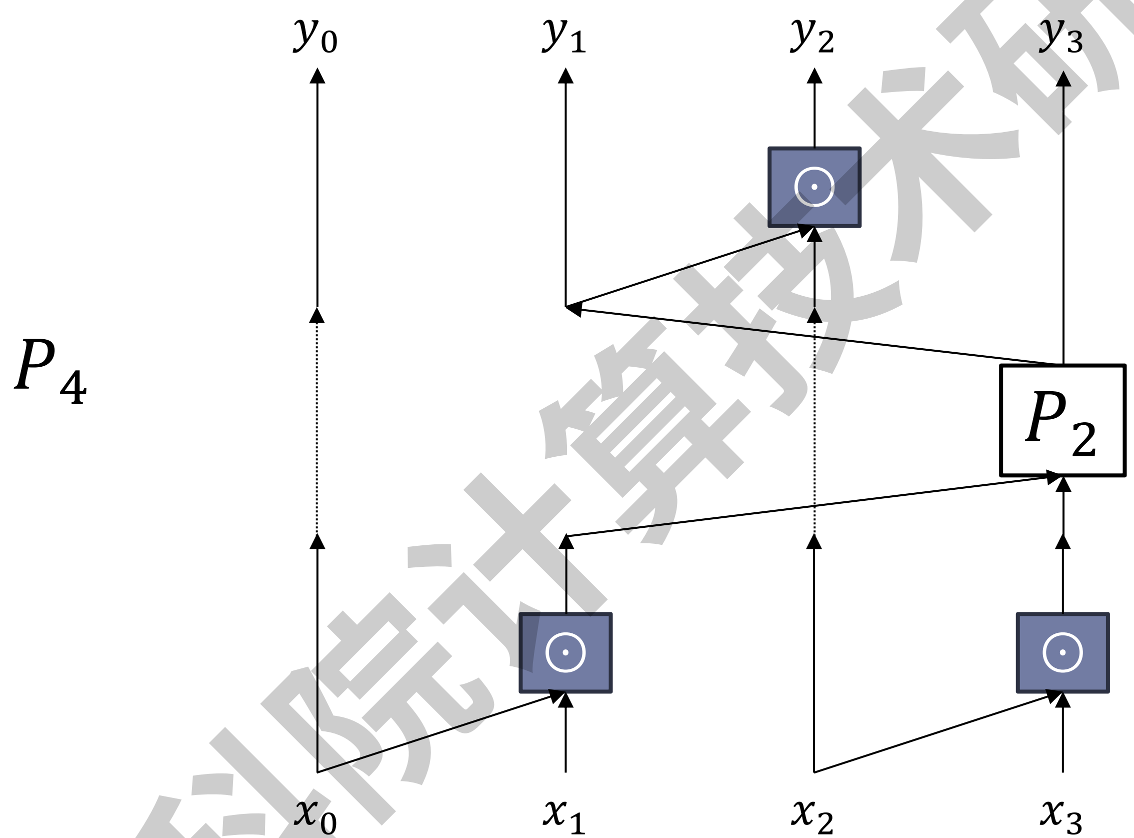
- 实现很简单。有高效的方式吗?

P_4



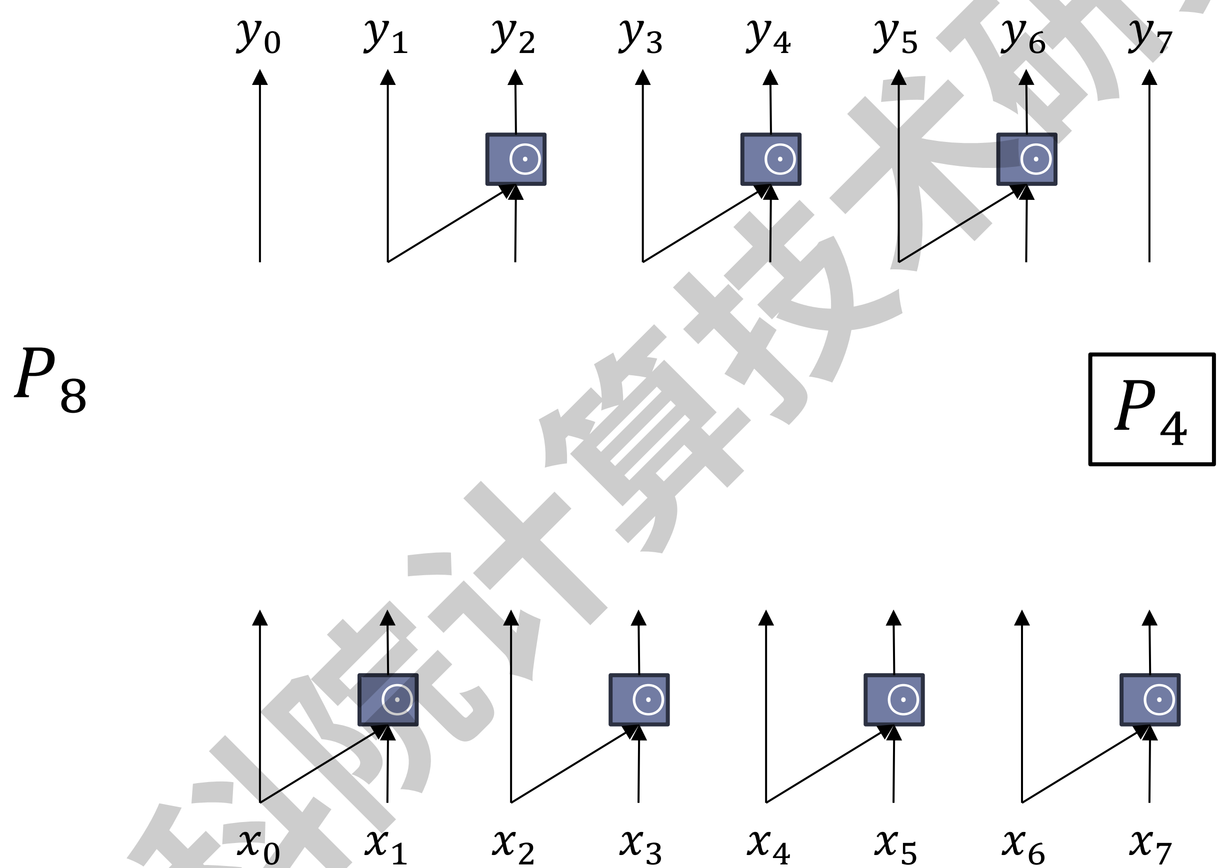
前缀计算

- 实现很简单。有高效的方式吗?



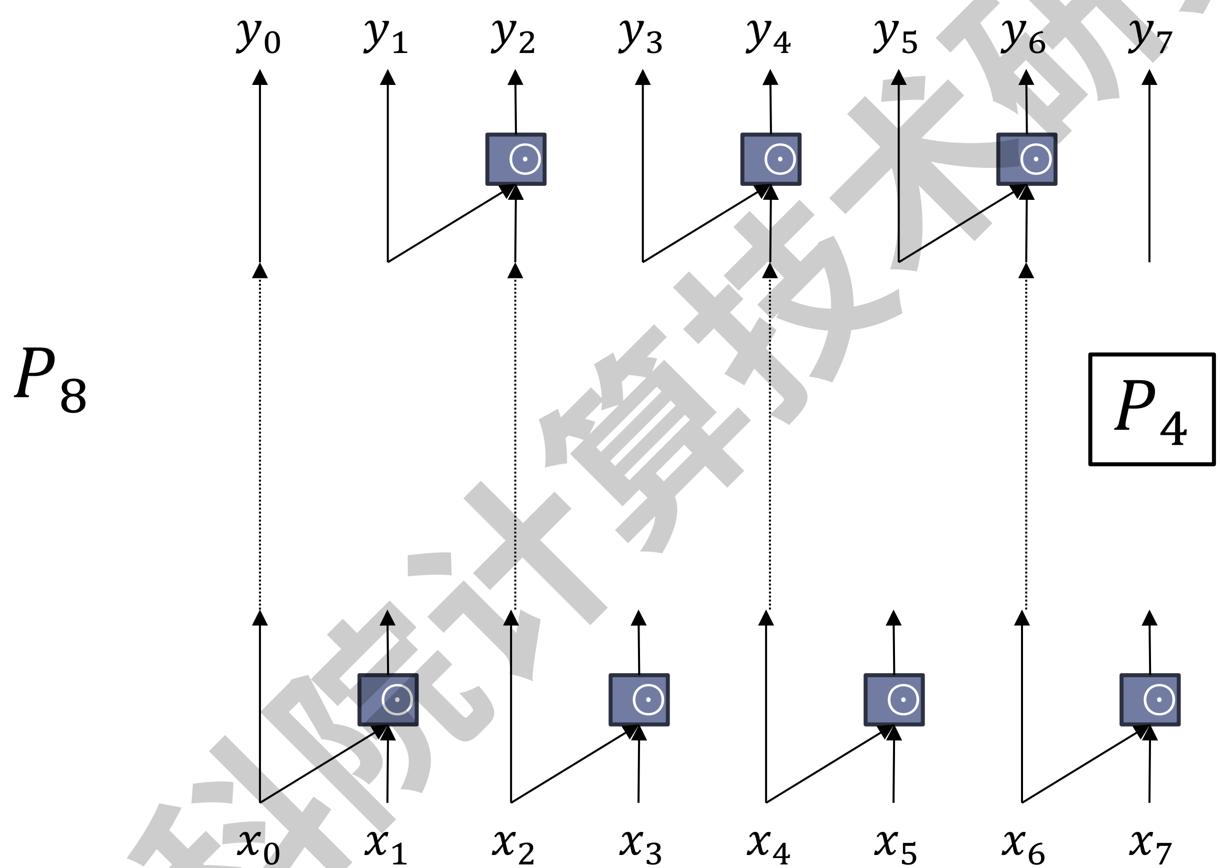
前缀计算

- 实现很简单。有高效的方式吗？



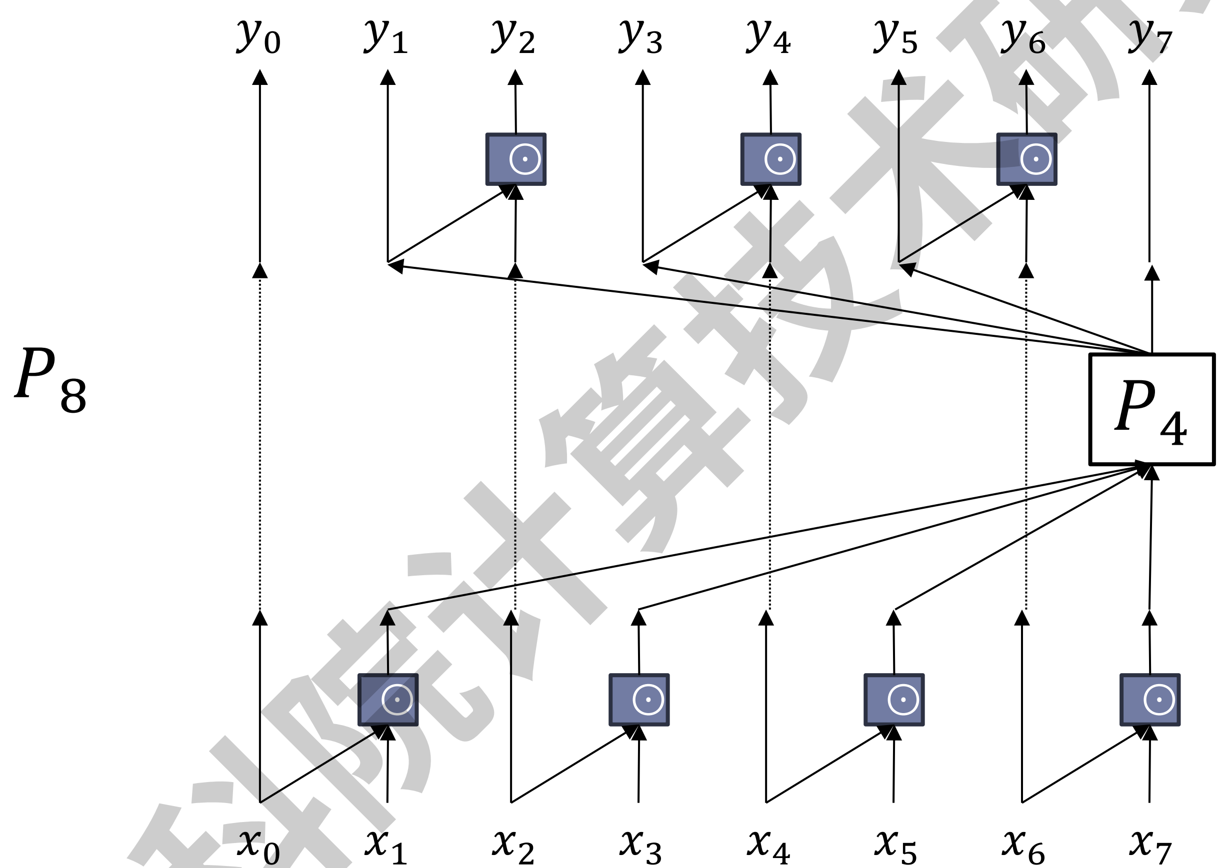
前缀计算

- 实现很简单。有高效的方式吗？



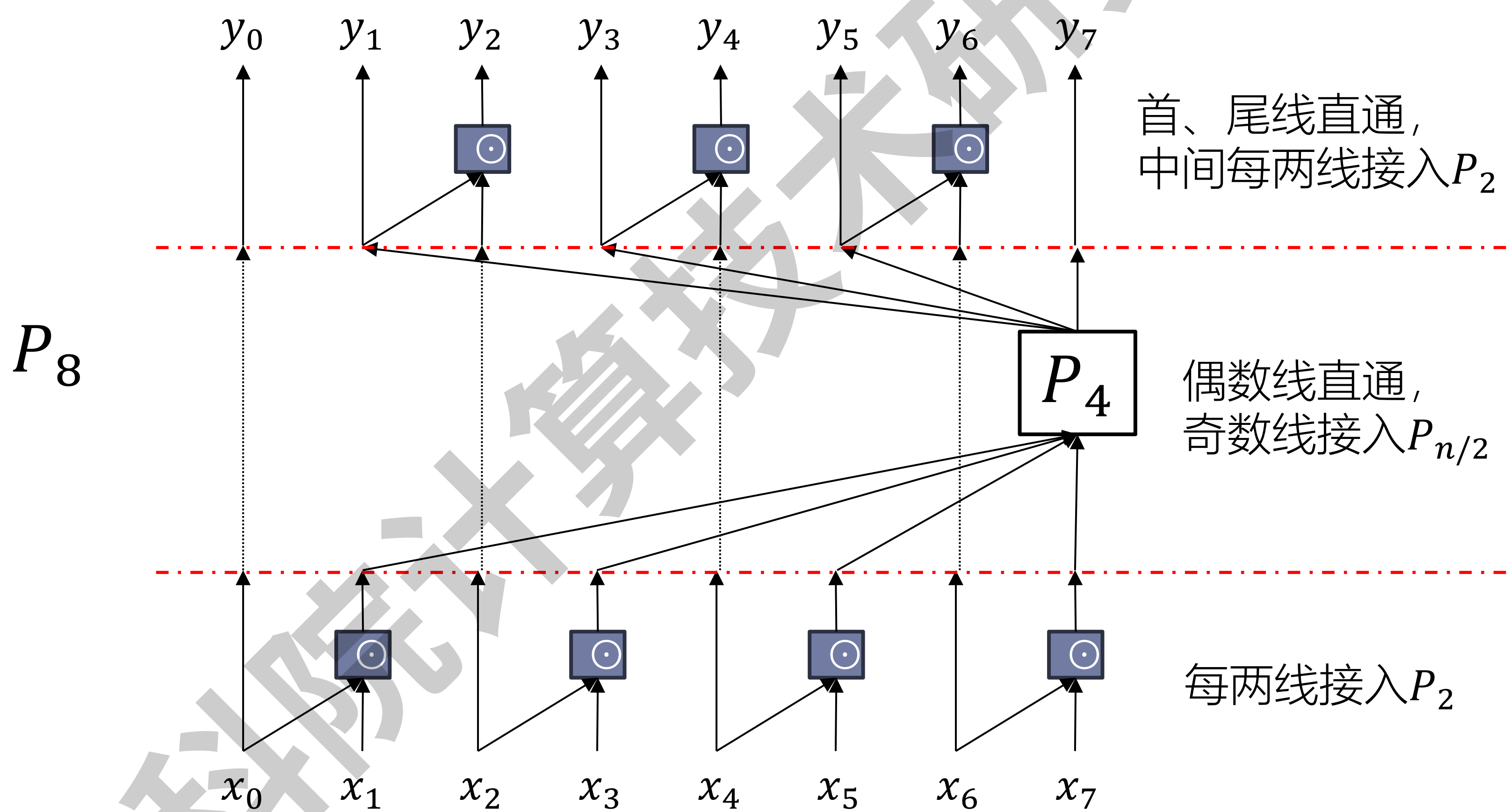
前缀计算

- 实现很简单。有高效的方式吗?



前缀计算

- 实现很简单。有高效的方式吗？

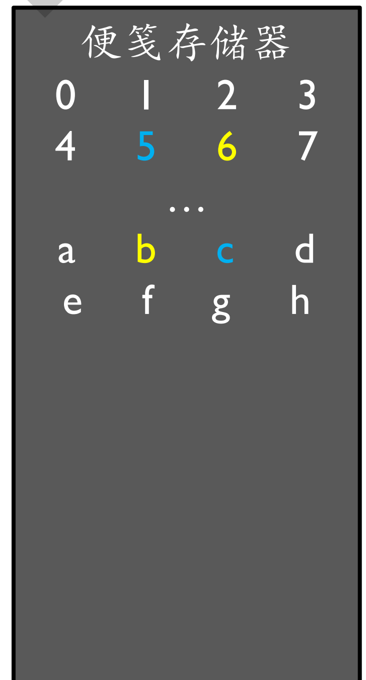


数据重排布

以向量为单位计算，很难使向量上不同位置的数据“相遇”

- 因为便笺访问是对齐的

例子：如何计算4a+5c+6b+7d?



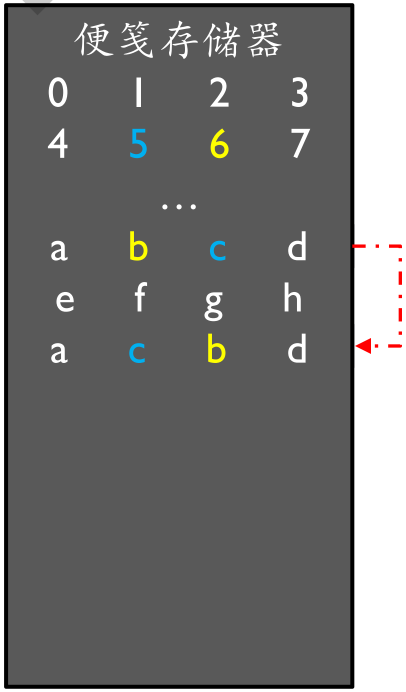
数据重排布

以向量为单位计算，很难使向量上不同位置的数据"相遇"

- 因为便笺访问是对齐的

例子：如何计算4a+5c+6b+7d?

- 先交换b和c的位置



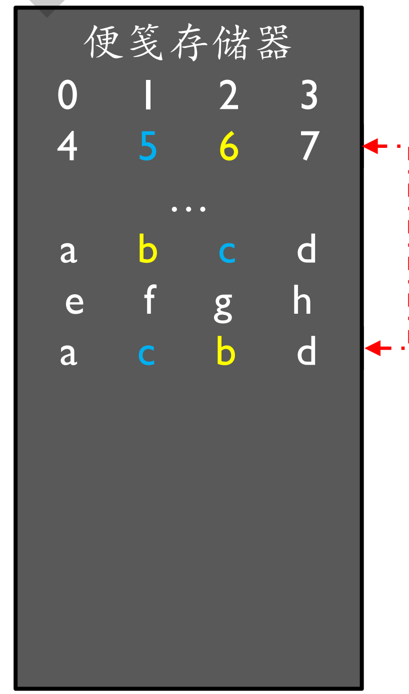
数据重排布

以向量为单位计算，很难使向量上不同位置的数据“相遇”

- 因为便笺访问是对齐的

例子：如何计算4a+5c+6b+7d?

- 先交换b和c的位置
- 再进行内积计算



数据重排布

以向量为单位计算，很难使向量上不同位置的数据“相遇”

- 因为便笺访问是对齐的

例子：如何计算4a+5c+6b+7d?

- 先交换b和c的位置

可以用标量指令编程完成交换

便笺存储器

0	1	2	3
4	5	6	7
	…		
a	b	c	d
e	f	g	h
a	c	b	d

数据重排布

以向量为单位计算，很难使向量上不同位置的数据“相遇”

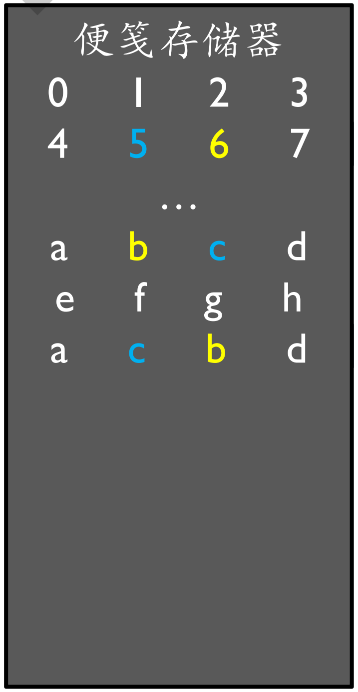
- 因为便笺访问是对齐的

例子：如何计算4a+5c+6b+7d?

- 先交换b和c的位置

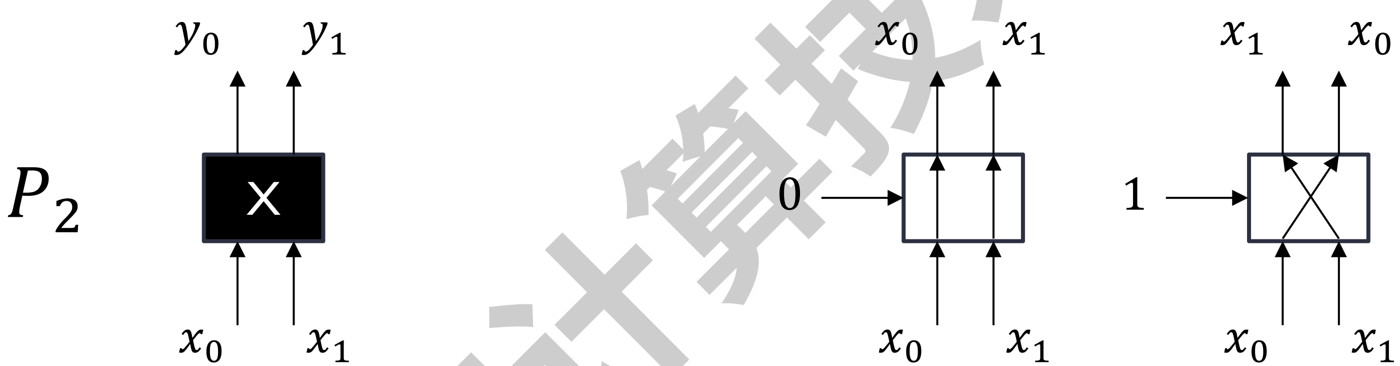
可以用标量指令编程完成交换

增加向量重排列功能，更高效！



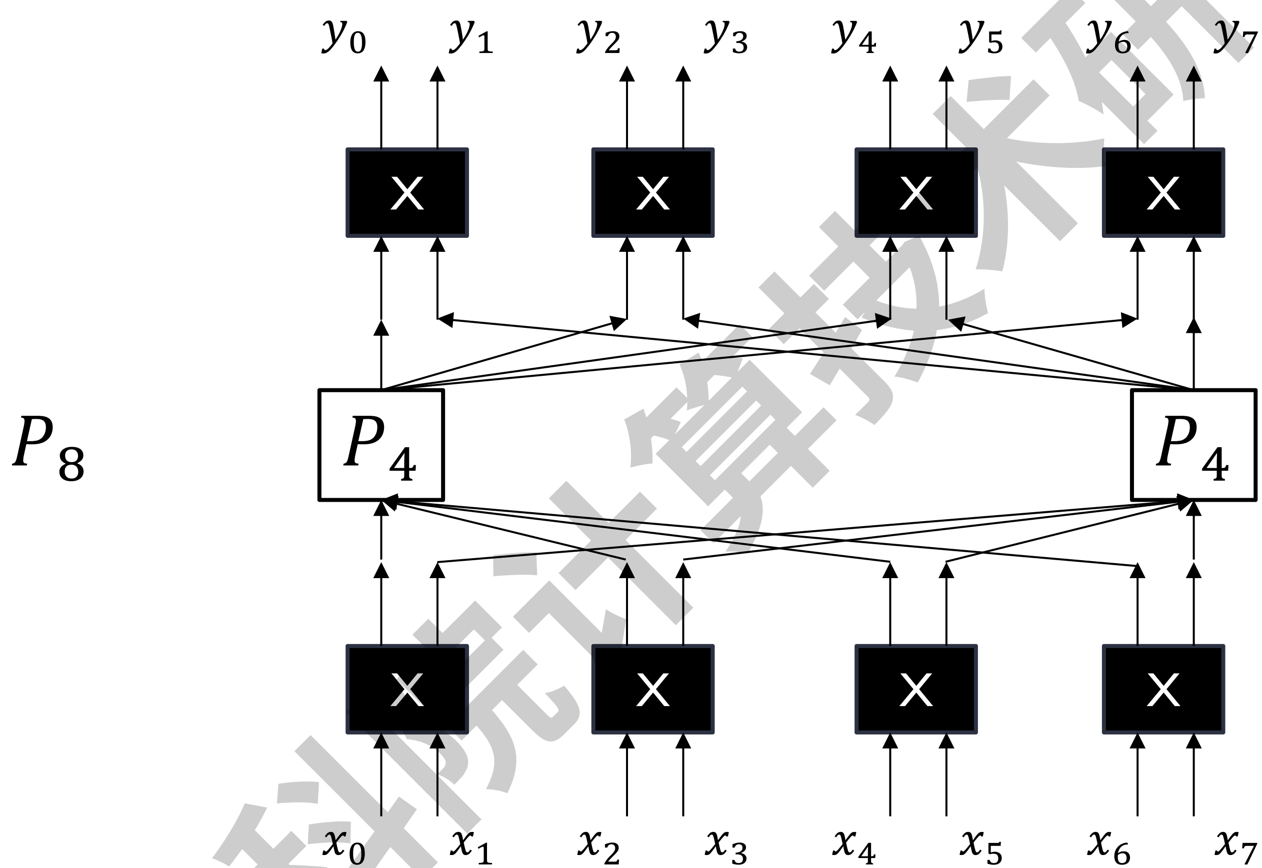
数据重排布

- 使用排列网络



数据重排布

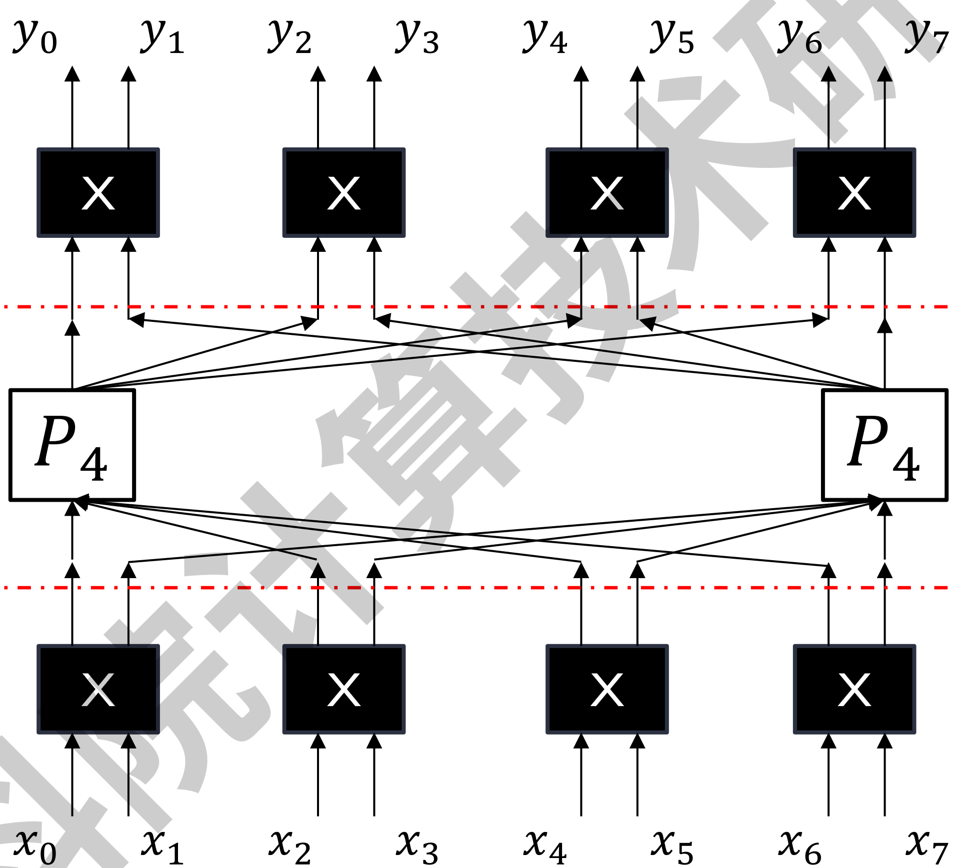
- 使用排列网络---变态式网络



数据重排布

- 使用排列网络

P_8



每两线接入开关

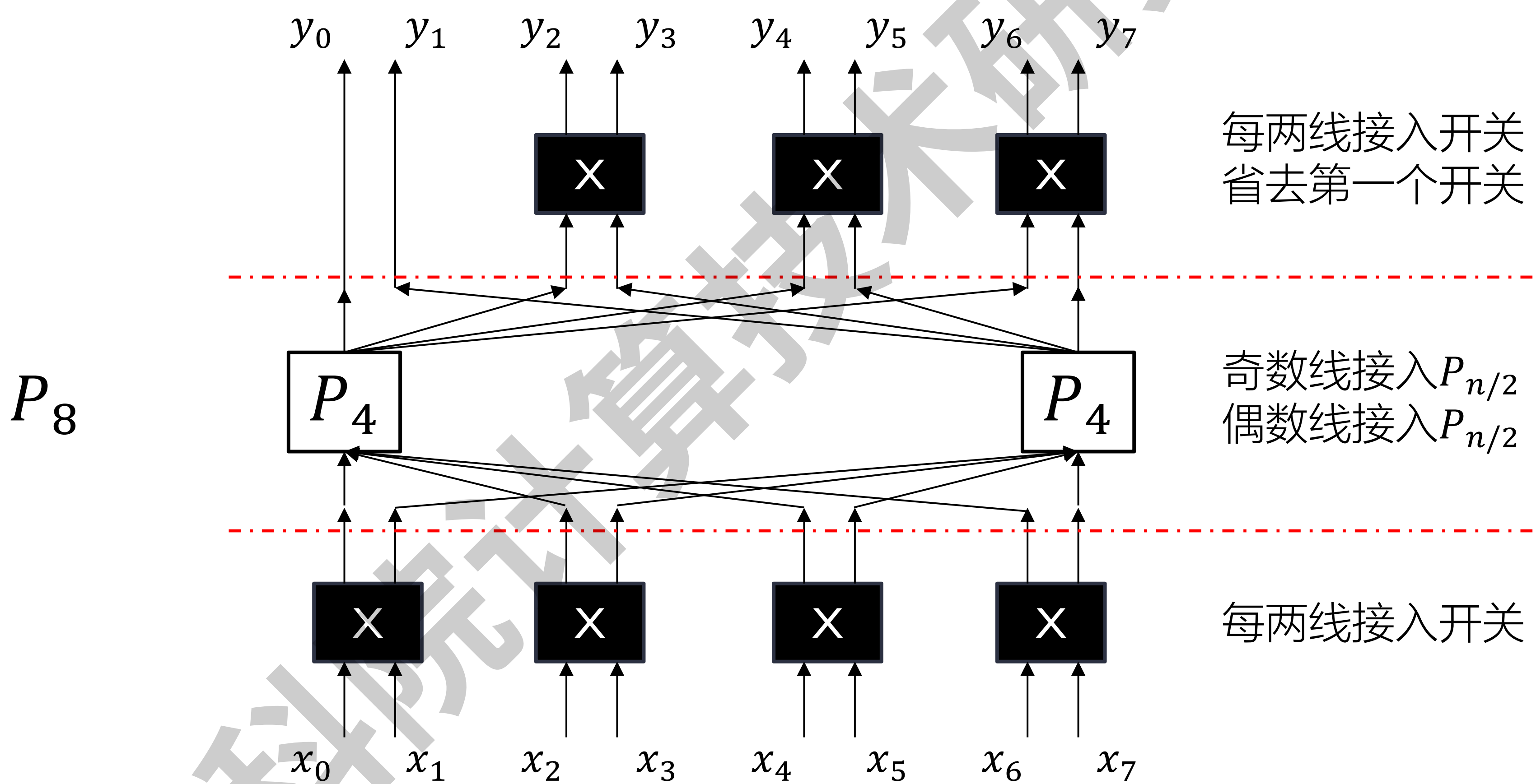
奇数线接入$P_{n/2}$
偶数线接入$P_{n/2}$

每两线接入开关

Beneš 网络

数据重排布

- 使用排列网络



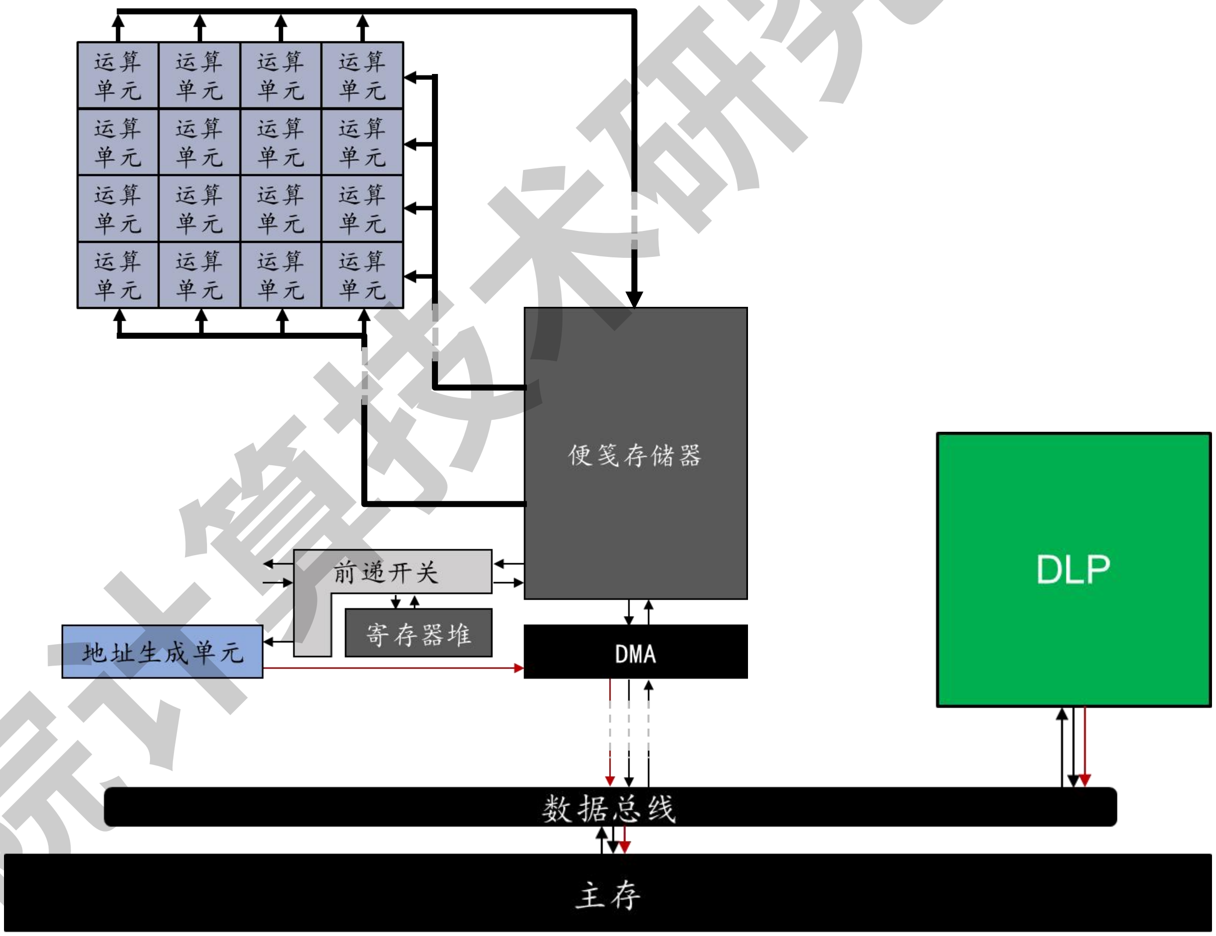
Waksman 网络

计算小结

- 矩阵运算单元
  - 可设计为矩阵乘向量单元、矩阵乘法单元、脉动阵列机等
  - 各有优势区间
- 向量/标量运算单元
  - 增设累加寄存器，可以实现池化
    - 一组硬件可以同时支持多种功能
  - 采用分段线性近似可以计算特殊函数
  - 增设前缀计算、重排布等功能，有助于拓展通用性

总体架构

- 计算
- 访存
- 通信



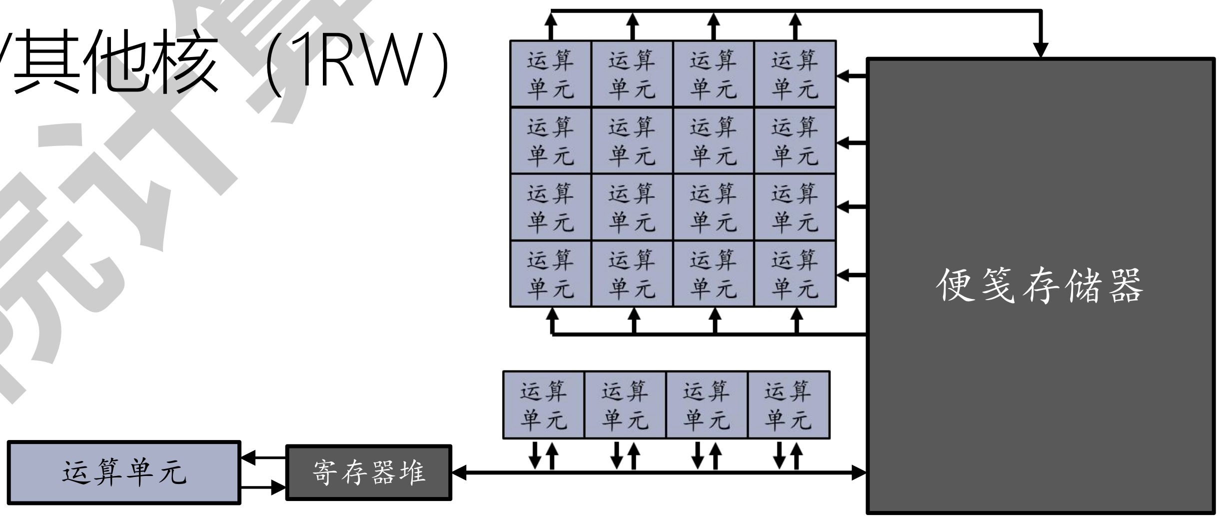
访存

- 访问便笺存储器
- 访问外部存储器
- 与计算的协同

便笺存储器

便笺存储器大多采用静态随机访问存储器（SRAM）实现

- 连接矩阵运算单元（2R,1W）
- 连接向量运算单元（2R,1W）
- 连接标量寄存器（1RW）
- 连接DMA/外存/其他核（1RW）
- …

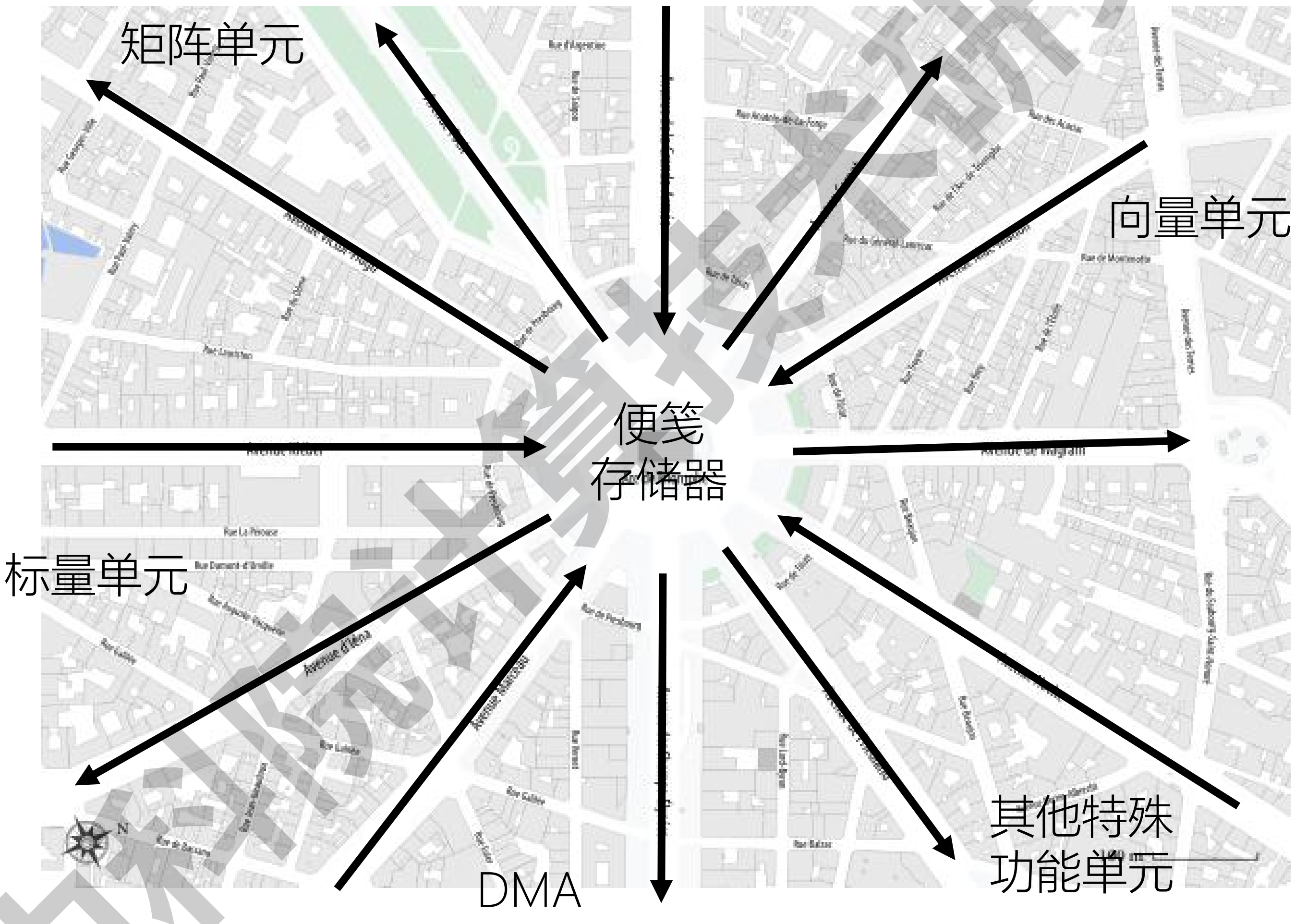


便笺存储器

- 便笺是DLP核当中的数据“枢纽”

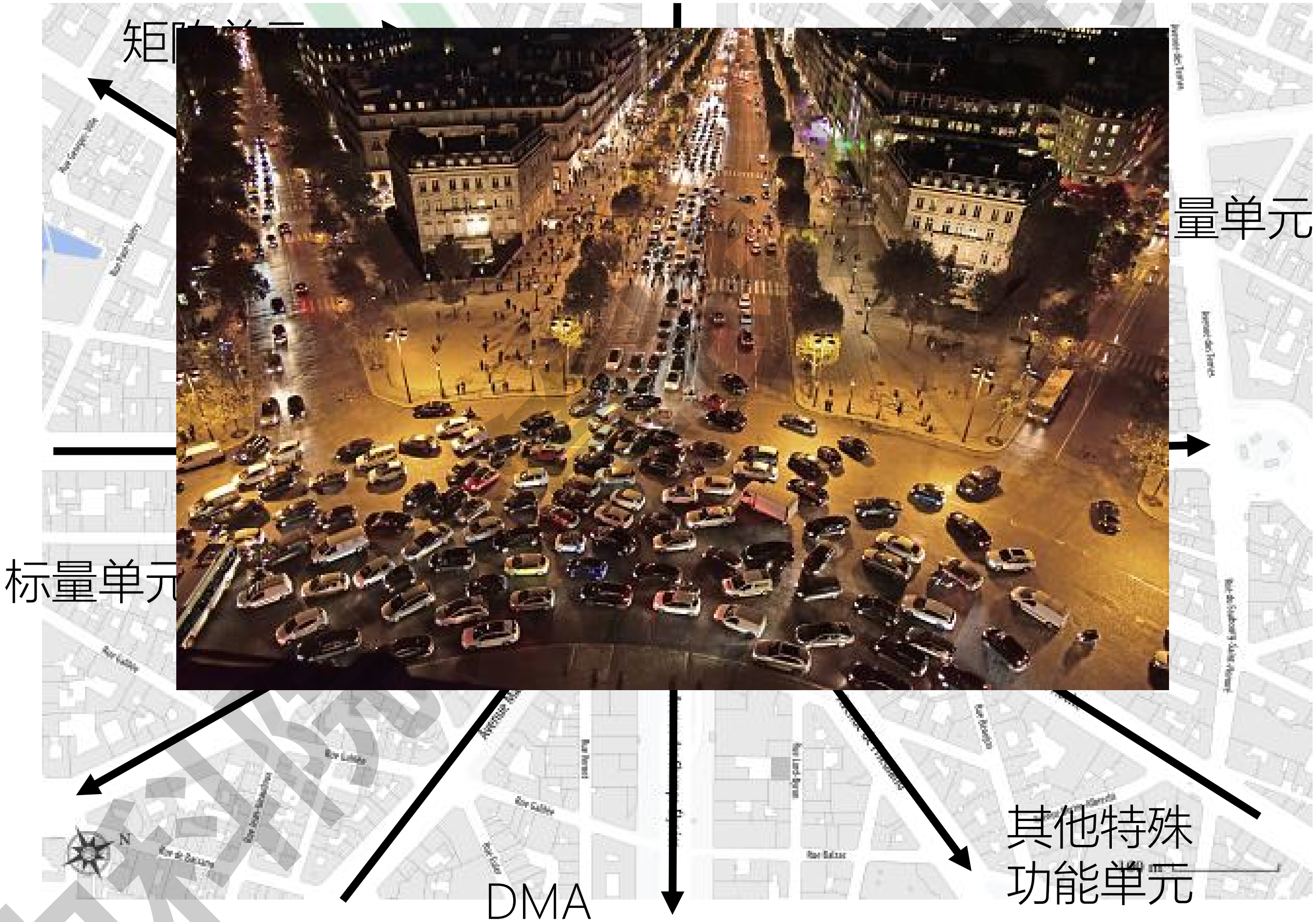
便笺存储器

- 便笺是DLP核当中的数据“枢纽”



便笺存储器

- 便笺是DLP核当中的数据“枢纽”



便笺存储器

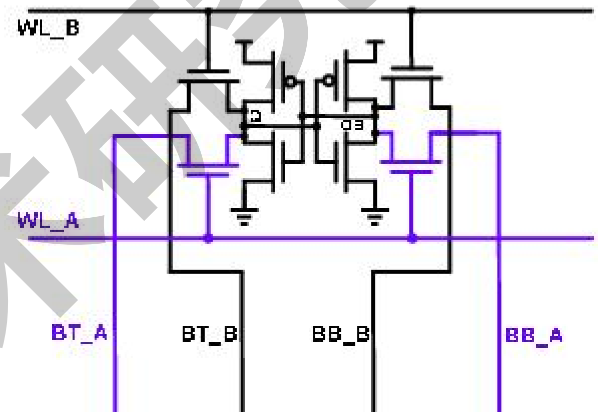
如何缓解拥堵?

- 拓宽"道路"
- 规划"车流"

便笺存储器

拓宽“道路”

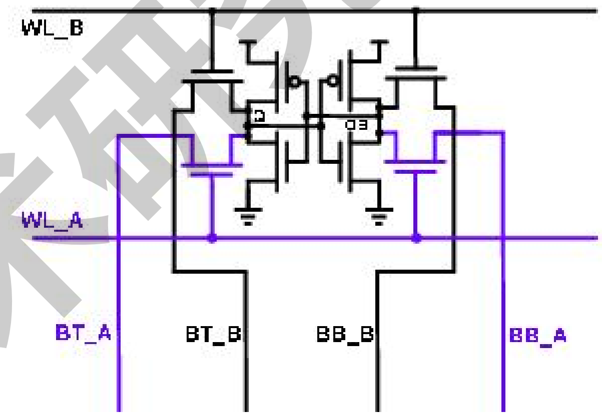
- 多端口SRAM
  - 增加一个端口，面积+50%~100%
  - 面积意味着成本、能耗、延时



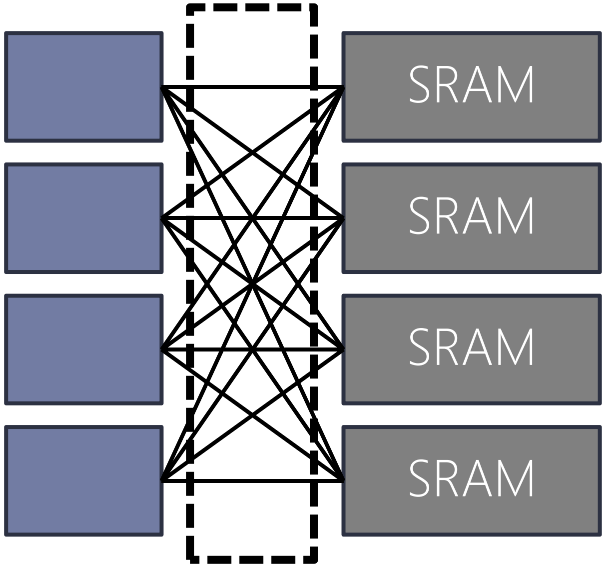
便笺存储器

拓宽“道路”

- 多端口SRAM
  - 增加一个端口，面积+50%~100%
  - 面积意味着成本、能耗、延时



- 分组SRAM
  - 开关阵列面积~O(分组数量^2)
  - 分组冲突（bank conflict)

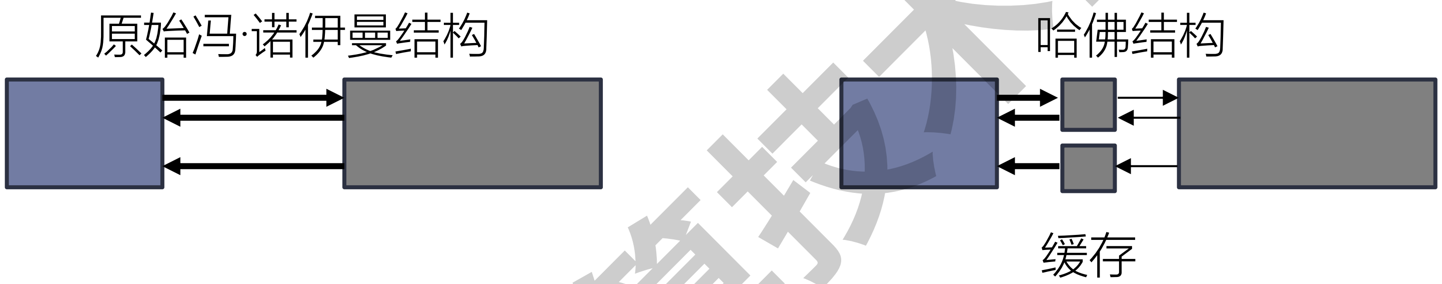


开关阵列

便笺存储器

规划“车流”

- 通用处理器中，采用哈佛结构解决取指-取数据冲突



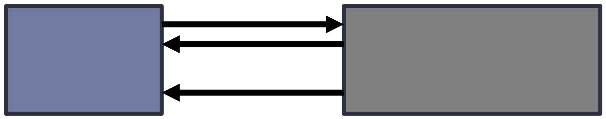
- 深度学习处理器中的哈佛结构?

便笺存储器

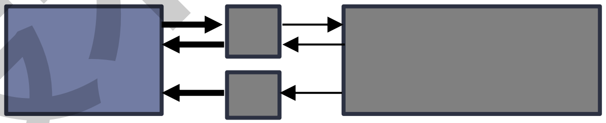
规划“车流”

- 通用处理器中，采用哈佛结构解决取指-取数据冲突

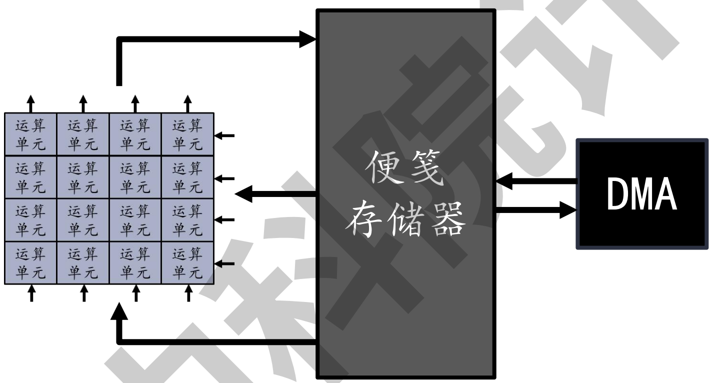
原始冯·诺伊曼结构



哈佛结构



- 深度学习处理器中的哈佛结构？

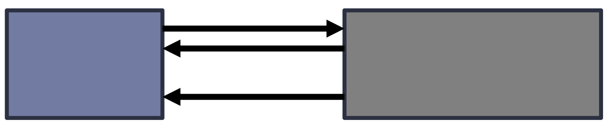


便笺存储器

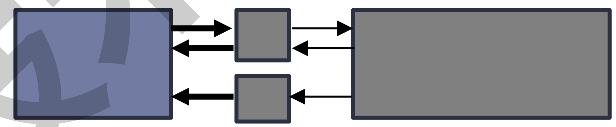
规划“车流”

- 通用处理器中，采用哈佛结构解决取指-取数据冲突

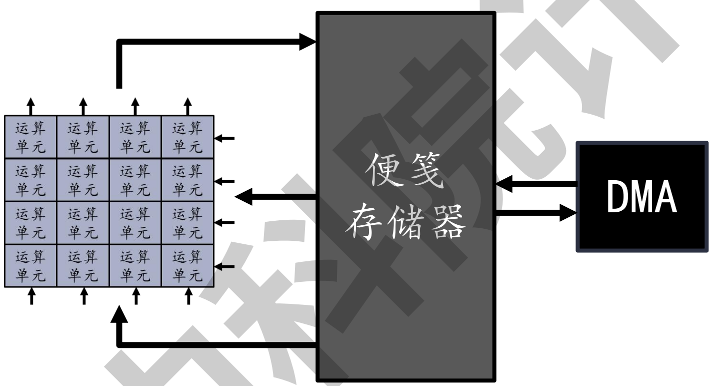
原始冯·诺伊曼结构

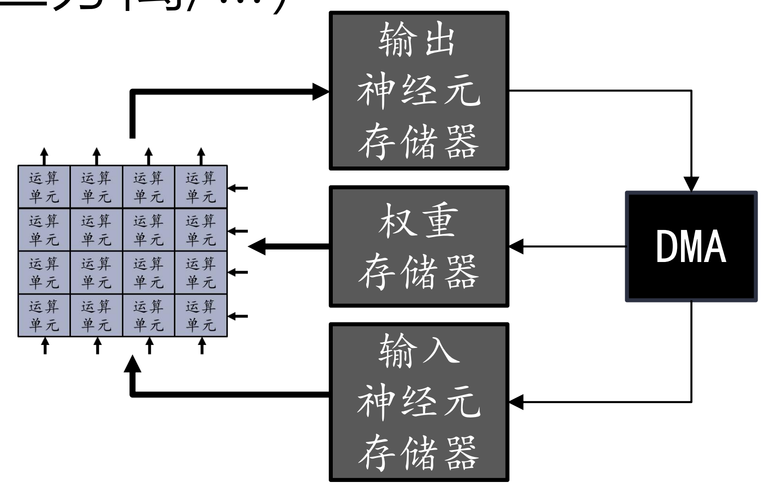


哈佛结构



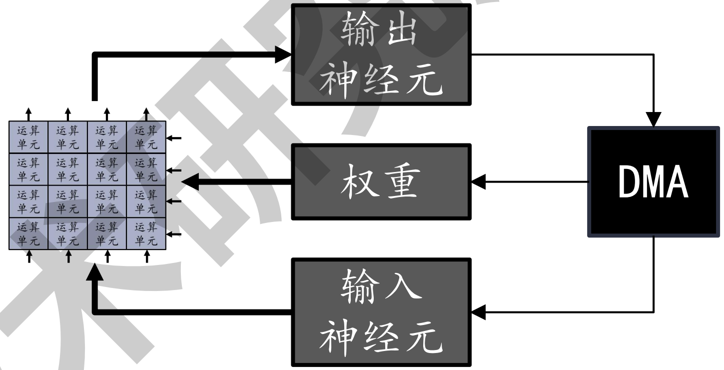
- 分离式便笺存储器（二分离/三分离/…）





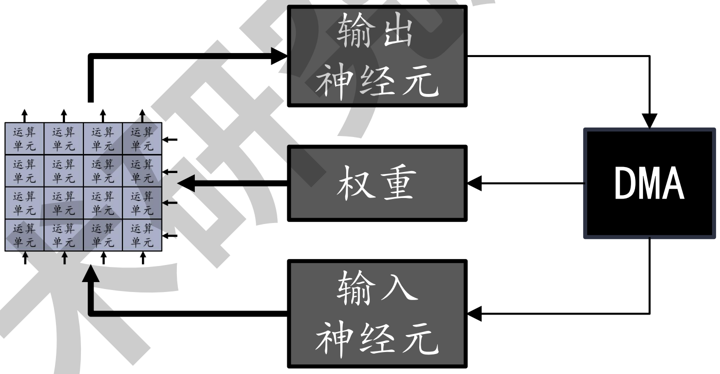
分离式便笺存储器

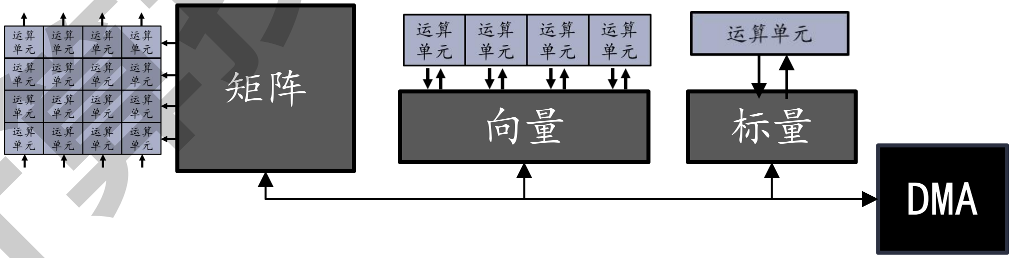
- 按数据划分
  - 神经元/权值
  - 输入神经元/输出神经元/权值



分离式便笺存储器

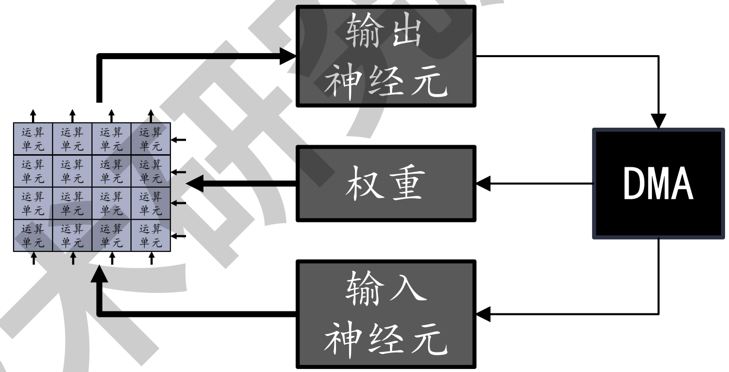
- 按数据划分
  - 神经元/权值
  - 输入神经元/输出神经元/权值
- 按功能单元划分
  - 向量/标量
  - 矩阵/向量/标量



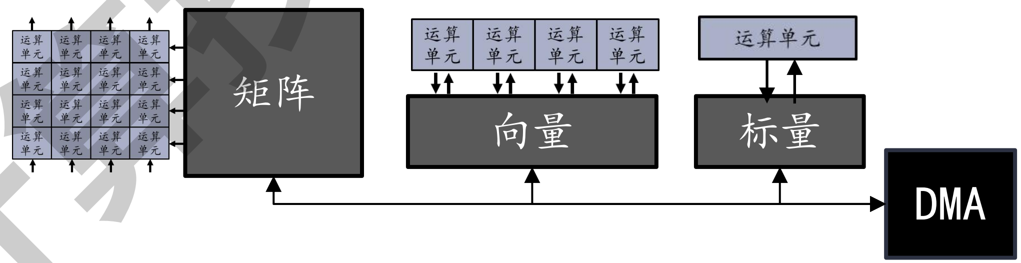


分离式便笺存储器

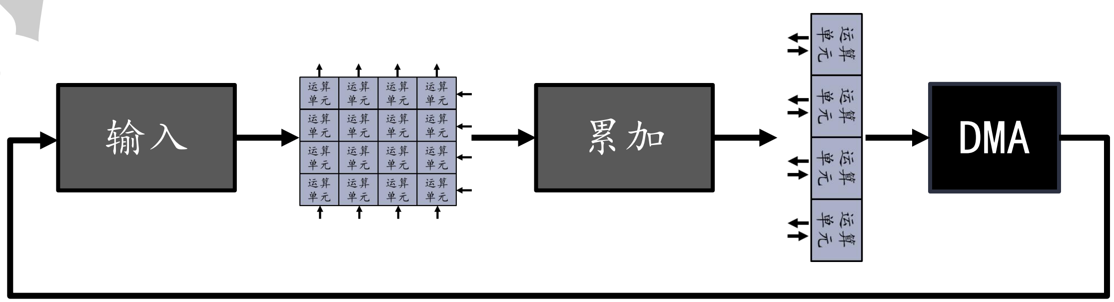
- 按数据划分
  - 神经元/权值
  - 输入神经元/输出神经元/权值



- 按功能单元划分
  - 向量/标量
  - 矩阵/向量/标量



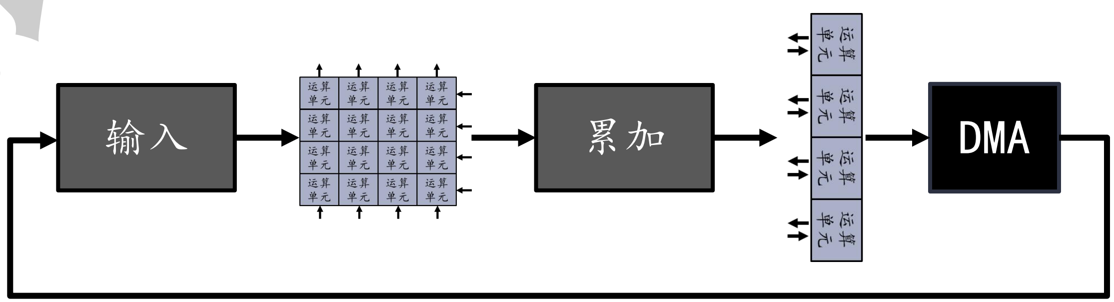
- 按处理阶段划分
  - 输入数据/累加器



分离式便笺存储器

对数据进行分流

- 提高了处理效率
- 对使用方式进行了约束（损失通用性）



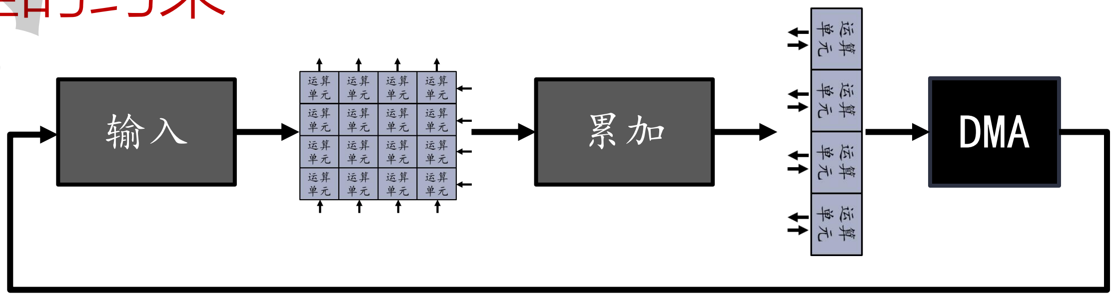
分离式便笺存储器

对数据进行分流

- 提高了处理效率
- 对使用方式进行了约束（损失通用性）

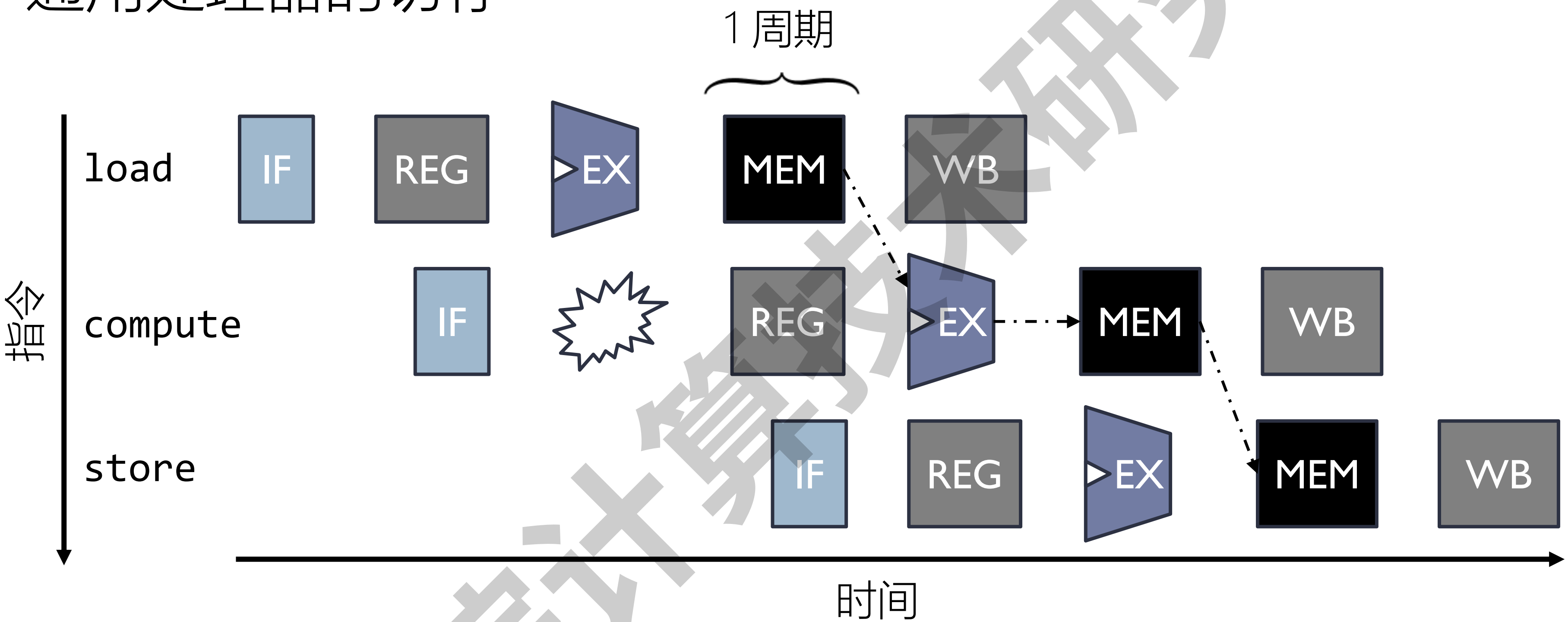
体系结构设计人员的职责：

- 寻找一组高效、合理的约束



外部存储器访问

通用处理器的访存

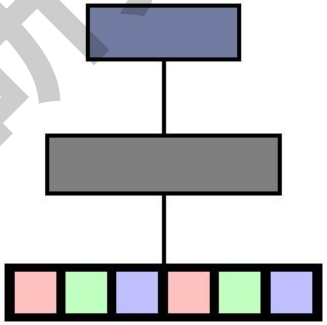


- 持续数个周期

外部存储器访问

处理大小为224×224×3的图像

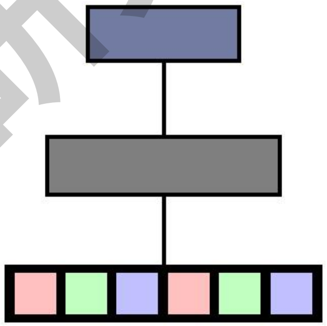
- 通用处理器
  - 工作在内存上
  - 需要执行30万条load/store指令

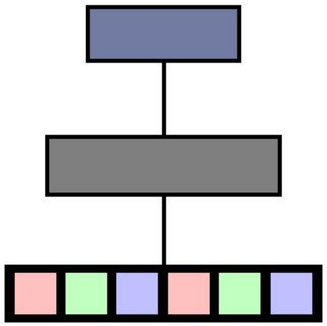


外部存储器访问

处理大小为224×224×3的图像

- 通用处理器
  - 工作在内存上
  - 需要执行30万条load/store指令
- 深度学习处理器
  - 工作在便笺存储器上
  - 1条load指令装载一整块图像
  - 1条指令完成计算
  - 1条store指令送回内存

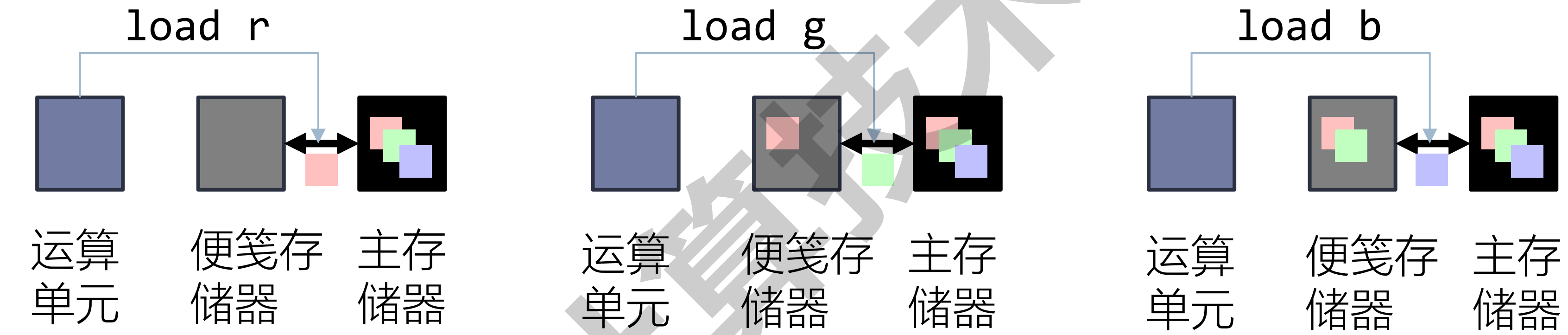




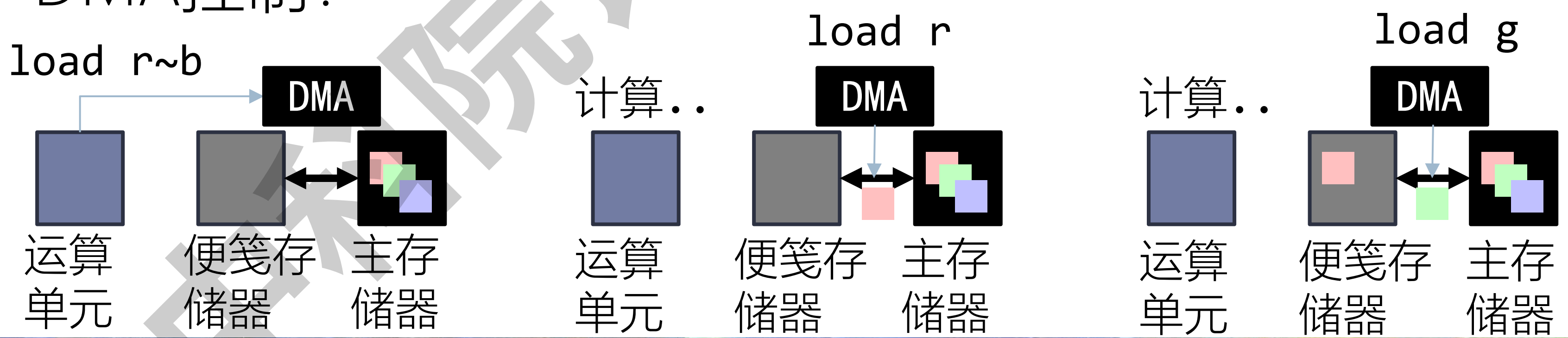
直接内存访问（DMA）

如何实现"1条load指令装载一整块图像"？

- 处理器控制：

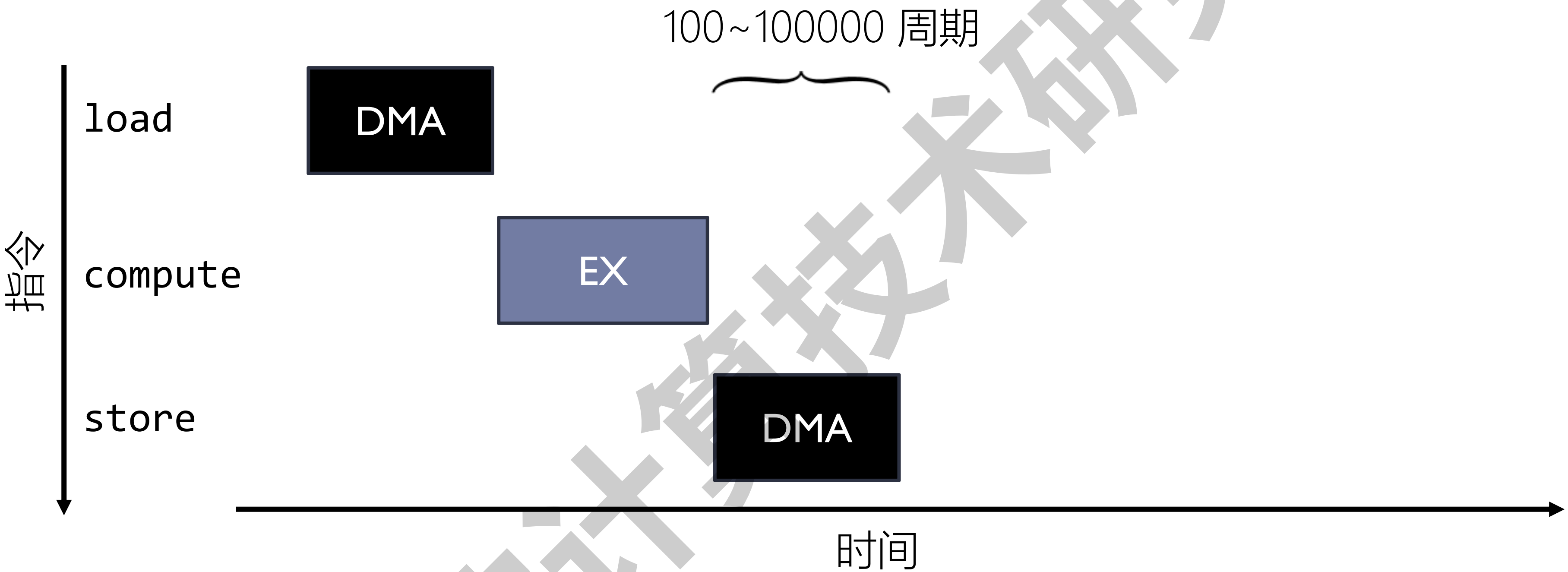


- DMA控制：



外部存储器访问

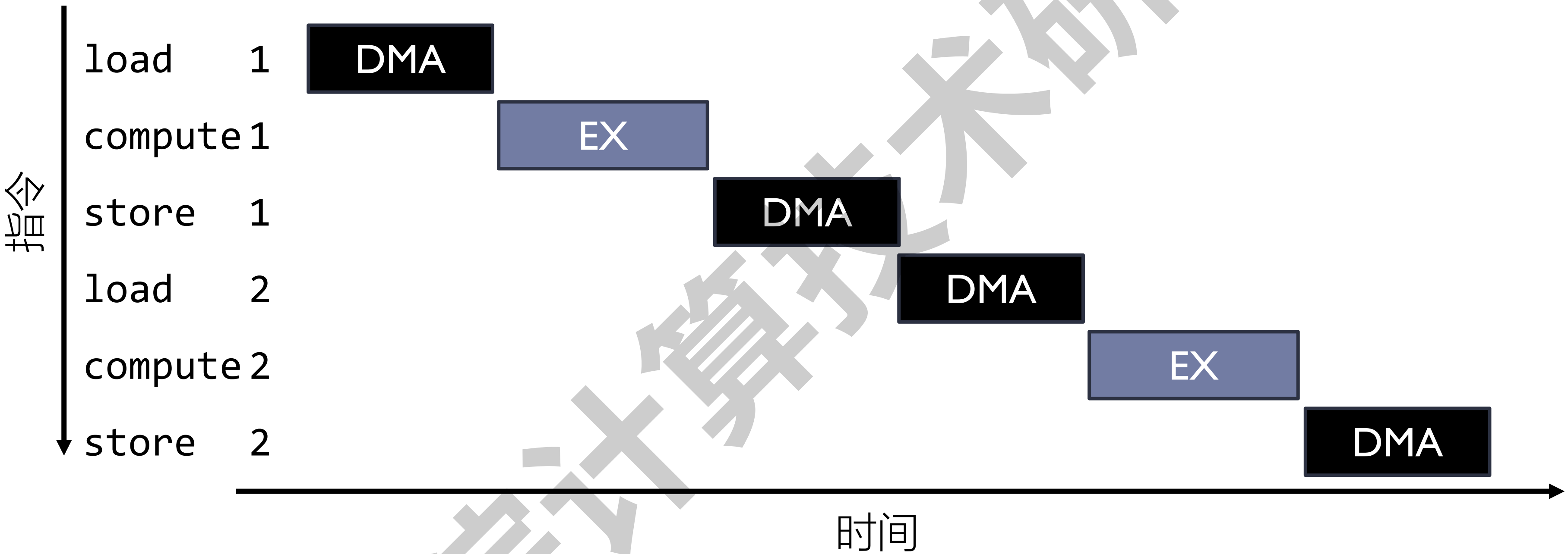
深度学习处理器的访存



- 持续数百至数十万个周期

外部存储器访问

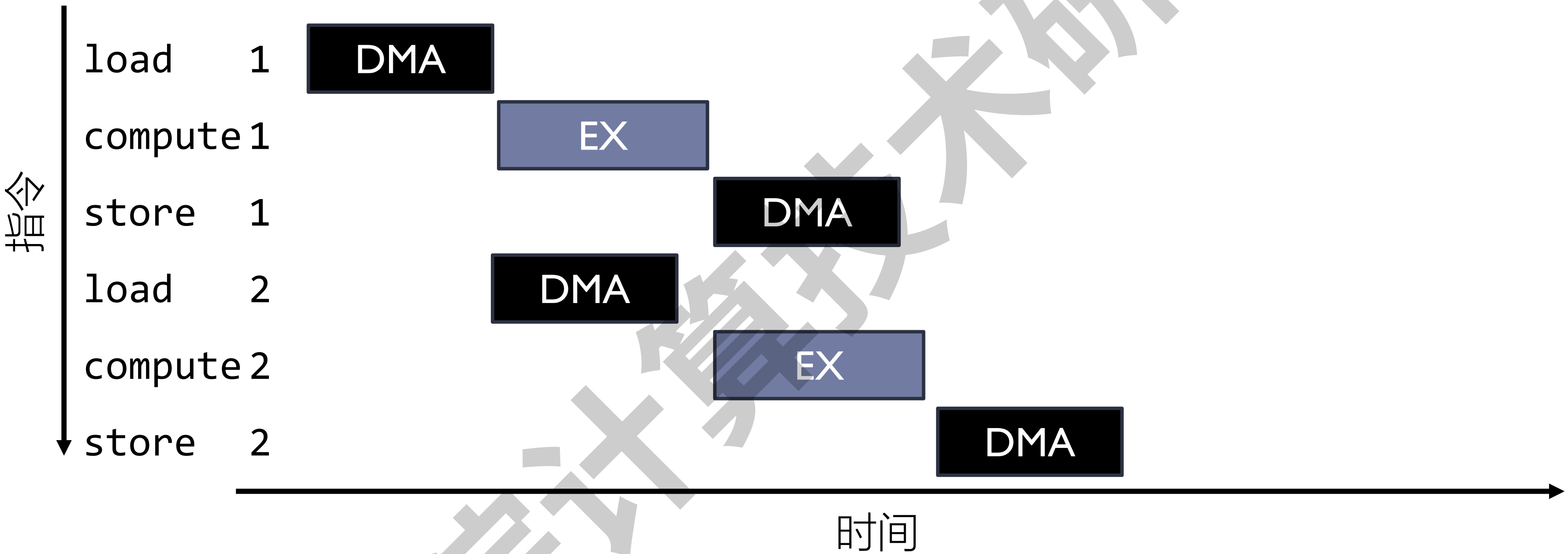
深度学习处理器的访存



- 持续数百至数十万个周期

外部存储器访问

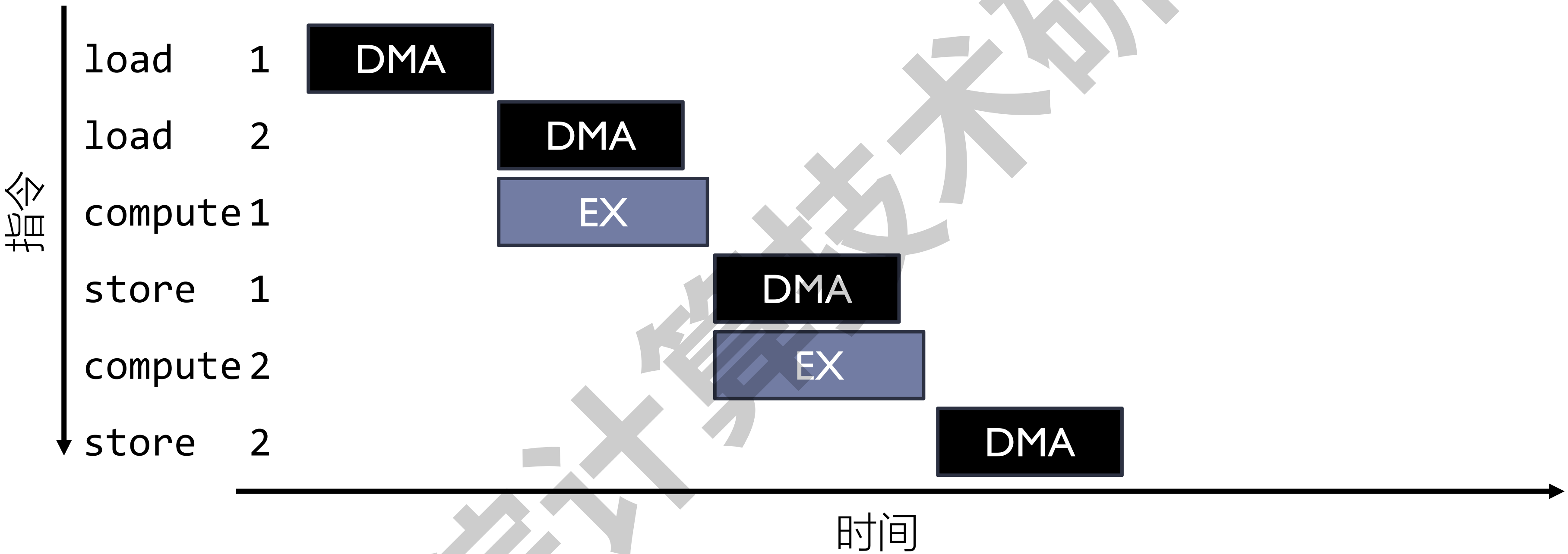
深度学习处理器的访存



- “软件流水线”

外部存储器访问

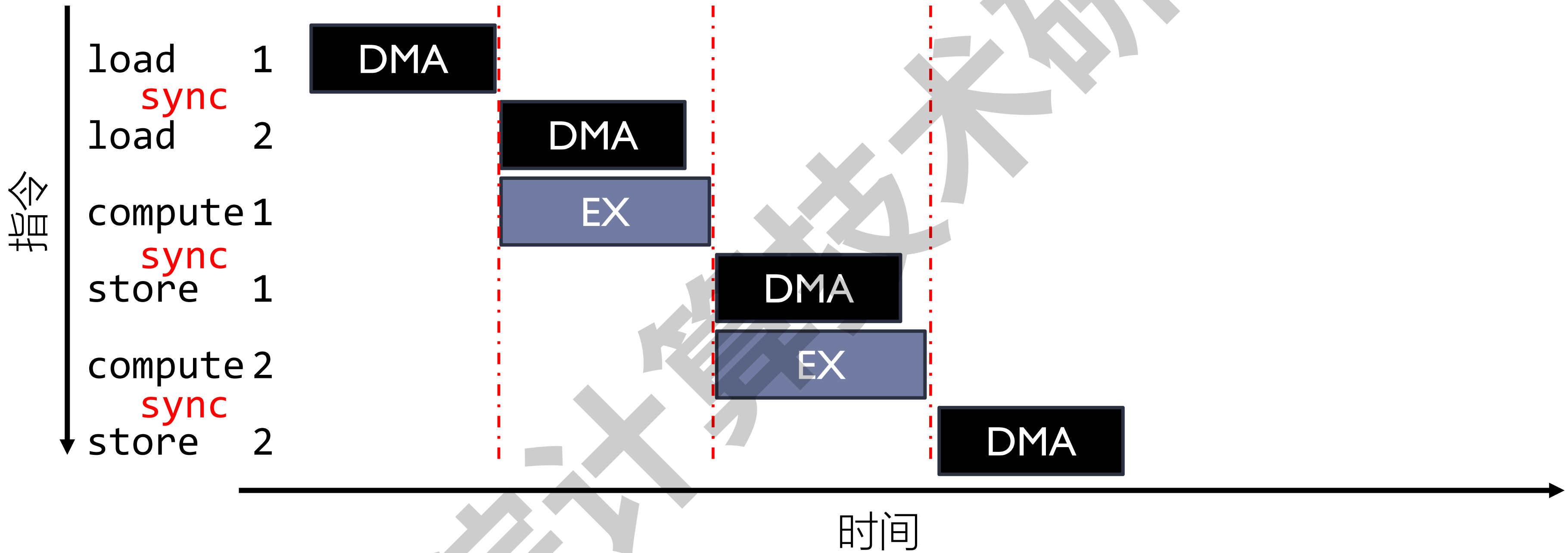
深度学习处理器的访存



- 通过编译器或编写程序重新安排指令顺序，简化硬件

外部存储器访问

深度学习处理器的访存

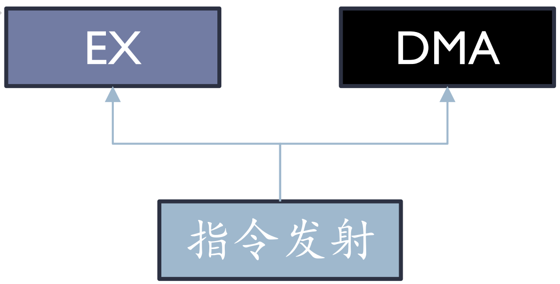


- 通过编译器或编写程序重新安排指令顺序，简化硬件
- 显式控制同步，简化硬件

软件流水线

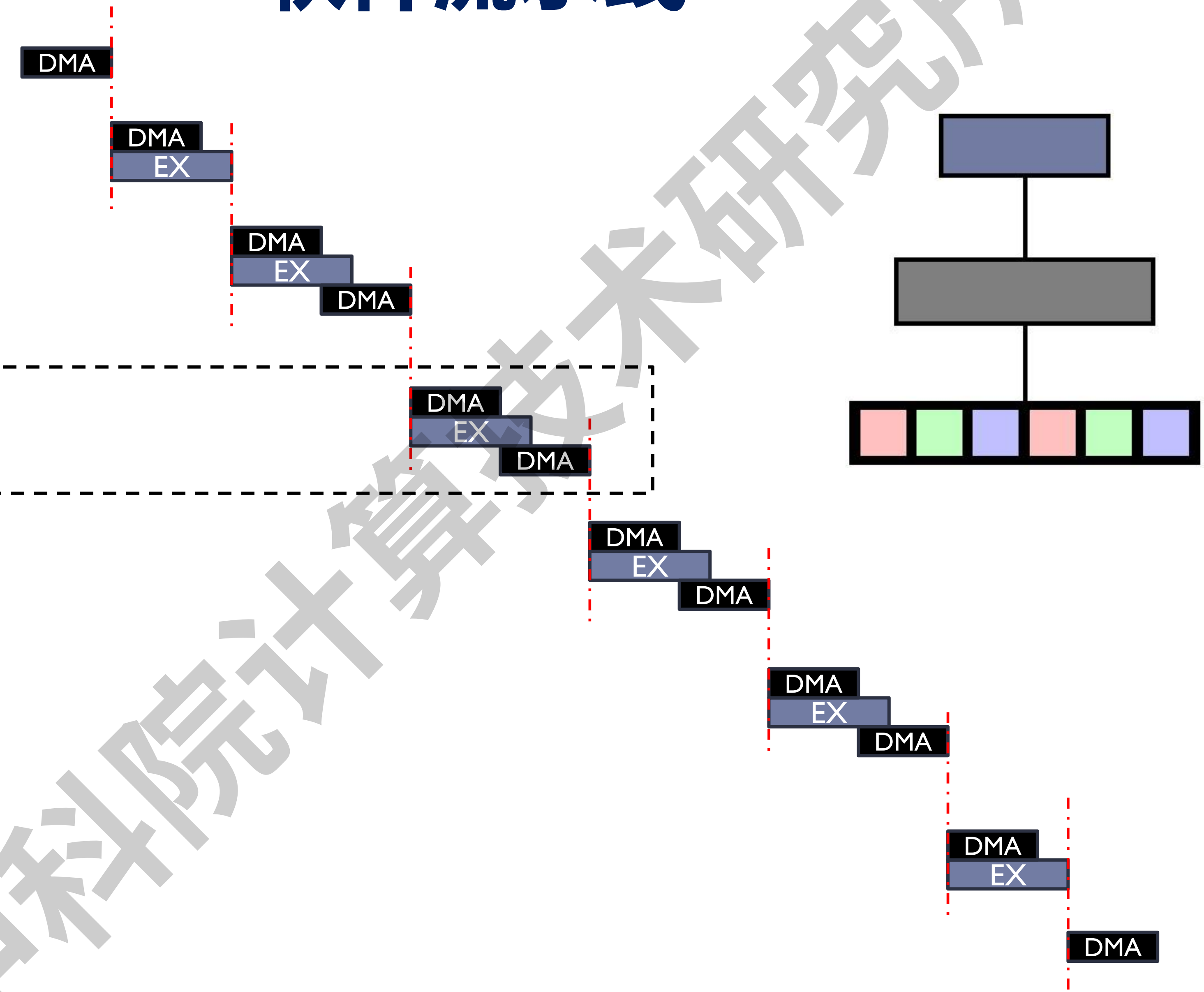
如何实现同步指令（sync）？

- 简化硬件模型描述：
  - 计算模块：随时执行收到的指令
  - DMA模块：随时执行收到的指令
  - 指令发射模块：
    - 计算指令发射到计算模块
    - 访存指令发射到DMA模块
    - 遇到sync时：阻塞，直到整个处理器空闲下来，再发射新的指令



软件流水线

```
load      1
sync
load      2
compute   1
sync
store     1
compute   2
load      3
sync
store     2
compute   3
load      4
sync
store     3
compute   4
load      5
sync
store     4
compute   5
load      6
sync
store     5
compute   6
sync
store     6
```

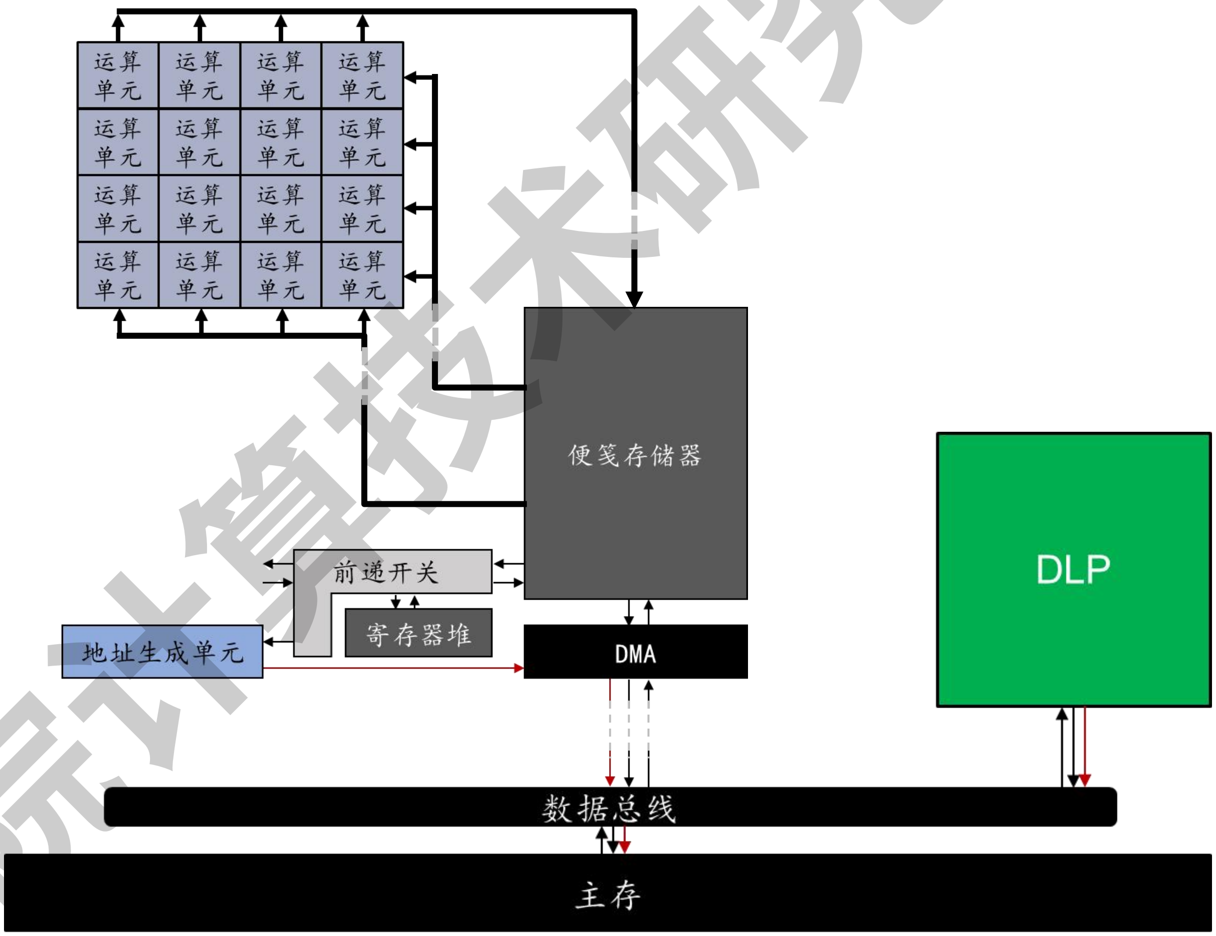


访存小结

- 便笺存储器是DLP核心的数据枢纽
  - 访问便笺可能成为瓶颈
  - “拓宽道路”：增加端口、设计为分组SRAM
    - 代价：硬件开销增加
  - “规划车流”：根据算法特征，采用分离式设计
    - 代价：降低通用性
- 通过软件流水线（而不再是硬件）使计算/访存并行起来
  - 指令通过编译器或编写程序重新排序，不需要乱序执行
  - 显式控制同步，不需要依赖检查

总体架构

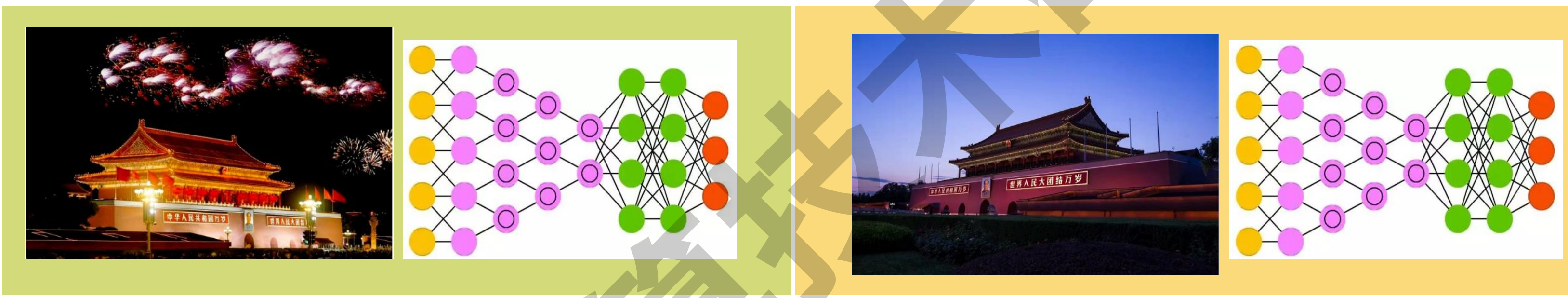
- 计算
- 访存
- 通信



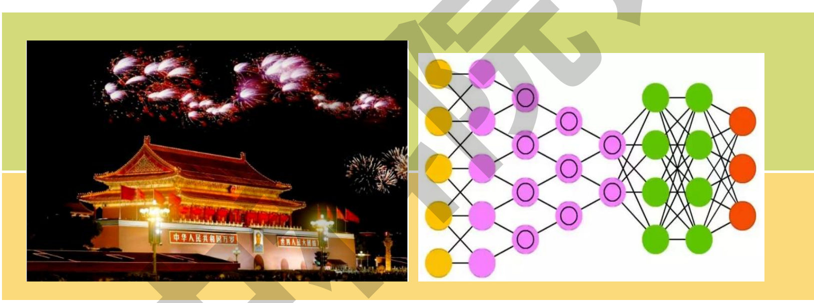
通信

几种任务划分模式

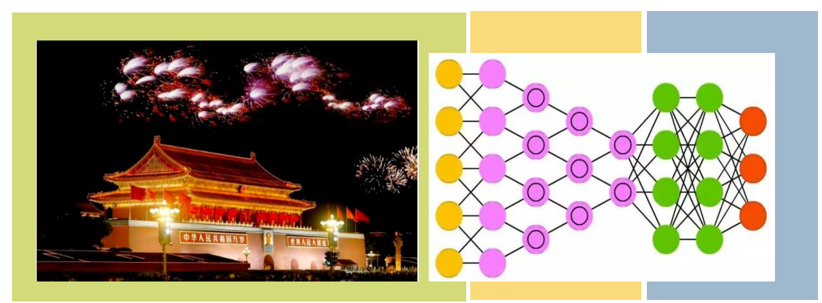
- 数据并行



- 模型并行
  - 算子内并行



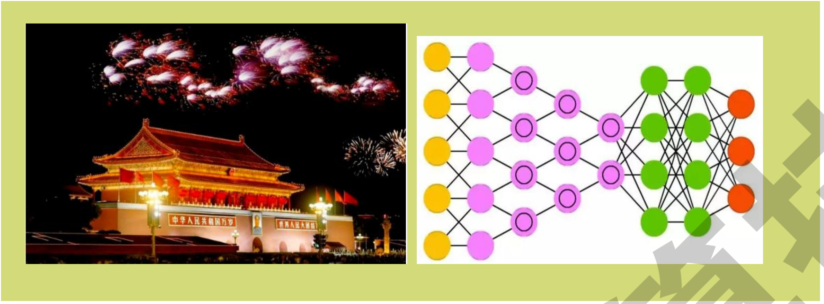
  - 算子间并行

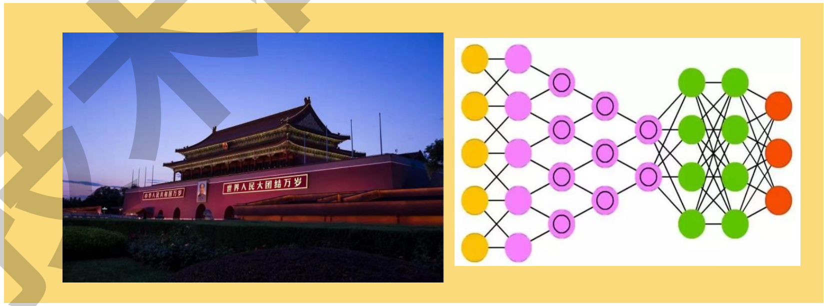


通信

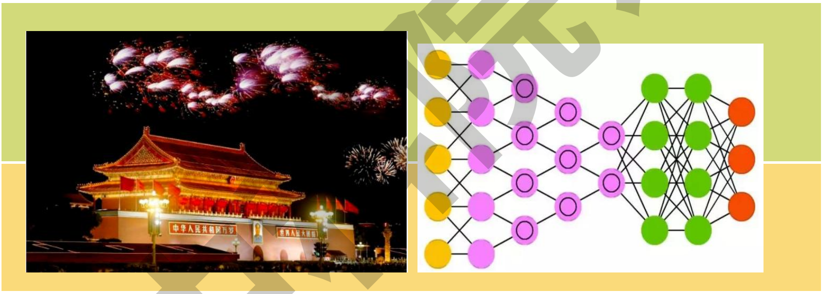
几种任务划分模式

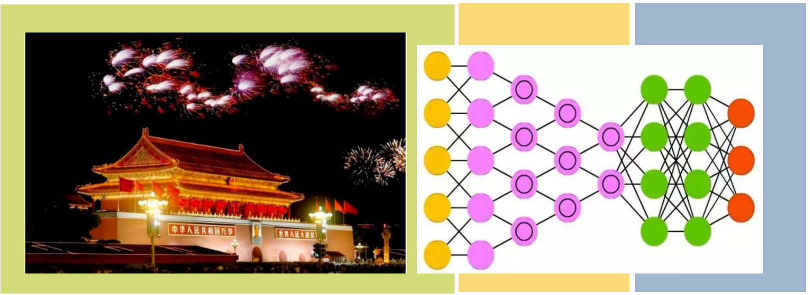
- 数据并行：全局归约（all-reduce)





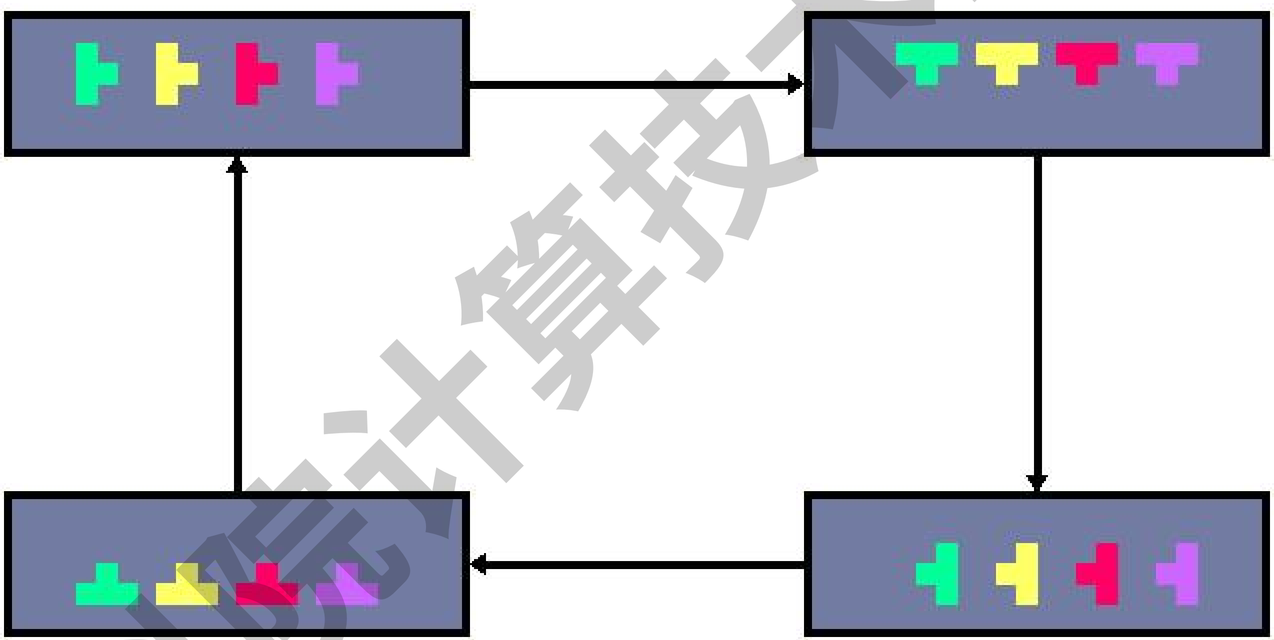
- 模型并行
  - 算子内并行：全局交换（all-to-all ）
  - 算子间并行：局部通信





全局归约（all-reduce）

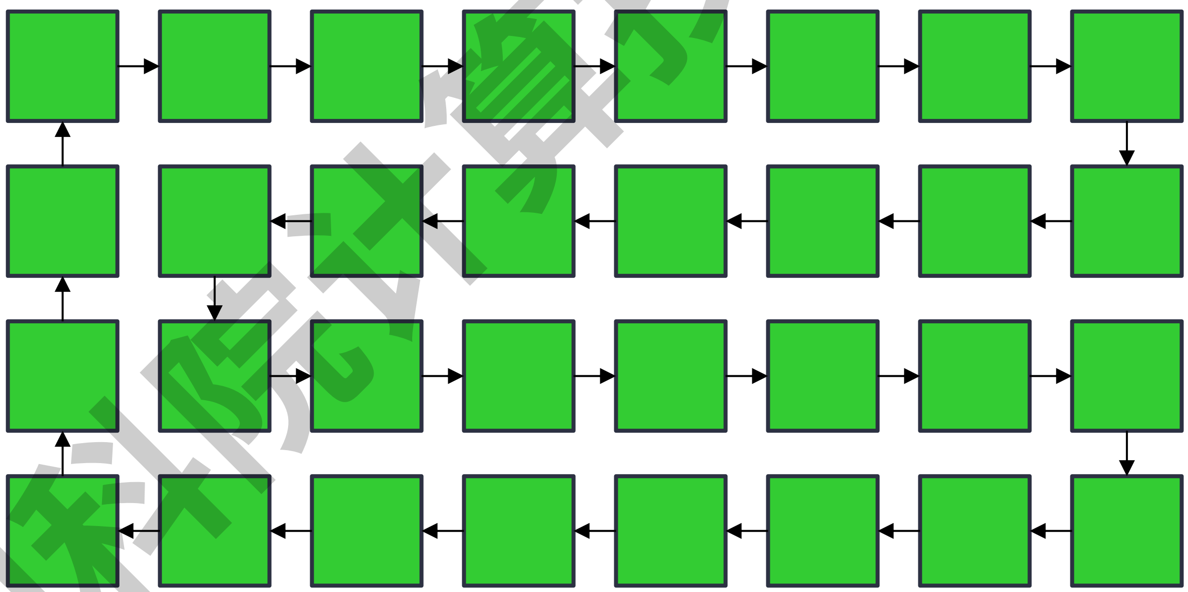
通过一个环（ring），就可以高效实现全局归约



全局归约

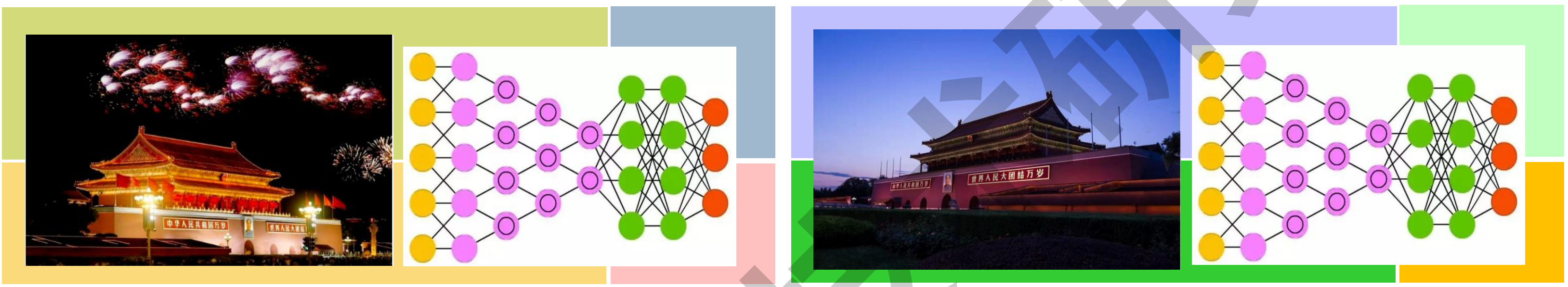
通过一个环（ring），就可以高效实现全局归约

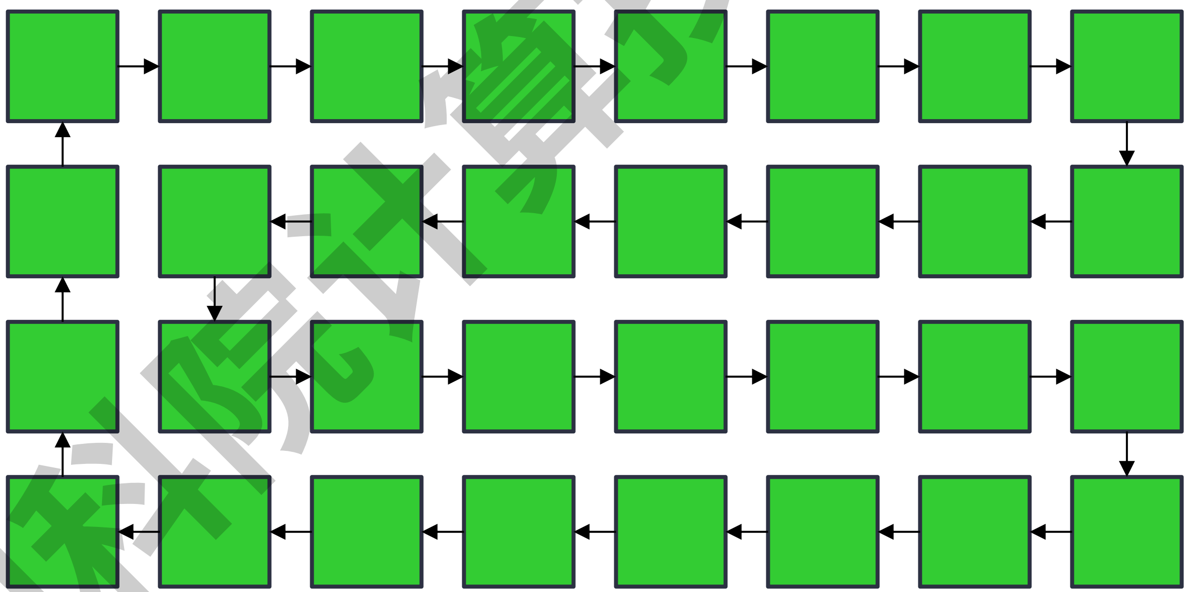
- 是否意味着多核心DLP架构应该设计为环状?



通信

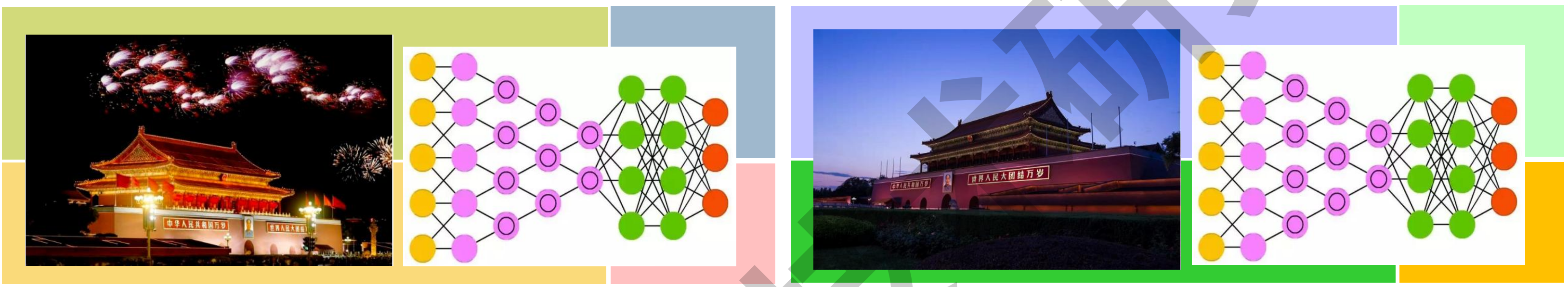
混合使用几种任务划分模式

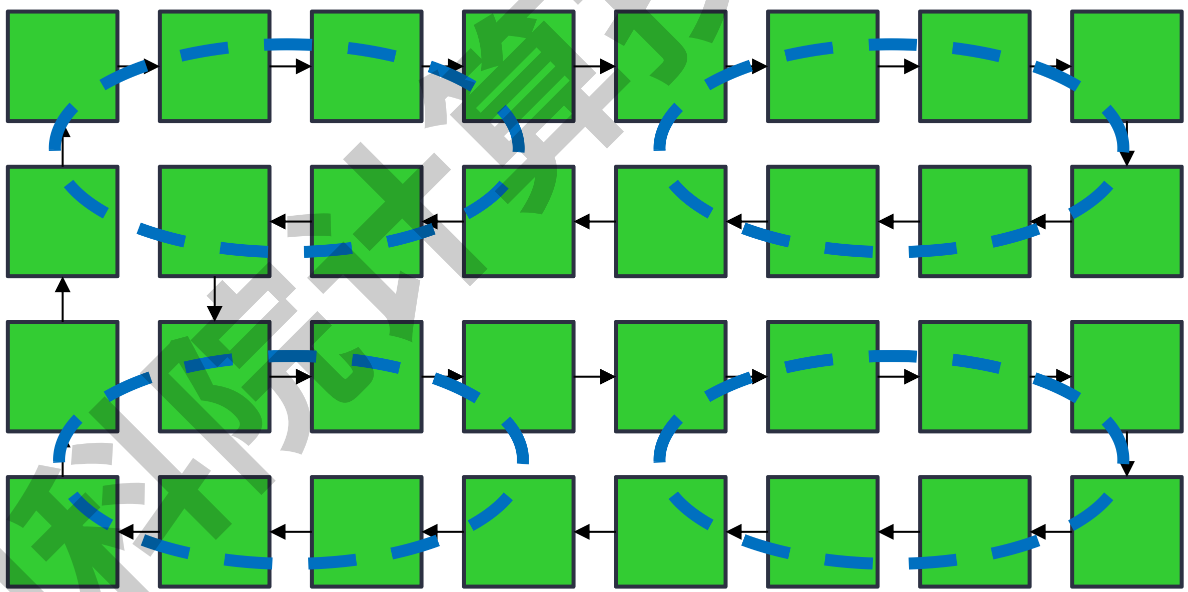




通信

混合使用几种任务划分模式

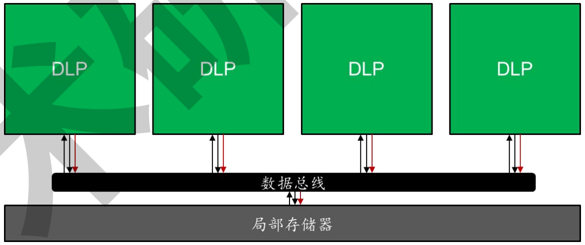


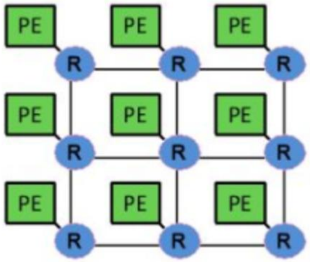


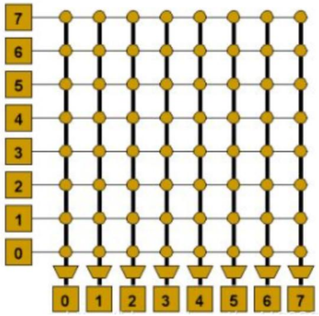
通信

常见物理链路设计：

- 总线（bus）
  - 常用AXI、PCI-E等标准
  - 性能差，成本低
- 片上网络（NoC）
  - 常用胖树、二维环面等拓扑
  - 性能较好，成本可控
- 交叉开关阵列（crossbar）
  - 性能最佳，成本高







通信小结

通信结构的设计原则：

- **逻辑上**：环状链路足以高效完成通信
- **物理上**：链路设计适当增加冗余，按需配成环路
- 综合考虑性能和成本约束做出选择

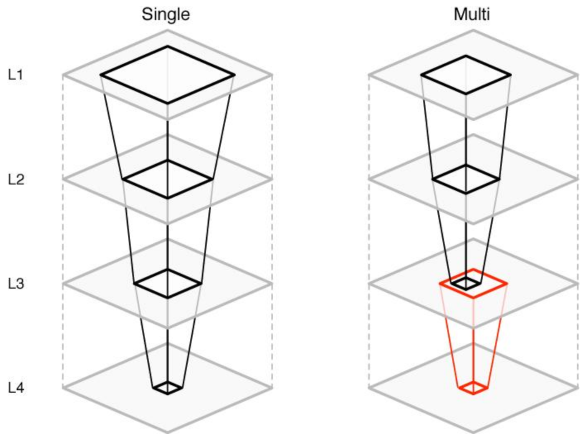
优化设计

一些常用的/前沿的优化设计：

- 变换
  - 对算法进行变换，削减计算强度
- 压缩
  - 对算法进行压缩，直接减少算法的参数量和计算量
- 近似
  - 对算法进行近似替代，降低计算成本
- 非传统结构和器件
  - 探索采用CMOS数字电路以外的技术，改写计算范式

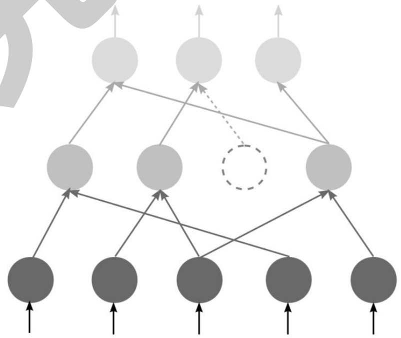
变换

- 快速矩阵乘法算法
  - Strassen算法、Coppersmith-Winograd算法、FFT
  - 降低矩阵乘法算法复杂度
- 快速卷积算法
  - Winograd最小滤波算法
  - 降低卷积算法复杂度
- 算子融合
  - 多个串行算子融合在一起计算
  - 降低通信复杂度，减少访存

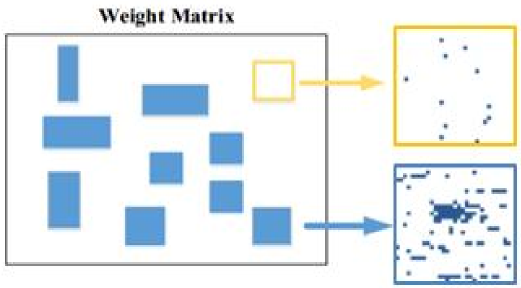


压缩

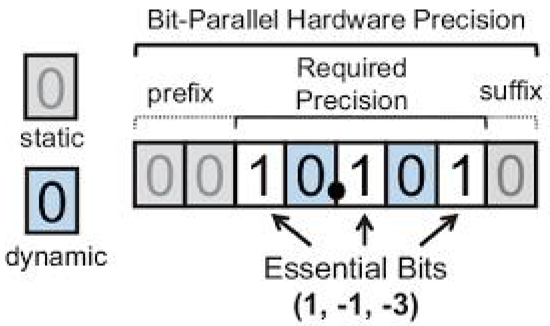
- 网络裁剪
  - 思路：权值或激活值为零时，不需要计算
  - 方法：增加硬件，选出非零部分送入运算器



- 结构化稀疏
  - 思路：改动算法，让非零值呈规律分布
  - 方法：简化选数模块

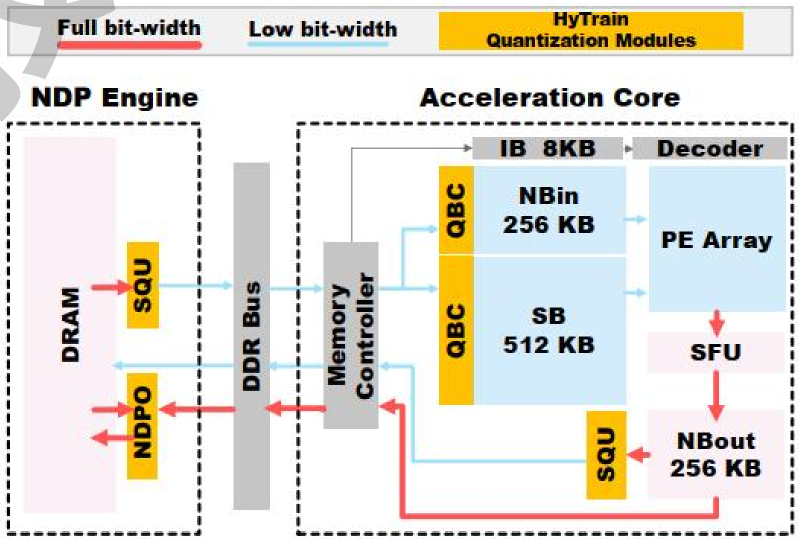


- 串行计算
  - 思路：单个数值中比特0也可以跳过
  - 方法：改为串行运算器，逐比特计算



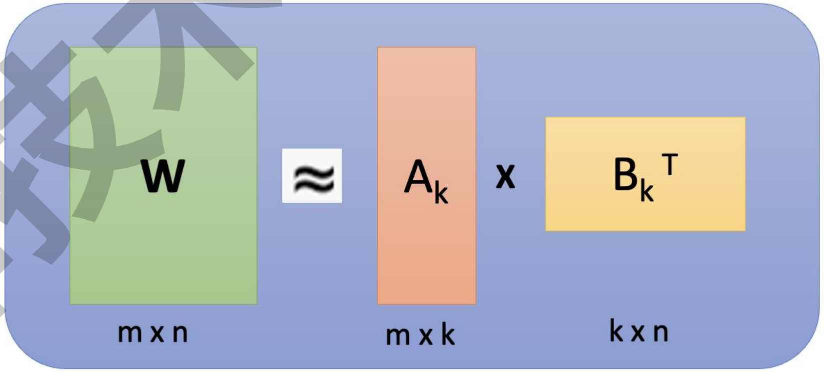
近似

- 数值量化
  - **思路**：将数值替换为更容易计算的近似值
  - **方法**：用低精度数值近似高精度，辅以误差校正
    - 定点数 （例：INT4）
    - 低精度浮点数 （例：FP4）
    - 块浮点数 （例：BFP）
    - 对数域计算 （例：PoT）
    - 二值化 （例：BNN）
    - 可变精度

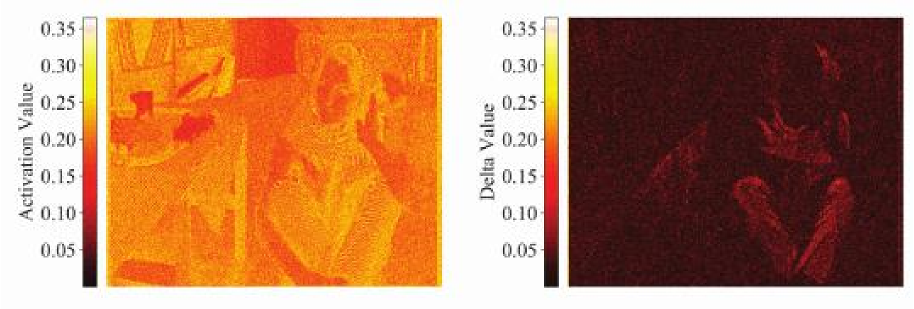


近似

- 算法近似
  - **思路**：从算法上将计算分解，提取出主要部分
  - 低秩分解
    - 将权值矩阵进行特征分解
    - 只保留主要特征



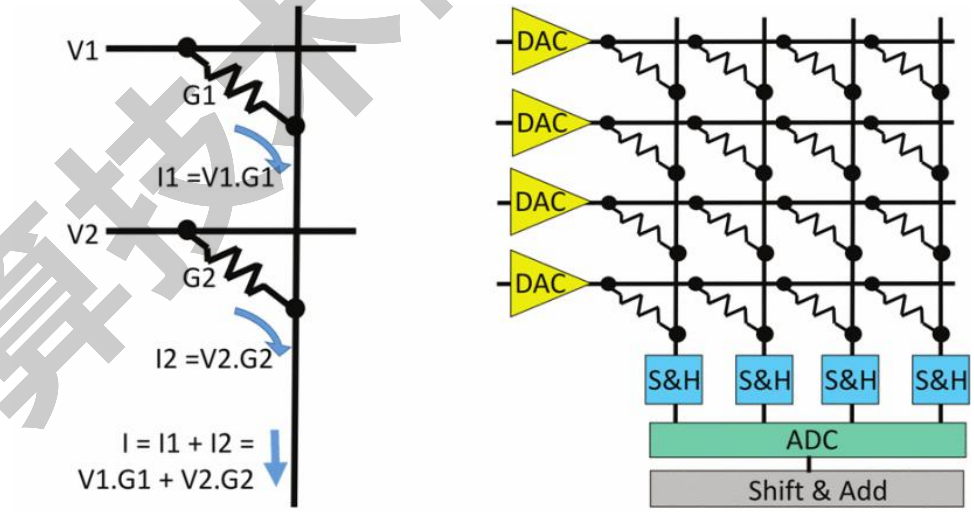
  - 差分计算
    - 图像具有局部相似性
    - 邻域差分值比原值更易表达



(a) Activation values (b) Deltas along the X-Axis

非传统结构和器件

- 神经拟态计算
  - **思路**：数值可以用模拟物理量（电压、电导、电流）来表达
    - 激活值→电压
    - 权值→电导
    - 神经元输出→电流



(a) Multiply-Accumulate operation (b) Vector-Matrix Multiplier

  - 计算发生在权值的存储矩阵内

总结

- 计算部分：矩阵、向量、标量各司其职
  - 讨论了多种实现方式
- 访存部分："拓宽道路"，"规划车流"
  - 合理利用应用特性
- 通信部分：高效完成全局归约、全局交换
  - 逻辑链路要简单，物理链路要灵活

- 概览了常见的和前沿的优化方法

谢谢大家！