



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

陈云霁 研究员

cyj@ict.ac.cn

架构设计需要解决的两个主要问题

- ▶ 如何提高处理器的能效比（性能/功耗）-硬化算法
- ▶ 如何提高处理器的可编程性（通用性） - CPU
- ▶ 找到平衡点。 需要深度学习算法知识（tensor描述，运算op）， 电路设计知识， 架构设计知识等。
- ▶ 如何基于tensor设计高能效比， 可编程性的深度学习处理器

提纲

- ▶ 单核深度学习处理器 (DLP-S)
- ▶ 多核深度学习处理器 (DLP-M)

中科院计算所

单核深度学习处理器 (DLP-S)

控制模块

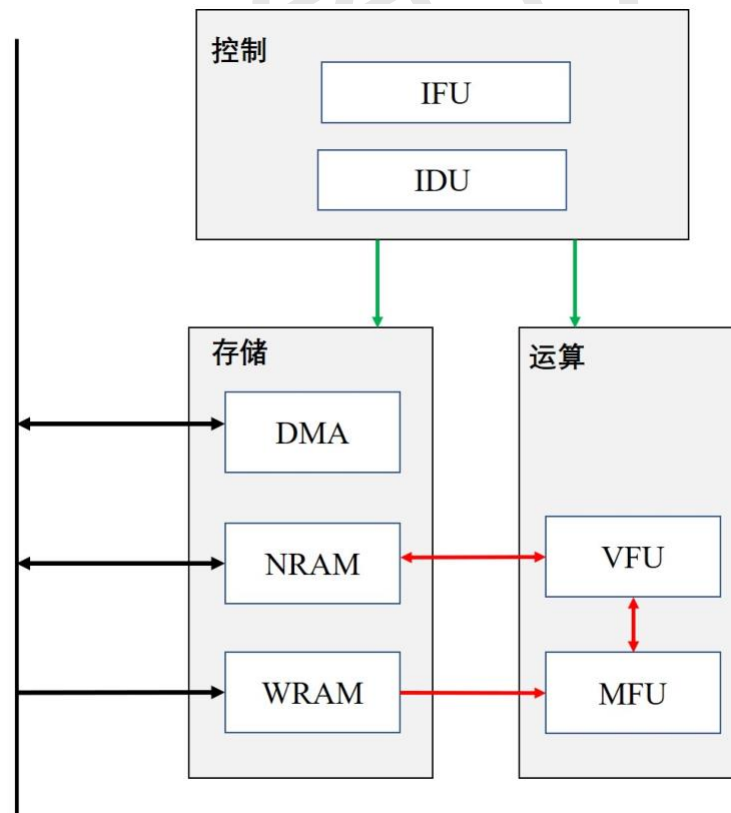
- ▶ 指令的语义粒度 (提供专用指令, 操作粒度为tensor)
- ▶ 领域专用指令 vs. RISC vs. CISC

运算模块

- ▶ 基于tensor语义设计运算模块

存储模块

- ▶ 基于tensor语义设计存储模块



总体架构

▶ 控制模块

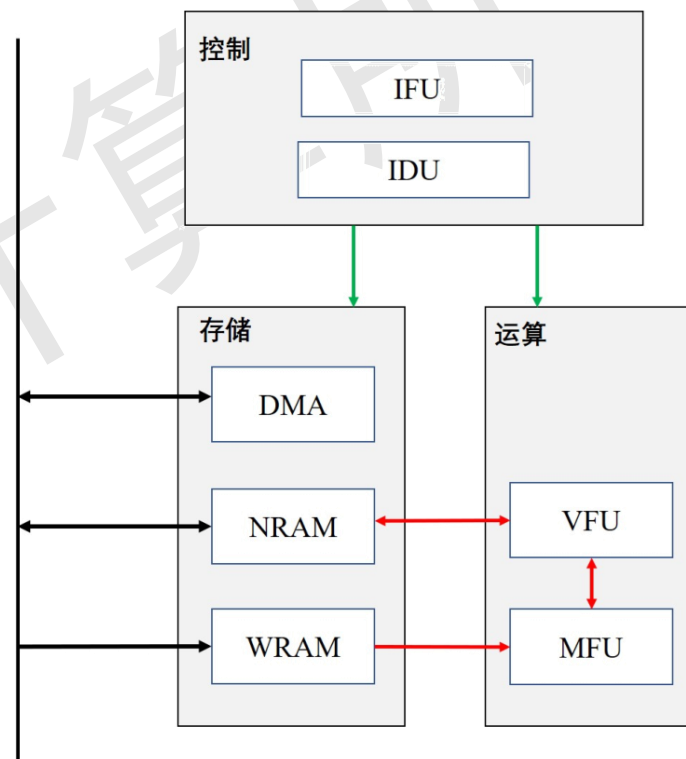
- ▶ 取指单元IFU (Instruction Fetch Unit)
- ▶ 指令译码单元IDU (Instruction Decode Unit)

▶ 运算模块

- ▶ 向量运算单元VFU (Vector Function Unit)
- ▶ 矩阵运算单元MFU (Matrix Function Unit)

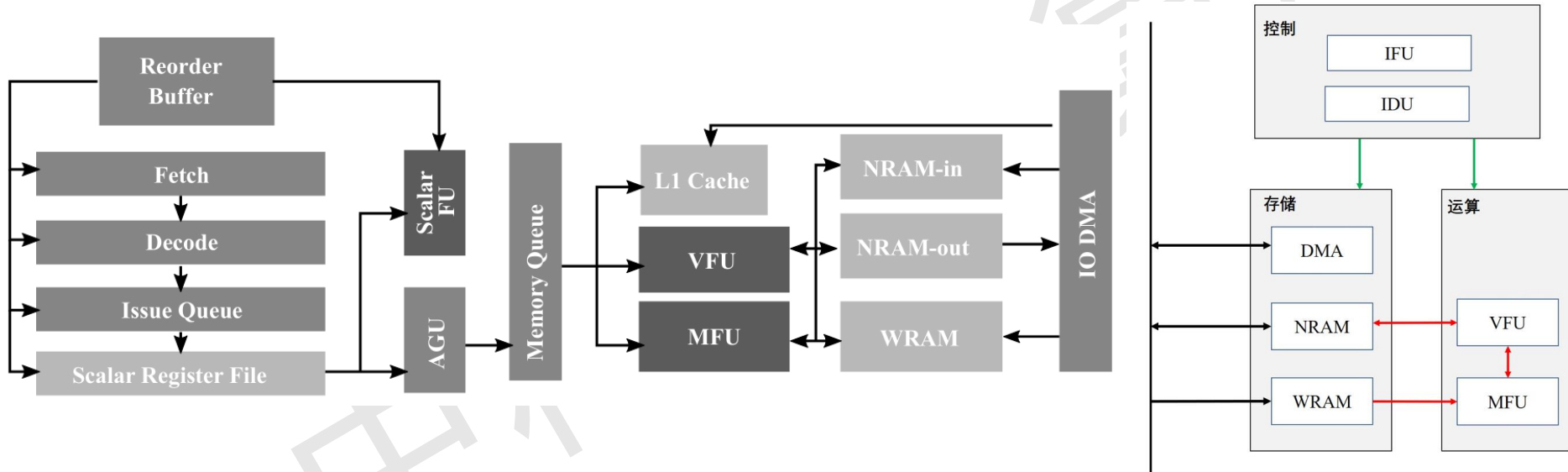
▶ 存储模块

- ▶ 权重存储单元WRAMW (Weight RAM)
- ▶ 神经元存储单元NRAM (Neuron RAM)
- ▶ 直接内存存取单元DMA (Direct Memory Access)



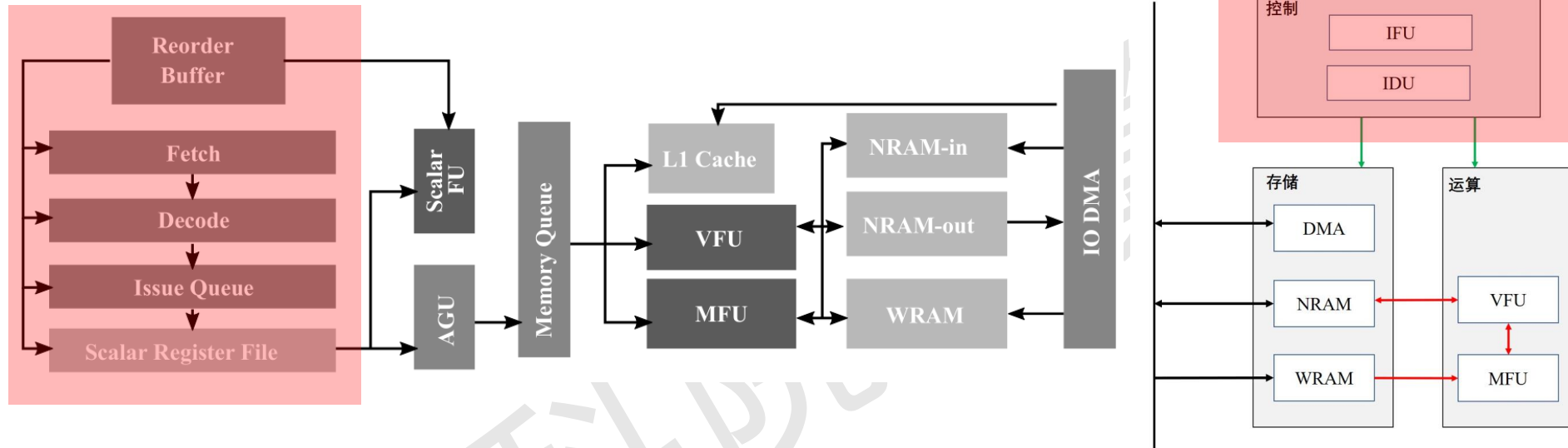
总体架构

从DLP到DLP-S



总体架构

从DLP到DLP-S

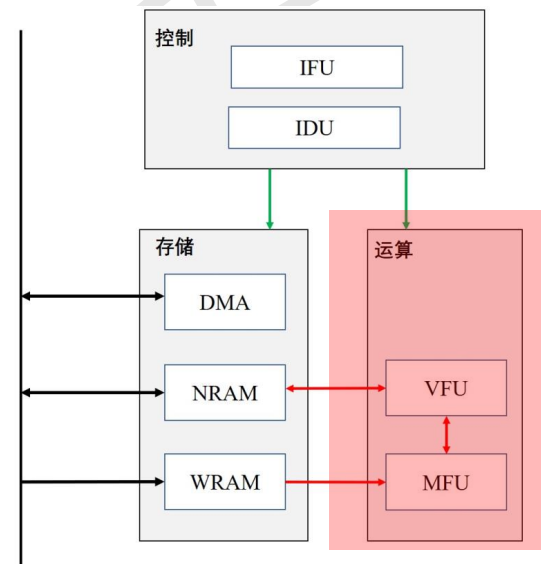
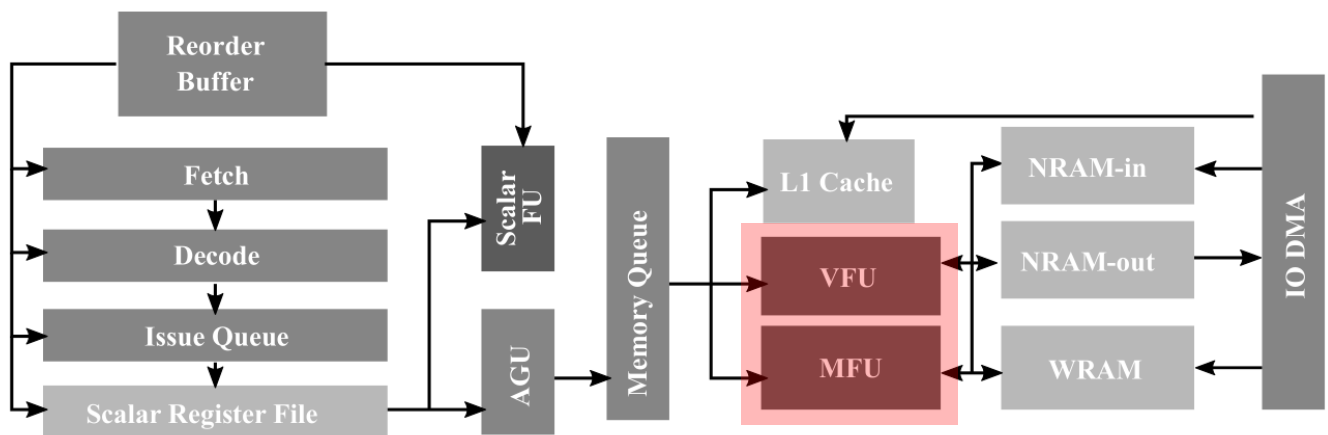


控制:

多发射队列, 支持指令级并行, 寄存器重命名

总体架构

从DLP到DLP-S

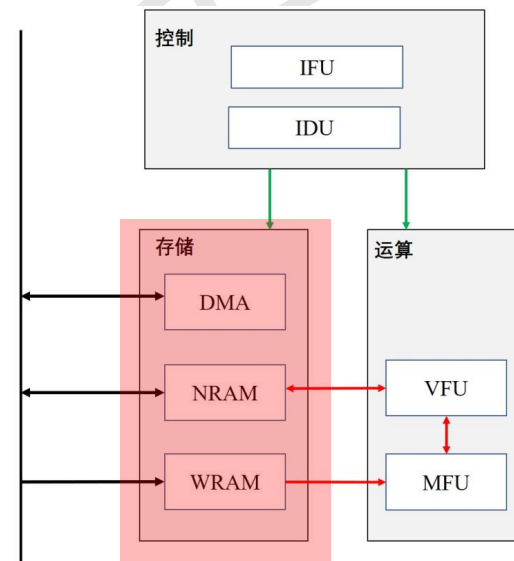
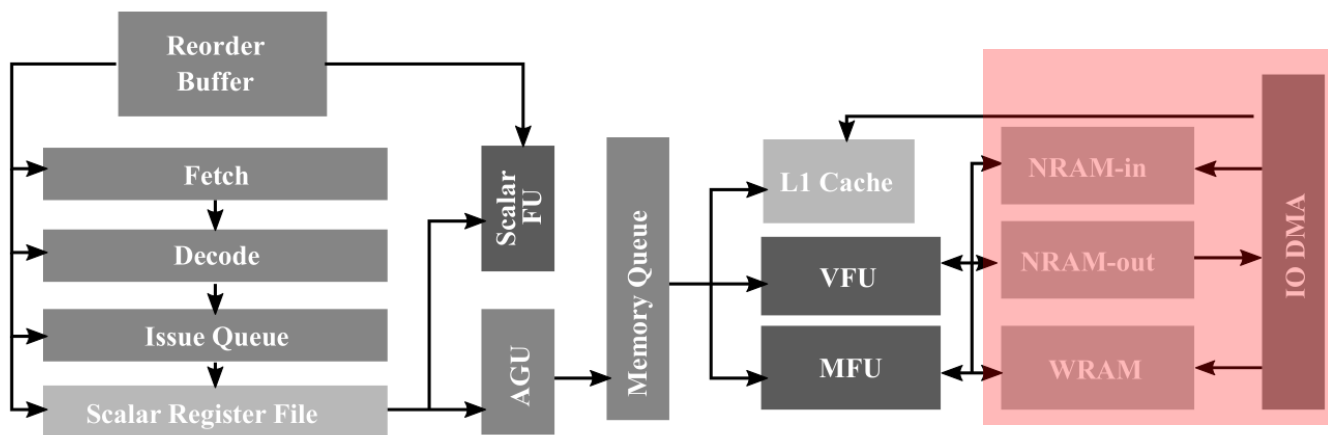


计算:

多发射队列, 支持指令级并行, 寄存器重命名
增加运算器中的操作, 支持硬件高效执行的操作
低位宽运算器 (量化), 提高执行能效
稀疏运算, 提高计算效率

总体架构

从DLP到DLP-S



访存:

多发射队列，支持指令级并行，寄存器重命名
增加运算器中支持的操作
低位宽运算器

稀疏数据的稠密化访存，降低访存开销

转换检测缓冲区 (TLB, Translation Lookaside Buffer)，降低访存延迟

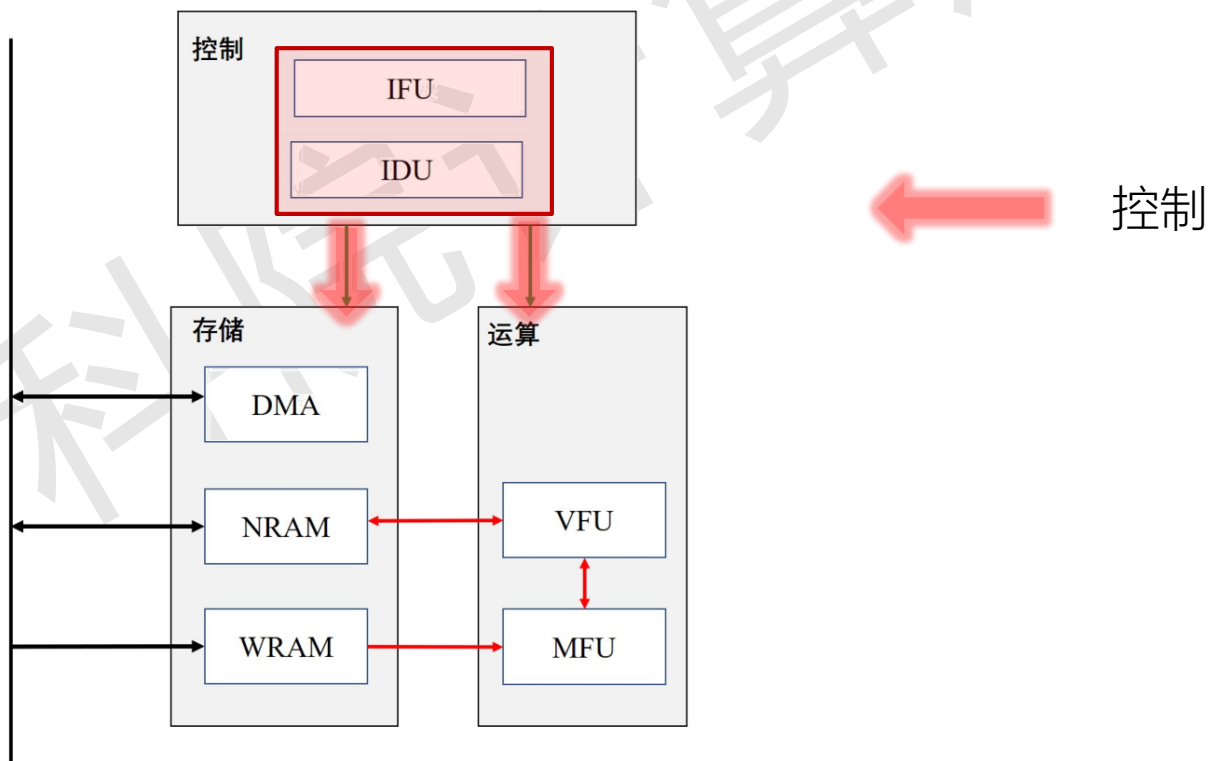
最后一级cache (LLC, Last Level Cache)，降低访存延迟

总体架构

- ▶ 从DLP到DLP-S
 - ▶ 多发射队列，支持指令级并行
 - ▶ 增加运算器中支持的操作
 - ▶ 低位宽运算器
 - ▶ 稀疏支持
 - ▶ 转换检测缓冲区 (TLB, Translation Lookaside Buffer)
 - ▶ 最后一级cache (LLC, Last Level Cache)

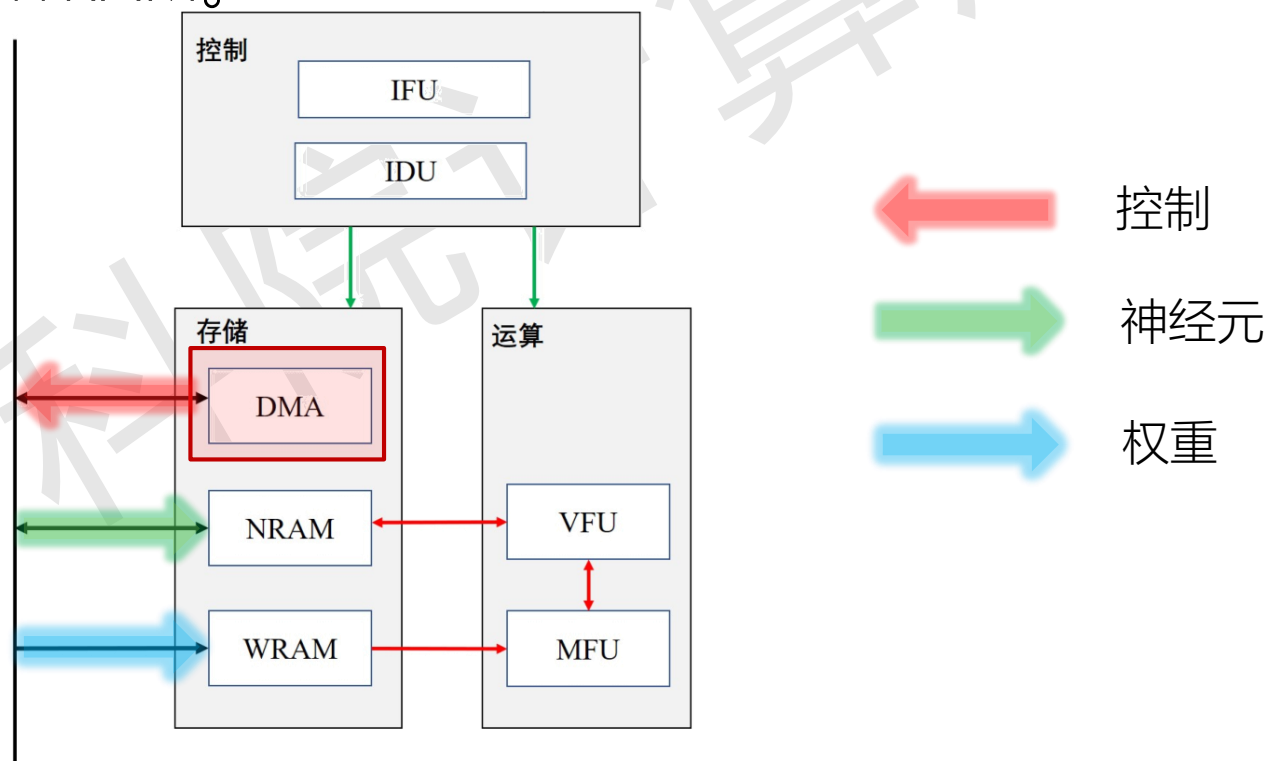
执行流程

- Step #1: IFU 通过 DMA 从 DRAM 中读取程序指令，然后经过 IDU 进行译码后分发给 DMA、VFU 和 MFU



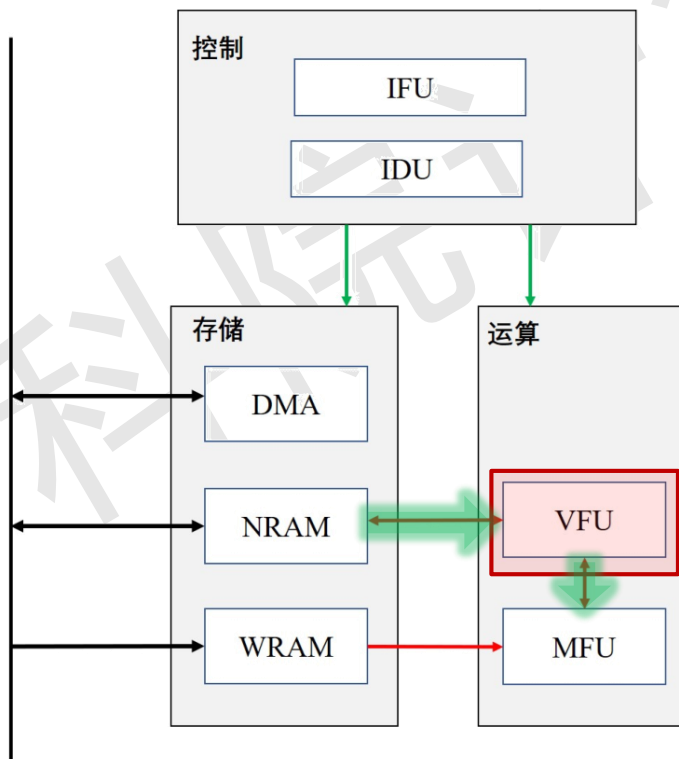
执行流程

- Step #2: DMA 接收到访存指令（读tensor指令，包括地址，数据量等信息）后从 DRAM 读取神经元tensor至 NRAM，读取权值tensor至 WRAM。



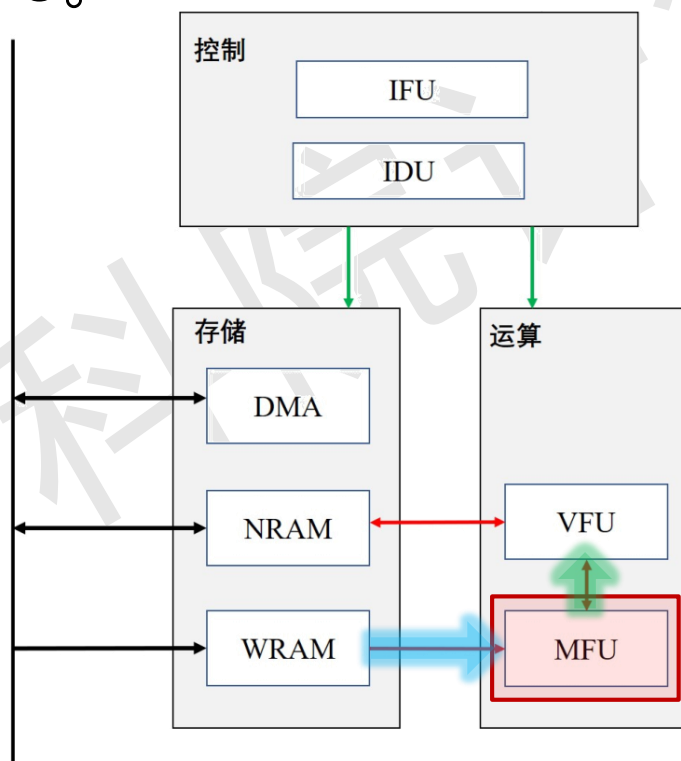
执行流程

- Step #3: VFU 接收到指令后从 NRAM 中读取神经元tensor, 并对神经元tensor进行预处理 (如边界扩充等), 然后发送给 MFU。



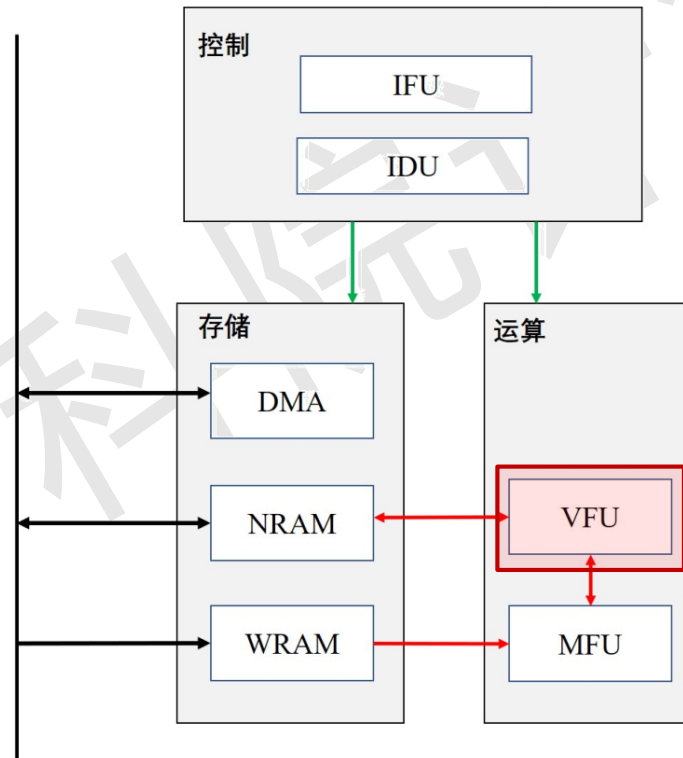
执行流程

- Step #4: MFU 接收到指令后从 VFU 接收经过预处理的神经元tensor，并从 WRAM 中读取权重tensor，完成矩阵运算后将结果发送给 VFU。



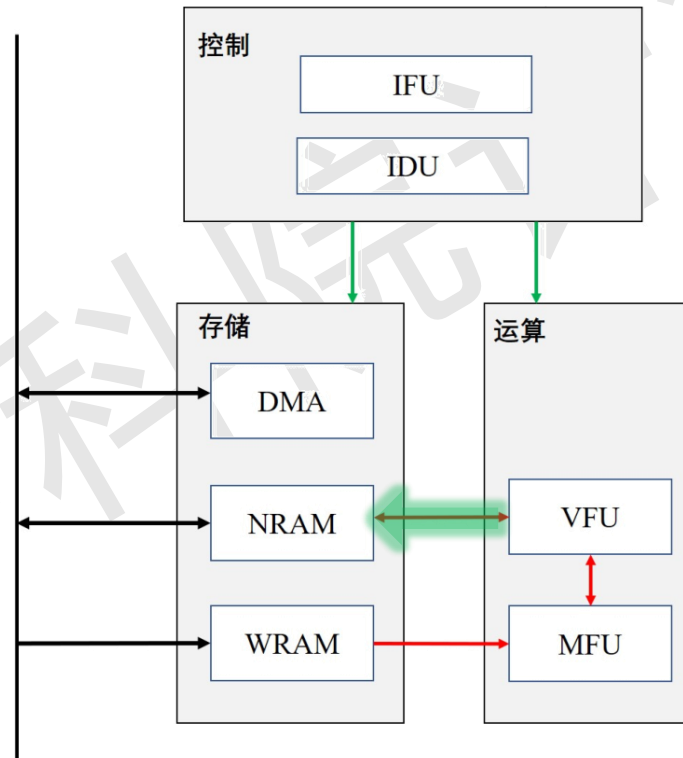
执行流程

- ▶ Step #5: VFU 对输出神经元tensor进行后处理 (如激活、池化等)



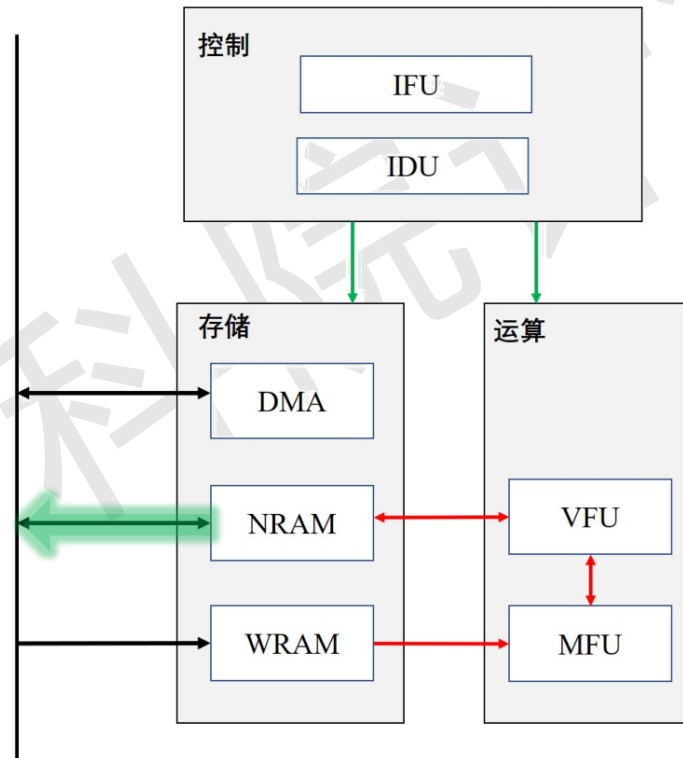
执行流程

- ▶ Step #6: VFU 将运算结果tensor写回 NRAM



执行流程

- ▶ Step #7: DMA 将输出神经元tensor从 NRAM 写回到 DRAM



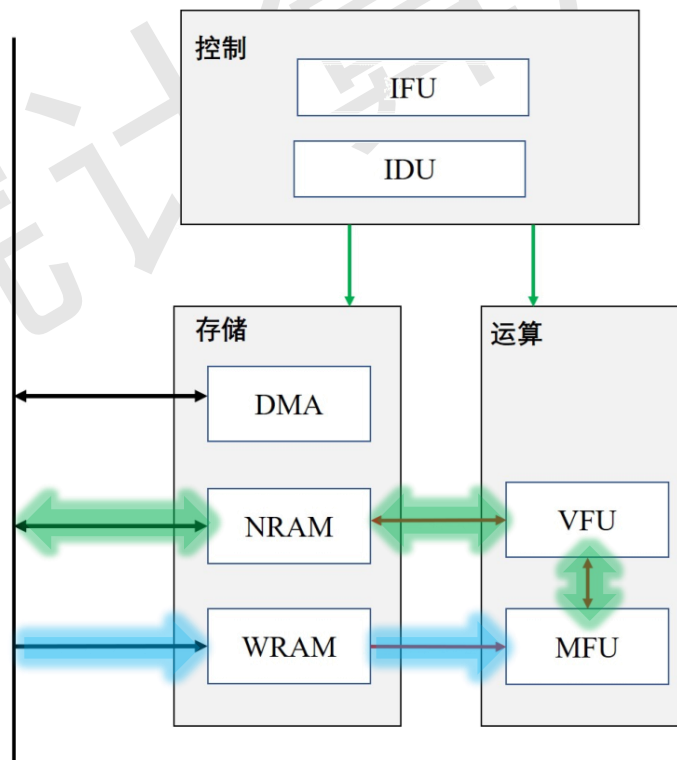
执行流程

▶ 神经元tensor数据流

▶ DRAM->NRAM->VFU-> (MFU->VFU->) NRAM->DRAM

▶ 权值tensor数据流

▶ DRAM->WRAM->MFU

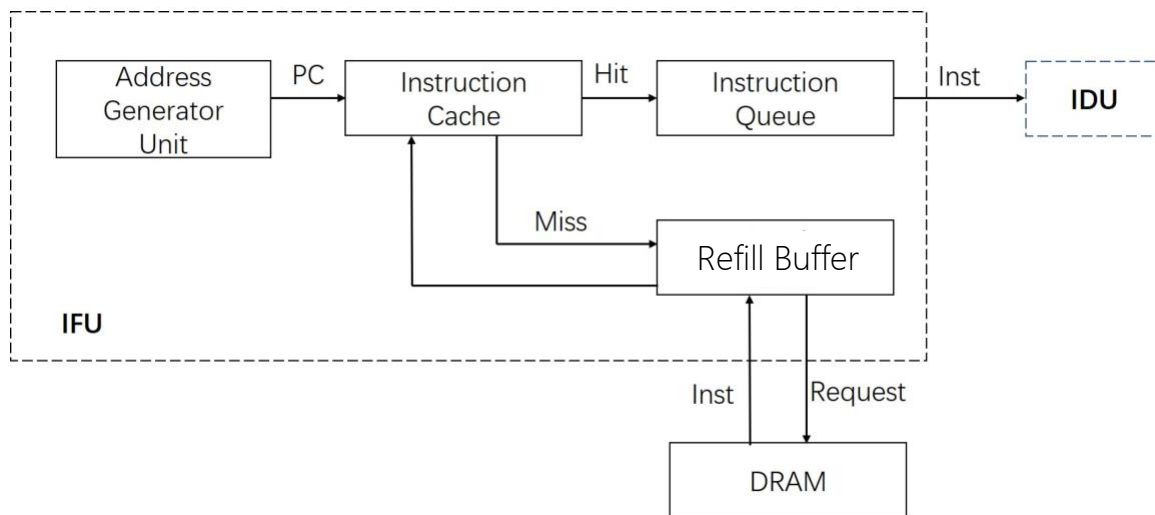


控制模块

▶ IFU

- ▶ 地址生成器AGU (Address Generator Unit)
- ▶ 指令高速缓存ICache (Instruction Cache)
- ▶ 指令回填单元RB (Refill Buffer)
- ▶ 指令队列IQ (Instruction Queue)

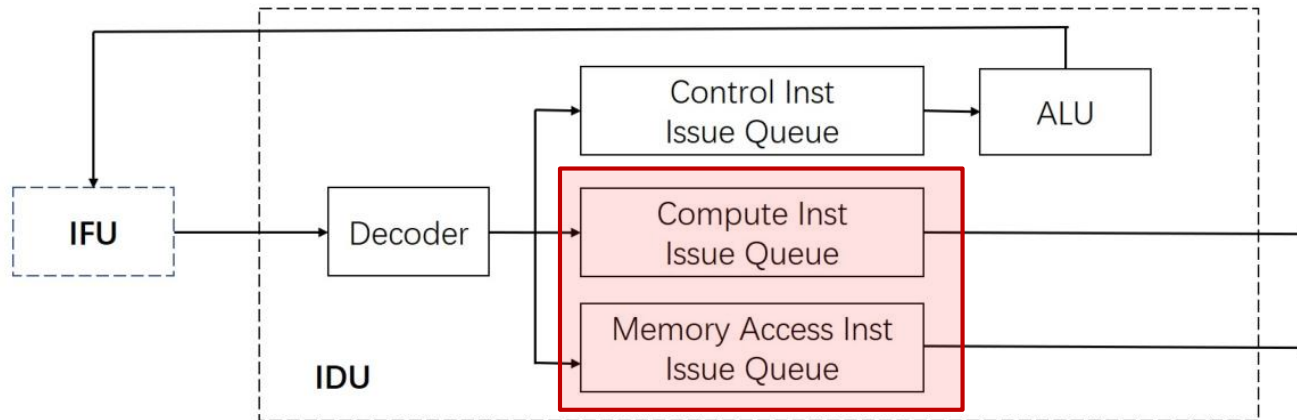
PC = 0
PC = PC + 1
PC = Jump/CB Reg



控制模块

▶ IDU

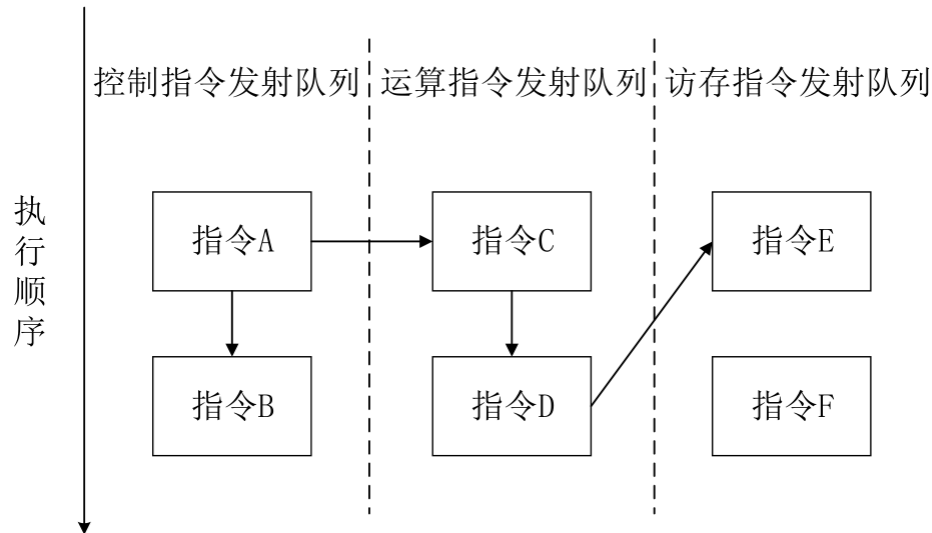
- ▶ 译码单元Decoder
- ▶ 指令发射队列 (Issue Queue)
 - ▶ Control IQ, Compute IQ, Memory Access IQ
- ▶ 算术逻辑单元ALU



控制模块

▶ 指令发射队列IQ

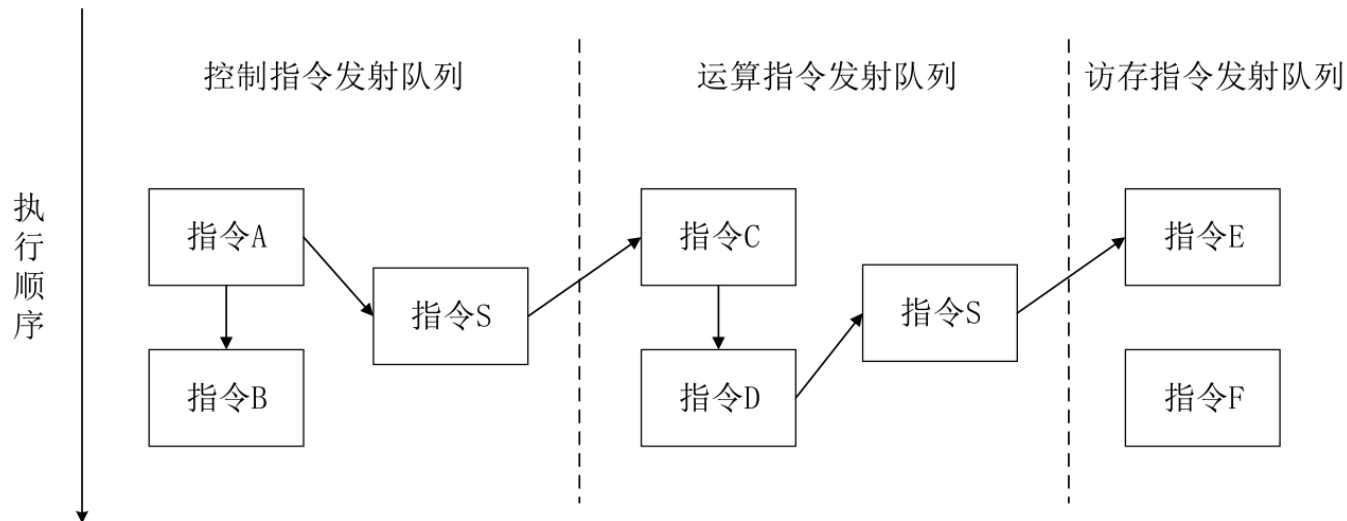
- ▶ 三个指令队列乱序发射，指令队列内顺序发射
- ▶ 两条同类型指令有依赖：位于同一发射队列顺序发射
- ▶ 两条不同类型指令有依赖：添加SYNC同步指令



控制模块

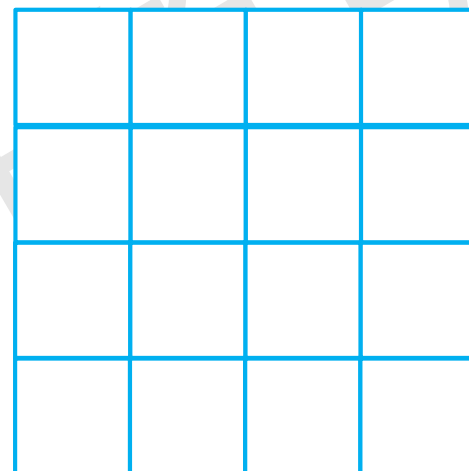
▶ 指令发射队列IQ

- ▶ 三个指令队列乱序发射，指令队列内顺序发射
- ▶ 两条同类型指令有依赖：位于同一发射队列顺序发射
- ▶ 两条不同类型指令有依赖：添加SYNC同步指令

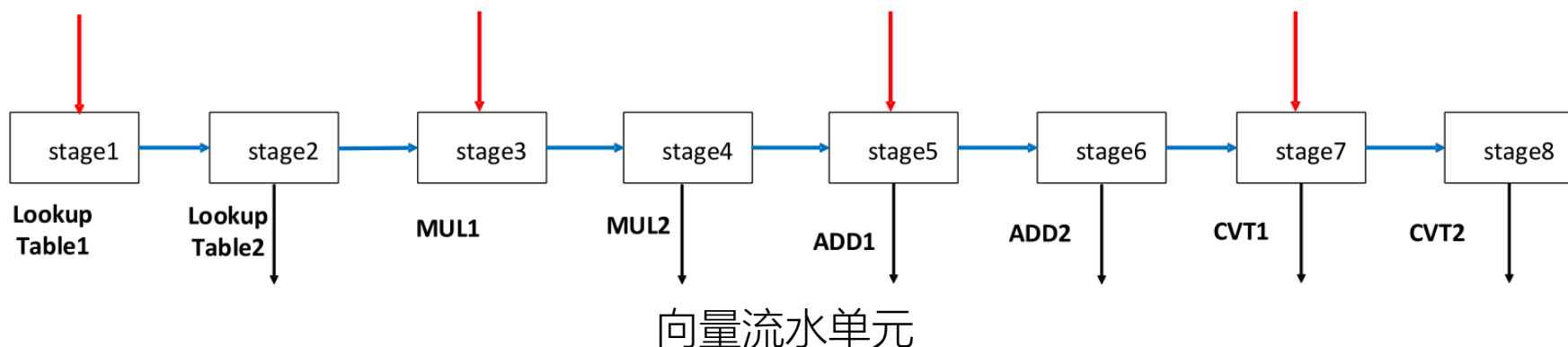


运算模块VFU

- ▶ 完成输入神经元的前处理和输出神经元的后处理
- ▶ 包括向量流水单元和转置单元
- ▶ 向量流水单元承载向量运算功能
- ▶ 转置单元承载数据重新摆放功能



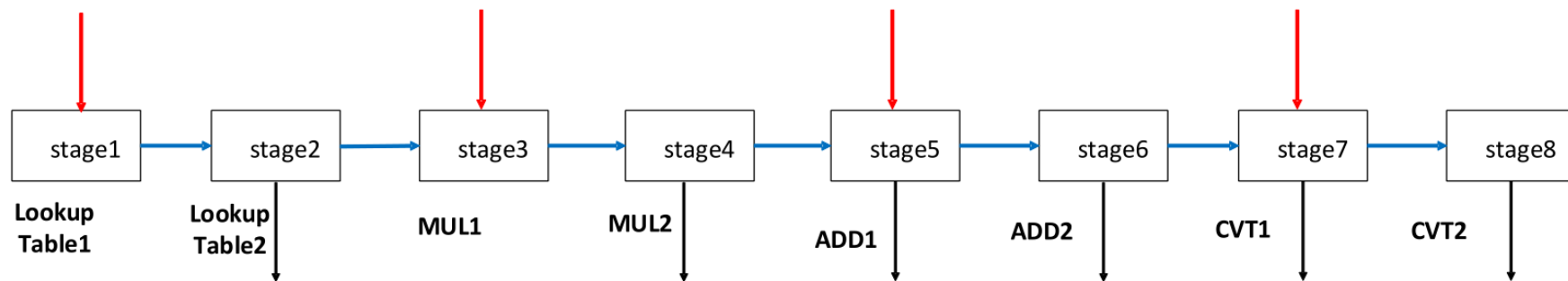
转置单元



向量流水单元

▶ 向量流水单元

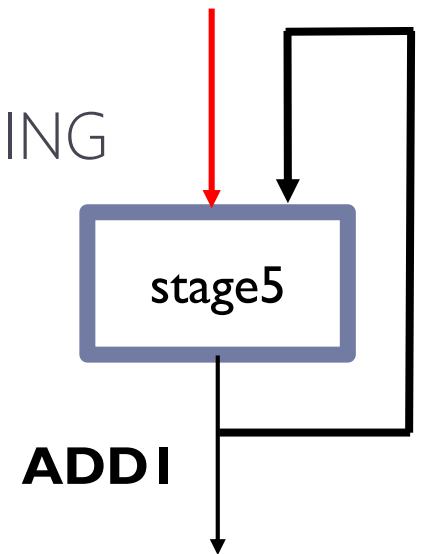
- ▶ 多种数据类型：INT8、INT16、INT32、FP16 和 FP32
- ▶ 新增运算：查表、边缘扩充、数据格式转换等
- ▶ 多个stage可以输入，多个stage可以输出
- ▶ 从stage1到stage8串联可以完成激活+转数的操作



向量流水单元

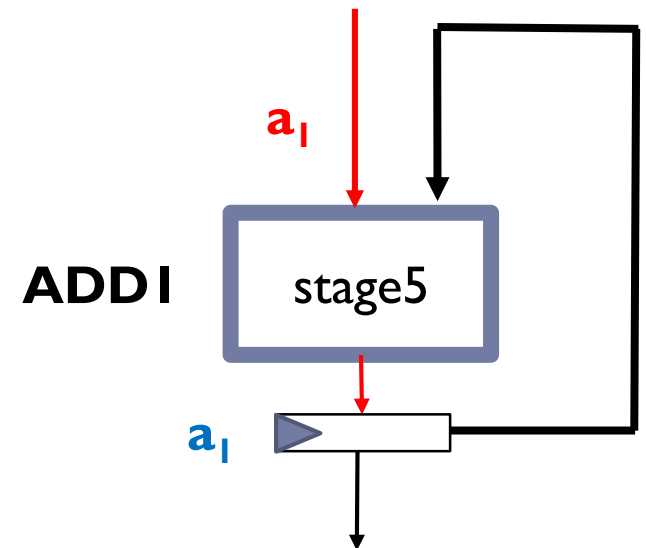
向量流水单元如何完成AVGPOOLING

- ▶ 当输入数据类型是INT型时
 - ▶ AVGPOOLING本质是 $k_x * k_y$ 个向量的累加 (k_x 和 k_y 是pooling核大小)
 - ▶ INT型的加法延迟是1个cycle
 - ▶ 使用stage5即可完成INT数据类型的AVGPOOLING



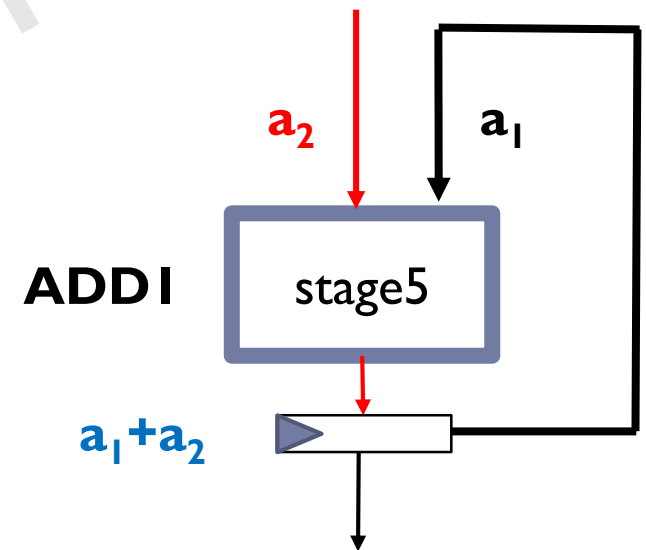
向量流水单元如何完成AVGPOOLING

- ▶ 当输入数据类型是INT型时
 - ▶ 步骤1: 输入向量1 BYPASS stage5后进入stage5输出寄存器



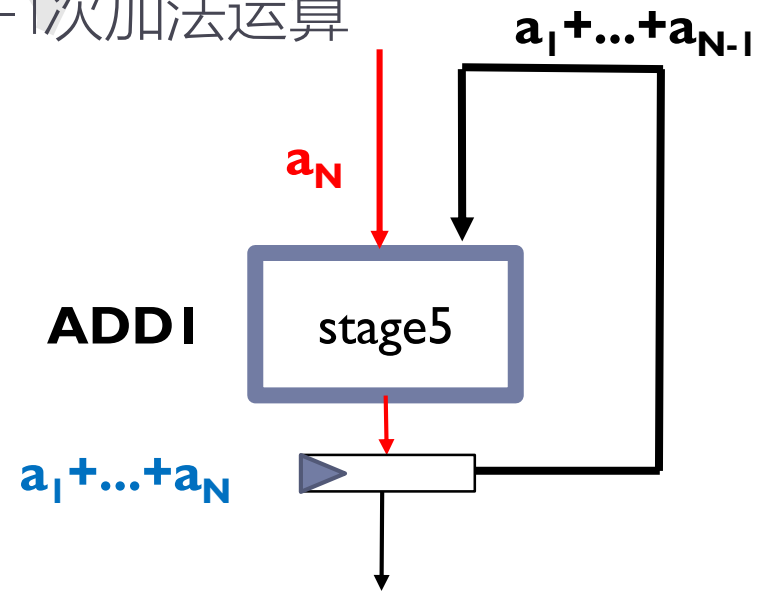
向量流水单元如何完成AVGPOOLING

- ▶ 当输入数据类型是INT型时
 - ▶ 步骤1: 输入向量1 BYPASS stage5后进入stage5输出寄存器
 - ▶ 步骤2: 输入向量2与输入向量1经过stage5的定点加法器, 完成加法运算后, 写入stage5输出寄存器



向量流水单元如何完成AVGPOOLING

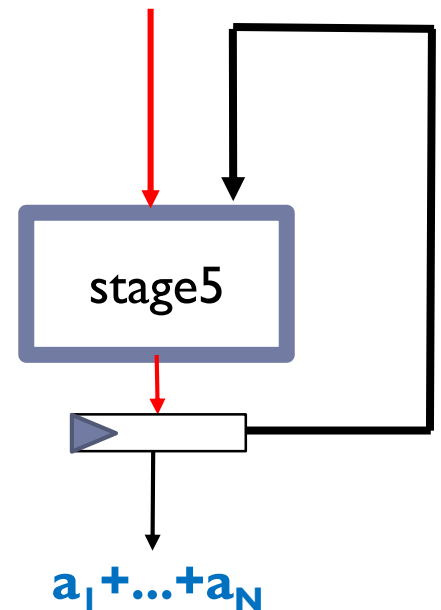
- ▶ 当输入数据类型是INT型时
 - ▶ 步骤1: 输入向量1 BYPASS stage5后进入stage5输出寄存器
 - ▶ 步骤2: 输入向量2与输入向量1经过stage5的定点加法器, 完成加法运算后, 写入stage5输出寄存器
 - ▶ 步骤3: 重复步骤2直至完成 $k_x * k_y - 1$ 次加法运算



向量流水单元如何完成AVGPOOLING

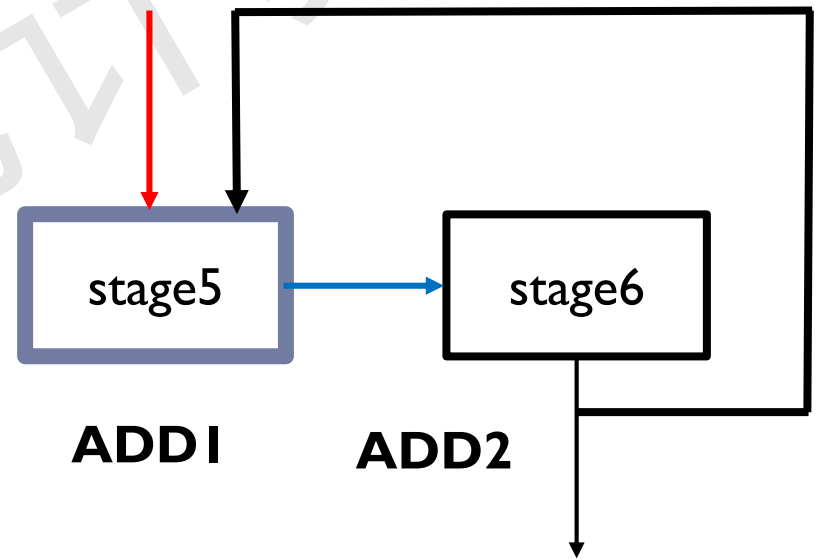
- ▶ 当输入数据类型是INT型时
 - ▶ 步骤1: 输入向量1 BYPASS stage5后进入stage5输出寄存器
 - ▶ 步骤2: 输入向量2与输入向量1经过stage5的定点加法器, 完成加法运算后, 写入stage5输出寄存器
 - ▶ 步骤3: 重复步骤2直至完成 $kx*ky-1$ 次加法运算
 - ▶ 步骤4: 将结果从stage5的输出寄存器输出
 - ▶ 每个步骤是1个cycle, 所以完成AVGPOOLING需要 $kx*ky$ 个cycle

ADDI



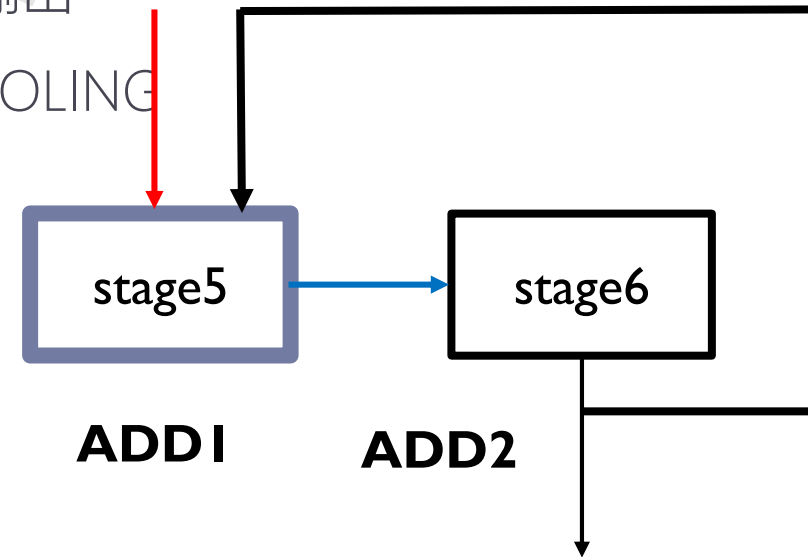
向量流水单元如何完成AVGPOOLING

- ▶ 当输入数据类型是FLOAT型时
 - ▶ FLOAT型的加法延迟是2个cycle
 - ▶ 使用stage5和stage6即可完成FLOAT数据类型的AVGPOOLING



向量流水单元如何完成AVGPOOLING

- ▶ 当输入数据类型是FLOAT型时
 - ▶ 步骤1: 输入向量1 BYPASS stage5和stage6后进入stage6输出寄存器
 - ▶ 步骤2: 输入向量2与输入向量1经过stage5和stage6的浮点加法器, 完成加法运算后, 写入stage6输出寄存器
 - ▶ 步骤3: 重复步骤2直至完成 $kx*ky-1$ 次加法运算
 - ▶ 步骤4: 将结果从stage6的输出寄存器输出
 - ▶ 每个步骤是2个cycle, 所以完成AVGPOOLING需要 $2*kx*ky$ 个cycle



扩展和思考

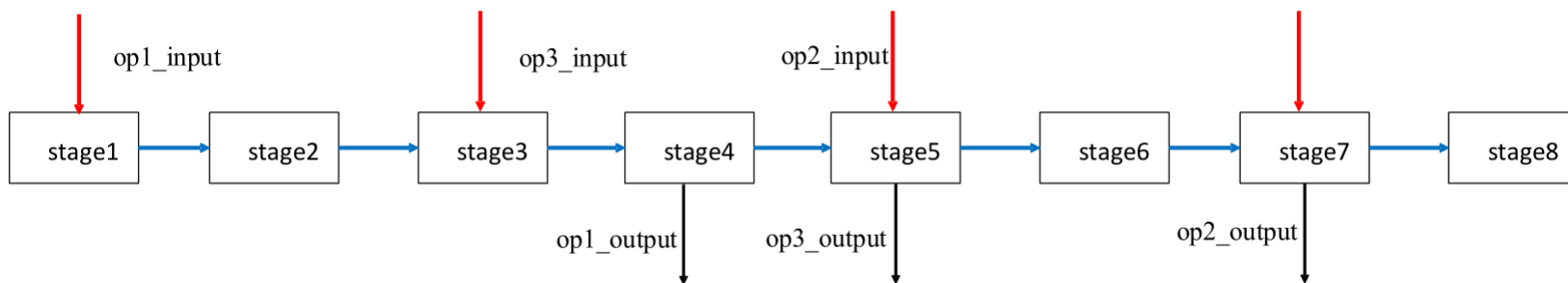
- ▶ 需要增加什么逻辑单元，使VFU能够完成MAXPOOLING?

中科院计算所

向量流水单元超车问题

▶ VFU：向量运算单元

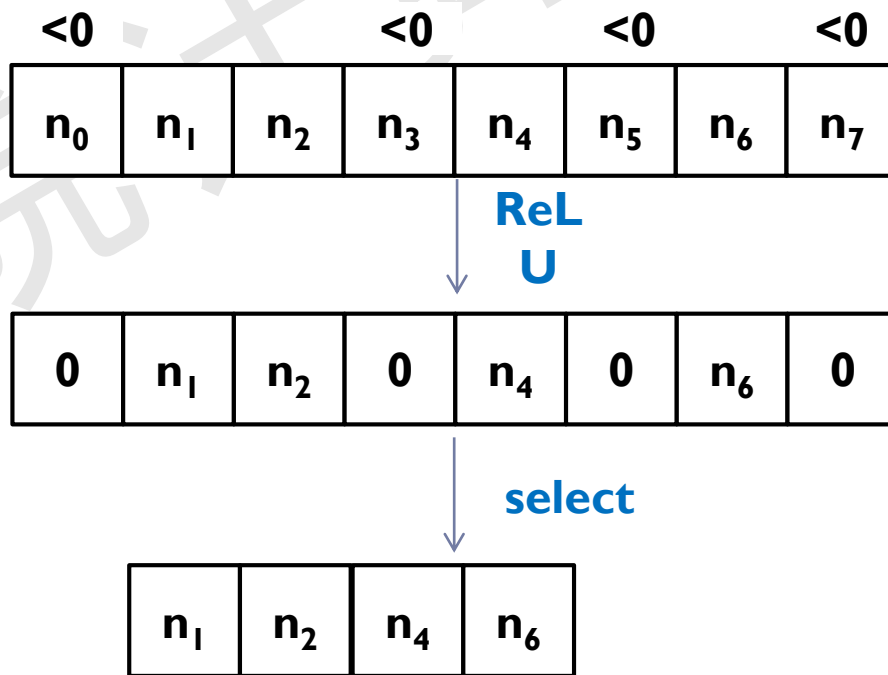
- ▶ 向量超车：后执行的运算指令比先执行的指令执行的更快
- ▶ 规则：
 - ▶ 指令从第 n 个stage进入，需要等待stage 1到stage n 的输入寄存器清空
 - ▶ 指令从第 m 个stage输出，需要等待stage $(m+1)$ 后所有输入寄存器清空



向量运算指令超车

向量流水单元完成神经元压缩

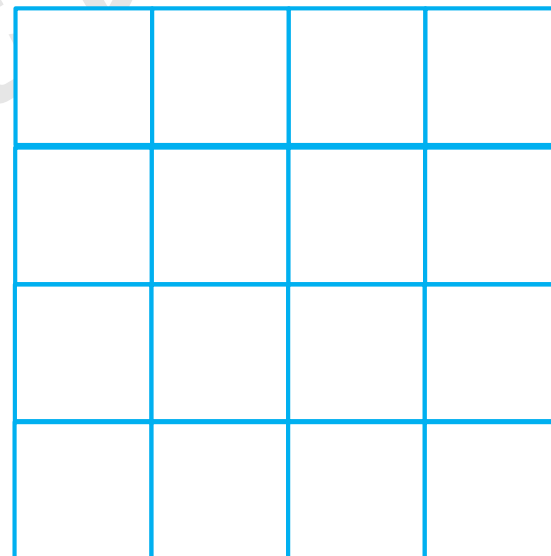
- ▶ 神经元经过ReLU激活层后会产生很多0
- ▶ 如何将这0进行过滤，生成稠密的神经元？
- ▶ 从而减少数据搬运，减少能耗
- ▶ 还能够应用于快排等算法



转置单元

▶ 转置单元

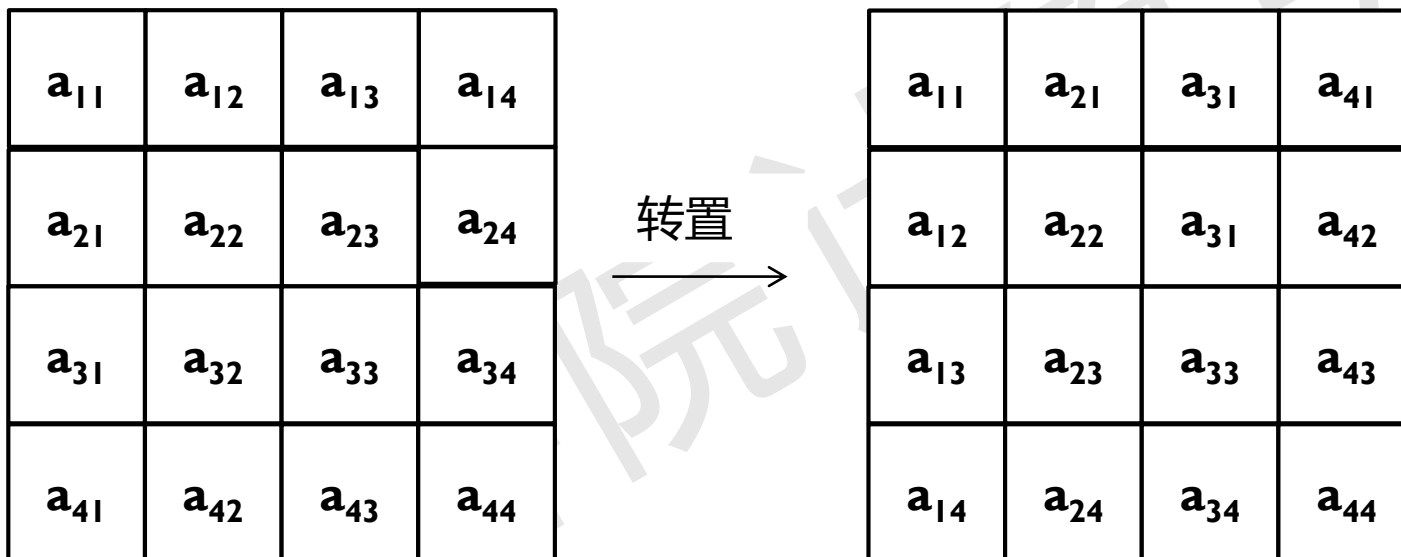
- ▶ 支持多种数据类型，包括INT8、INT16、INT32、FP16和FP32
- ▶ 功能包括转置、镜像、旋转等。
- ▶ 主要由一个数据缓存和读写控制逻辑组成
- ▶ 读写控制逻辑能够对数据缓存进行多种模式的读写



转置单元

转置操作

▶ 转置操作

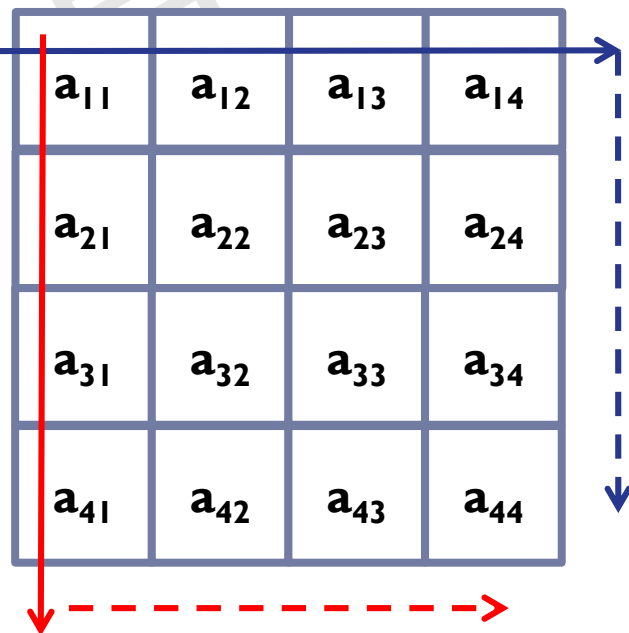


转置单元实现转置操作

▶ 转置单元实现转置操作

- ▶ 步骤1: 从第一行开始, 将数据从左向右写入缓存, 直至填满数据缓存
- ▶ 步骤2: 从第一列开始, 将数据从上到下依次读出, 直至读完缓存中的所有数据

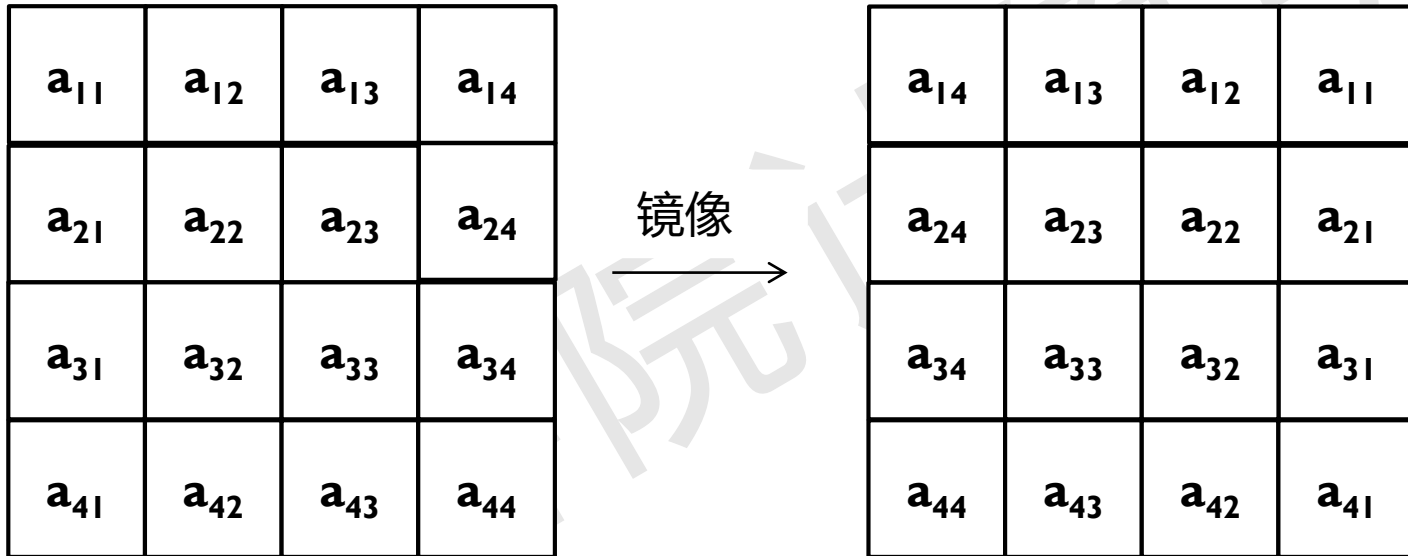
横向写入



纵向读出

镜像操作

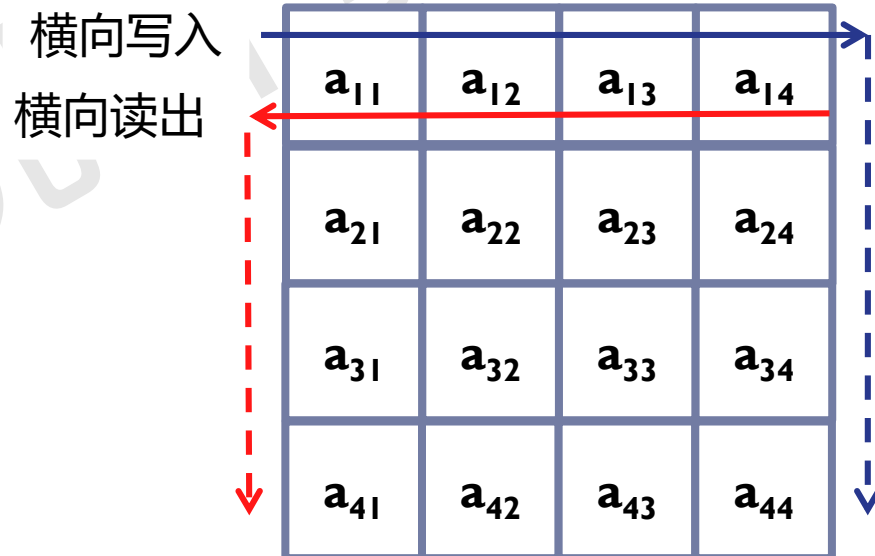
▶ 镜像操作



转置单元实现镜像操作

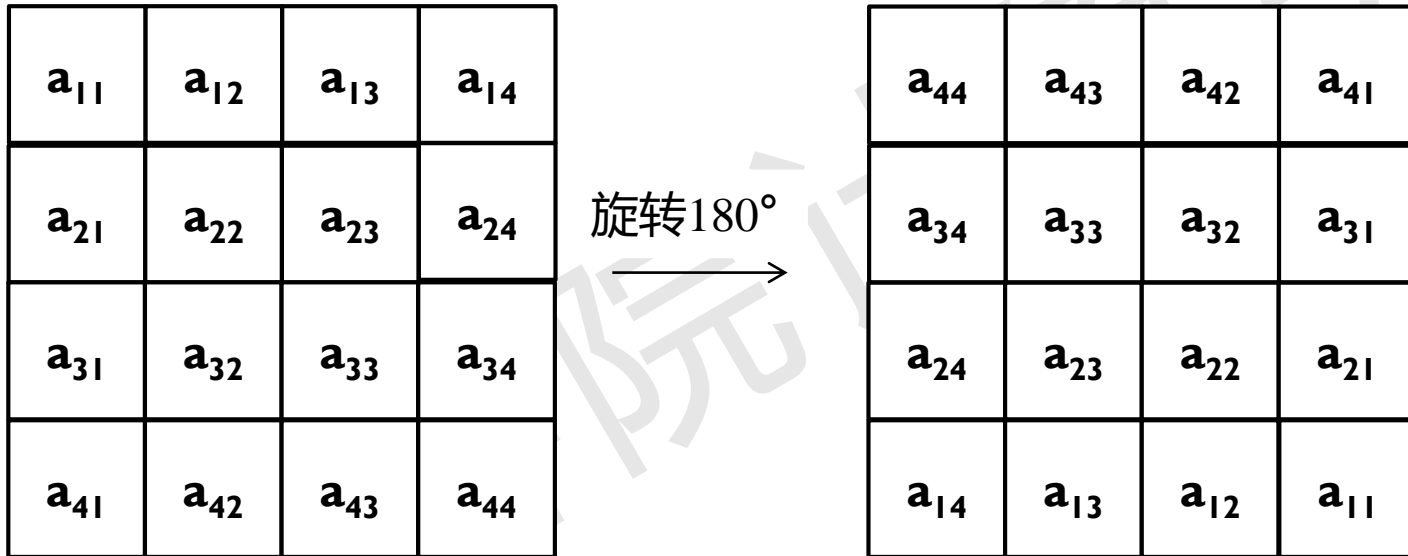
▶ 转置单元实现镜像操作

- ▶ 步骤1: 从第一行开始, 将数据从左向右写入缓存
- ▶ 步骤2: 从第一行开始, 将数据从右向左依次读出



旋转180°操作

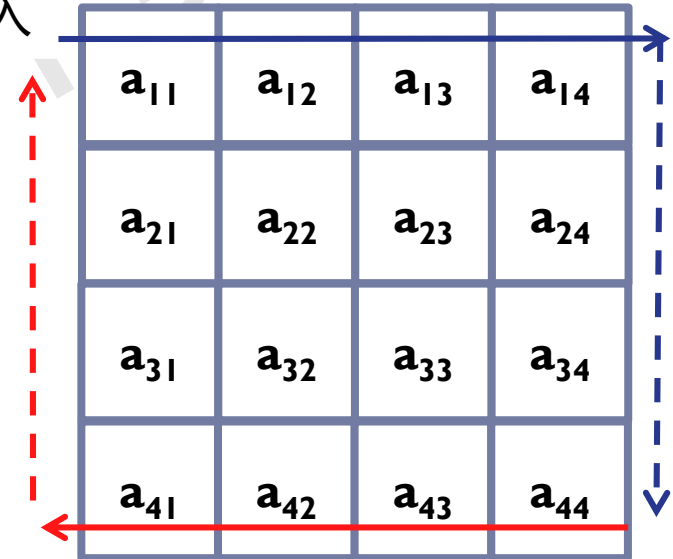
▶ 旋转180°操作



转置单元实现旋转180°操作

- ▶ 转置单元实现旋转180°操作
 - ▶ 步骤1: 从第一行开始, 将数据从左向右写入缓存
 - ▶ 步骤2: 从最后一行开始, 将数据从右向左依次读出

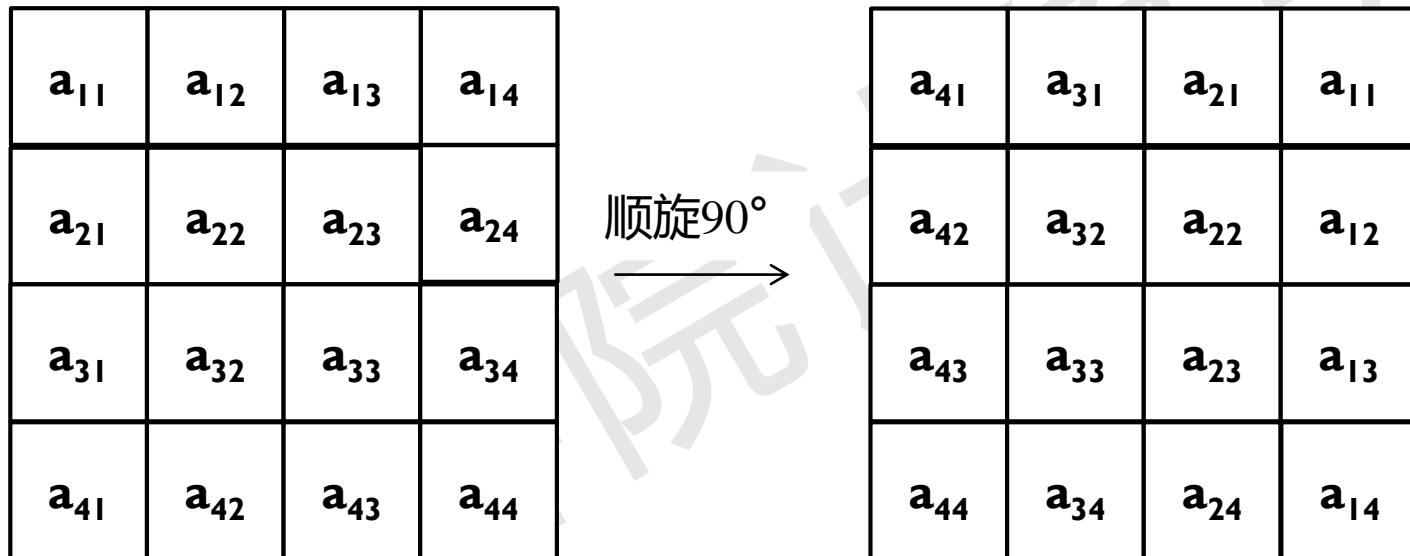
横向写入



横向读出

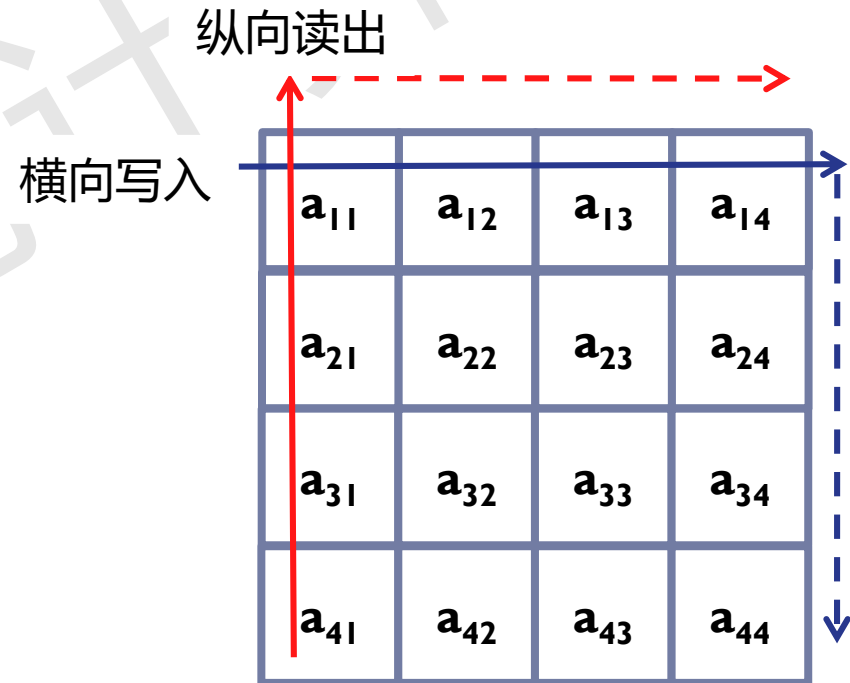
顺时针旋转90°操作

▶ 顺时针旋转90°操作



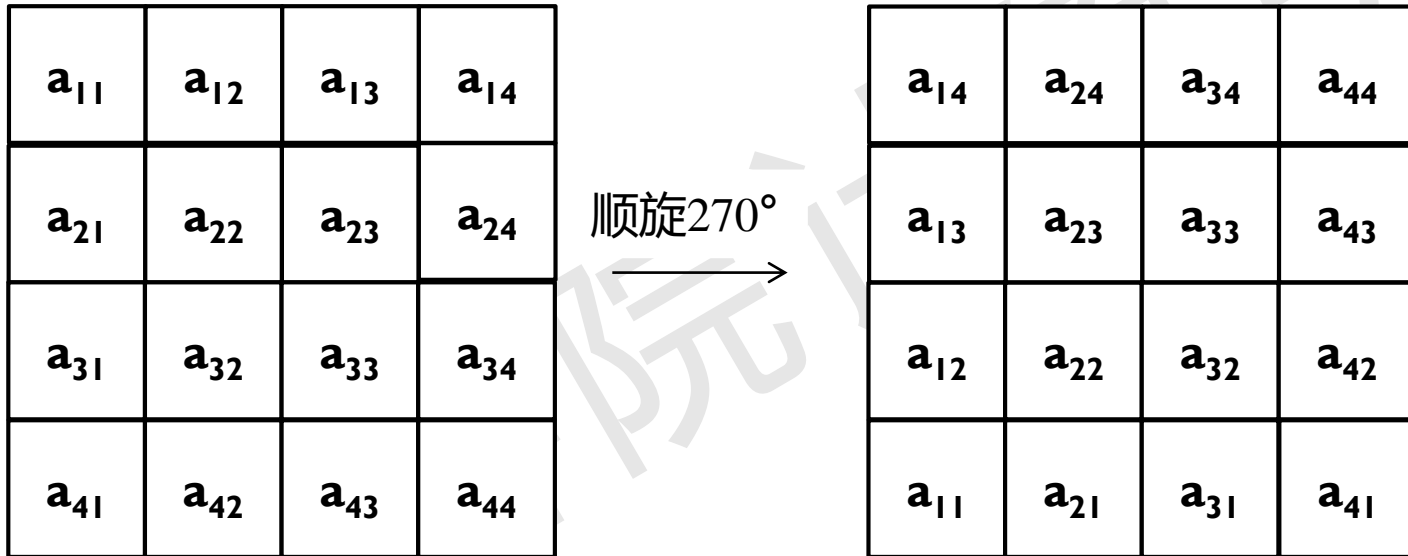
转置单元实现顺时针旋转90°操作

- ▶ 转置单元实现顺时针旋转90°操作
 - ▶ 步骤1: 从第一行开始, 将数据从左向右写入缓存
 - ▶ 步骤2: 从第一列开始, 将数据从下至上依次读出



顺时针旋转270°操作

▶ 顺时针旋转270°操作

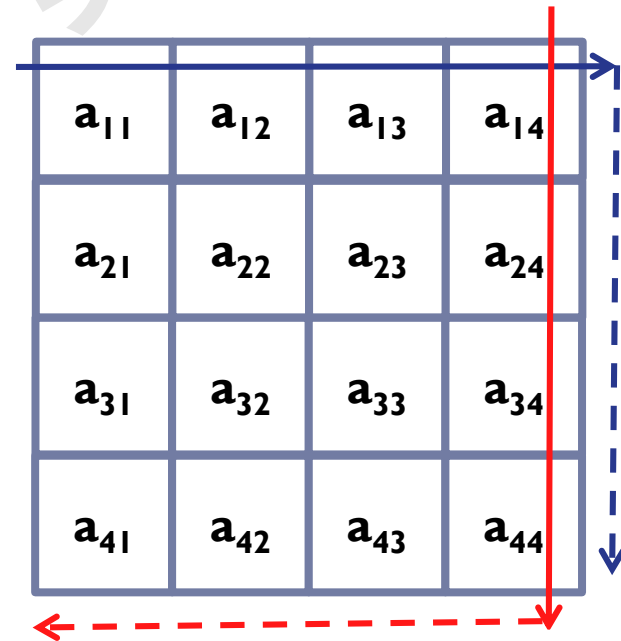


转置单元实现顺时针旋转270°操作

▶ 转置单元实现顺时针旋转270°操作

- ▶ 步骤1: 从第一行开始, 将数据从左向右写入缓存
- ▶ 步骤2: 从最后一列开始, 将数据从上至下依次读出

横向写入



纵向读出

运算模块

- ▶ MFU：矩阵运算单元（分布式设计）

- ▶ H-tree互联（广播神经元）

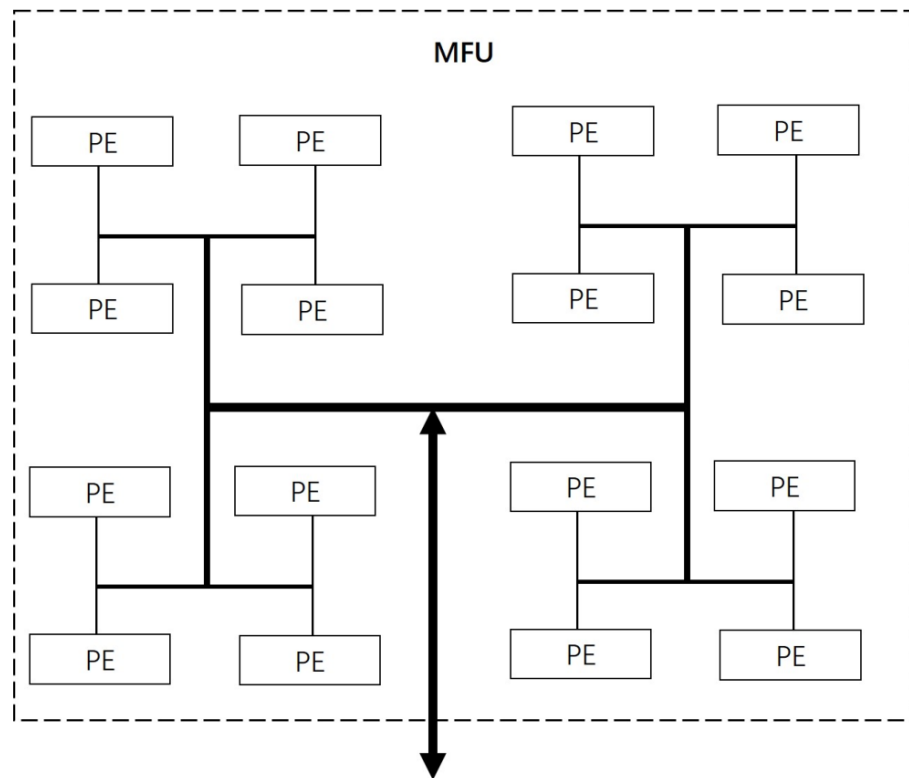
- ▶ 低位宽定点运算器

- ▶ 三种模式：

- ▶ $\text{int16} \times \text{int16}$

- ▶ $\text{int8} \times \text{int8}$

- ▶ $\text{int8} \times \text{int4}$



存储单元

- ▶ 存储管理
 - ▶ NRAM, WRAM, DMA
 - ▶ 虚拟存储：片内片外统一编址
 - ▶ 片内无需虚实地址转换
 - ▶ 片内外需要虚实地址转换

中科院计算所

存储单元

- ▶ 降低访问延迟

- ▶ TLB

- ▶ 缓存常用页表

- ▶ LLC

- ▶ 缓存经常访问的 DRAM 数据

- ▶ 降低访存量

- ▶ 稀疏化存储

- ▶ 数据压缩

DLP-S总结

- ▶ 终端智能应用
- ▶ 控制：基于tensor语义进行设计专用指令
- ▶ 计算：基于tensor进行运算操作。流水排布，转置
- ▶ 存储：基于tensor进行数据搬运。稀疏，压缩

提纲

- ▶ 单核深度学习处理器 (DLP-S)
- ▶ 多核深度学习处理器 (DLP-M)
 - ▶ 由于工艺/面积受限，提升单核处理器性能会使得能耗快速升高，能效比下降
 - ▶ 需要解决核间同步，核间通讯，拓扑结构，任务划分等问题

多核深度学习处理器 (DLP-M)

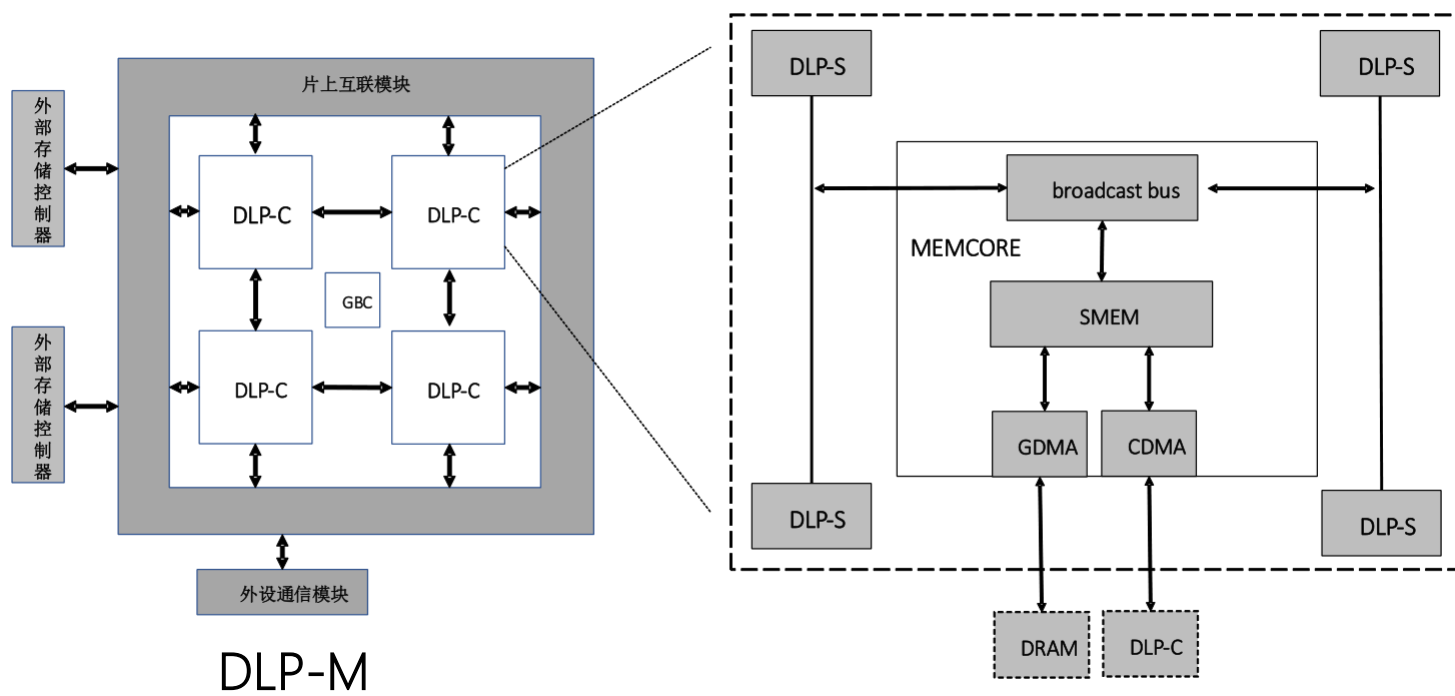
- ▶ 总体架构
- ▶ Cluster架构
- ▶ 互联结构

中科院计算所

总体架构

多核处理器分层结构设计

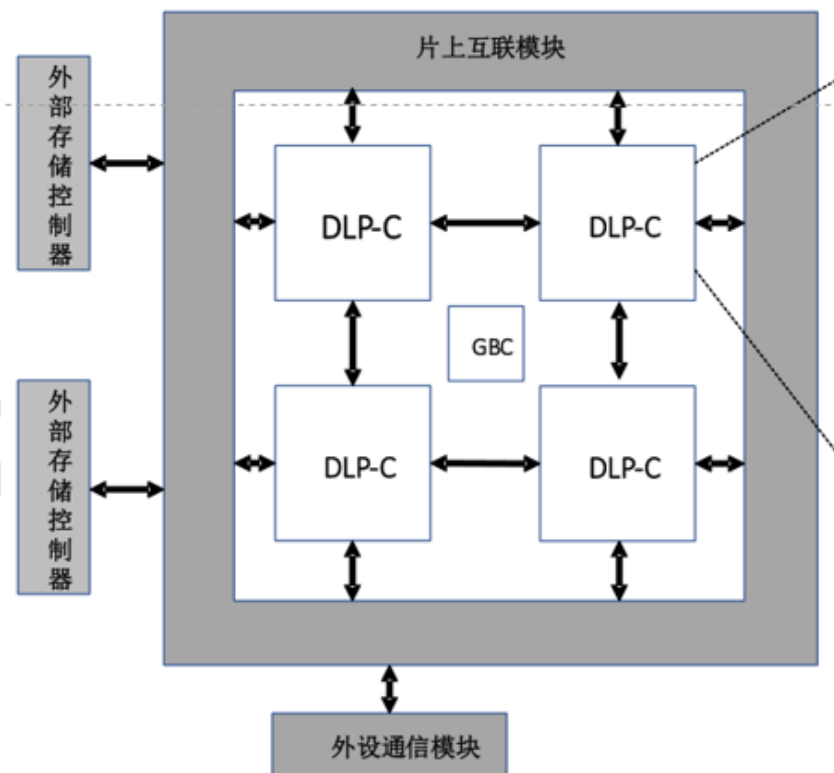
- ▶ 一个DLP-M由多个DLP-C构成
- ▶ 一个DLP-C由多个DLP-S构成
- ▶ 为什么需要进行分层结构设计-减少NoC的负载核开销



总体架构

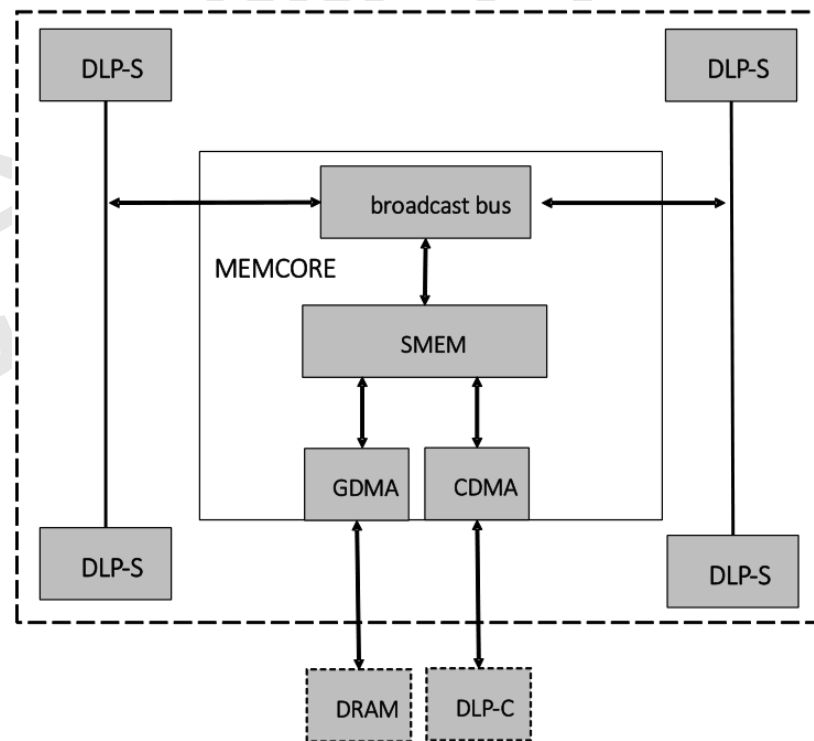
▶ DLP-M

- ▶ 外部存储控制器
- ▶ 外设通信模块
- ▶ 片上互联模块
- ▶ 同步模块GBC (Global Barrier Controller)
- ▶ 四个DLP-C



总体架构

- ▶ DLP-C
 - ▶ 四个DLP-S
 - ▶ 存储核MEMCORE (Memory Core)
 - ▶ 存储: DLP-S共享数据
 - ▶ 通信: DLP-C与片外DRAM, DLP-C之间, 多个DLP-S之间



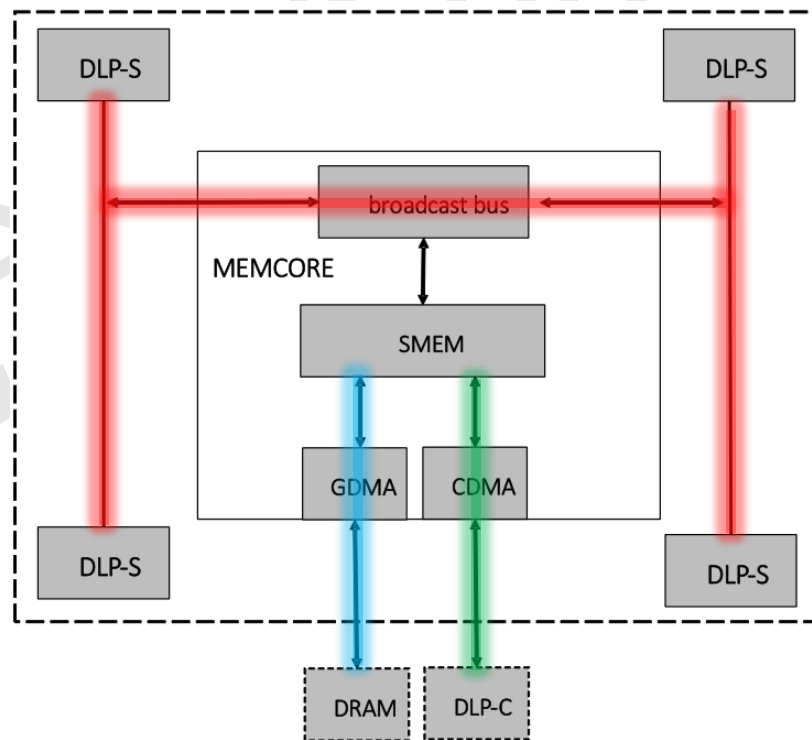
Cluster架构

MEMCORE

- ▶ 共享存储模块SMEM (Shared Memory)
- ▶ 广播总线 (Broadcast Bus)
- ▶ Cluster 直接内存访问CDMA (Cluster Direct Memory Access)
- ▶ 全局直接内存访问GDMA (Global Direct Memory Access)

存储: DLP-S共享数据

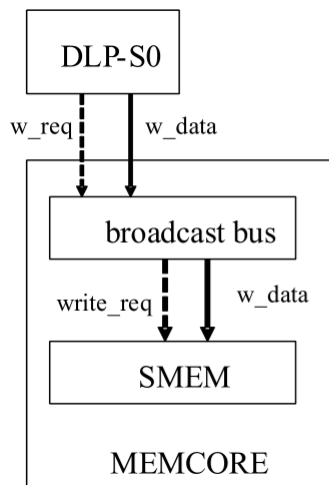
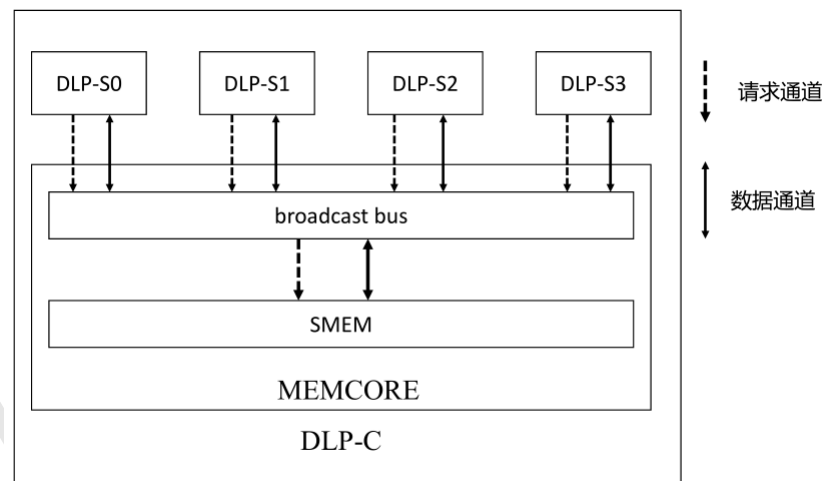
通信: DLP-C与片外DRAM
DLP-C之间,
多个DLP-S之间



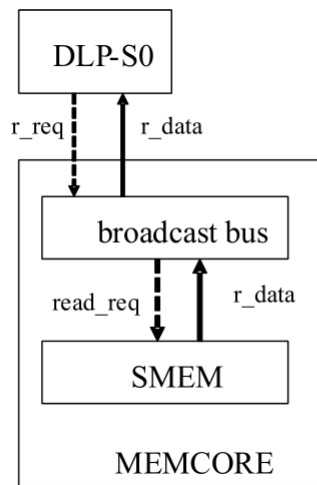
Cluster架构

广播总线

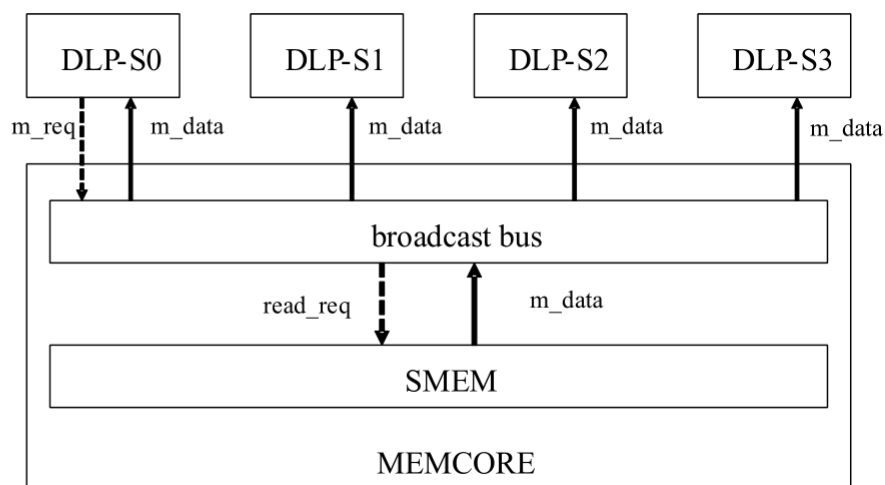
- 深度学习数据的复用特性
- 读写请求：单播写、单播读，多播



单播写



单播读



多播

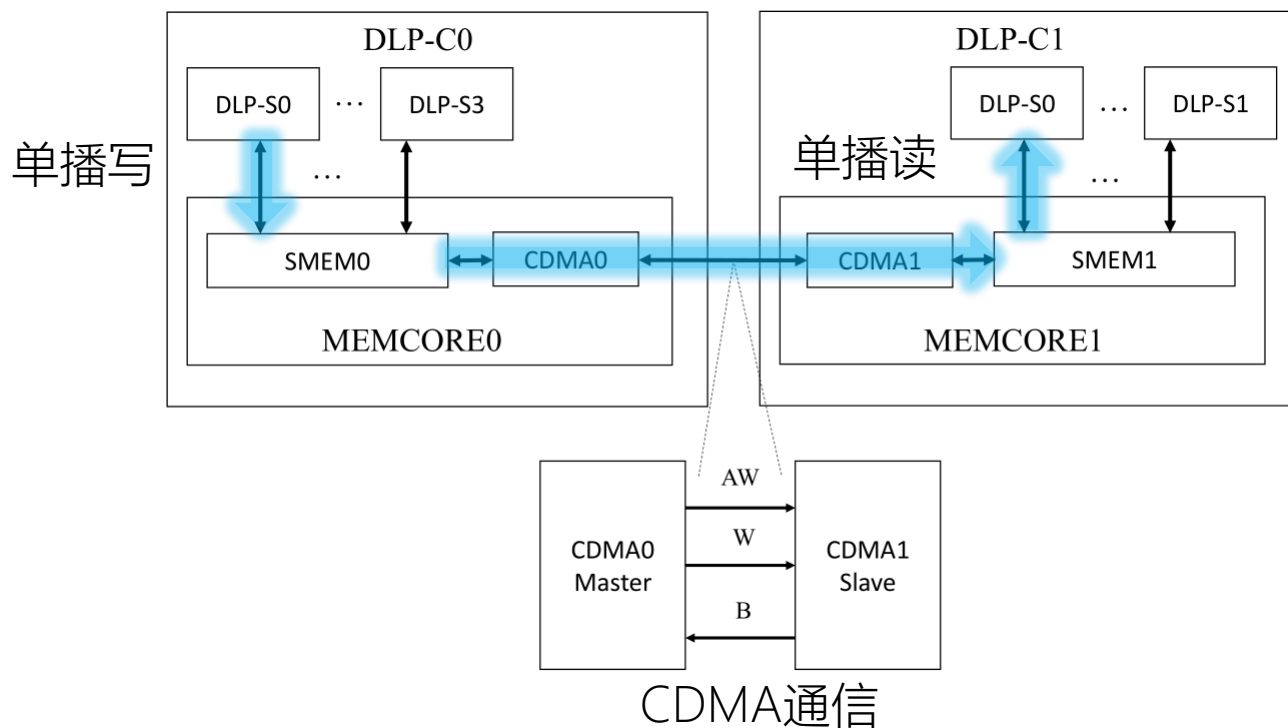
Cluster架构

- ▶ 多播的应用：大规模卷积运算
 - ▶ DLP-S中NRAM中存入不同输入数据，权重位于片外DRAM中，需要将权重加载至所有DLP-S的WRAM中
 - ▶ 执行过程：
 - ▶ 访存指令：通过DMA载入权重至SMEM中
 - ▶ 广播指令：广播总线广播权重至所有DLP-S中
 - ▶ 计算指令：DLP-S执行卷积运算
 - ▶ 如果没有 SMEM，则相同的权重数据需要重复 4 次从 DRAM 读出，片外访存的数据量变为原来的 4 倍
 - ▶ 如果使用 SMEM 但不使用广播，则相同的权重数据也需要重复 4 次从 SMEM 读出，对 SMEM 访存的数据总量变为原来的 4 倍

Cluster架构

▶ CDMA

- ▶ 执行过程：单播写，CDMA通信，单播读
- ▶ 访存指令：目标Cluster号，源地址，目的地址，数据大小



Cluster架构

▶ GDMA

- ▶ 每个 DLP-C 可能对应多个 DRAM 控制器，因此 GDMA 发出的请求地址需要进行路由；
- ▶ GDMA 发出的请求地址是虚地址，需要使用 MMU 进行虚实地址转换；
- ▶ GDMA 会利用 TLB 加速虚拟地址到物理地址的转换；
- ▶ GDMA 会利用 LLC 缩短片外访存的平均延时。

Cluster架构

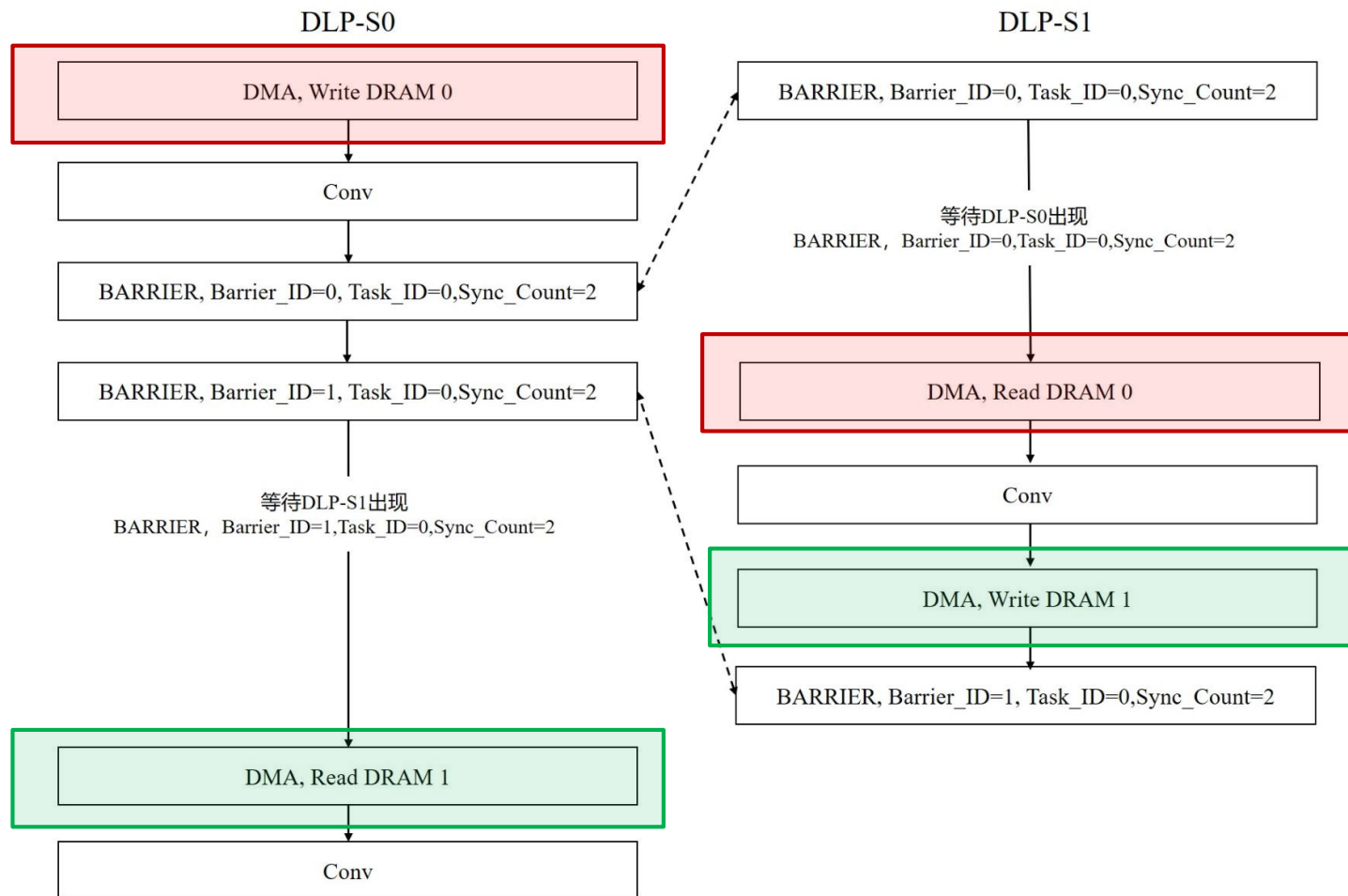
▶ 多核同步模型

- ▶ BARRIER指令：多核同步指令，解决访存冲突
 - ▶ BARRIER: Opcode
 - ▶ Barrier_ID: Barrier序号
 - ▶ Task_ID: 同步的任务编号（同一个任务才需要同步）
 - ▶ Sync_Count: 需要同步的Barrier个数

BARRIER	Barrier_ID	Task_ID	Sync_Count
---------	------------	---------	------------

Cluster架构

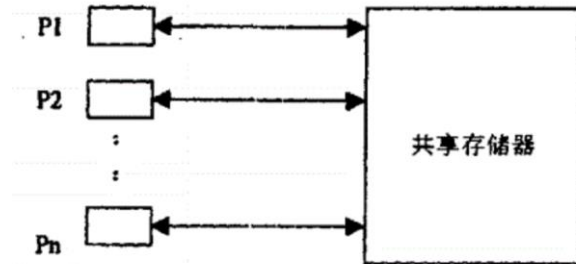
多核同步模型：双核协同指令流



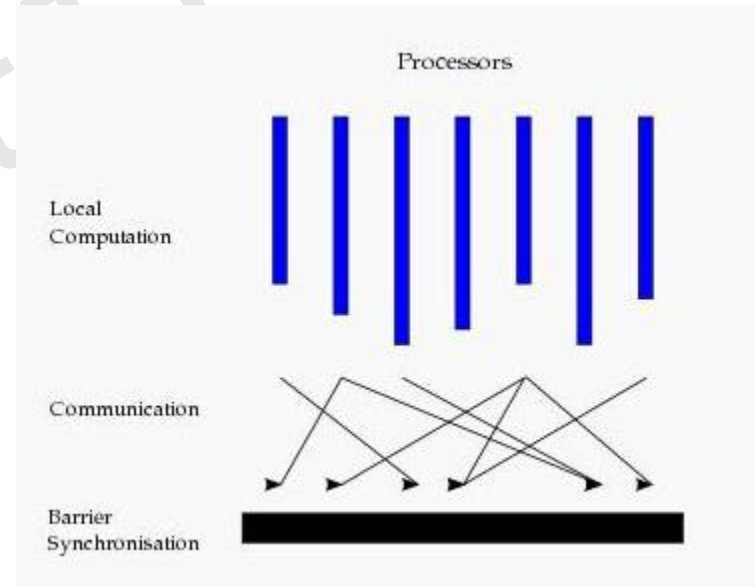
互联架构

▶ 多核协同

- ▶ 数据共享来减少对片外 DRAM 的访问
- ▶ 提高处理单个任务时的计算能力
- ▶ 需要实时数据交互



PRAM

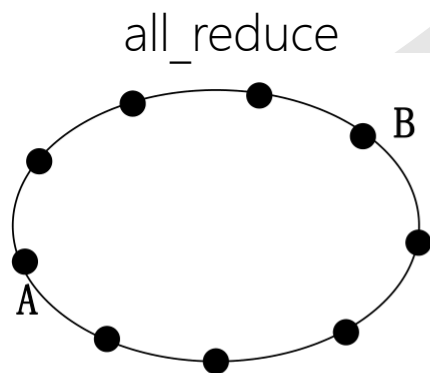


BSP

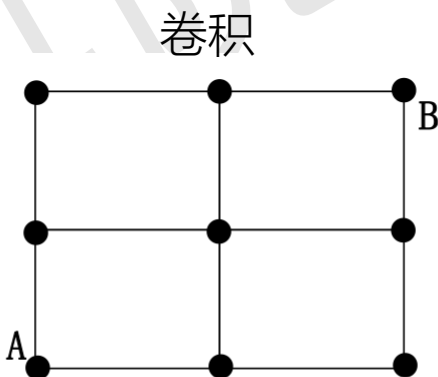
互联架构

核间互联拓扑结构

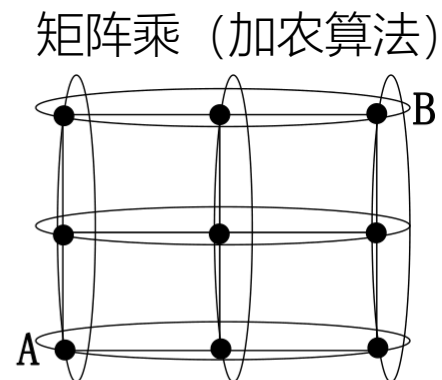
- 不同的核到同一个核的延时相同。这样可以提供所有核完全对等的编程模型，方便软件编写和性能优化，另一方面也使得多核系统在调度时可以做任意的任务分配。
- 核间的互联通路尽量稠密。这样可以减少单个通路的负载，同时降低访问延时。理论上，只有多核之间对称的全连接拓扑才能完全满足上述要求。



(a) 环形



(b) 网状



(c) Torus

互联架构

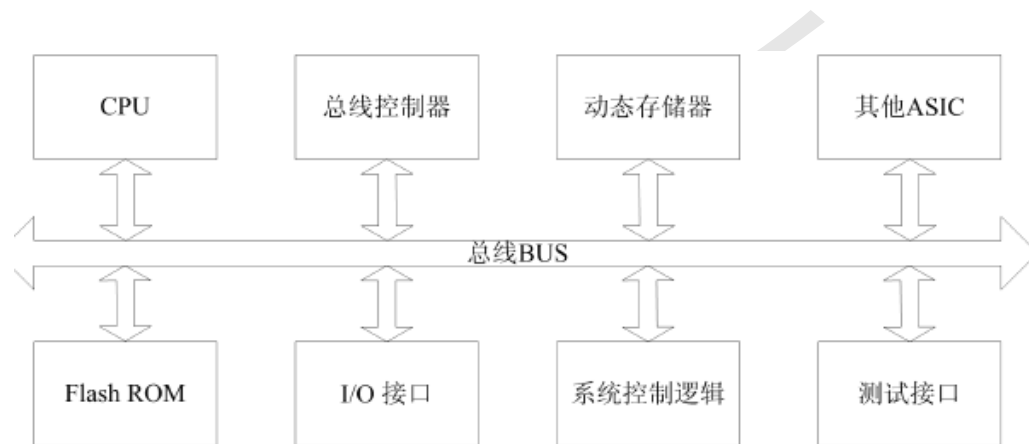
▶ 互联方式

▶ 总线互联

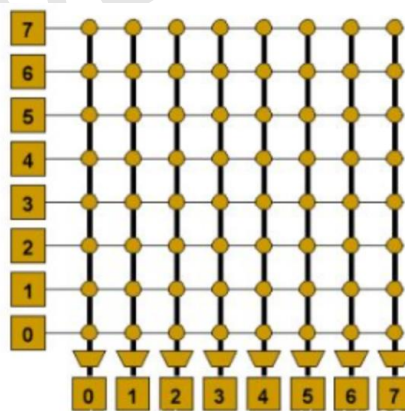
- ▶ 公共数据干线
- ▶ 扩展性差

▶ 片上网络

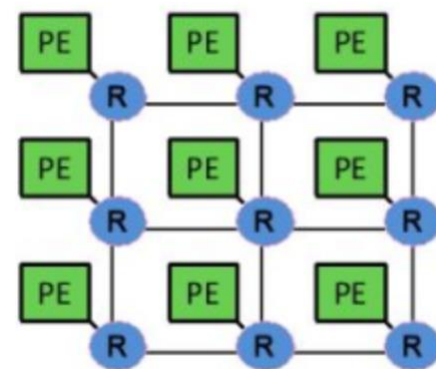
- ▶ 片上互联
- ▶ 性能和功耗有优势
- ▶ 扩展性好



总线互联



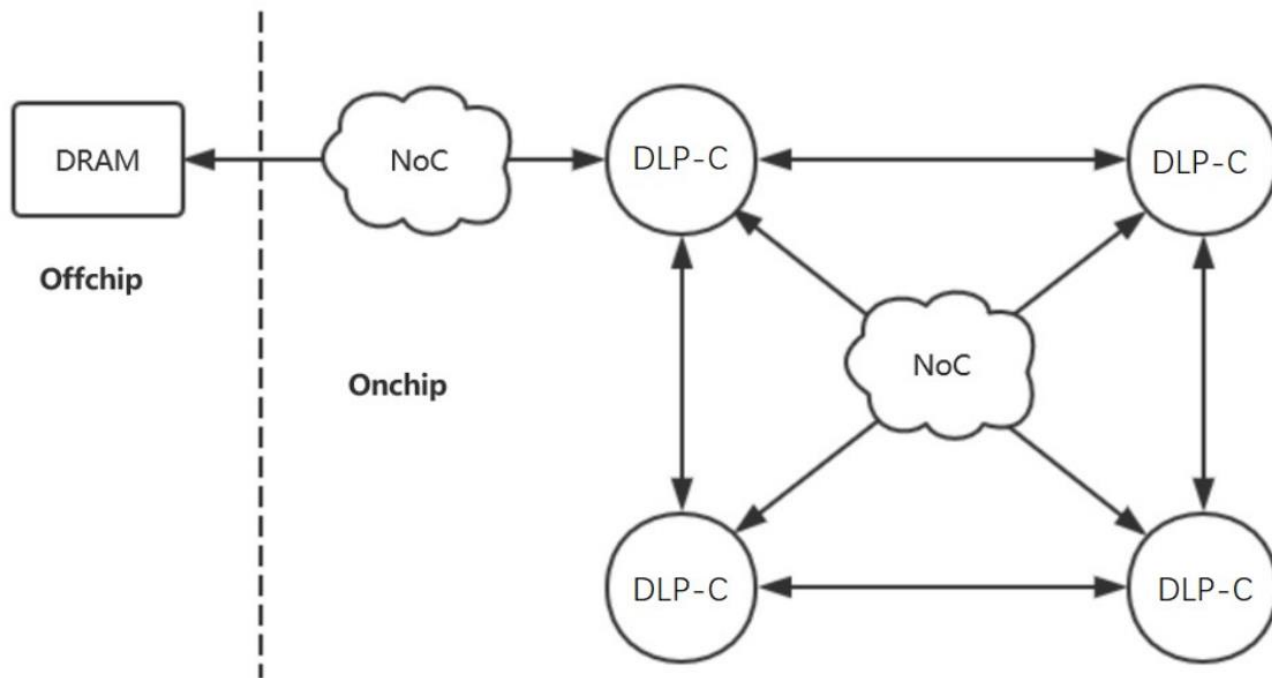
Crossbar



NoC

互联架构

▶ DLP-C互联



多核深度学习处理器 (DLP-M)

- ▶ 云端智能领域应用
- ▶ 分层结构
 - ▶ Chip级->Cluster-级>Core级
- ▶ 通信模型
 - ▶ MEMCORE
 - ▶ Cluster互联架构



谢谢大家!

中科院计算所