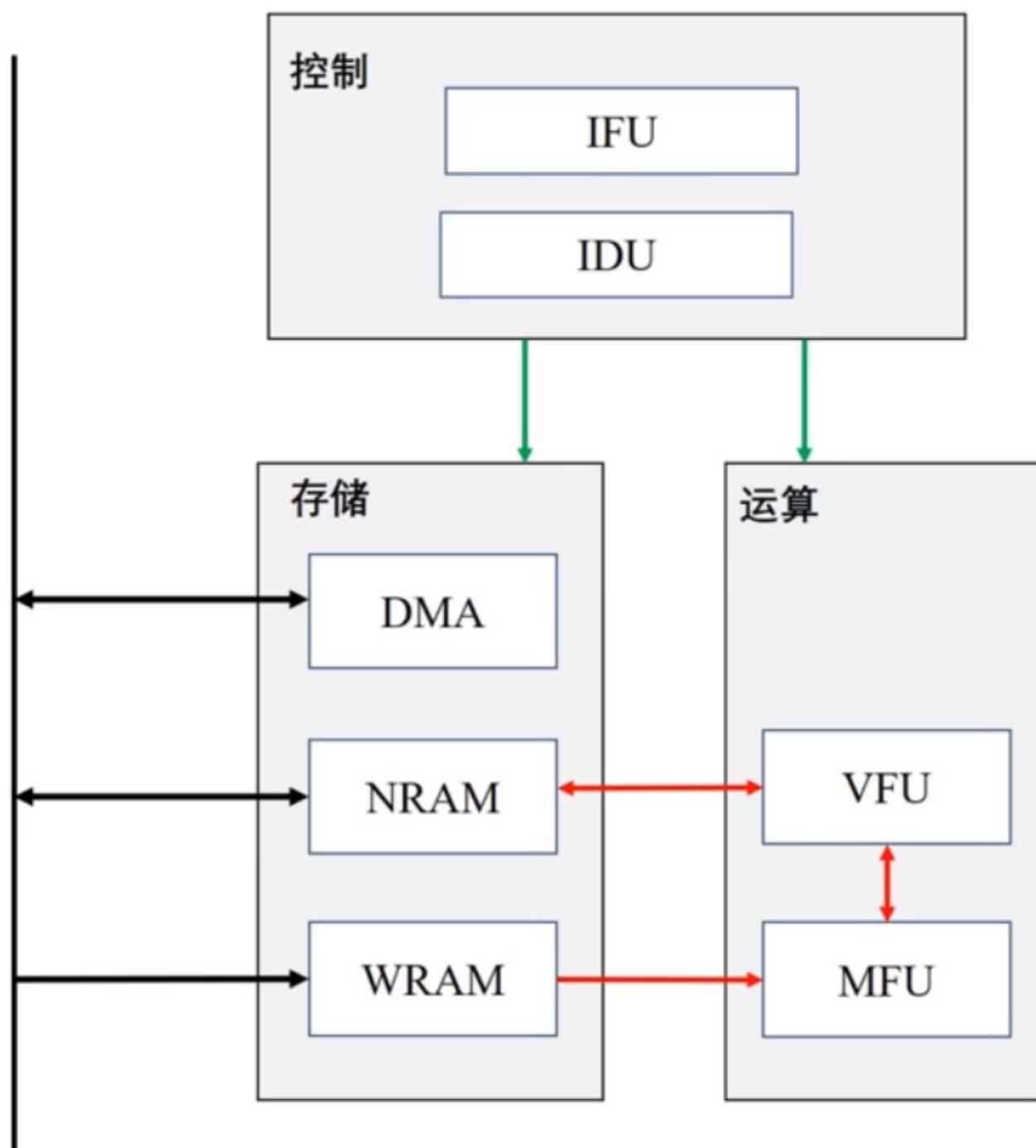


# Chapter 7 深度学习处理器架构

Revision: 1

## 7.1 单核深度学习处理器(DLP-S)



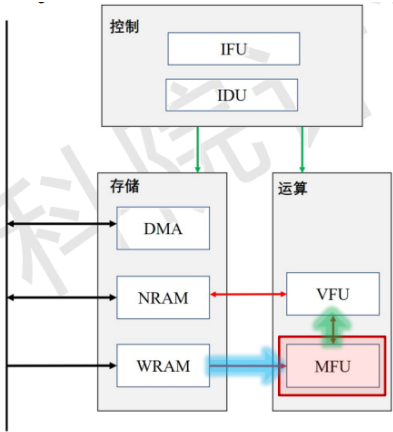
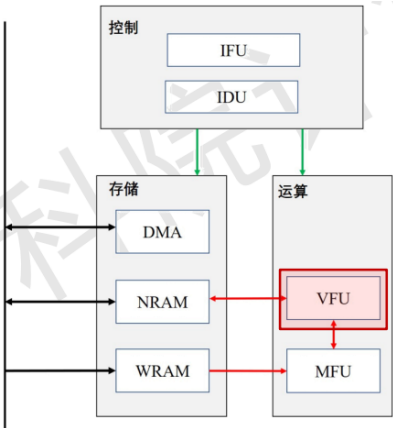
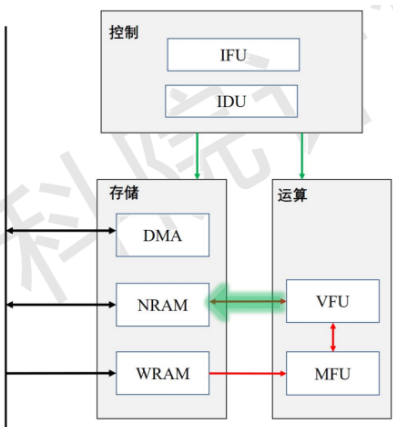
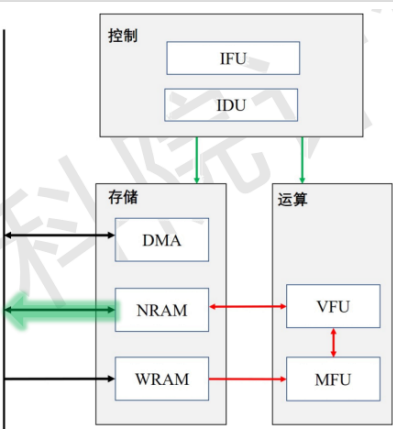
从DLP到DLP-S：

- 控制模块
  - 多发射队列，支持指令级并行；

- 运算模块
  - 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，提高执行能效；
  - 稀疏运算，提高计算效率；
- 存储单元
  - 稀疏数据的稠密化访存，降低开销；
  - 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；

## 执行流程

步骤	描述
	IFU通过DMA从DRAM中读取程序指令，经IDU进行译码后分发给DMA、VFU和MFU
	DMA接收到指令后从DRAM读取神经元至NRAM，读取权重至WRAM
	VFU接收到指令后从NRAM中读取神经元数据，并进行预处理，然后发送至MFU

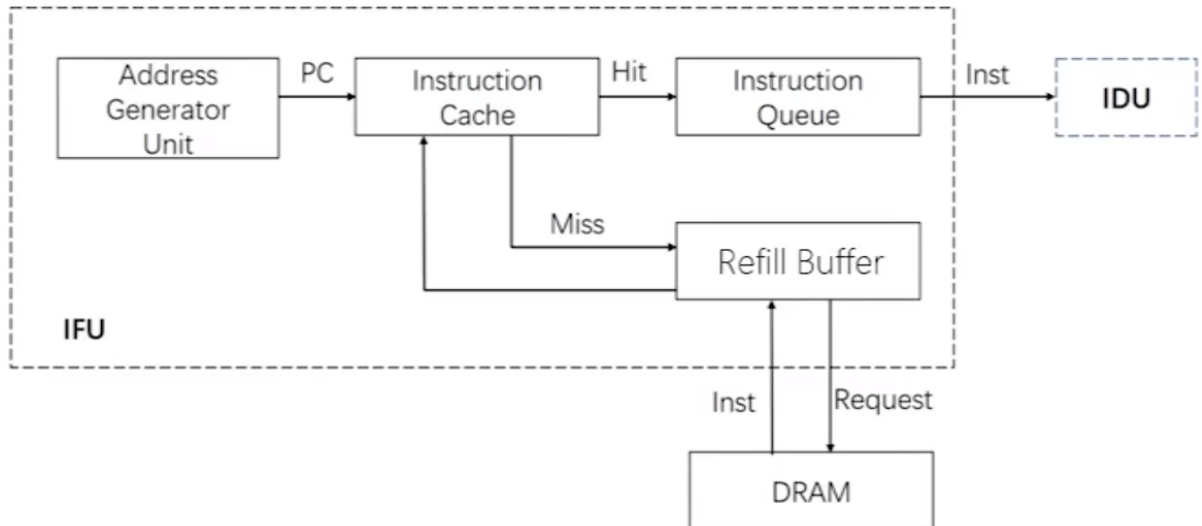
步骤	描述
 <p>The diagram shows a control unit with IFU and IDU. The storage unit contains DMA, NRAM, and WRAM. The operation unit contains VFU and MFU. A green arrow points from IFU to VFU. A green arrow points from IDU to MFU. A red arrow points from NRAM to VFU. A blue arrow points from WRAM to MFU. A red box highlights the MFU.</p>	<p>MFU接收到指令后从VFU接收仅预处理后的神经元数据，并从WRAM中读取权重数据，完成矩阵运算后将结果发给VFU</p>
 <p>The diagram shows the same components as Step 1. A red box highlights the VFU. A red arrow points from NRAM to VFU. A red arrow points from WRAM to MFU. A red arrow points from MFU to VFU.</p>	<p>VFU对输出神经元进行后处理（如激活、池化）</p>
 <p>The diagram shows the same components as Step 1. A green arrow points from VFU to NRAM. A red arrow points from NRAM to VFU. A red arrow points from WRAM to MFU. A red arrow points from MFU to VFU.</p>	<p>VFU将运算结果写回NRAM</p>
 <p>The diagram shows the same components as Step 1. A green arrow points from NRAM to the left, representing DRAM. A red arrow points from NRAM to VFU. A red arrow points from WRAM to MFU. A red arrow points from MFU to VFU.</p>	<p>DMA将输出神经元从NRAM写回到DRAM</p>

- 神经元数据流：DRAM→NRAM→VFU→(MFU→VFU→)NRAM→DRAM；

- 权重数据流：DRAM→WRAM→MFU；

## 控制模块

- IFU

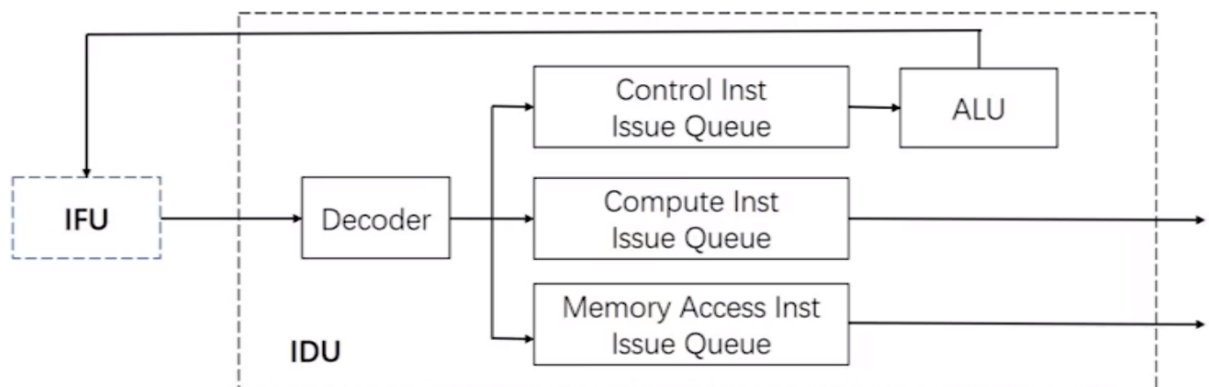


- 地址生成器AGU(Address Generator Unit)：

1.  $PC=0$
2.  $PC=PC+1$
3.  $PC=Jump/CB\ Reg$

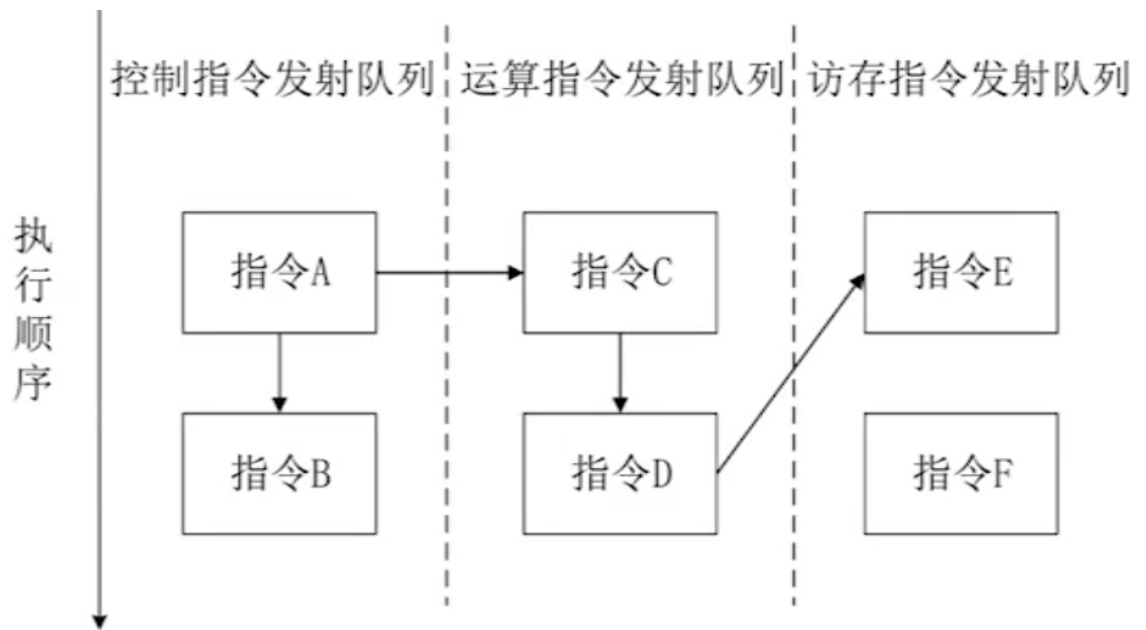
- 指令高速缓存ICache(Instruction Cache)；
- 指令回填单元RB(Refill Buffer)；
- 指令队列IQ(Instrcuton Queue)；

- IDU



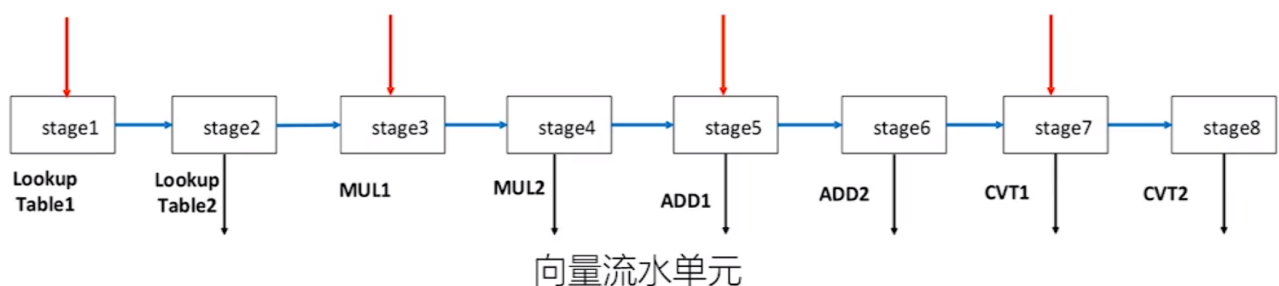
- 译码单元Decoder

- 指令发射队列(Issue Queue)：



- Control IQ、Compute IQ、Memory Access 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；
- 算数逻辑单元ALU

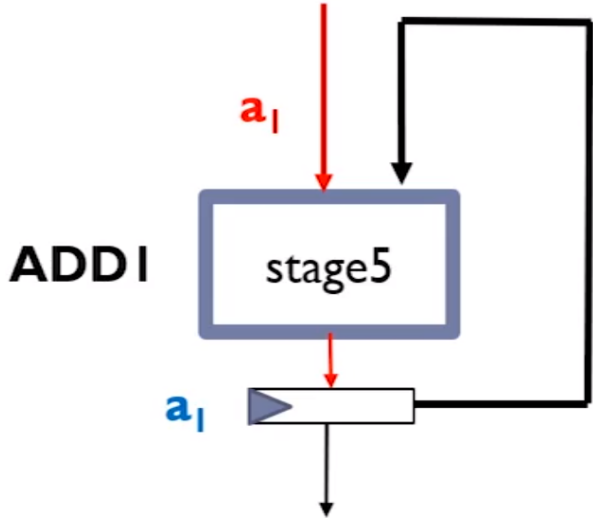
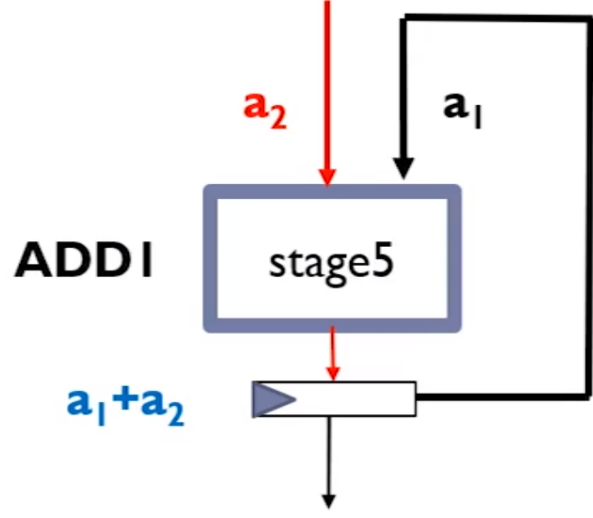
## 运算模块VFU

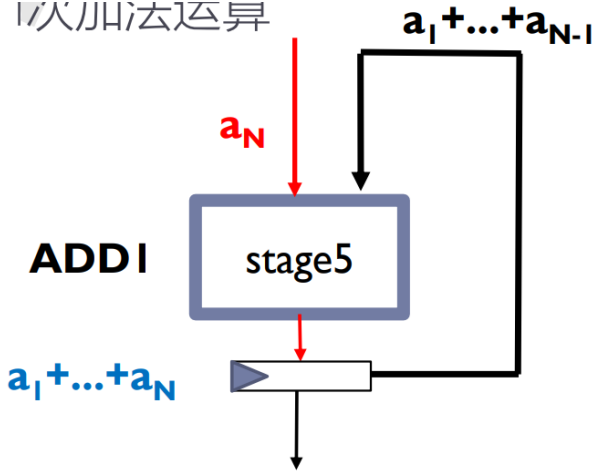
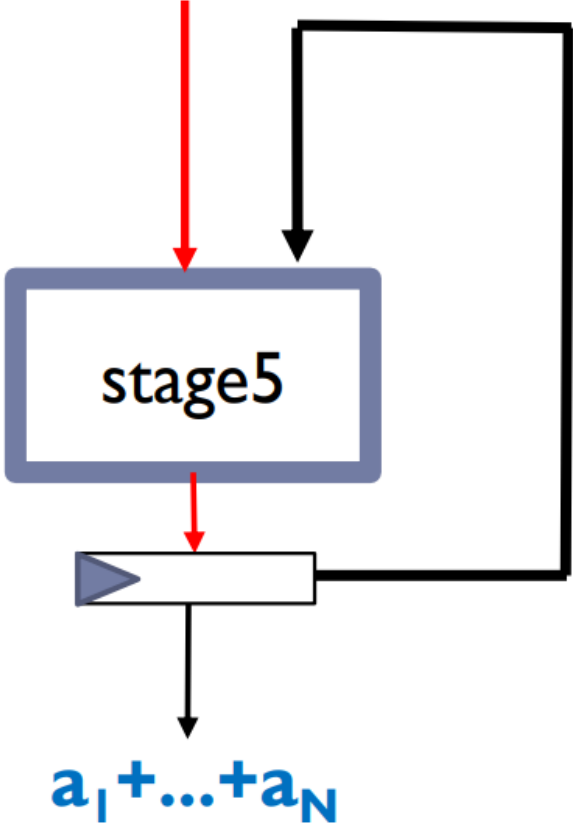


- 完成输入神经元的前处理和输出神经元的后处理；
- 包括向量流水单元和转置单元；
  - 多种数据类型：INT8/16/32、FP16/32；
  - 新增运算：查表、边缘扩充、数据格式转换等；
  - 多个stage可以输入，多个stage可以输出；
- 向量流水单元承载向量运算功能；
- 转置单元承载数据重新摆放功能；

# 向量流水单元如何完成Avg Pooling

- 当输入数据类型是INT时：
  - Avg Pooling本质是  $k_x \times k_y$  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Avg Pooling；
  - 每个步骤是一个cycle，所以完成Avg Pooling需要  $k_x \times k_y$  个cycles；
  - 步骤：

步骤	描述
	输入向量1 BYPASS stage5后进入stage5 输出寄存器
	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

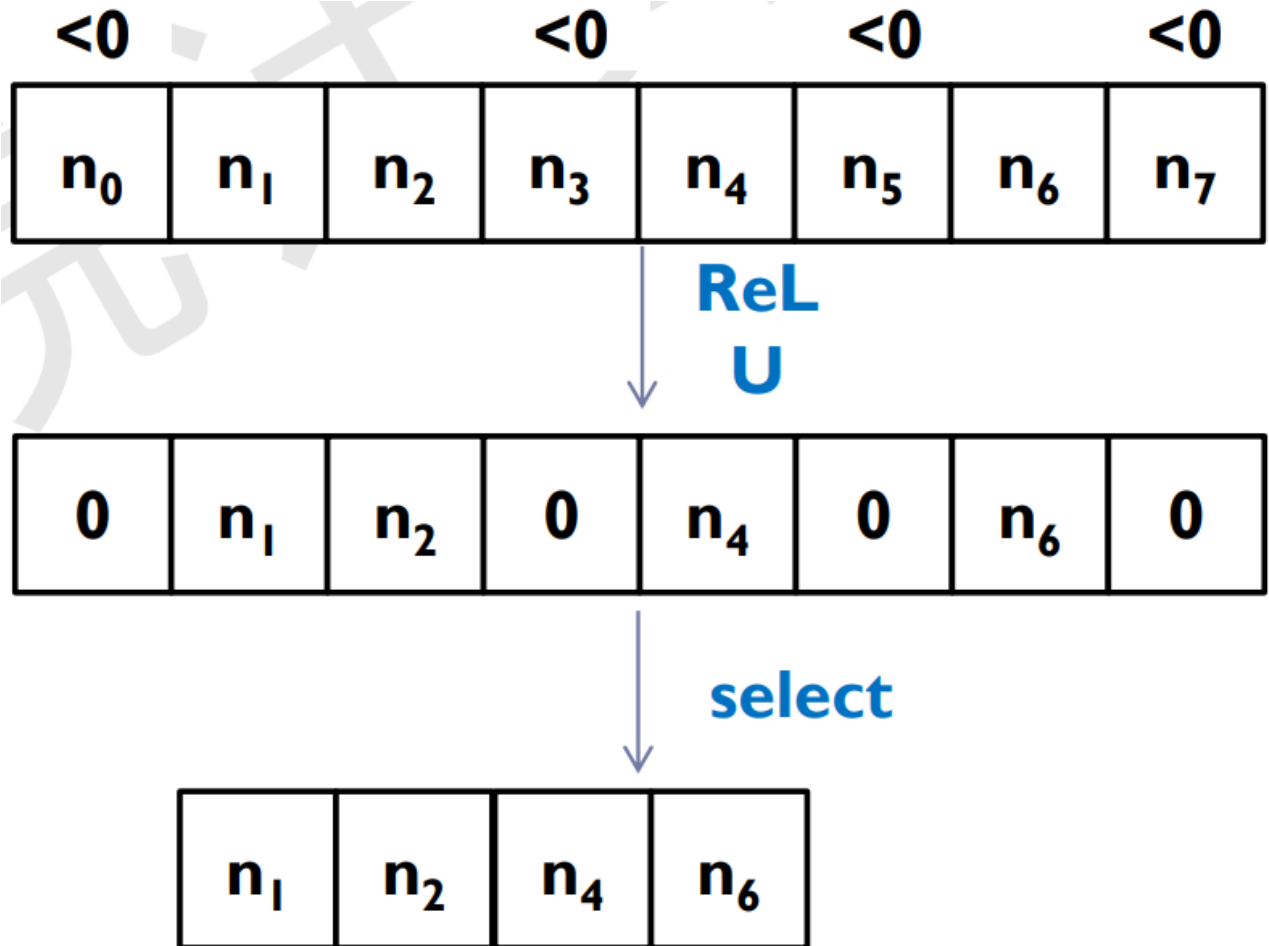
步骤	描述
<p>1次加法运算</p> 	<p>重复步骤2直至完成 <math>k_x \times k_y - 1</math> 次加法运算</p>
	<p>将结果从stage5的输出寄存器输出</p>

- 当输入数据类型是Float型时
  - Float型的加法延迟是2 cycles；
  - 使用stage5和stage6即可完成Float数据类型的Avg Pooling；
  - 步骤：
    1. 输入向量1 BYPASS stage5和stage6后进入stage6输出寄存器；
    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；
    3. 重复步骤2直至完成  $k_x \times k_y - 1$  次加法运算；
    4. 将结果从stage6的输出寄存器输出；

- 每个步骤是2 cycles，所以完成Avg Pooling需要  $2 \times k_x \times k_y$  个cycles；

## 向量流水单元完成神经元压缩

- 神经元经过ReLU会产生很多0；
- 如何将这些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；

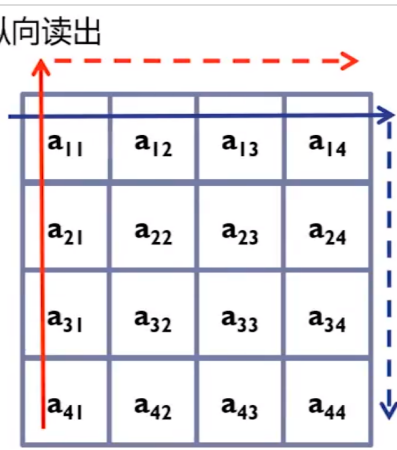


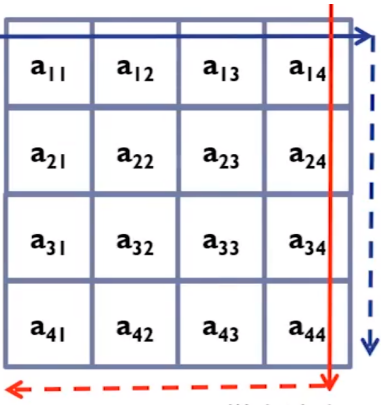
## 转置单元

- 支持多种数据类型，INT8/16/32、FP16/32；
- 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；
- 实现操作

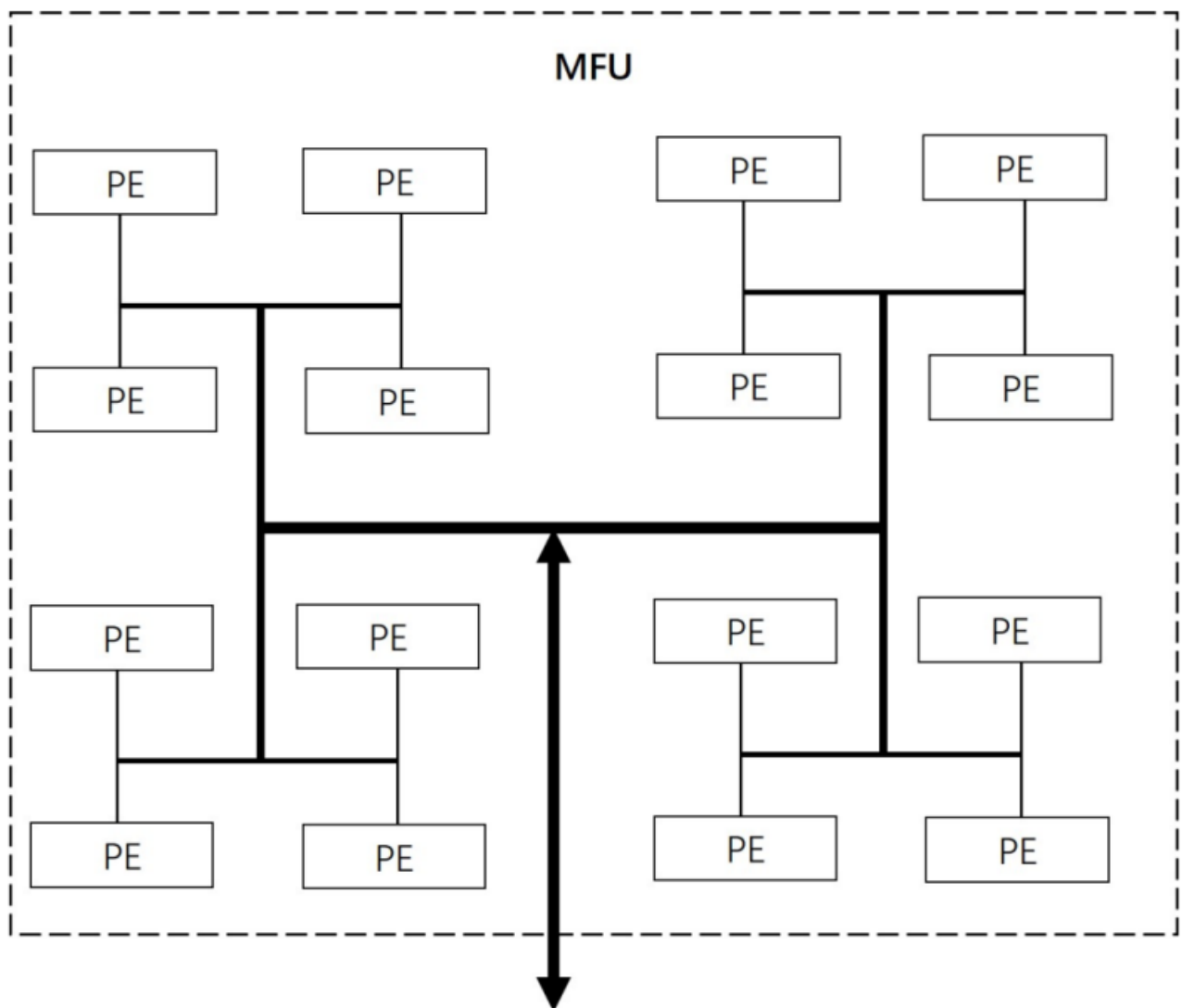
操作	描述	步骤
----	----	----



操作	描述	步骤
转置	<p>横向写入</p>  <p>纵向读出</p>	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；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；
镜像	<p>横向写入</p>  <p>横向读出</p>	1. 从第一行开始，将数据从左向右写入缓存；2. 从第一列开始，将数据从右向左依次读出；
旋转180度	<p>横向写入</p>  <p>横向读出</p>	1. 从第一行开始，将数据从左向右写入缓存；2. 从最后一行开始，将数据从右向左依次读出
顺时针旋转90度	<p>纵向读出</p>  <p>横向写入</p>	1. 从第一行开始，将数据从左向右写入缓存；2. 从第一列开始，将数据从下到上依次读出

操作	描述	步骤
顺时针旋转 270 度	<p>横向写入</p>  <p>纵向读出</p>	<p>1. 从第一行开始，将数据从左向右写入缓存；2. 从最后一列开始，将数据从上到下依次读出</p>

## 运算模块



- MFU：矩阵运算单元
  - H-tree互联；
  - 低位宽定点运算器；
  - 三种模式：

- INT16×INT16；
- INT8×INT8；
- INT8×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；
  - 片内外需虚实地址转换；
- 降低访存延迟
  - 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；

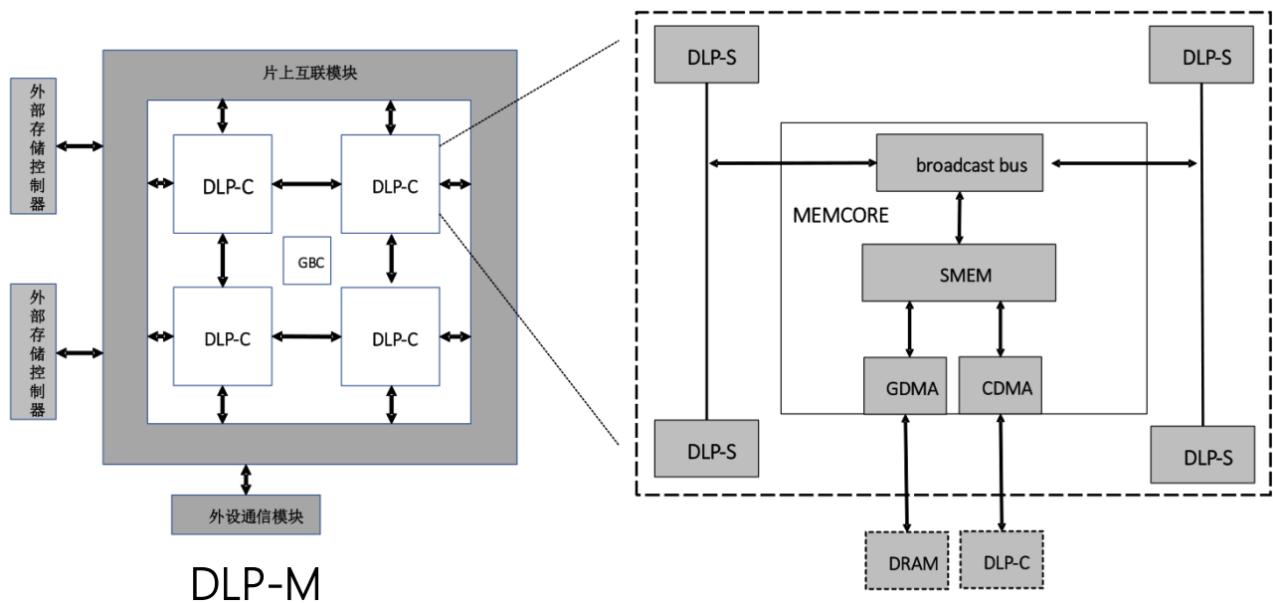
## DLP-S总结

- 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；

## 7.2 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；

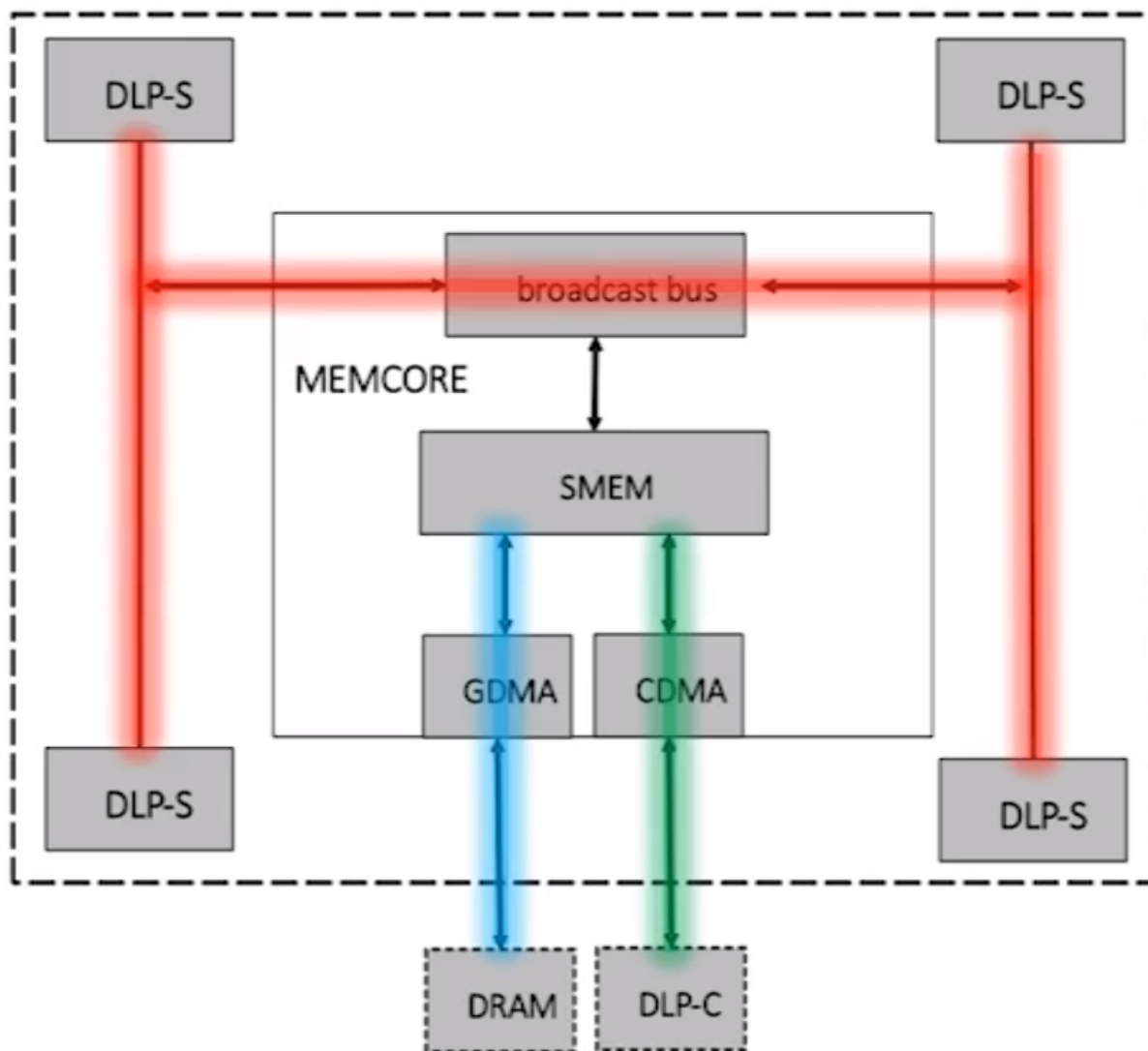
## 总体架构



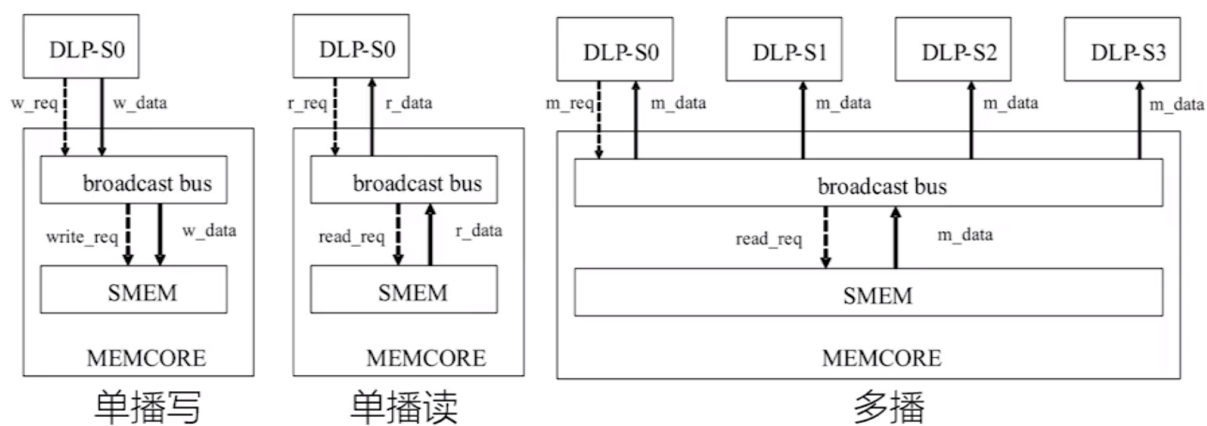
- 多核处理器分层结构设计
  - 一个DLP-M由多个DLP-C构成；
  - 一个DLP-C由多个DLP-S构成；
  - 减少NoC的负载核开销；
- DLP-M
  - 外部存储控制器；
  - 外设通信模块；
  - 片上互联模块；
  - 同步模块GBC(Global Barrier Controller)；
  - 四个DLP-C；
- DLP-C
  - 四个DLP-S；
  - 存储核MEMCORE(Memory Core)
    - 存储：DLP-S共享数据；
    - 通信：DLP-C与片外DRAM，DLP-C之间，多个DLP-S之间；

## Cluster架构

- MEMCORE

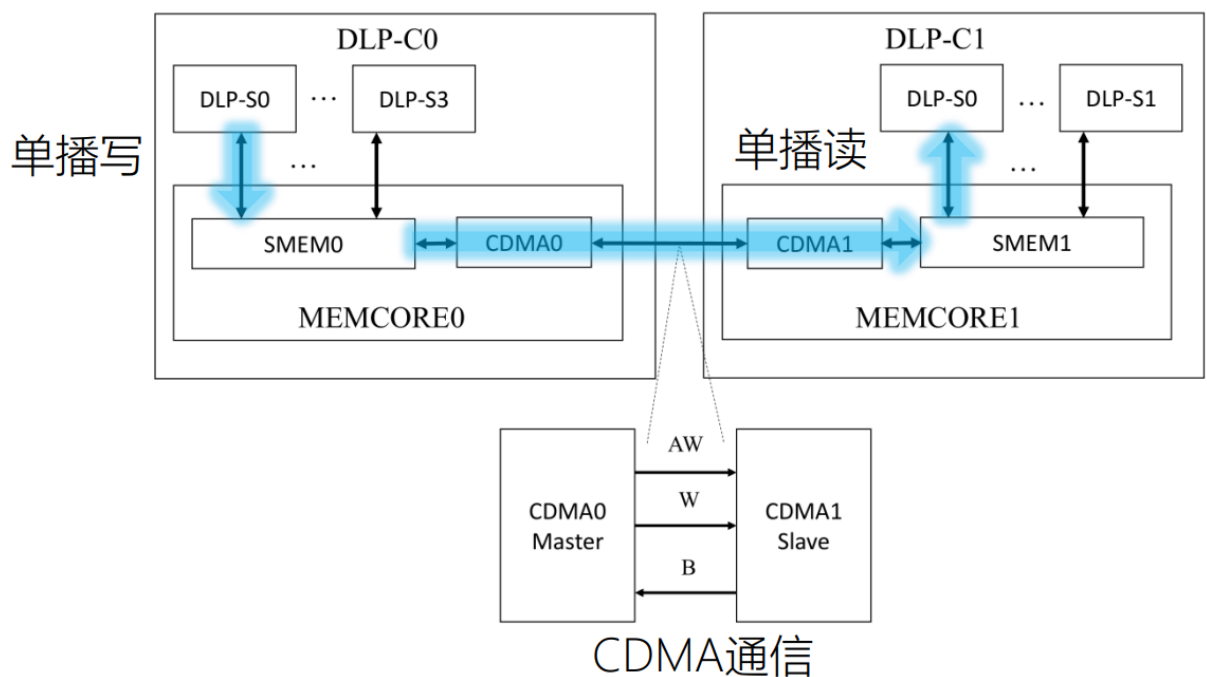


- 共享存储模块SMEM(Shared Memory)；
- 广播总线(Broadcast Bus)；
- Cluster直接内存访问CDMA(Cluster Direct Memory Access)；
- 全局直接内存访问GDMA(Global Direct Memory Access)；
- 广播总线



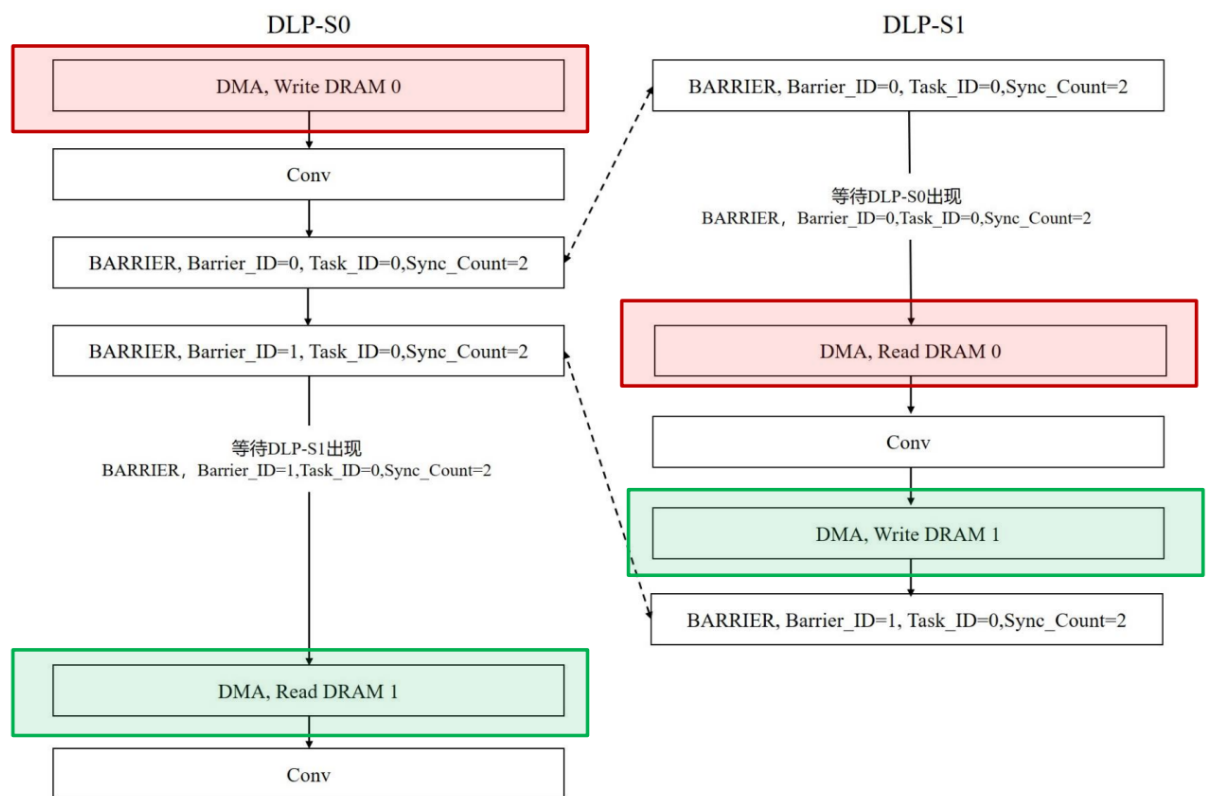
- 深度学习数据的复用特性；

- 读写请求：单播写、单播读，多播；
- 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；
- 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；
- GDMA
  - 每个DLP-C可能对于多个DRAM终止其，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；
- 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；
- 双核协同指令流

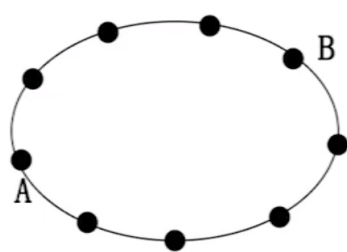


## 互联架构

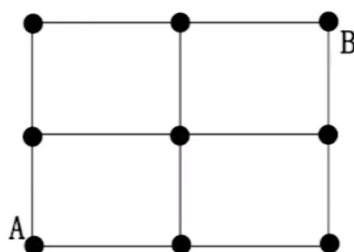
- 多核协同
  - 数据共享来减少对片外DRAM的访问；
  - 提高处理单个任务时的计算能力；
  - 需要实时数据交互；

- 核间互联拓扑结构

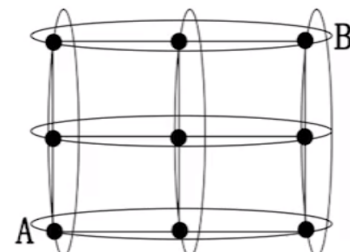
- 不同核到同一个核的延时相同：提供所有核完全对等的编程模型，方便软件编写和性能优化，也使得多核系统在调度时可以做任意的任务分配；
- 核间的互联通路尽量稠密：减少单个通路负载，同时降低访问延时。理论上，只有多核之间对称的全连接拓扑才能完全满足上述要求；



(a) 环形



(b) 网状

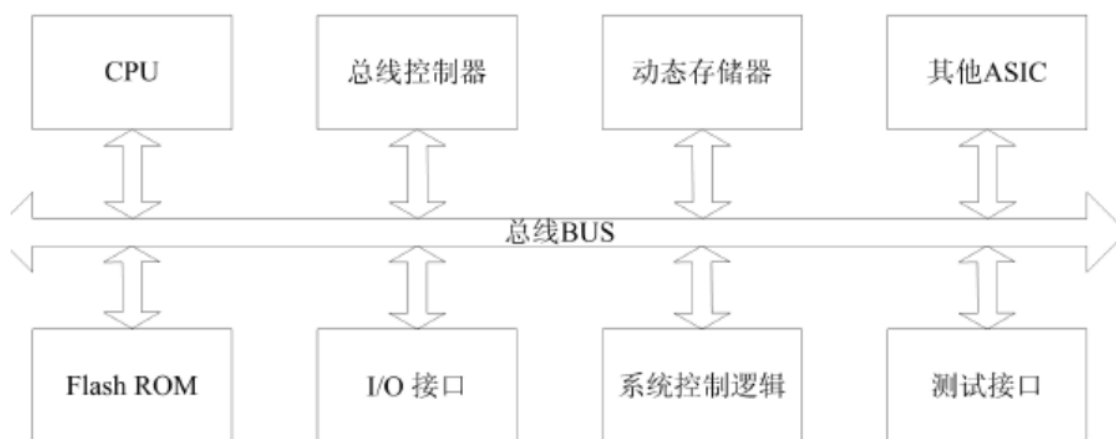


(c) Torus

- 互联方式

- 总线互联

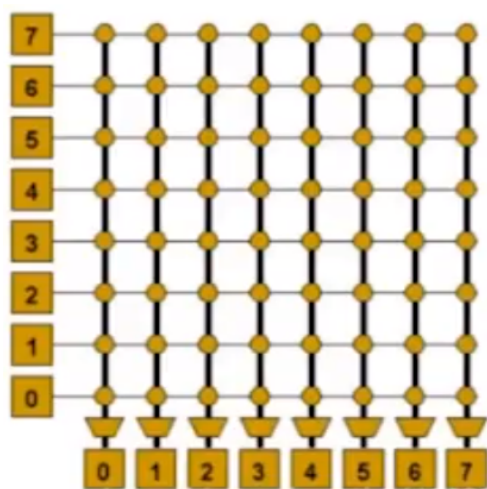
- 公共数据干线
- 拓展性差



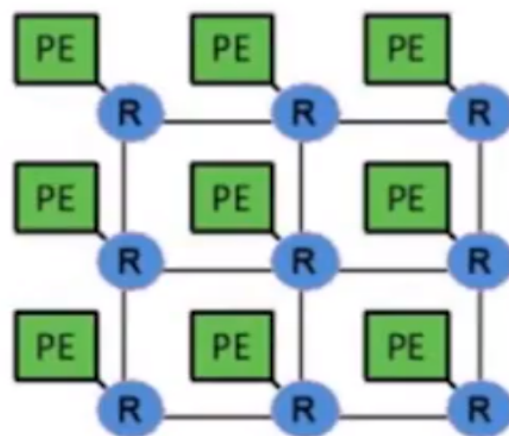
- 片上网络

- 片上互联
- 性能和功耗有优势
- 拓展性好



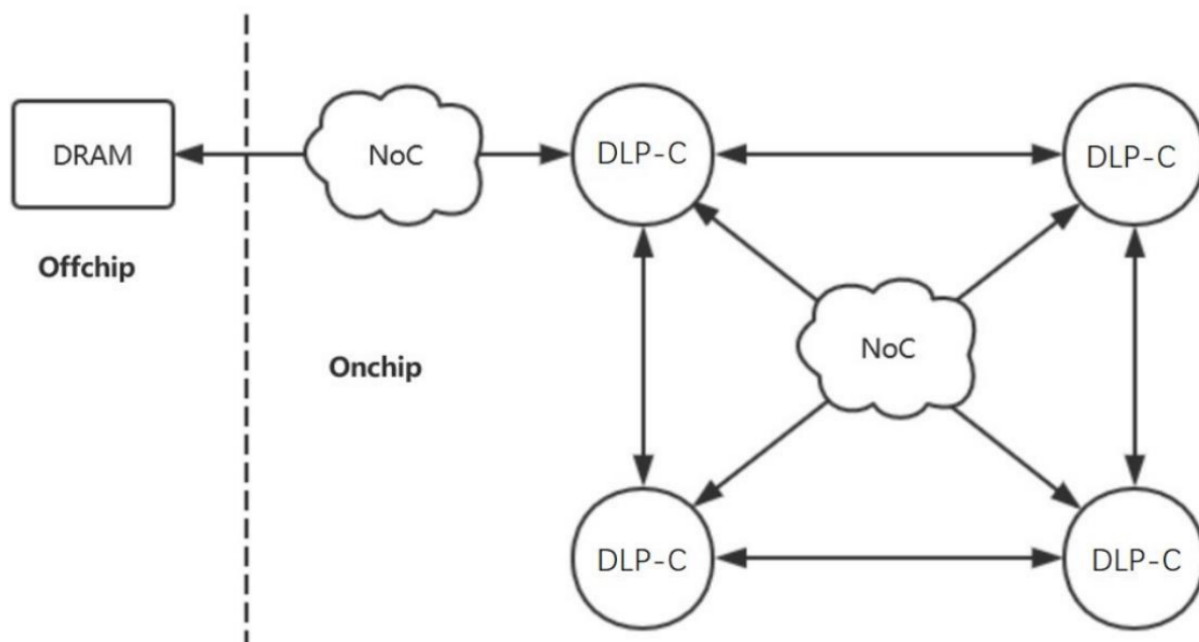


Crossbar



NoC

- DLP-C互联



## DLP-M总结

- 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

