北京大學
PEKING UNIVERSITY

人工智能的硬件基石
从物理器件到计算架构

第十讲：AI芯片设计-II

主讲：陶耀宇

2026年春季

注意事项

- 课程作业情况

- **作业将在3月底-4月中旬、4月中旬-5月初、5月中旬-6月初**

  第二次作业时间：5.11-5.25

  第三次作业时间：5.28-6.13
- **第1次lab时间：4月13日-5月13日（因CLAB故障延期）**
- **第2次lab时间：5月13日-6月13日（因CLAB故障延期，将适当简化）**
- **文献调研与报告（6月1日、6月8日）**

北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

北京大学 PEKING UNIVERSITY

期末汇报

- **自行选择器件（IEDM等）、电路（ISSCC、VLSI等）或架构（MICRO、ISCA等）方面的论文，或Nature/Science系列相关论文，进行3-5篇文献阅读**
  - **占总成绩20%**，分组（2-3人一组）深入论文技术细节，做**10分钟汇报**

AI加速器芯片
- 传统AI加速器：Fused-layer cnn accelerators、Eyeriess，Google TPU等
- 新兴AI加速器（大模型Transformer、Neural ODE、MANN/DNC、PINN等）

GPGPU芯片
- 流式多处理器（Multithreaded Streaming Multiprocessors，CUDA的来源）

FPGA芯片等（可编程逻辑块、可编程路由等）

安全与通信领域处理器芯片
- 各类密钥编码（AES、RSA）、视频编码（MPEG等）、通信编码（LDPC、Polar等）

传统CPU芯片
- 优化Branch Predictor、Load-Store、缓存预读取、众核缓存一致性等

新兴智能计算芯片
- 存算一体/感存算一体、量子计算、生物信息处理、高维NoC、区块链
- 基于后摩尔非CMOS器件的架构（模拟计算架构、动力学计算架构等）

北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

目 录

CONTENTS

01. 多核多线程数据并行

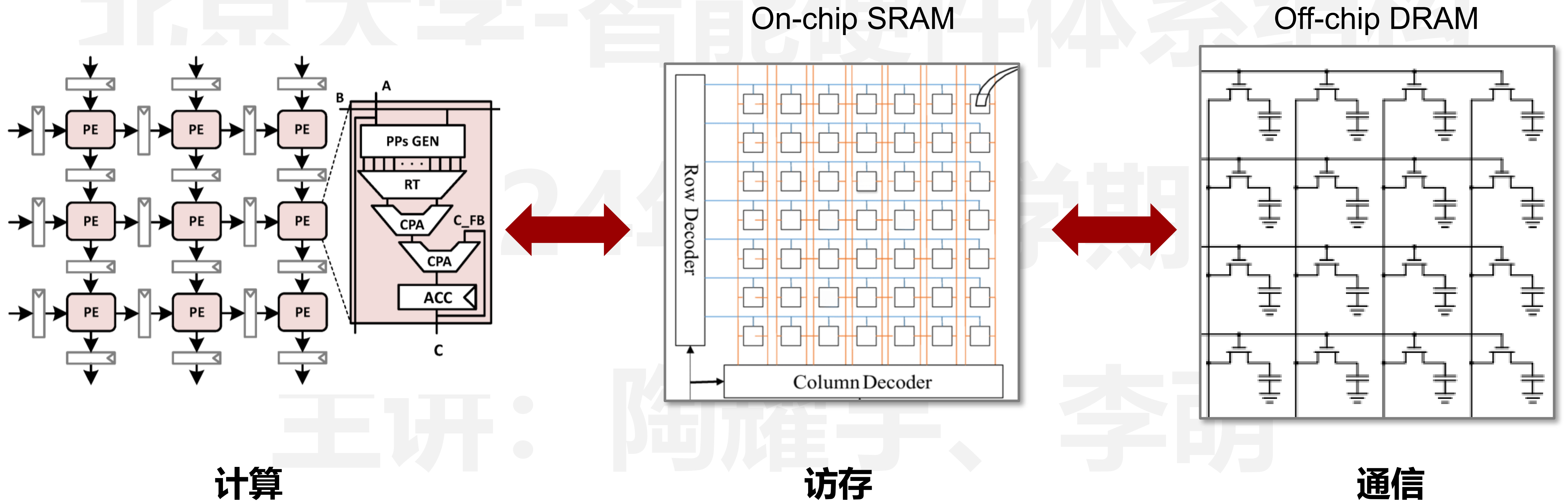
02. 基于编译的静态优化

03. GPGPU架构基础入门

04. AI芯片架构基础

AI加速器架构基础

- **AI加速器整体架构**



北京大学-智能硬件体系结构
24年春季学期
主讲：陶耀宇、李萌

AI加速器架构基础——计算单元

北京大学 PEKING UNIVERSITY

- 计算单元：主要包含面向矩阵、向量和标量三种
  - 分别对应神经网络计算过程中矩阵、向量和标量计算算子
- 针对计算单元，我们经常关心的指标是**计算访存比**，**高计算访存比，通常能带来更高的能效**
- 计算访存比同时受到**计算负载**和**硬件架构**的影响

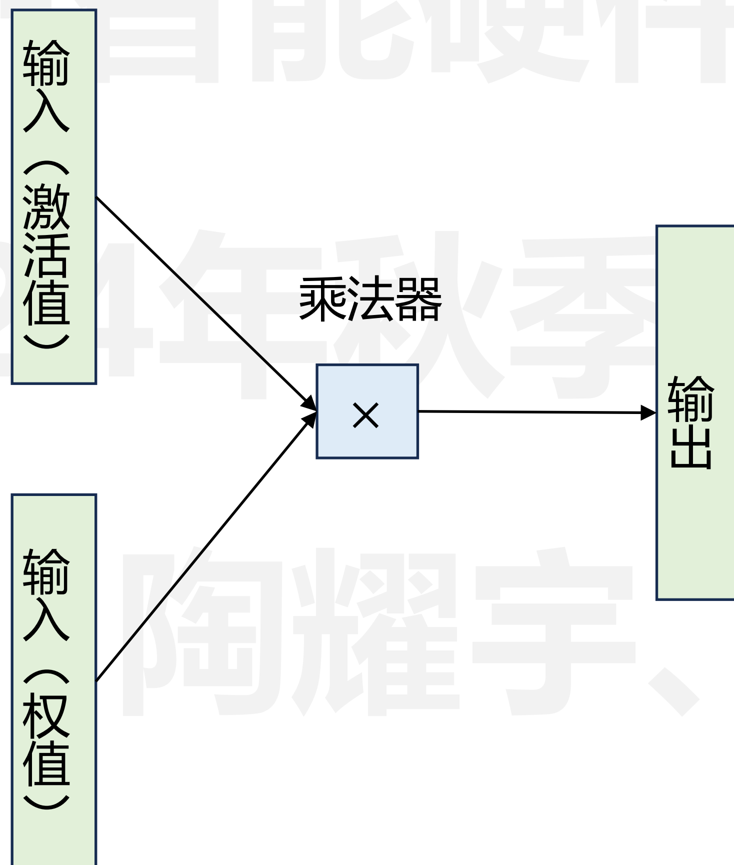
北京大学 智能硬件体系结构

$$\begin{bmatrix} 1 & 0 & 2 \\ 3 & 1 & 0 \\ 5 & -1 & 2 \end{bmatrix} \times \begin{bmatrix} 2 & -1 & 0 \\ 5 & 1 & -1 \\ -2 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -2 & -1 & 0 \\ 11 & -2 & -1 \\ 1 & -6 & 1 \end{bmatrix}$$

AI加速器架构基础——计算单元

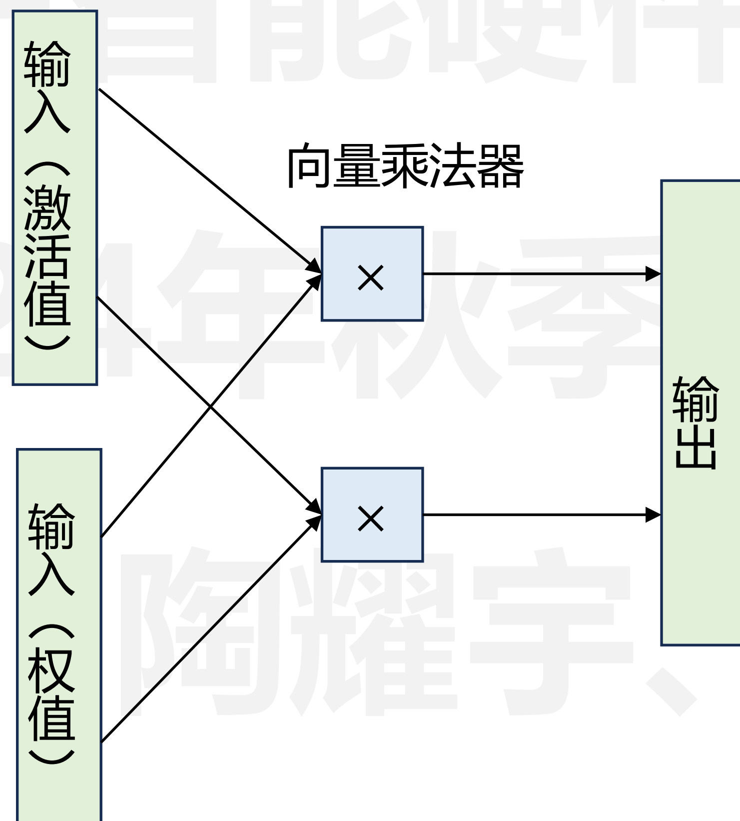
- **矩阵计算单元：通过内积计算，最大化数据复用**

北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌



北京大学 PEKING UNIVERSITY

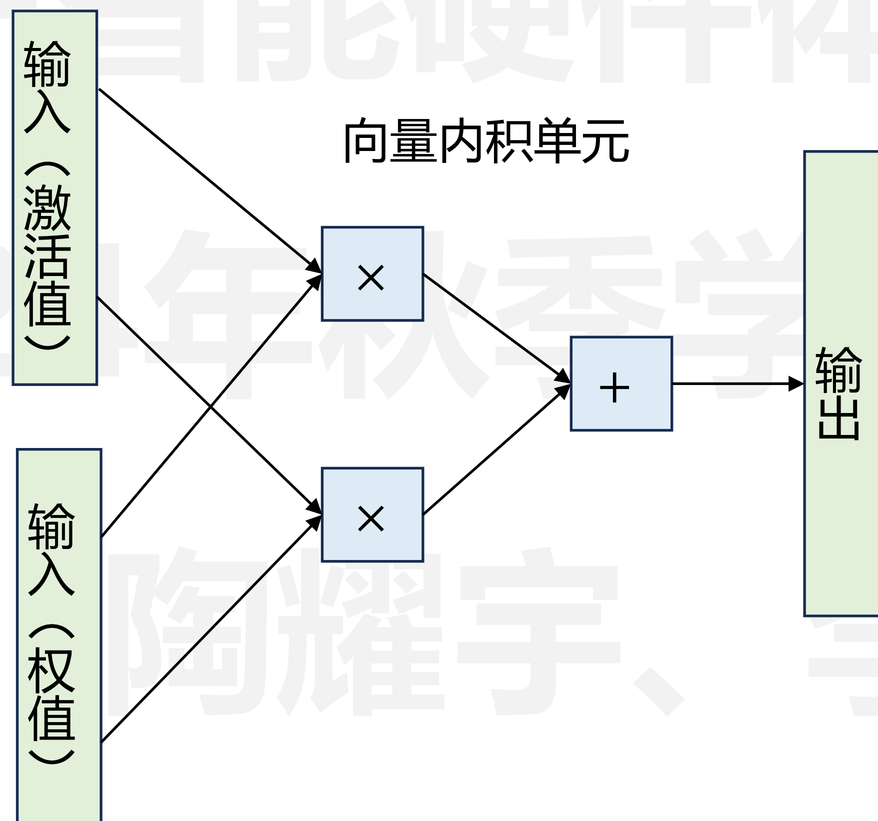
- **矩阵计算单元：通过内积计算，最大化数据复用**



北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

北京大学 PEKING UNIVERSITY

- **矩阵计算单元：通过内积计算，最大化数据复用**

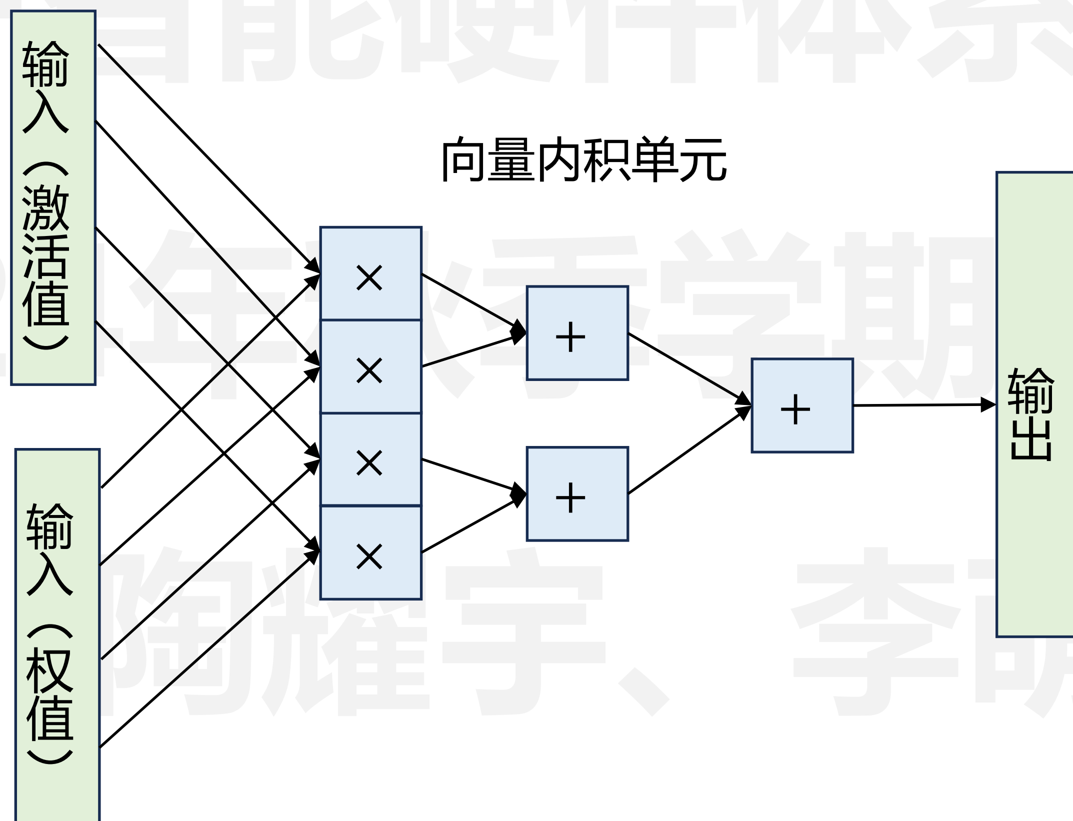


北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

北京大学 PEKING UNIVERSITY

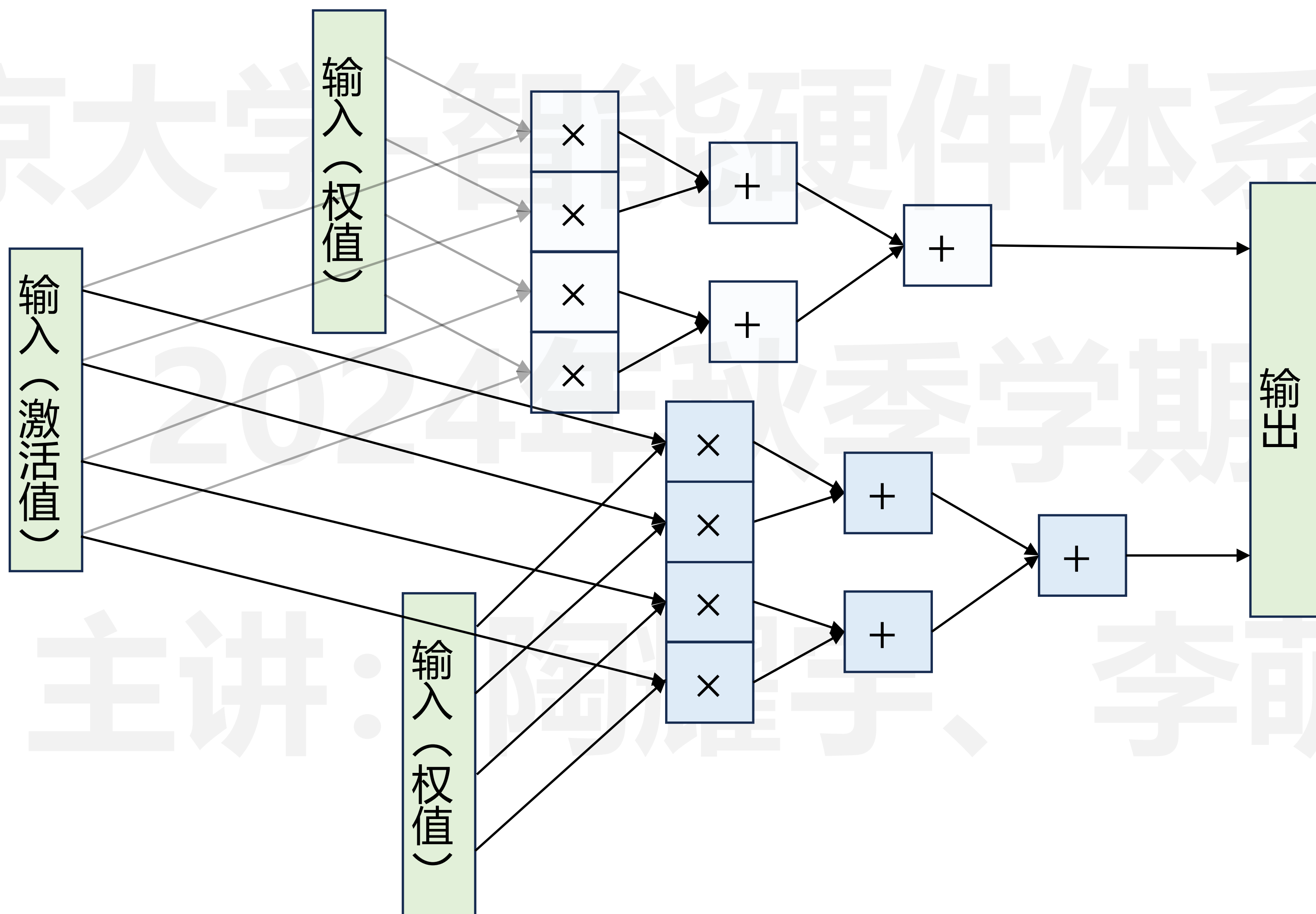
- **矩阵计算单元：通过内积计算，最大化数据复用**

北京大学-智能硬件体系结构
2021年秋季学期
主讲：陶耀宇、李萌



AI加速器架构基础

- **矩阵计算单元：通过内积计算，最大化数据复用**

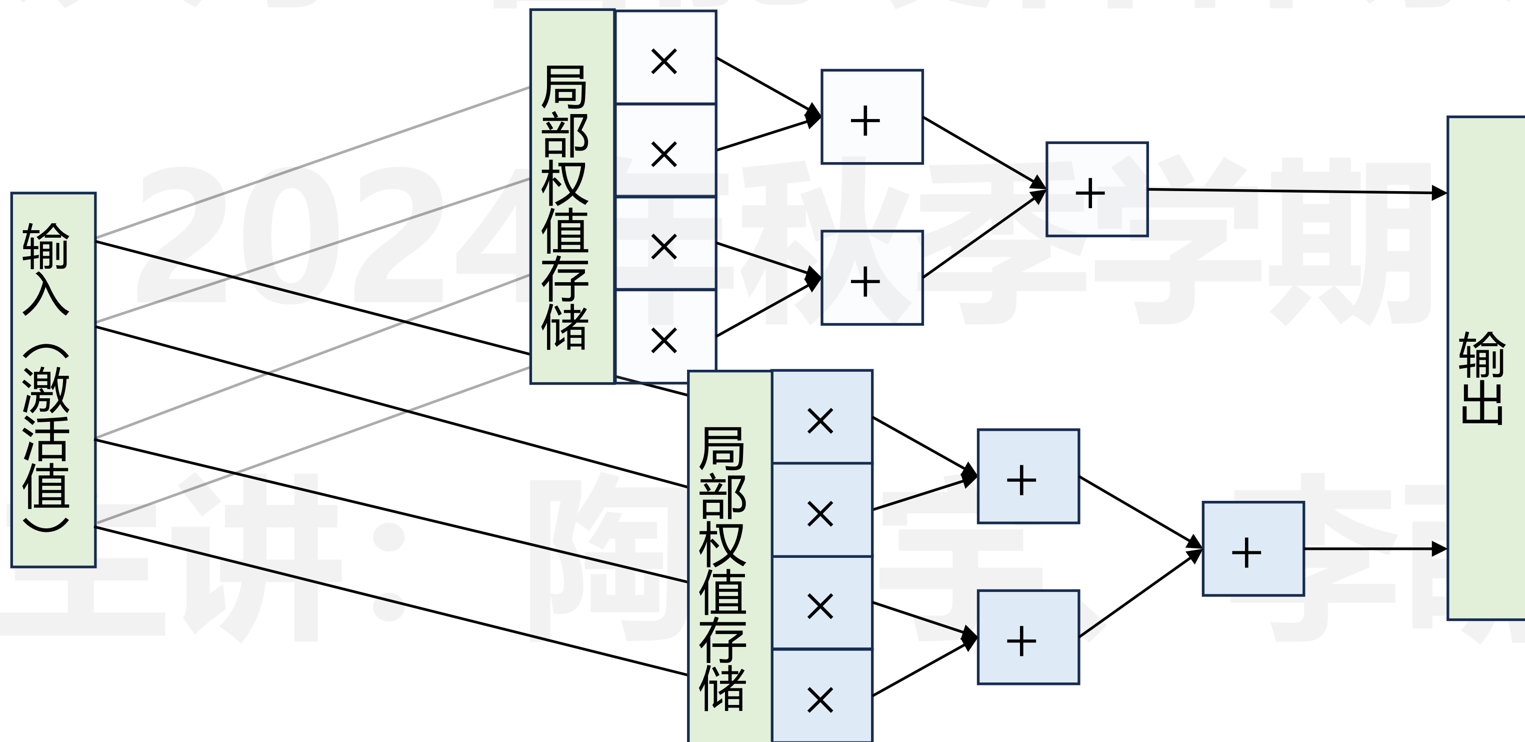


北京大学 PEKING UNIVERSITY
北京大学 智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

AI加速器架构基础

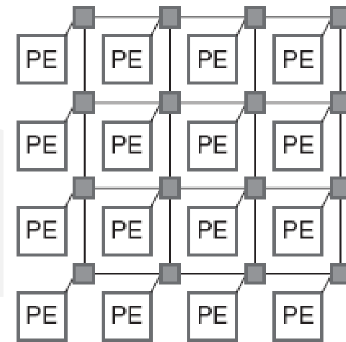
- **矩阵计算单元：通过内积计算，最大化数据复用**
- **权重采用局部小而快的存储器，直接存储在内积单元附近的电路中**

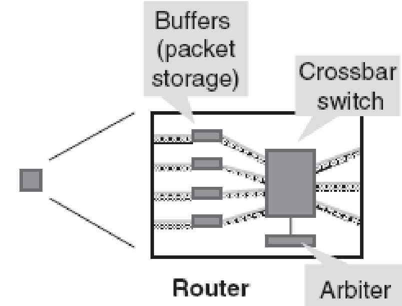
北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

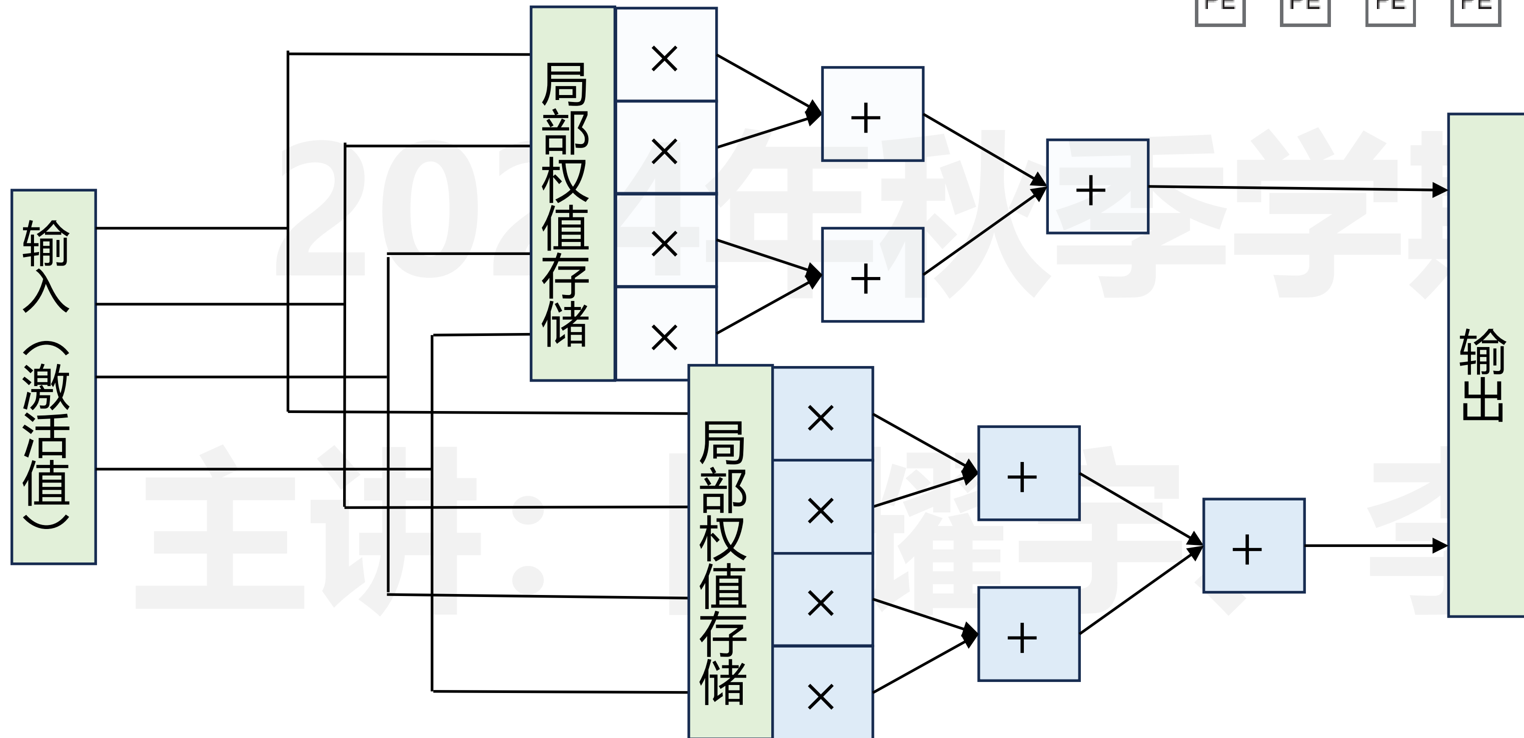


AI加速器架构基础

- **矩阵计算单元：通过内积计算，最大化数据复用**
- **所有内积单元共享激活值，采用广播的形式**

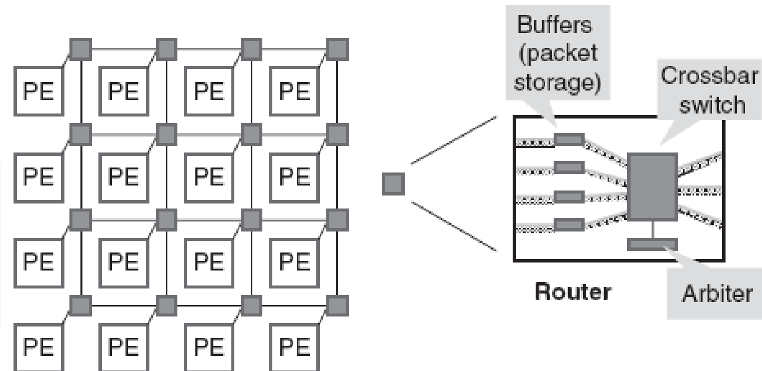


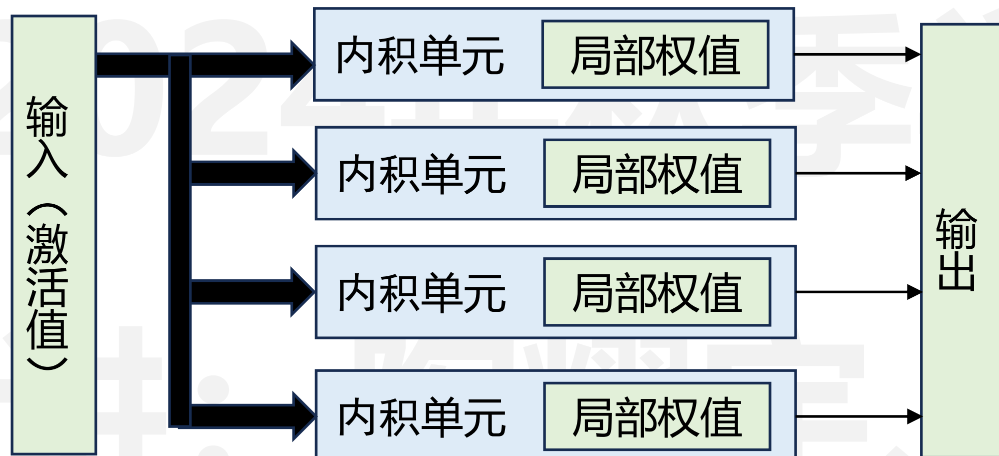




AI加速器架构基础

- **矩阵计算单元：通过内积计算，最大化数据复用**
- **所有内积单元共享激活值，采用广播的形式**



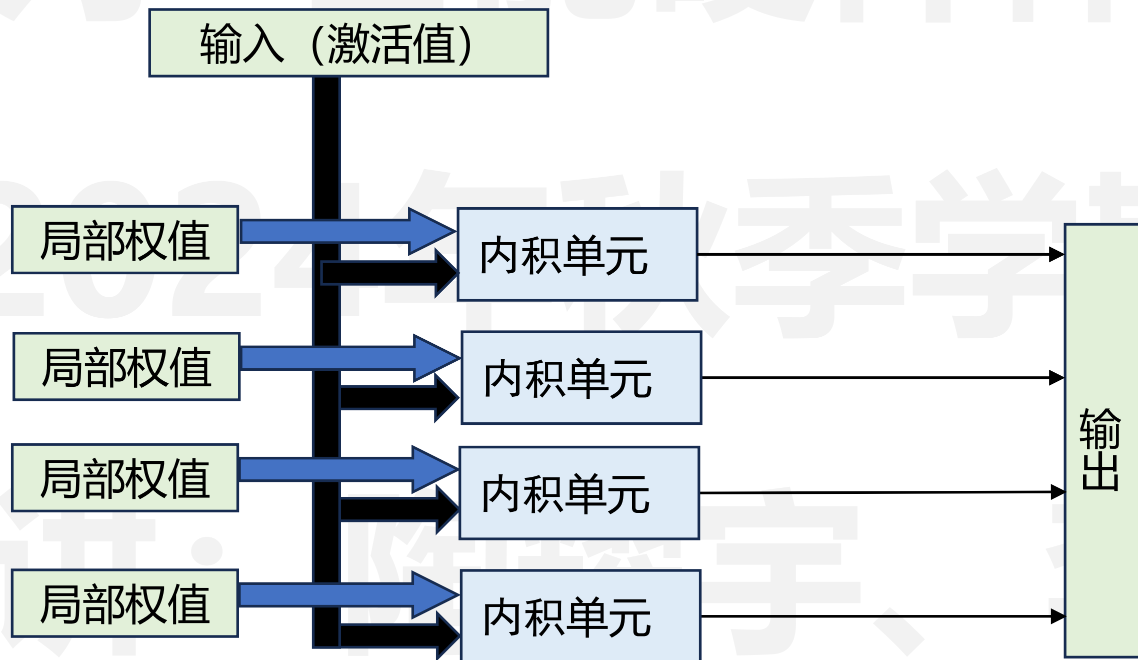


北京大学-智能硬件… 2024年秋季学期 主讲：… 李萌

AI加速器架构基础

- **矩阵计算单元：通过内积计算，最大化数据复用**
- **如果我们把权值提取出来**

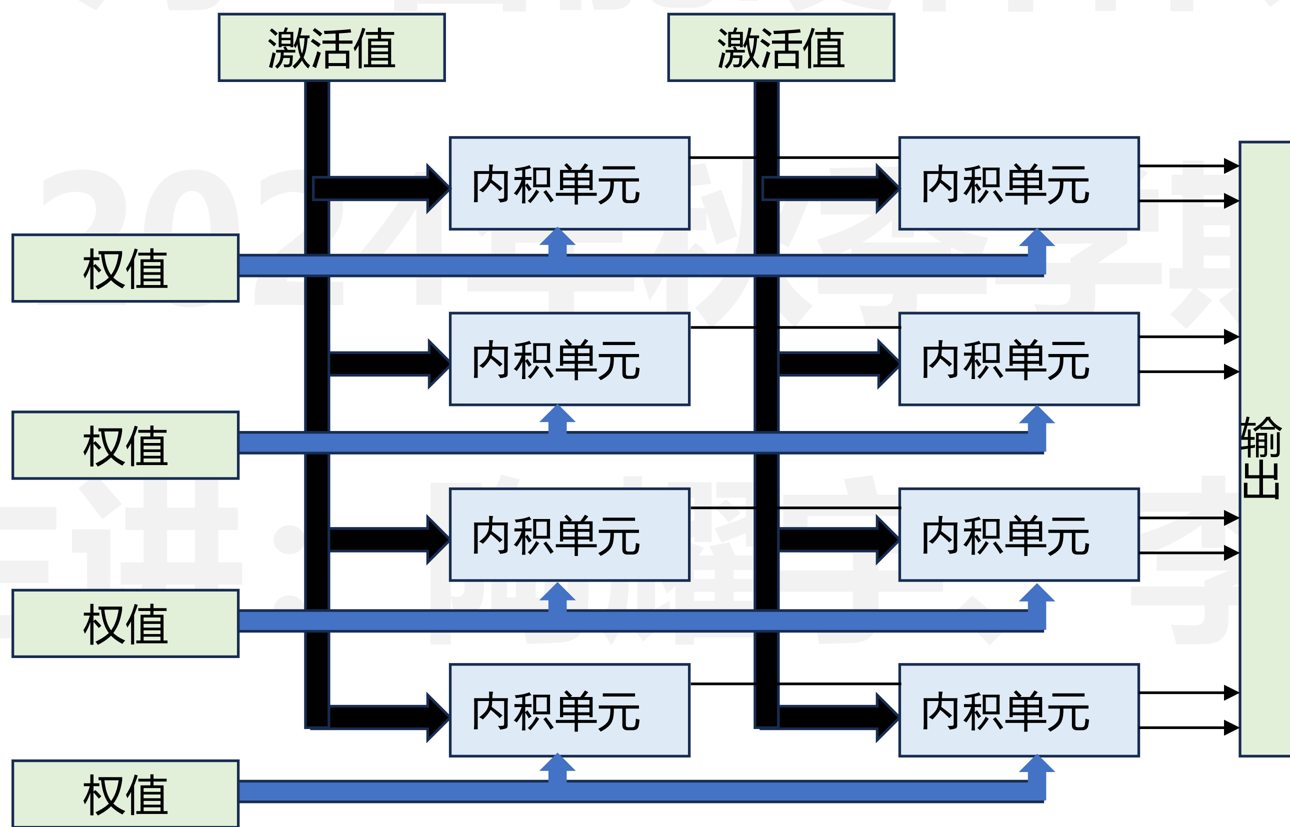
北京大学-智能硬件体系结构
2024年秋季学期
主讲：陈晓宇、李萌



AI加速器架构基础

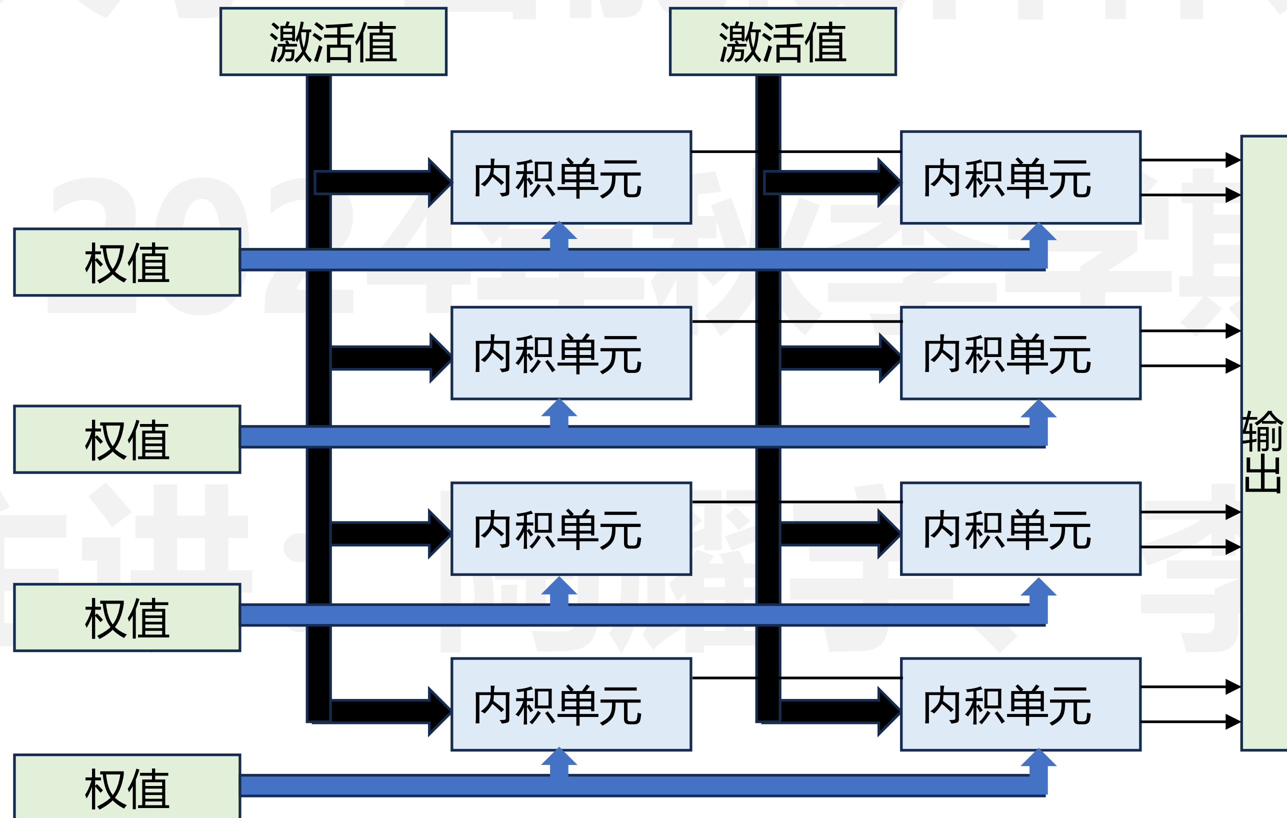
- **矩阵计算单元：通过内积计算，最大化数据复用**
- **如果我们把权值提取出来，并进一步扩大规模，能够显著增加数据复用**

北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌



AI加速器架构基础

- 矩阵计算单元：通过内积计算，最大化数据复用
- 矩阵乘法单元在规模大时，能够实现很大的计算访存比
- 但是，面临连线复杂、距离远、扇出（Fanout）多等问题



北京大学-智能硬件体系结构
2024年秋季学期
主讲：高峰、李萌

AI加速器架构基础

- 针对连线距离复杂、距离远、扇出多的问题，提出**脉动阵列**进行改进
- 只存在相邻计算单元之间的数据传输，连线短、扇出少
- 但是，**脉动阵列的延迟高（需要等待启动/排空）**，专用性更强，难以改造支持其他功能

北京大学-智能硬件体系结构 2024年秋季学期 主讲：陶耀宇、李萌

a44 a34 a24 a14
a43 a33 a23 a13
a42 a32 a22 a12
a41 a31 a21 a11

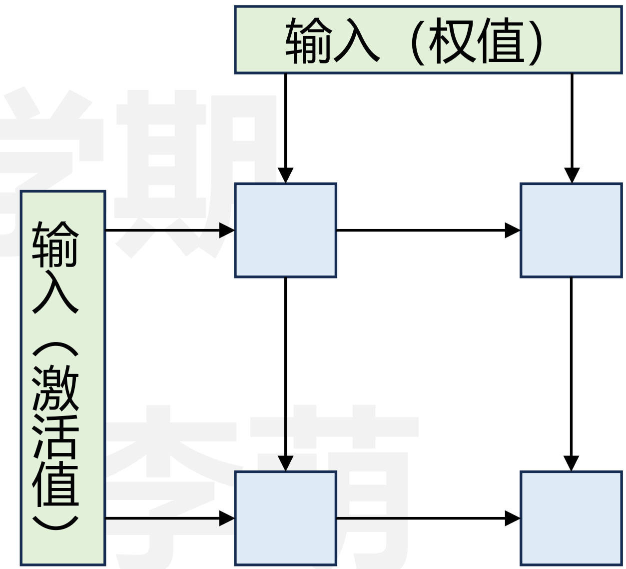
b41 b42 b43 b44
b31 b32 b33 b34
b21 b22 b23 b24
b11 b12 b13 b14

P11
P21 P12
P31 P22 P13
P41 P32 P23 P14
P42 P33 P24
P43 P34
P44

$a_{i,k}$ $b_{k,j}$

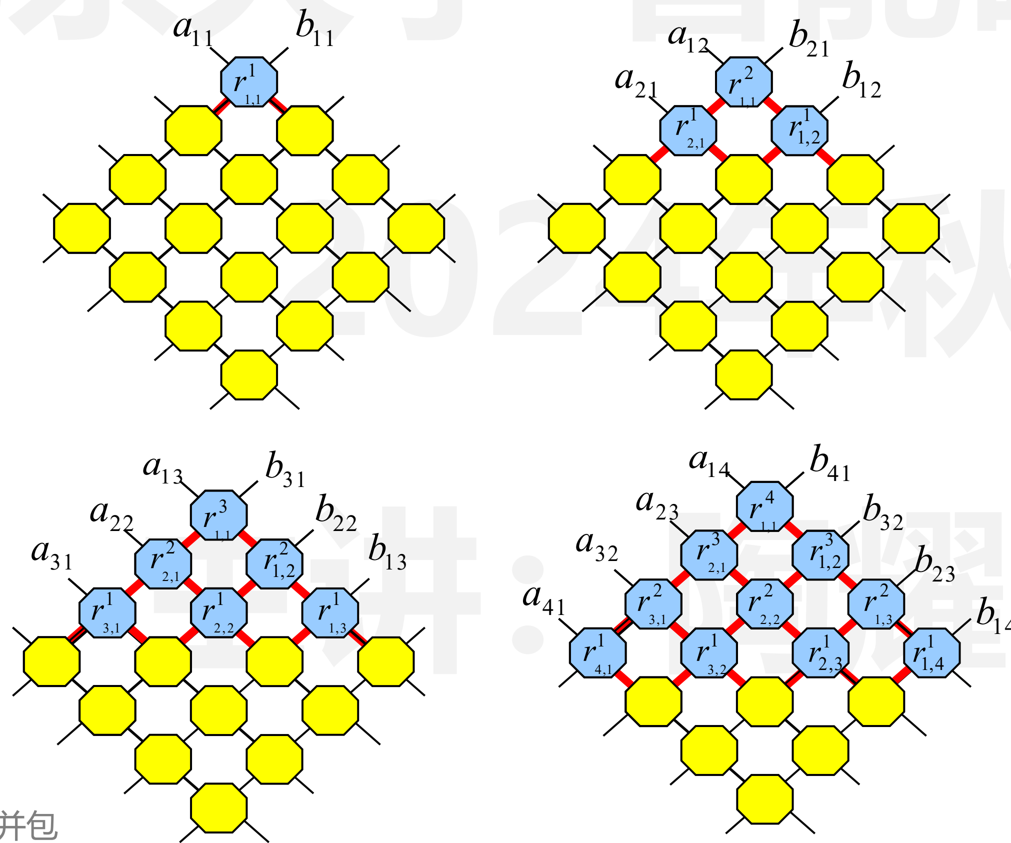
$r^k_{i,j} = r^k_{i,j} + a_{i,k} b_{k,j}$

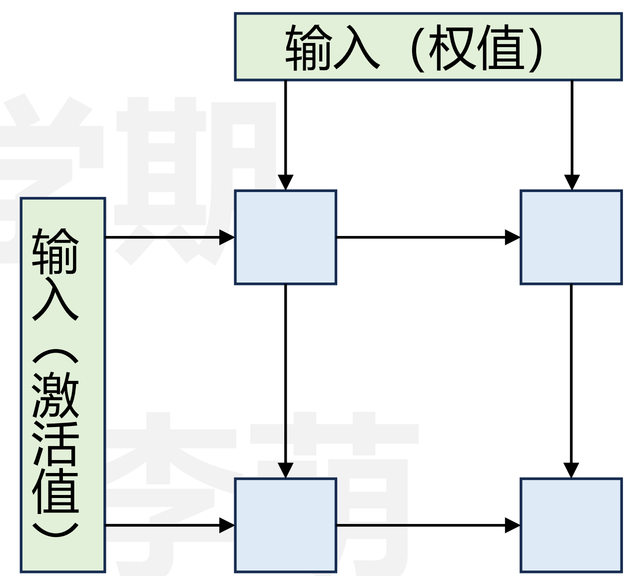
$b_{k,j}$ $a_{i,k}$



AI加速器架构基础

- 针对连线距离复杂、距离远、扇出多的问题，提出**脉动阵列**进行改进
- 只存在相邻计算单元之间的数据传输，连线短、扇出少
- 但是，**脉动阵列的延迟高（需要等待启动/排空）**，专用性更强，难以改造支持其他功能



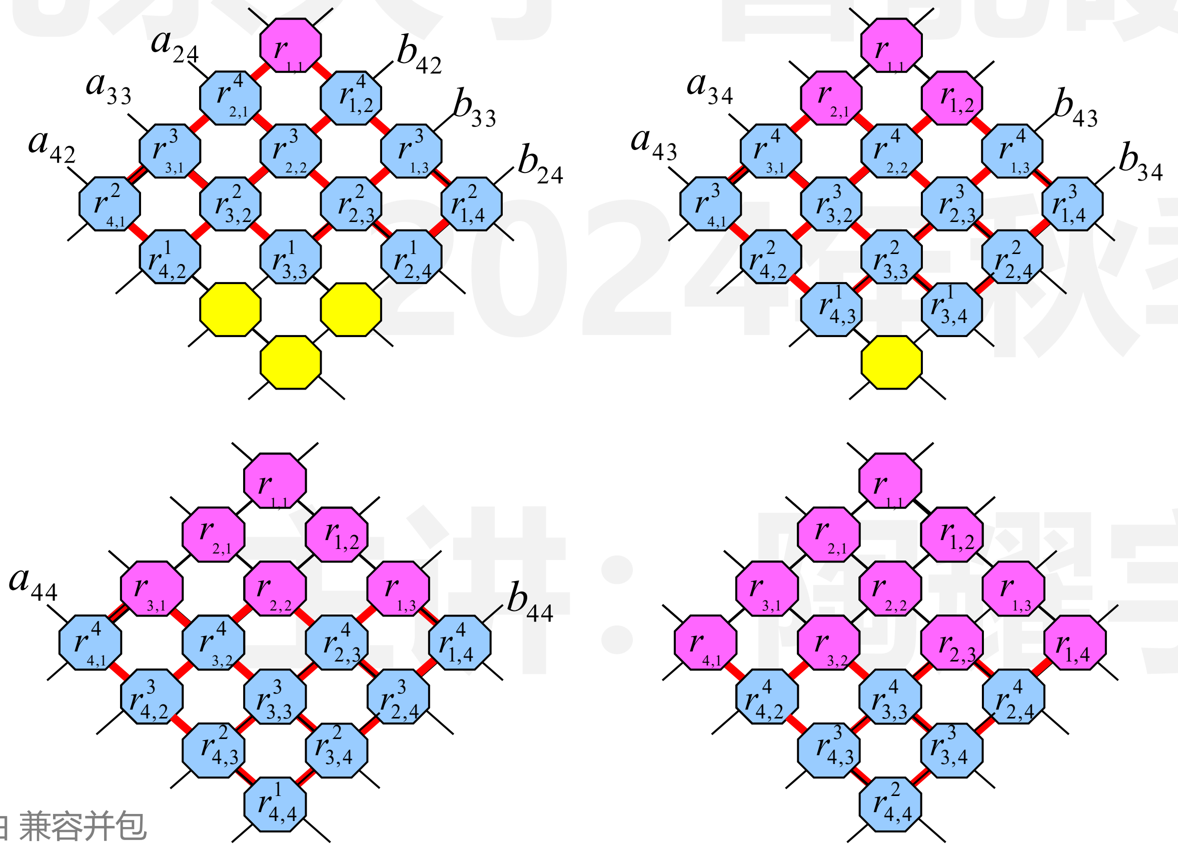


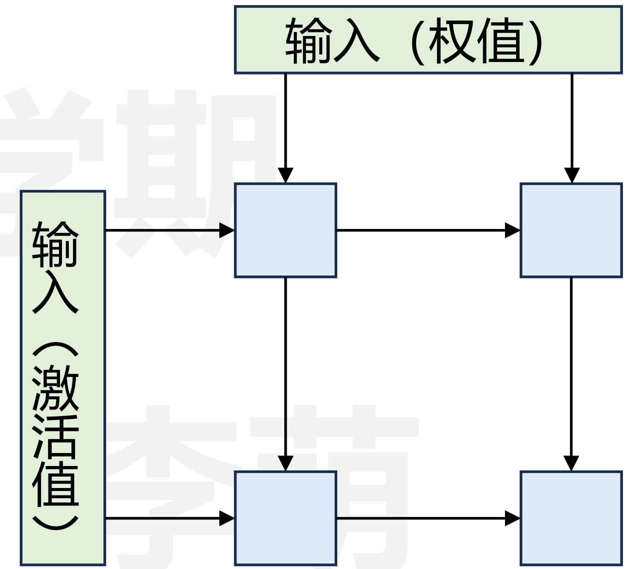
北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

AI加速器架构基础

- 针对连线距离复杂、距离远、扇出多的问题，提出**脉动阵列**进行改进
- 只存在相邻计算单元之间的数据传输，连线短、扇出少
- 但是，**脉动阵列的延迟高（需要等待启动/排空），**专用性更强，难以改造支持其他功能

北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

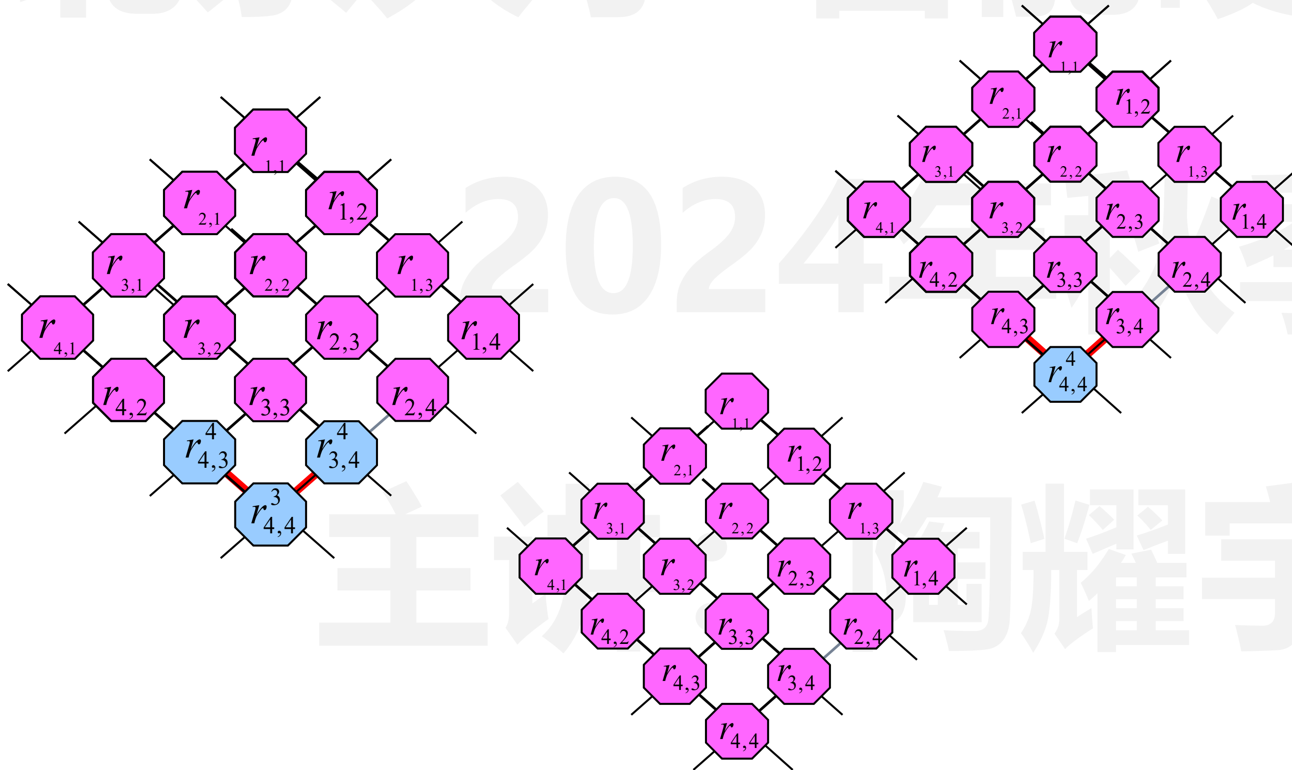


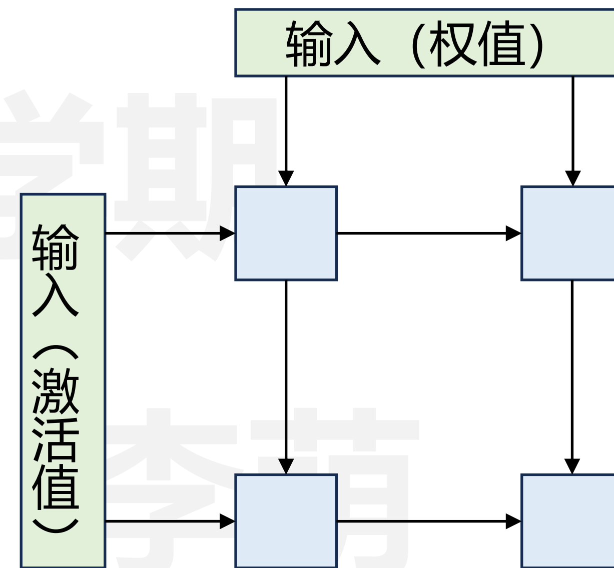


AI加速器架构基础

北京大学 PEKING UNIVERSITY

- 针对连线距离复杂、距离远、扇出多的问题，提出**脉动阵列**进行改进
- 只存在相邻计算单元之间的数据传输，连线短、扇出少
- 但是，**脉动阵列的延迟高（需要等待启动/排空），**专用性更强，难以改造支持其他功能





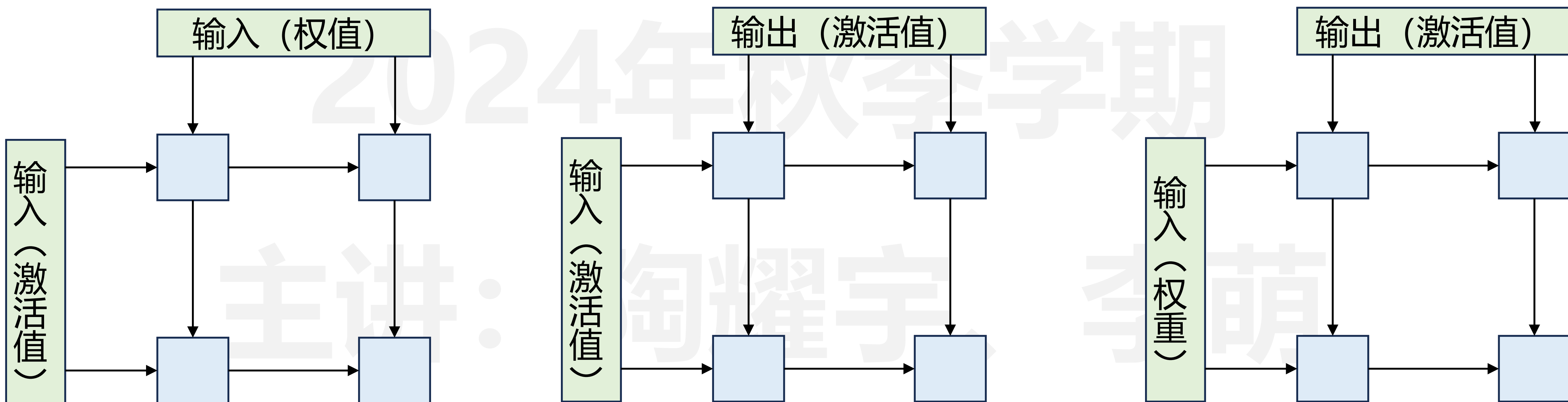
北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

AI加速器架构基础——张量单元

PEKING UNIVERSITY 北京大学

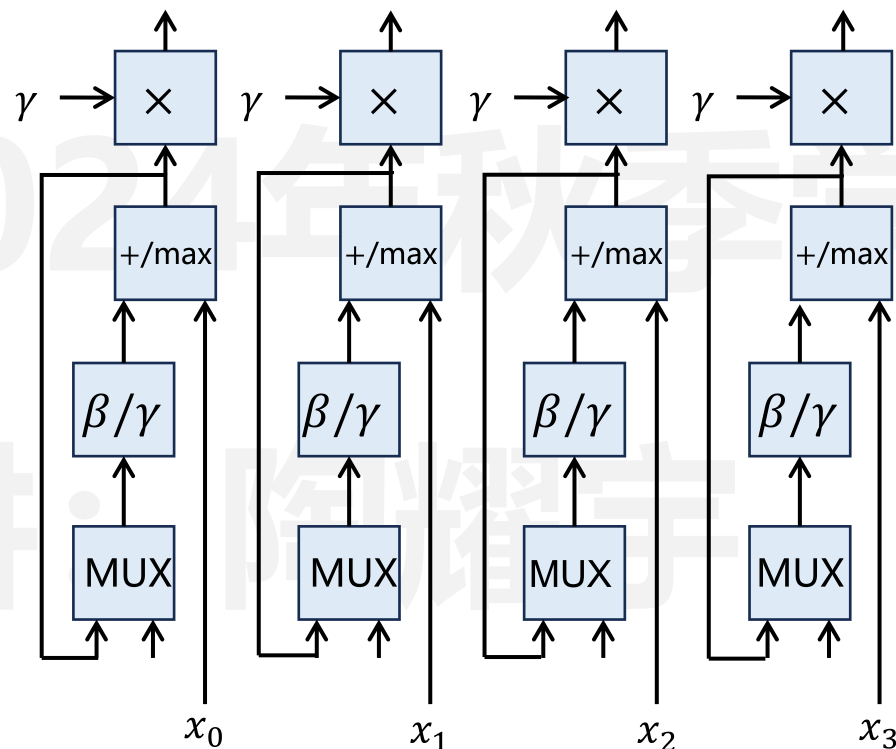
- 矩阵乘法阵列设计核心思想：设计数据流架构，利用计算模式本身的数据复用，提升计算效率
- 针对数据流架构，核心是明确哪些数据是**移动**的，哪些数据是**固定不动**
- **典型数据流**：output stationary、weight stationary、input stationary
  - 不同的数据流，具有不同的神经网络算子适用性

北京大学-智能硬件体系结构 2024年秋季学期 主讲：陶耀宇、李萌



AI加速器架构基础——向量和标量单元

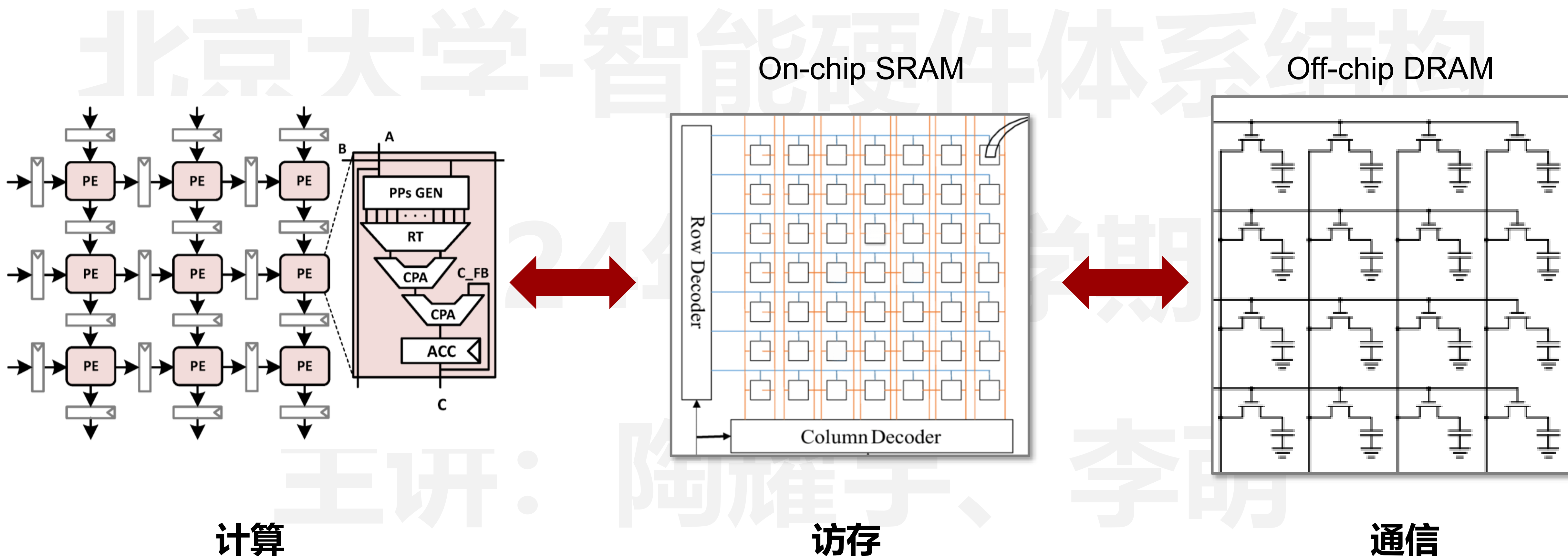
- 向量计算单元主要用于计算池化、归一化、ReLU、Sigmod、Softmax等特殊函数
- 尽管向量运算占神经网络总运算量并不大，但是，如果没有合适的加速硬件，仍然可能成为瓶颈
- 思考：向量计算单元和矩阵计算单元都有哪些区别?



北京大学 智能硬件体系结构
2024年秋季学期
主讲：陶耀宇 李萌

AI加速器架构基础

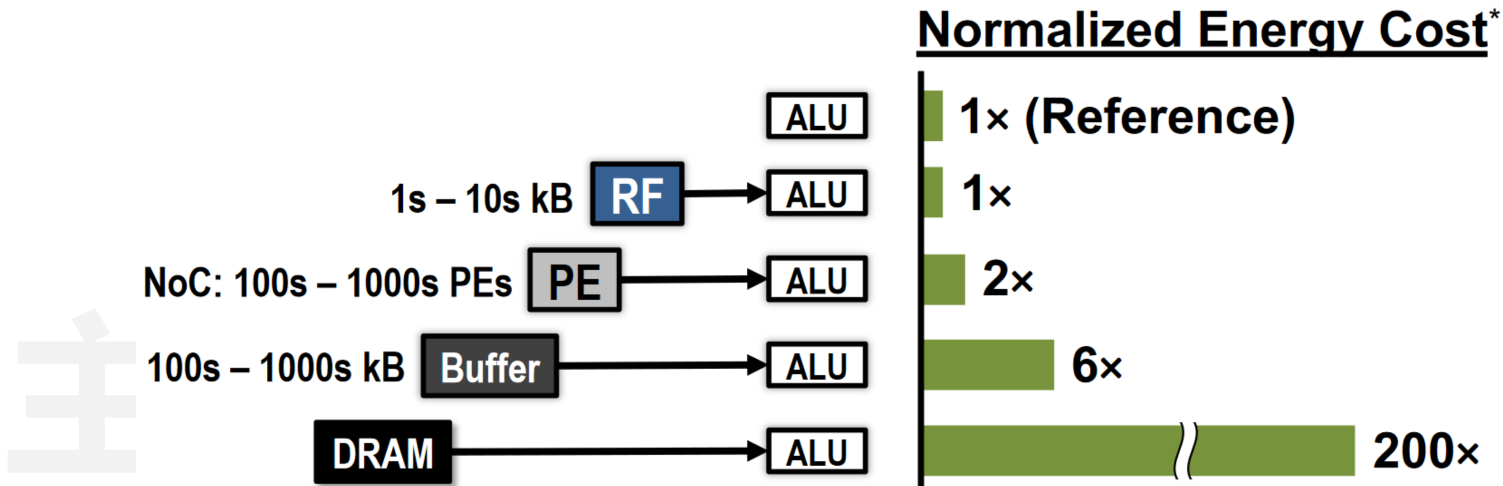
- **AI加速器整体架构**



北京大学-智能硬件体系结构
24年春季学期
主讲：陶耀宇、李萌

AI加速器架构基础——片上访存

- 针对AI芯片，通过引入片上SRAM，能够有效提升数据复用，降低DRAM访存开销
- 思考：AI芯片的片上存储是否可以延用传统CPU的缓存?
  - 可以但是没必要，由于AI芯片往往采用确定数据流，数据复用明确，无需借助类似CPU的缓存，仅使用便签存储器（Scratch Pad Memory）即可



AI加速器架构基础——片上访存

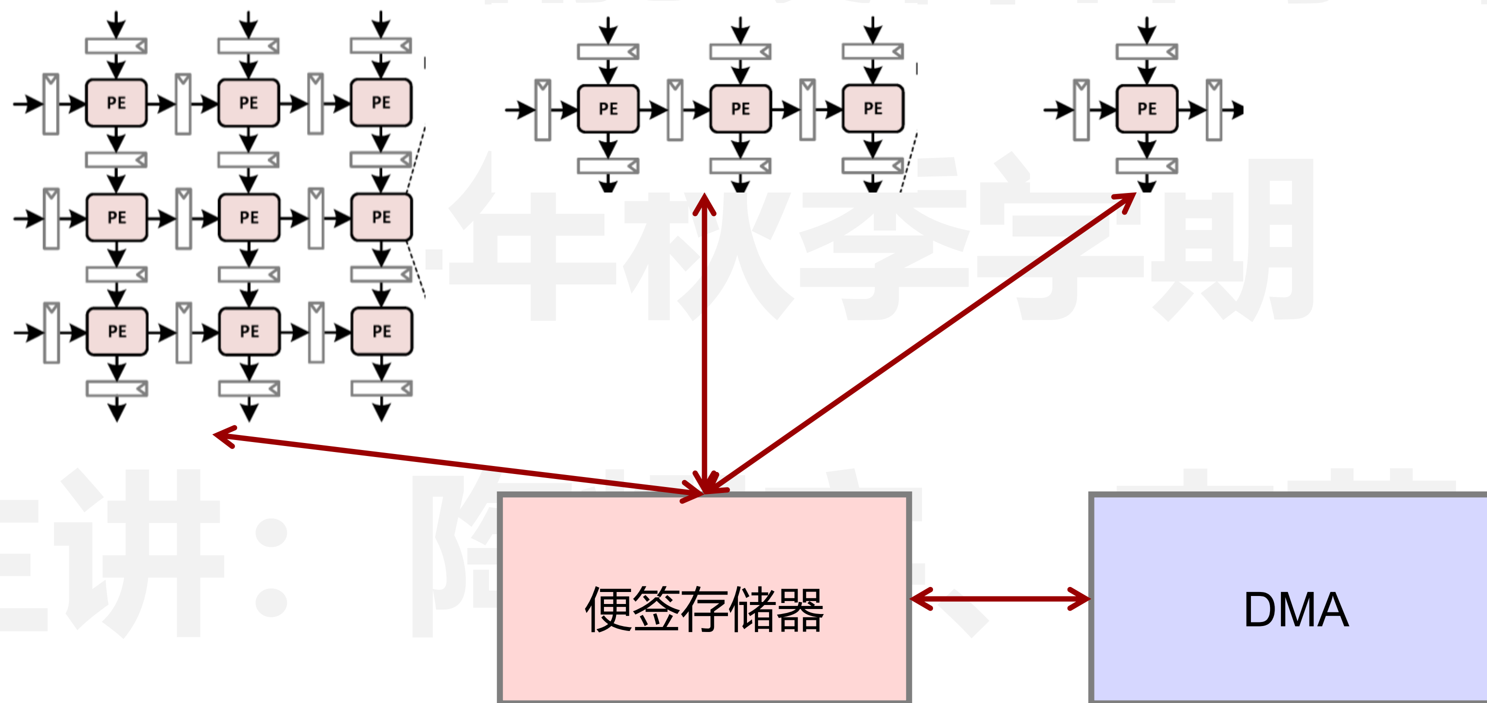
- **便签存储器 VS CPU缓存**
- **同等工艺下，CPU缓存面积约为便签存储器的1.5倍**

北京大学-智能硬件体系结构

便签存储器	CPU缓存
软件控制	硬件控制
软件编程	复杂的缓存管理策略 + 相应硬件支持
访存方式较为固定，多为串行	访存方式多样、随机
访问速度快、功耗低、轻量化	功耗较高
以读取为主	读写比例不确定

AI加速器架构基础——片上访存

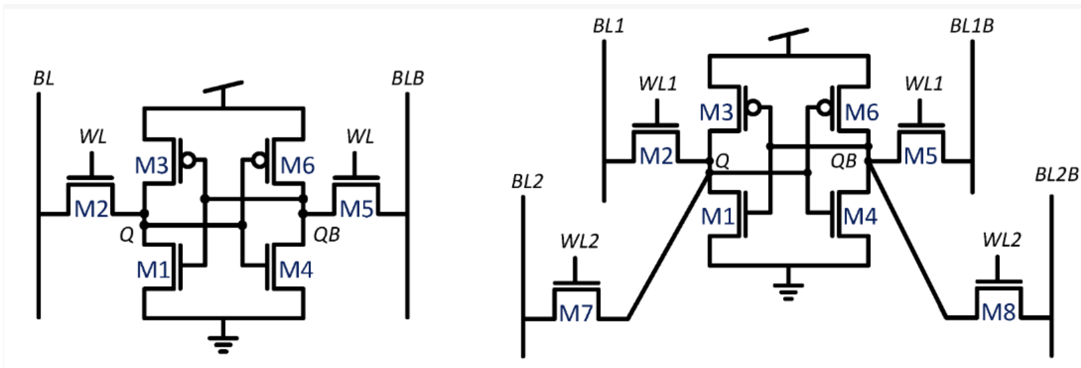
- AI芯片中，便签存储器需要连接矩阵运算单元、向量运算单元、寄存器、DMA/外存等
- 便签存储器是**数据传输的枢纽**，也非常容易造成读写冲突

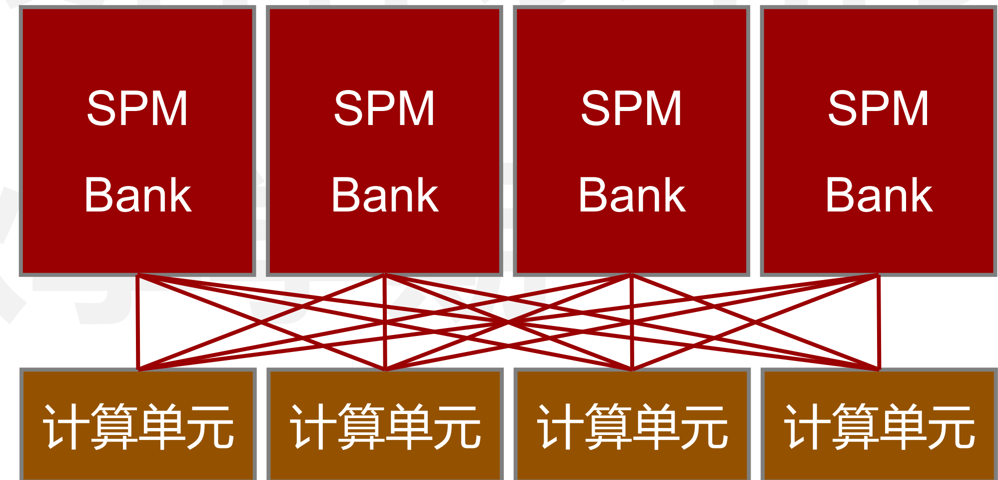


北京大学-智能硬件体系结构
年秋季学期
主讲：

AI加速器架构基础——片上访存

- AI芯片中，便签存储器需要连接矩阵运算单元、向量运算单元、寄存器、DMA/外存等
- 便签存储器是**数据传输的枢纽**，也非常容易造成读写冲突
- 为了缓解便签存储器的读写冲突，有以下方法





多端口SRAM：

- 增加一个端口，面积增长50%~100%
- 面积进一步影响成本、能效、演示

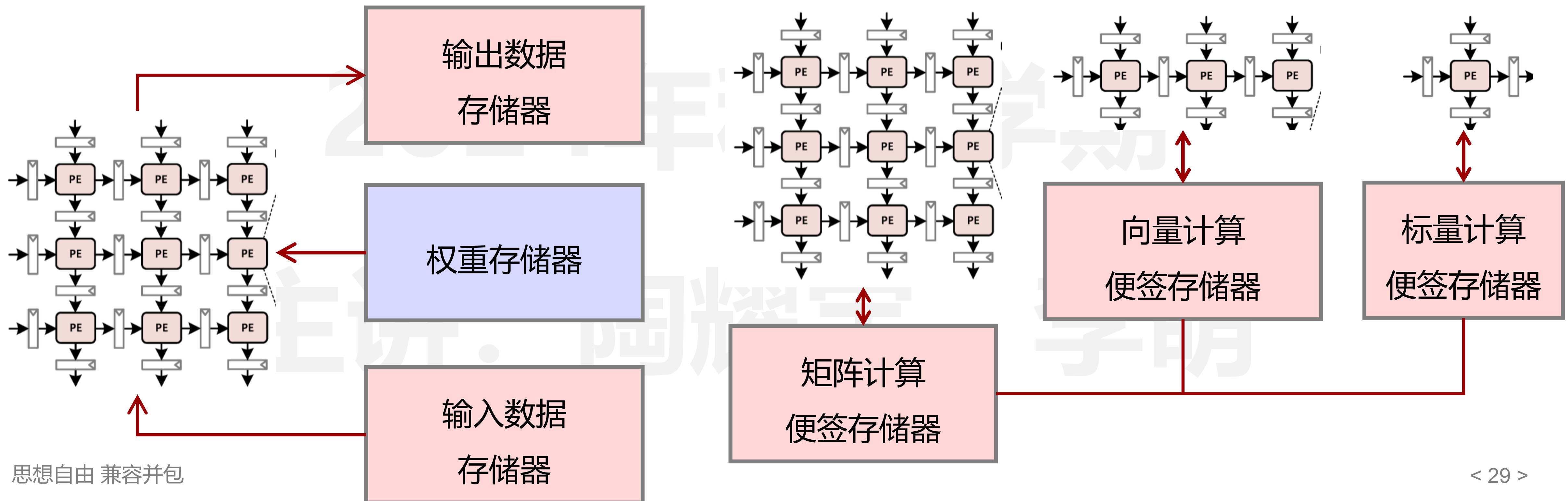
分组SRAM：

- 仍然存在bank conflict
- 连线复杂度提升~ $O(\text{分组数量}^2)$

北京大学 PEKING UNIVERSITY
北京大学-智能硬件体系结构 主讲：陶耀宇、李萌

AI加速器架构基础——片上访存

- 此外，还可以模仿CPU中数据和指令分开存储（**哈佛架构**）的方式，采用**分离式**便签存储器
- 按照**数据**划分，例如输入数据/输出数据/权重、输入输出数据/权重
- 按照**功能单元**划分，例如矩阵/向量/标量
- 按照**处理阶段**划分，例如输入数据/累加器



AI加速器架构基础——片上访存

- 分离式便签存储器对于数据进行了分流，从而提高了处理效率
  - 对于不同便签存储器的使用方式进行了约束，损失了架构的通用性

"Computer architecture is all about making trade-offs"

北京大学-智能硬件体系结构
主讲：陶耀宇、李萌

AI加速器架构基础——外部存储器访问

北京大学 PEKING UNIVERSITY

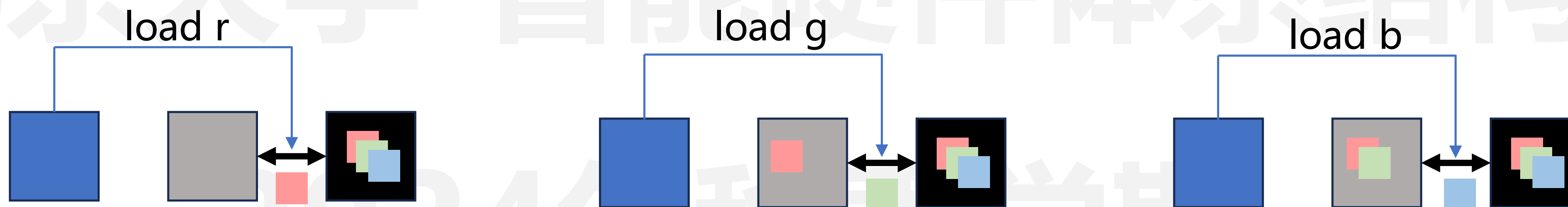
- AI芯片计算过程中，往往需要访问外部存储器（主要为DRAM）
  - 输入数据（例如ImageNet数据集$224 \times 224 \times 3$）的传输
  - 模型权重的传输（ResNet50模型参数为25.6M，可以完全存储于片上；大模型的模型参数为Billion以上量级，难以完全存储于片上）
  - 神经网络中间计算结果
- 对于传统CPU，需要执行30万条load/store指令，才能完成ImageNet数据读取
- 对于AI处理器，我们希望
  - 1条load指令装载一整张图片，1条指令完成计算，1条store指令送回内存

北京大学-智能硬件体系结构 2024年秋季学期 主讲：陶耀宇、李萌

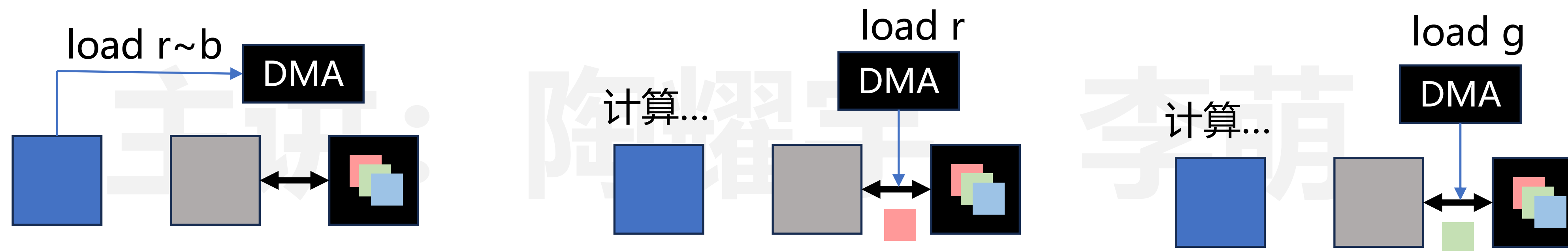
AI加速器架构基础——外部存储器访问

北京大学 PEKING UNIVERSITY

- 如何实现“1条load指令装载整张图像”？
- 处理器控制



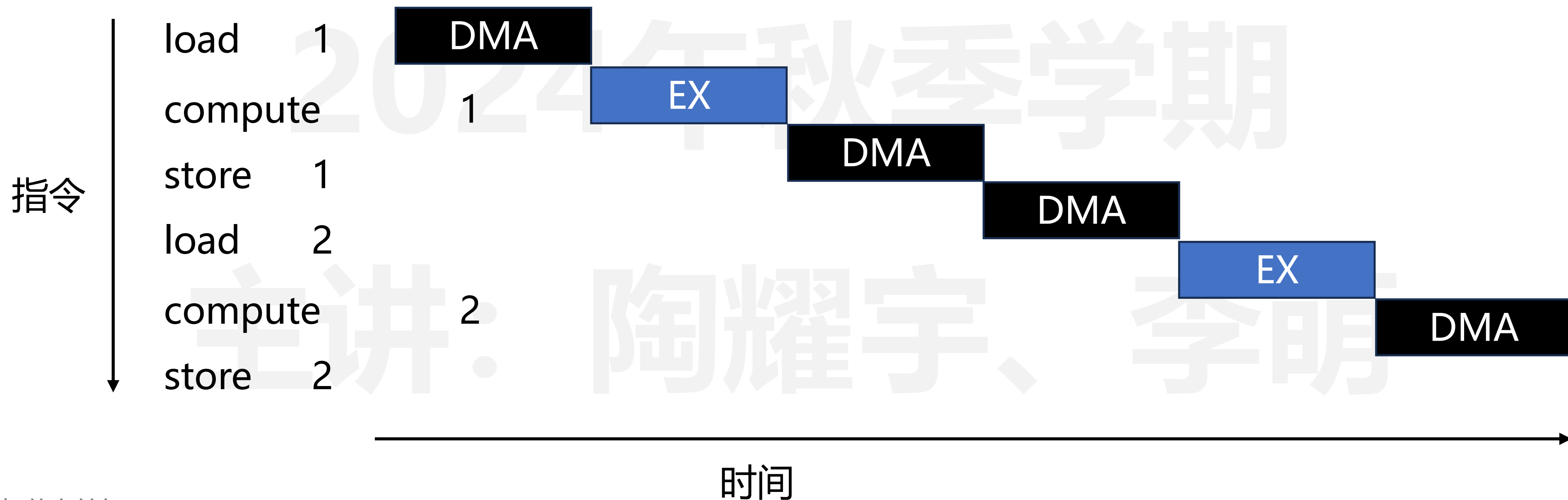
- DMA控制



北京大学-智能硬件体系结构 2024年秋季学期 主讲：陶耀宇 李萌

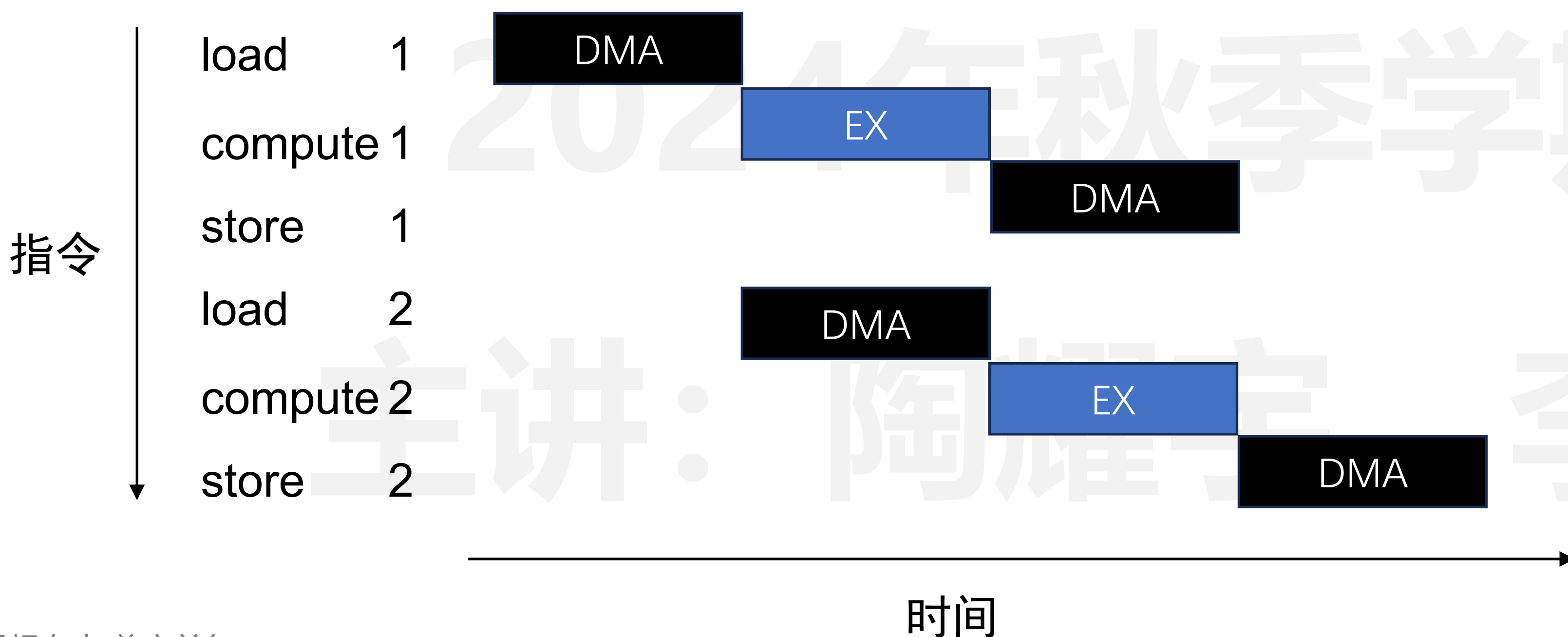
AI加速器架构基础——外部存储器访问

- DMA（direct memory access）：一种计算机系统中的数据传输机制，允许硬件子系统在**不经过CPU干预**的情况下，直接与主存进行数据交换
  - 能够显著减轻CPU负担，并且提高数据传输效率
- 针对AI处理器，单次DMA操作可能持续数百到数十万个周期



北京大学 智能硬件体系结构 2024年秋季学期 主讲：陶耀宇、李萌

- DMA（direct memory access）：一种计算机系统中的数据传输机制，允许硬件子系统在不经过CPU干预的情况下，直接与主存进行数据交换
  - 能够显著减轻CPU负担，并且提高数据传输效率
- 针对AI处理器，单次DMA操作可能持续数百到数十万个周期



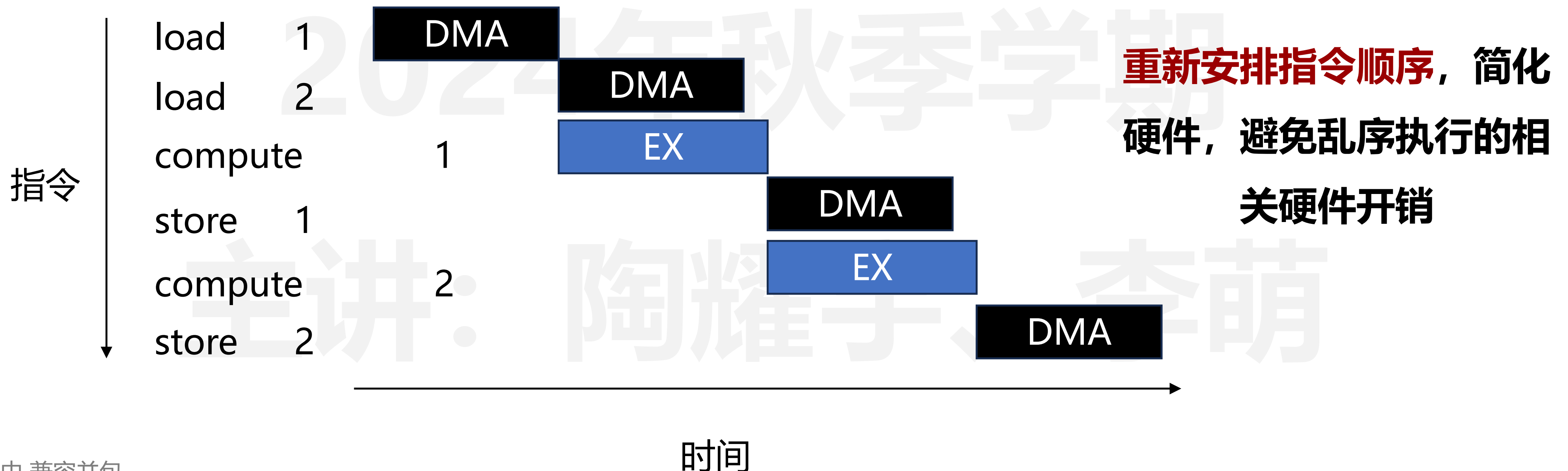
实现乱序执行提升不同硬件部分的利用率，有必要吗？有哪些硬件代价？

北京大学 智能硬件体系结构 2024年秋季学期 主讲：陶耀宇 李萌

AI加速器架构基础——外部存储器访问

- DMA（direct memory access）：一种计算机系统中的数据传输机制，允许硬件子系统在不经过CPU干预的情况下，直接与主存进行数据交换
  - 能够显著减轻CPU负担，并且提高数据传输效率
- 针对AI处理器，单次DMA操作可能持续数百到数十万个周期

北京大学 智能硬件体系结构 2024年秋季学期 主讲：陶耀宇、李萌



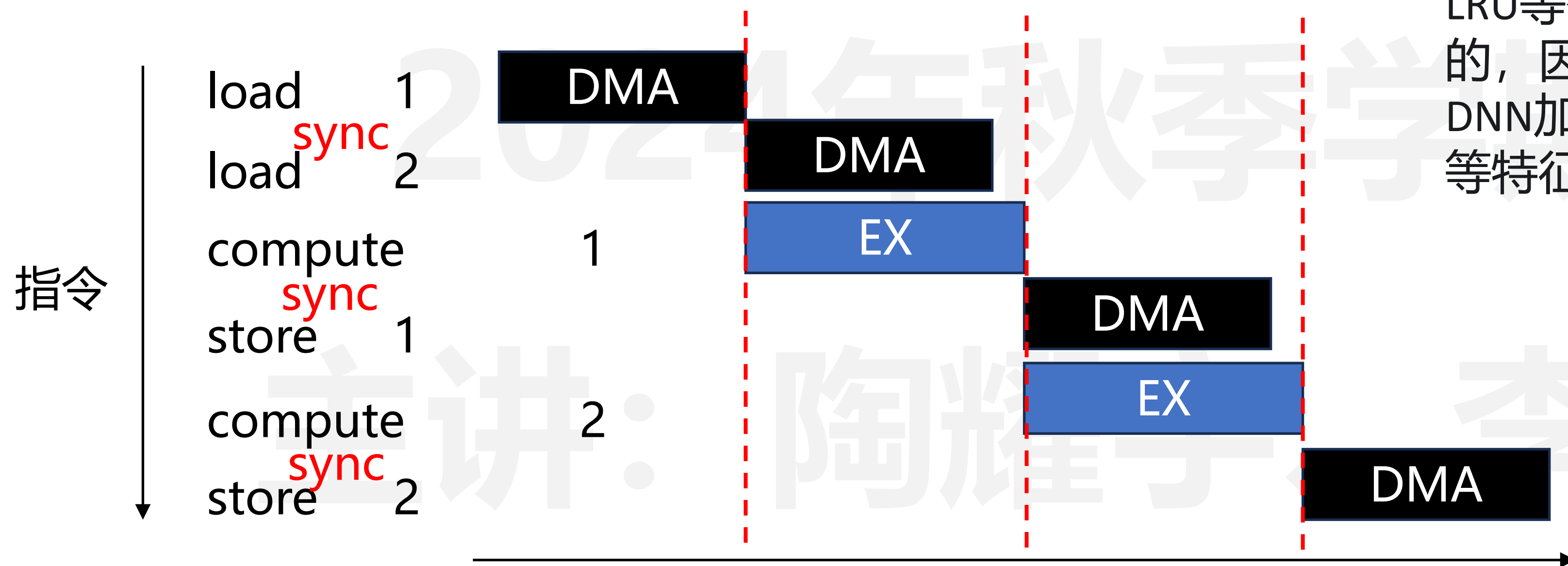
重新安排指令顺序，简化硬件，避免乱序执行的相关硬件开销

AI加速器架构基础——外部存储器访问

北京大学 PEKING UNIVERSITY

- **DMA（direct memory access）：一种计算机系统中的数据传输机制，允许硬件子系统在不经过CPU干预的情况下，直接与主存进行数据交换**
  - **能够显著减轻CPU负担，并且提高数据传输效率**
- **针对AI处理器，单次DMA操作可能持续数百到数十万个周期**

CPU Cache可以被描述为执行隐式式数据控制同步，因为加载请求发起者不直接控制响应数据是否保留的决策，也不直接控制何时删除响应数据。LRU等替换策略在通用场景中是有利的，因为它们与工作负载无关。对于DNN加速器来说，标签匹配和关联集等特征的面积和能量开销很高

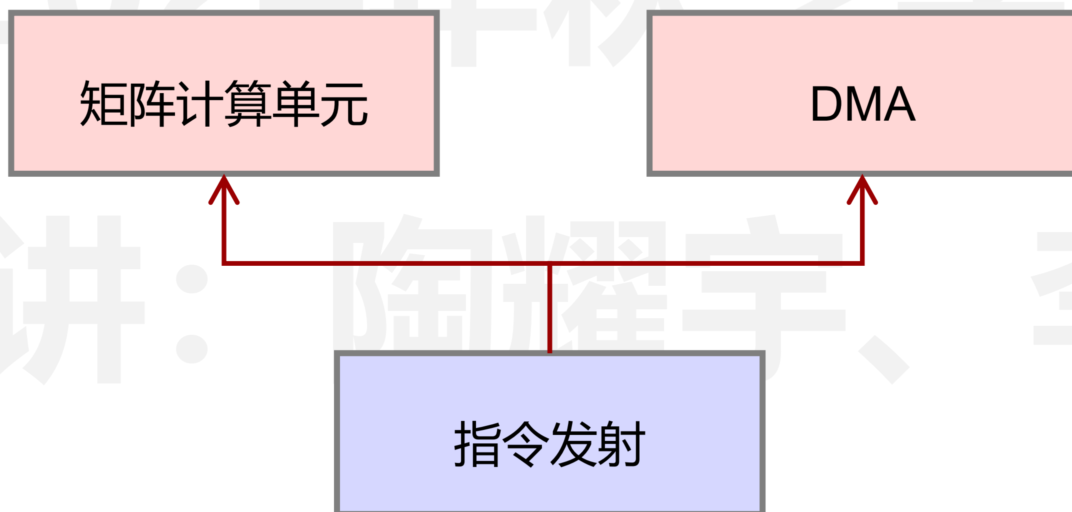


显式控制同步，硬件省去依赖检查，软件控制数据流

北京大学 智能硬件体系结构 2024年秋季学期 主讲：陶耀宇 李萌

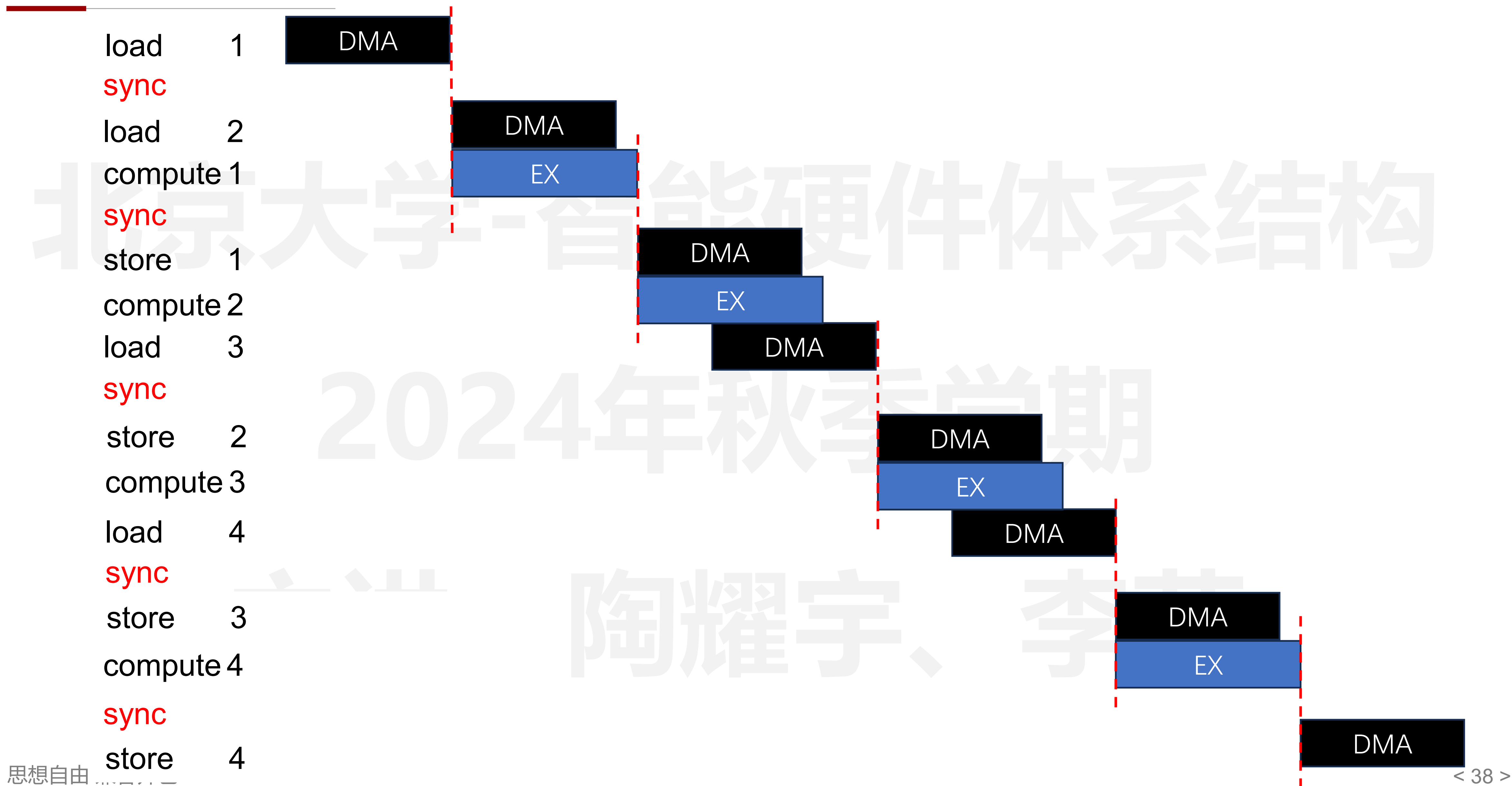
AI加速器架构基础——外部存储器访问

- **软件定义数据流**
- **如何实现同步指令（sync）？**
  - **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：将计算指令发射到计算模块，将访存指令发射到DMA模块，遇到sync时，阻塞直到整个处理器空闲下来**



北京大学 智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

AI加速器架构基础——外部存储器访问



北京大学-智能硬件体系结构
2024年秋季学期
陶耀宇、李萌

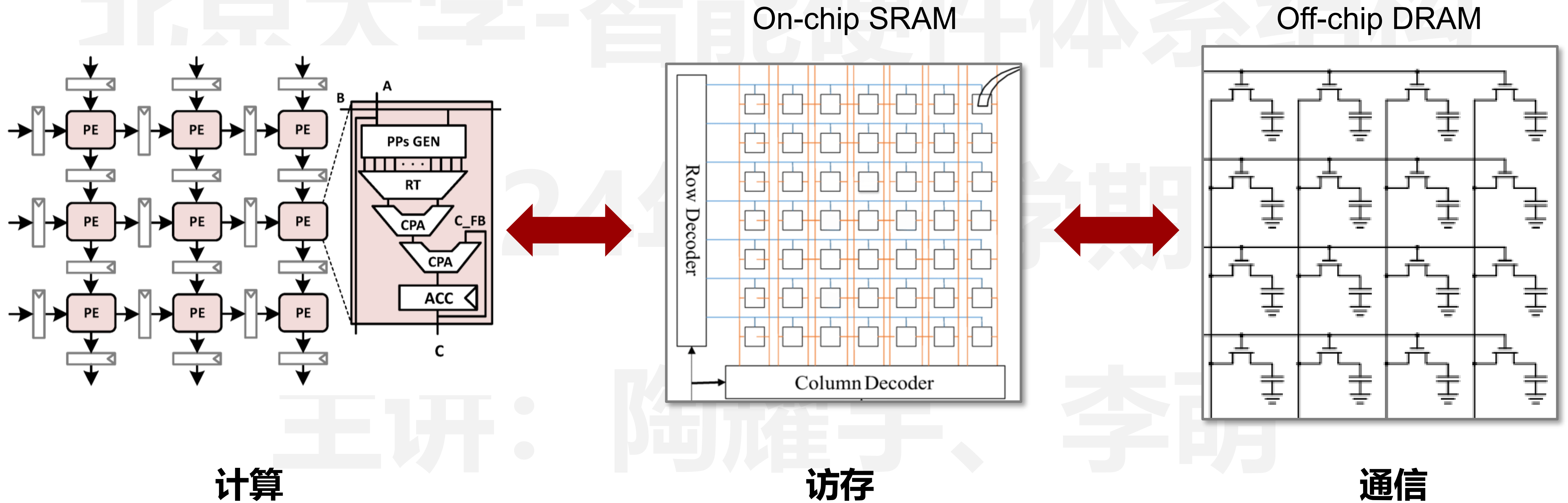
AI加速器架构基础——访存小结

北京大学 PEKING UNIVERSITY

- 便签存储器是数据传输的枢纽
  - 访问便签存储器成为瓶颈
  - 可以通过增加端口、设计分组SRAM的方法，实现更多读取支持（硬件开销增加）
  - 也可以根据算法特征，采用分离式设计
- 针对外部访存，设计DMA，并通过软件流水线，将计算和访存并行起来
  - 指令重新排序，不再需要乱序执行
  - 显示控制同步，省去硬件依赖检查

北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

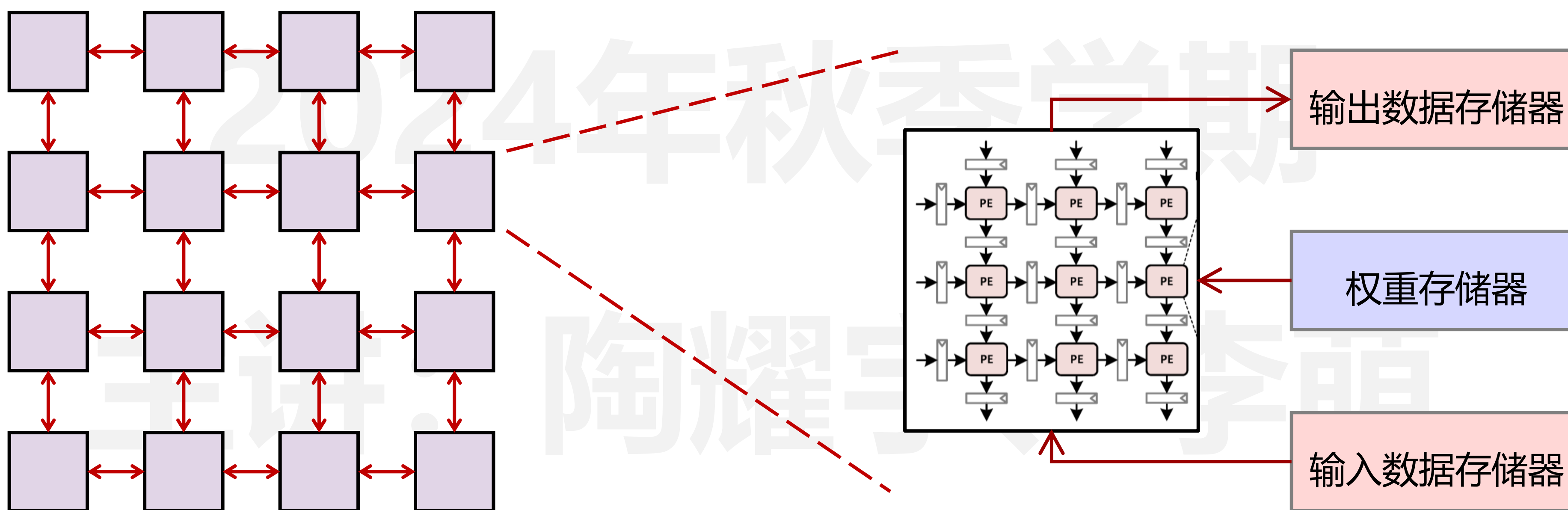
- AI加速器整体架构



北京大学-智能硬件体系结构 24年春季学期 主讲：陶耀宇、李萌

AI加速器架构基础 —— 通信

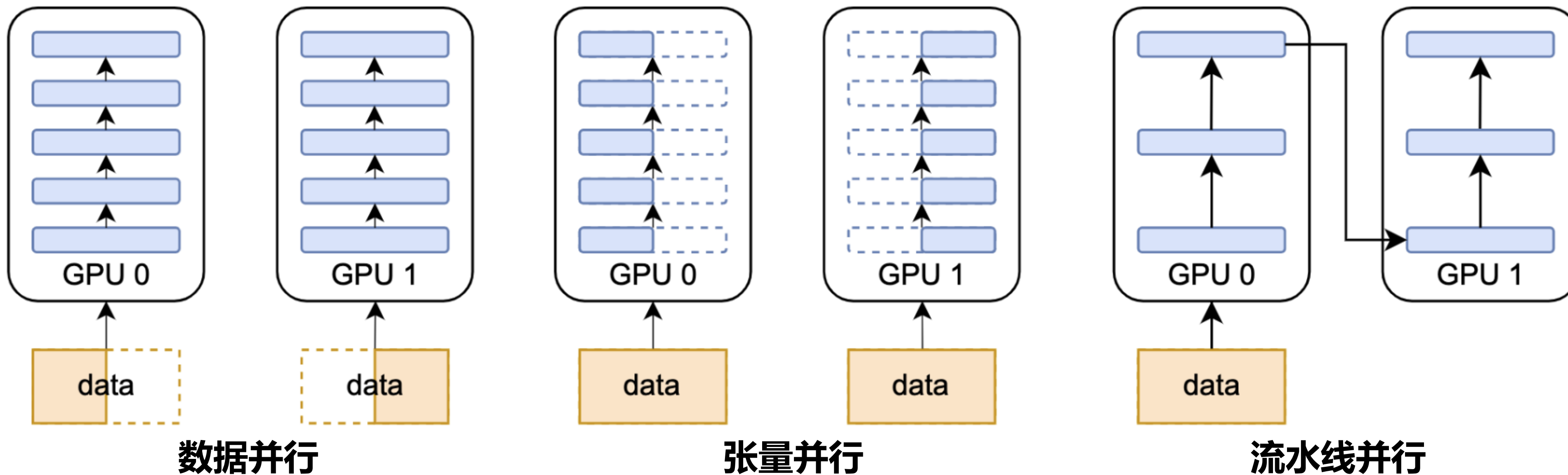
- 通信即包括计算核与**外存**的通信，也可以包括多核扩展中**计算核间**的通信
- 在上一讲中，我们介绍了不同的片上网络拓扑、路由设计等，本节课，我们重点说明在AI加速器中需要哪些通信算子，以及为什么会有此类需求
- 计算核间的通信需求主要来源于**并行计算**，即需要多计算核共同完成神经网络的计算



北京大学·智能硬件体系结构 2024年秋季学期 主讲：陶耀宇、李萌

典型模型并行计算方式简介

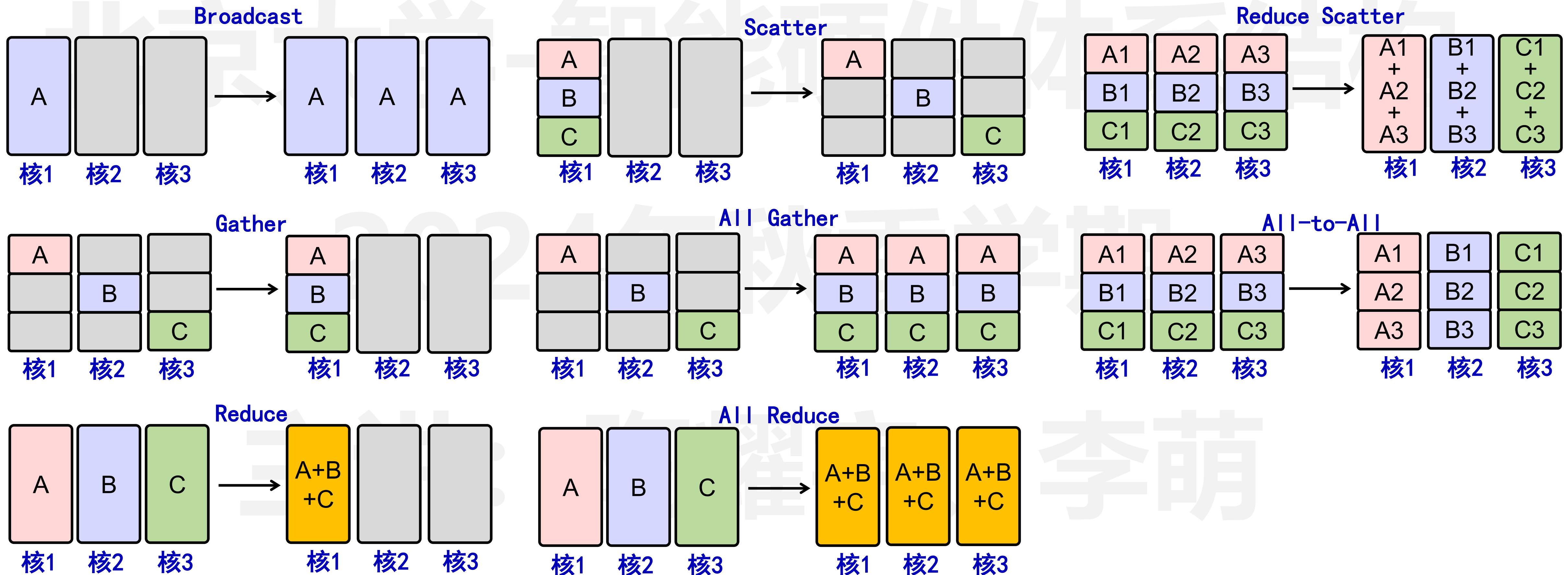
- 面向神经网络，典型的并行计算方式有以下几种



	数据并行	张量并行	流水线并行
部署方式	**模型重复存储 各组部署相互独立**	**单个weight分配至多个芯片 各计算单元结果需要累加**	**不同weight分配到不同芯片阵列 各计算单元结果存在数据依赖**
硬件特点	存算芯片算力较大模型较小	存算芯片算力较小模型较大	常规部署策略
推理特点	Batch 较大/卷积	无固定需求	Batch较大/算力有限

典型集合通信算子

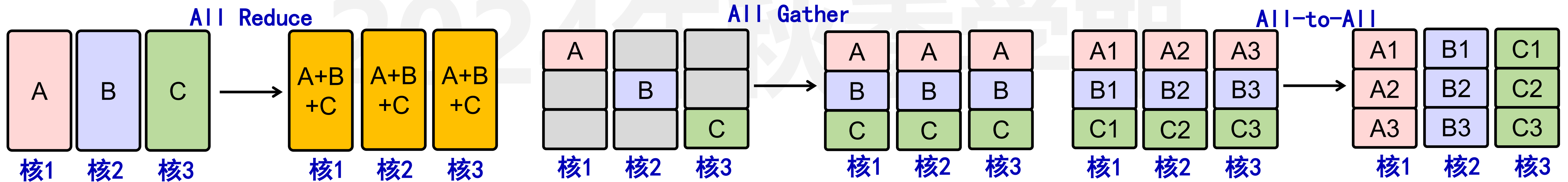
- **集合通信**：并行计算领域中的一种通信操作类型，用于多个并行计算单元（多个处理器、多个计算节点）间的数据交互和同步



典型集合通信算子

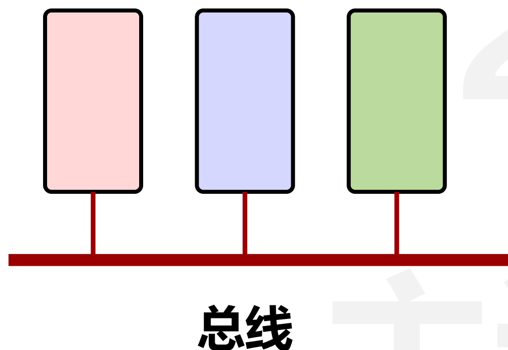
- 大部分的集合通信操作，可以通过二维环（Ring）互连拓扑实现
- 是否说明二维环就足够了呢?

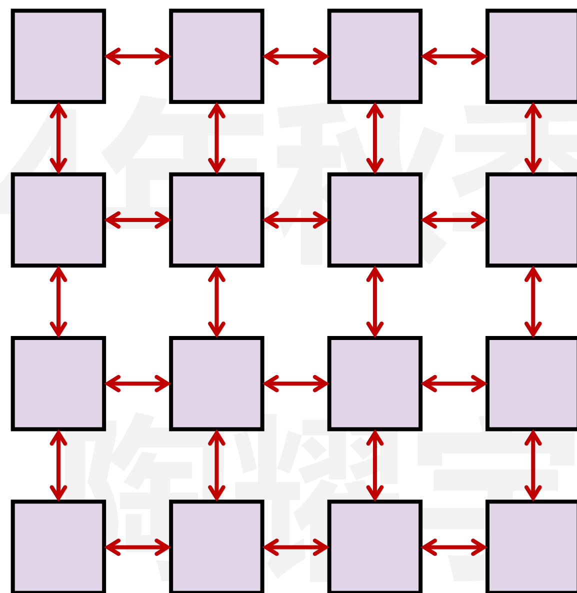
北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

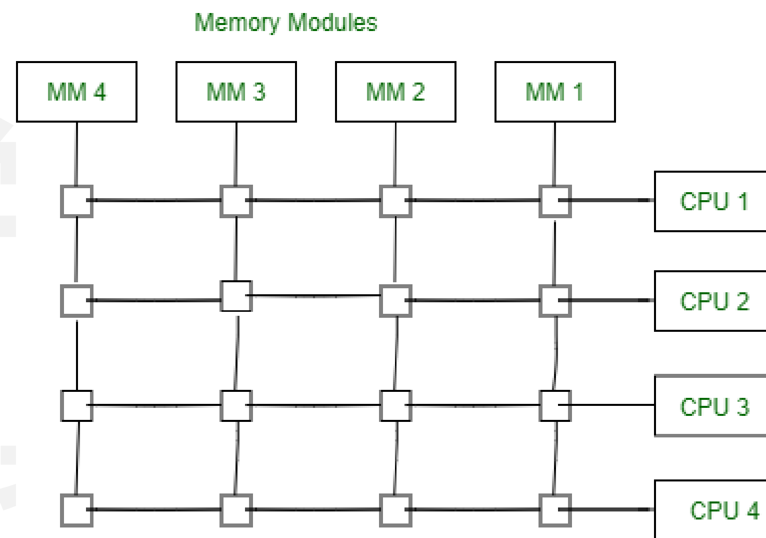


AI加速器架构基础 —— 通信

- 常见的物理链路设计：
  - 总线：常用AXI、PCIe等标准，性能差、成本低
  - 片上网络：常用胖树、二维环、二维网格等拓扑，性能较好、成本可控
  - 交叉开关：性能最佳，但是成本很高，难以拓展


总线

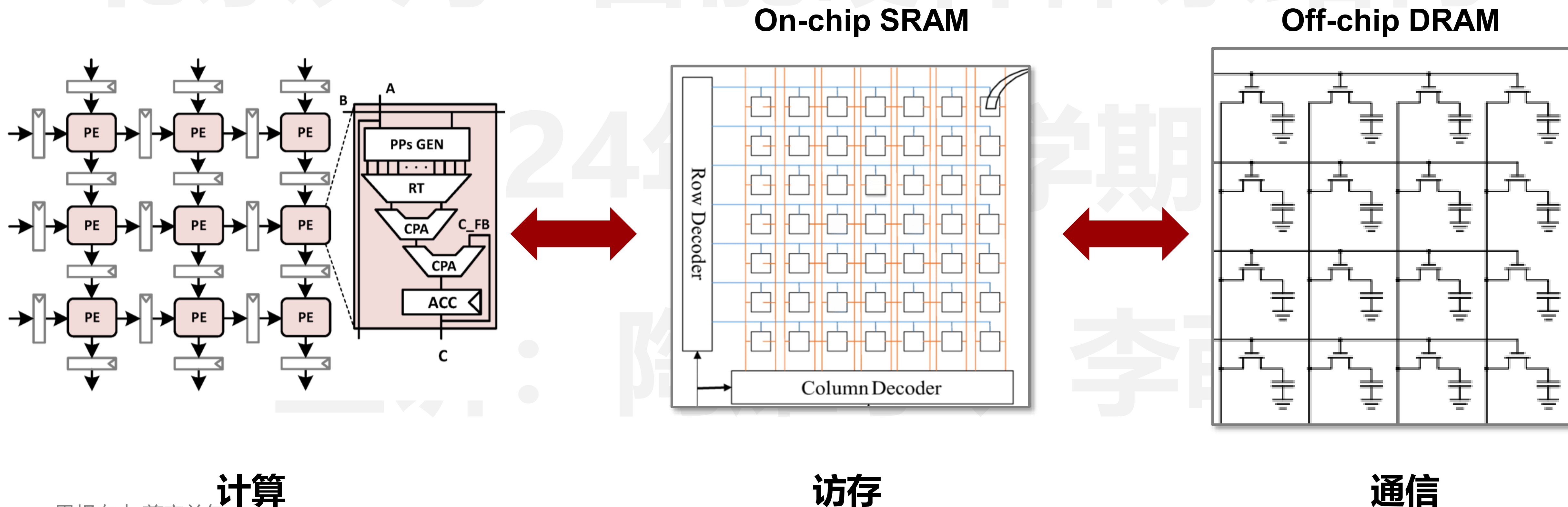

片上网络



交叉开关

AI加速器架构基础-Recap

- 我们介绍了AI加速器的基础架构，并着重介绍了AI加速器中的计算单元、片上访存以及片间/片外通信
- 我们将介绍有代表性的**AI加速器架构**



北京大学-智能硬件体系结构

AI加速器架构发展——DianNao

北京大学 PEKING UNIVERSITY

- **ASPLOS 2014最佳论文**
- **DianNao对于神经网络加速器设计进行了系统讨论**
  - **如何设计计算阵列?**
  - **如何设计片上存储?**
  - **如何提升数据复用，降低访存开销?**
  - **如何灵活支持不同操作?**
- **DianNao面向MLP和CNN，相关参数如下**

北京大学-智能硬件 2024年秋季

Performance	452 GOPS
Area	3.02 mm2
Power	485 mW
Technology Node	65 nm
Baseline	117.87X speedup over SIMD processor

DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning

Tianshi Chen, SKLCA, ICT, China
Zidong Du, SKLCA, ICT, China
Ninghui Sun, SKLCA, ICT, China
Jia Wang, SKLCA, ICT, China
Chengyong Wu, SKLCA, ICT, China
Yunji Chen, SKLCA, ICT, China
Olivier Temam, Inria, France

Abstract

Machine-Learning tasks are becoming pervasive in a broad range of domains, and in a broad range of systems (from embedded systems to data centers). At the same time, a small set of machine-learning algorithms (especially Convolutional and Deep Neural Networks, i.e., CNNs and DNNs) are proving to be state-of-the-art across many applications. As architectures evolve towards heterogeneous multi-cores composed of a mix of cores and accelerators, a machine-learning accelerator can achieve the rare combination of efficiency (due to the small number of target algorithms) and broad application scope.

Until now, most machine-learning accelerator designs have focused on efficiently implementing the computational part of the algorithms. However, recent state-of-the-art CNNs and DNNs are characterized by their large size. In this study, we design an accelerator for large-scale CNNs and DNNs, with a special emphasis on the impact of memory on accelerator design, performance and energy.

We show that it is possible to design an accelerator with a high throughput, capable of performing 452 GOP/s (key NN operations such as synaptic weight multiplications and neurons outputs additions) in a small footprint of 3.02 mm^2 and 485 mW; compared to a 128-bit 2GHz SIMD processor, the accelerator is 117.87x faster, and it can reduce the total energy by 21.08x. The accelerator characteristics are obtained after layout at 65nm. Such a high throughput in a small footprint can open up the usage of state-of-the-art machine-learning algorithms in a broad set of systems and for a broad set of applications.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASPLOS '14, March 1–5, 2014, Salt Lake City, Utah, USA.
Copyright © 2014 ACM 978-1-4503-2305-5/14/03…$15.00.
http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2541940.2541967

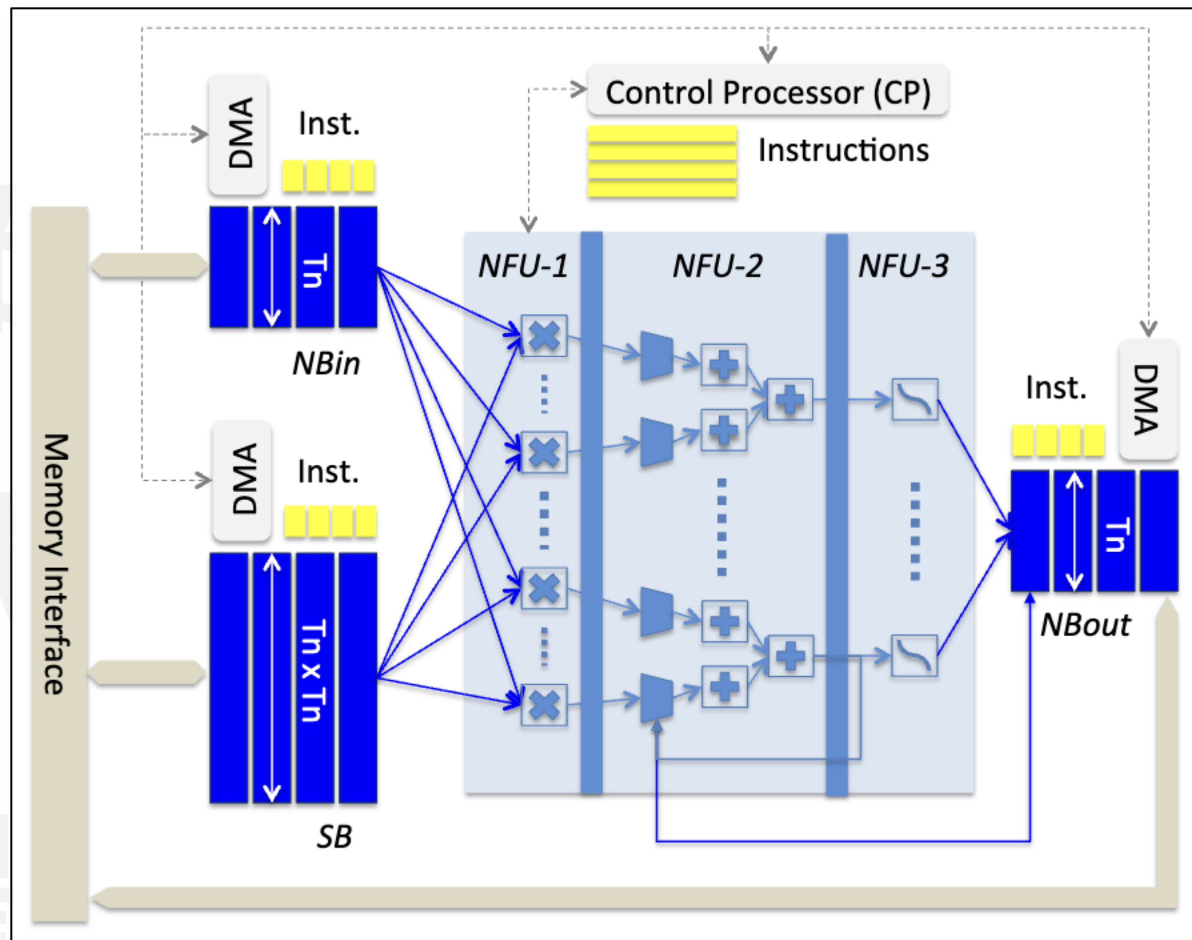
1. Introduction

As architectures evolve towards heterogeneous multi-cores composed of a mix of cores and accelerators, designing accelerators which realize the best possible tradeoff between flexibility and efficiency is becoming a prominent issue.

The first question is for which category of applications one should primarily design accelerators ? Together with the architecture trend towards accelerators, a second simultaneous and significant trend in high-performance and embedded applications is developing: many of the emerging high-performance and embedded applications, from image/video/audio recognition to automatic translation, business analytics, and all forms of robotics rely on *machine-learning techniques*. This trend even starts to percolate in our community where it turns out that about half of the benchmarks of PARSEC [2], a suite partly introduced to highlight the emergence of new types of applications, can be implemented using machine-learning algorithms [4]. This trend in application comes together with a third and equally remarkable trend in machine-learning where a small number of techniques, based on neural networks (especially Convolutional Neural Networks [27] and Deep Neural Networks

AI加速器架构发展——DianNao

- 如何进行神经网络加速器设计?
  - 直接实例化完整神经网络 vs 通用神经网络计算架构实现多算子支持
- 如何进行计算阵列设计?
- 如何设计访存?
  - High associative cache vs 便签存储器



北京大学-智能硬件 2024年秋 主讲：陶耀宇

AI加速器架构发展——DianNao

```
for (int nnn = 0; nnn ¡ Nn; nnn += Tnn)  {  // tiling for output neurons;
  for (int iii = 0; iii ¡ Ni; iii += Tii)  {  // tiling for input neurons;
    for (int nn = nnn; nn ¡ nnn + Tnn; nn += Tn)  {
      for (int n = nn; n ¡ nn + Tn; n++)
        sum[n] = 0;
      for (int ii = iii; ii ¡ iii + Tii; ii += Ti)
        // — Original code —
        for (int n = nn; n < nn + Tn; n++)
          for (int i = ii; i < ii + Ti; i++)
            sum[n] += synapse[n][i] ∗ neuron[i];
      for (int n = nn; n < nn + Tn; n++)
        neuron[n] = sigmoid(sum[n]);
} } }
```

全连接层

```
for (int yy = 0; yy ¡ Nyin; yy += Ty)  {
  for (int xx = 0; xx ¡ Nxin; xx += Tx)  {
    for (int nnn = 0; nnn ¡ Nn; nnn += Tnn)  {
      // — Original code — (excluding nn, ii loops)
      int yout = 0;
      for (int y = yy; y < yy + Ty; y += sy) {  // tiling for y;
        int xout = 0;
        for (int x = xx; x < xx + Tx; x += sx) {  // tiling for x;
          for (int nn = nnn; nn < nnn + Tnn; nn += Tn) {
            for (int n = nn; n < nn + Tn; n++)
              sum[n] = 0;
            // sliding window;
            for (int ky = 0; ky < Ky; ky++)
              for (int kx = 0; kx < Kx; kx++)
                for (int ii = 0; ii < Ni; ii += Ti)
                  for (int n = nn; n < nn + Tn; n++)
                    for (int i = ii; i < ii + Ti; i++)
                      // version with shared kernels
                      sum[n] += synapse[ky][kx][n][i]
                                  ∗ neuron[ky + y][kx + x][i];
                      // version with private kernels
                      sum[n] += synapse[yout][xout][ky][kx][n][i]}
                                  ∗ neuron[ky + y][kx + x][i];
            for (int n = nn; n < nn + Tn; n++)
              neuron[yout][xout][n] = non_linear_transform(sum[n]);
          } xout++; } yout++;
} } } }
```

卷积层

主讲：陶耀

AI加速器架构发展——DianNao

- 如何处理池化层?
  - 池化层和卷积层具有不同的访存需求
  - 通过局部转置的方式，在DMA读入输入激活值的时候，采用interleaving的方式，将输入激活值存储在便签存储器中
- 这种设计有没有什么问题?

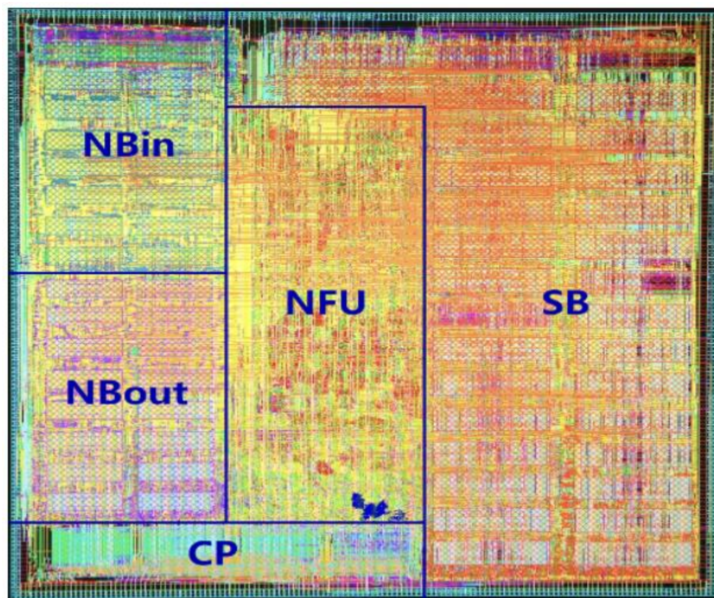
```
for (int yy = 0; yy ¡ Nyin; yy += Ty) {
  for (int xx = 0; xx ¡ Nxin; xx += Tx) {
    for (int iii = 0; iii ¡ Ni; iii += Tii)
      // — Original code — (excluding ii loop)
      int yout = 0;
      for (int y = yy; y < yy + Ty; y += sy) {
        int xout = 0;
        for (int x = xx; x < xx + Tx; x += sx) {
          for (int ii = iii; ii < iii + Tii; ii += Ti)
            for (int i = ii; i < ii + Ti; i++)
              value[i] = 0;
            for (int ky = 0; ky < Ky; ky++)
              for (int kx = 0; kx < Kx; kx++)
                for (int i = ii; i < ii + Ti; i++)
                  // version with average pooling;
                  value[i] += neuron[ky + y][kx + x][i];
                  // version with max pooling;
                  value[i] = max(value[i], neuron[ky + y][kx + x][i]);
        } } } }
        // for average pooling;
        neuron[xout][yout][i] = value[i] / (Kx * Ky);
        xout++; } yout++;
}}}
```

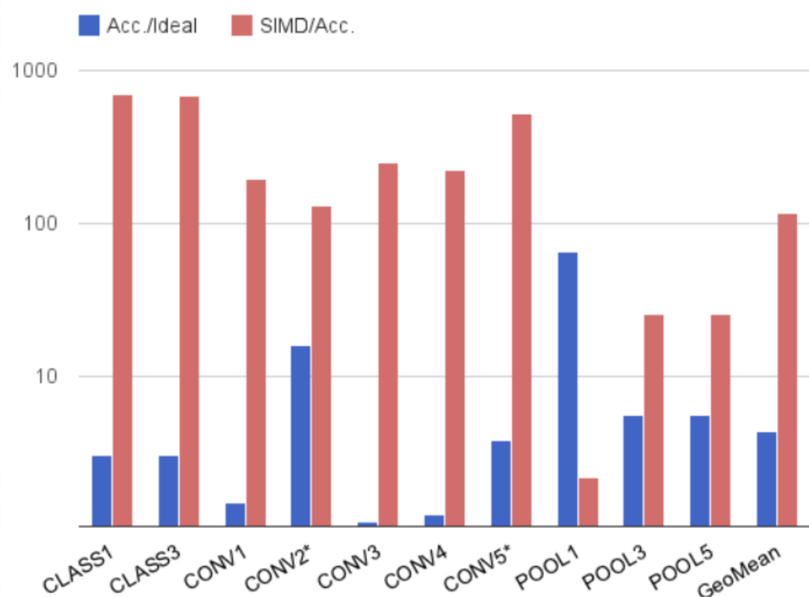
AI加速器架构发展——DianNao

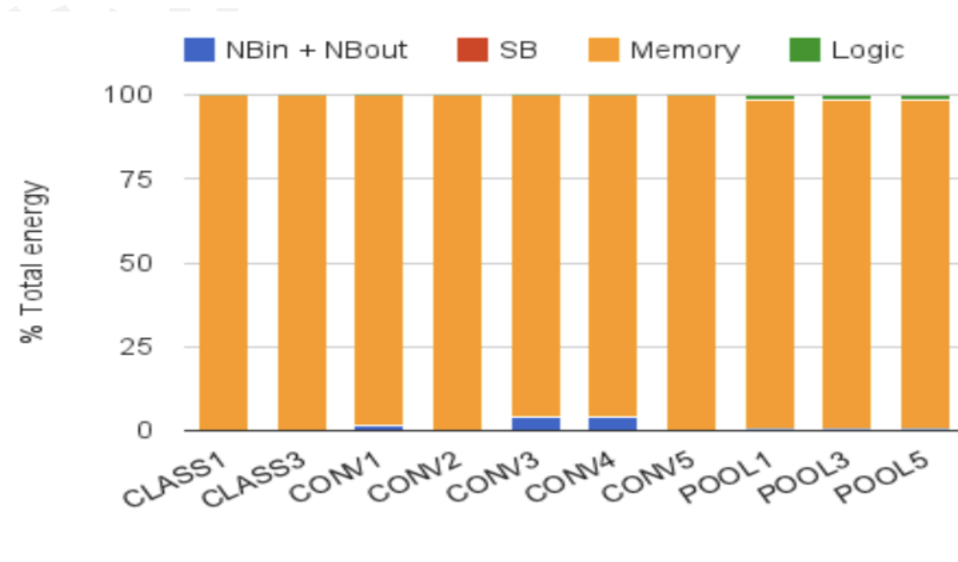
北京大学 PEKING UNIVERSITY

- 相较于CPU的SIMD计算，实现了显著的计算速度提升
- 但是，相较于理论加速比，仍然存在一定的差距，即硬件仍然面临欠利用问题
- 主要功耗来源于DRAM的访问

北京大学-智能硬件体系结构

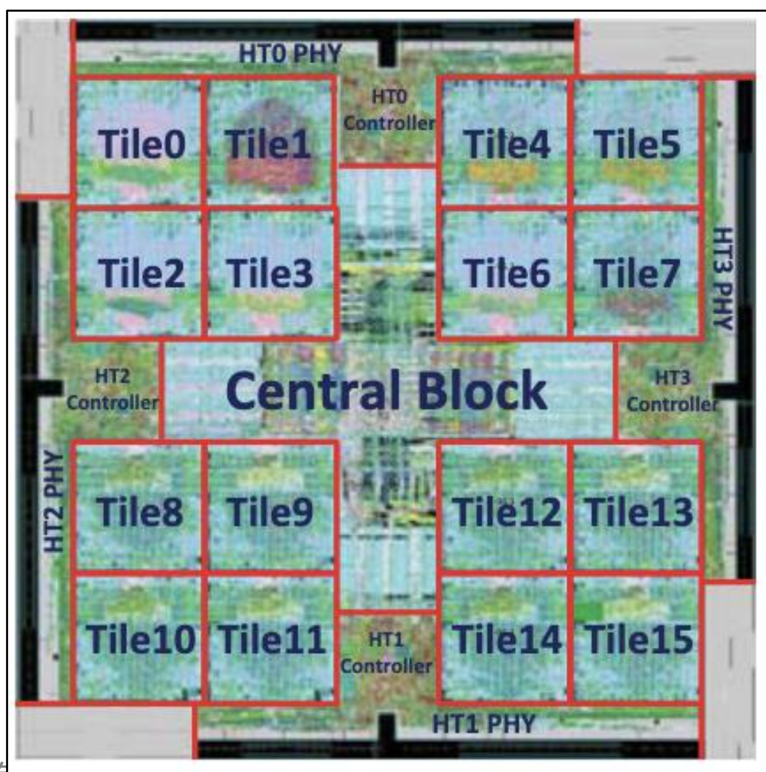






AI加速器架构发展——DaDianNao

- **MICRO 2014最佳论文**
- **Multi-chip神经网络加速器：机器学习超级计算机**



DaDianNao: A Machine-Learning Supercomputer

Yunji Chen[1], Tao Luo[1,3], Shaoli Liu[1], Shijin Zhang[1], Liqiang He[2,4], Jia Wang[1], Ling Li[1], Tianshi Chen[1], Zhiwei Xu[1], Ninghui Sun[1], Olivier Temam[2]

[1] SKL of Computer Architecture, ICT, CAS, China
[2] Inria, Scalay, France
[3] University of CAS, China
[4] Inner Mongolia University, China

***Abstract*—Many companies are deploying services, either for consumers or industry, which are largely based on machine-learning algorithms for sophisticated processing of large amounts of data. The state-of-the-art and most popular such machine-learning algorithms are Convolutional and Deep Neural Networks (CNNs and DNNs), which are known to be both computationally and memory intensive. A number of neural network accelerators have been recently proposed which can offer high computational capacity/area ratio, but which remain hampered by memory accesses.**

However, unlike the memory wall faced by processors on general-purpose workloads, the CNNs and DNNs memory footprint, while large, is not beyond the capability of the on-chip storage of a multi-chip system. This property, combined with the CNN/DNN algorithmic characteristics, can lead to high internal bandwidth and low external communications, which can in turn enable high-degree parallelism at a reasonable area cost. In this article, we introduce a custom multi-chip machine-learning architecture along those lines. We show that, on a subset of the largest known neural network layers, it is possible to achieve a speedup of 450.65x over a GPU, and reduce the energy by 150.31x on average for a 64-chip system. We implement the node down to the place and route at 28nm, containing a combination of custom storage and computational units, with industry-grade interconnects.

I. Introduction

Machine-Learning algorithms have become ubiquitous in a very broad range of applications and cloud services; examples include speech recognition, e.g., Siri or Google Now, click-through prediction for placing ads [27], face identification in Apple iPhoto or Google Picasa, robotics [20], pharmaceutical research [9] and so on. It is probably not exaggerated to say that machine-learning applications are in the process of displacing scientific computing as the major driver for high-performance computing. Early symptoms of this transformation are Intel calling for a refocus on Recognition, Mining and Synthesis applications in 2005 [14] (which later led to the PARSEC benchmark suite [3]), with Recognition and Mining largely corresponding to machine-learning tasks, or IBM developing the Watson supercomputer, illustrated with the Jeopardy game in 2011 [19].

Remarkably enough, at the same time this profound shift in applications is occurring, two simultaneous, albeit apparently unrelated, transformations are occurring in the machine-learning and in the hardware domains. Our community is well aware of the trend towards heterogeneous computing where architecture specialization is seen as a promising path to achieve high performance at low energy [21], provided we can find ways to reconcile architecture specialization and flexibility. At the same time, the machine-learning domain has profoundly evolved since 2006, where a category of algorithms, called Deep Learning (Convolutional and Deep Neural Networks), has emerged as state-of-the-art across a broad range of applications [33], [28], [32], [34]. In other words, at the time where architects need to find a good tradeoff between flexibility and efficiency, it turns out that just one category of algorithms can be used to implement a broad range of applications. In other words, there is a fairly unique opportunity to design highly specialized, and thus highly efficient, hardware which will benefit many of these emerging high-performance applications.

A few research groups have started to take advantage of this special context to design accelerators meant to be integrated into heterogeneous multi-cores. Temam [47] proposed a neural network accelerator for multi-layer perceptrons, though it is not a deep learning neural network, Esmaeilzadeh et al. [16] propose to use a hardware neural network called NPU for approximating any program function, though not specifically for machine-learning applications, Chen et al. [5] proposed an accelerator for Deep Learning (CNNs and DNNs). However, all these accelerators have significant neural network *size* limitations: either small neural networks of a few tens of neurons can be executed, or the neurons and synapses (i.e., weights of connections between neurons) intermediate values have to be stored in main memory. These two limitations are severe, respectively from a machine-learning or a hardware perspective.

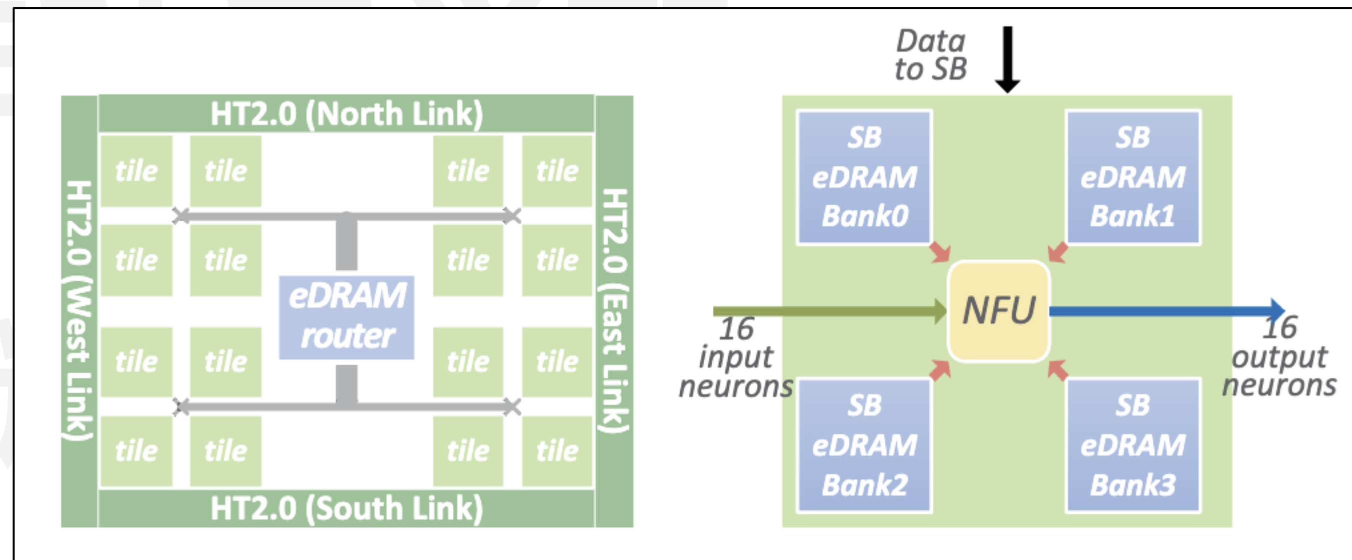
From a machine-learning perspective, there is a significant trend towards increasingly large neural networks. The recent work of Krizhevsky et al. [32] achieved state-of-the-art accuracy on the ImageNet database [13] with "only" 60

AI加速器架构发展——DaDianNao

- DaDianNao的设计特点：
  - 分块（tile）设计，模型权重存储靠近计算单元，采用4-bank eDRAM进行存储
  - 采用胖树的路由设计，进行输入和输出激活值的广播
  - 芯片中间设置2个eDRAM bank，进行输入和输出激活值的存储
  - 主要针对MLP和CNN设计，同时支持训练和推理

Parameters	Settings	Parameters	Settings
Frequency	606MHz	tile eDRAM latency	~3 cycles
# of tiles	16	central eDRAM size	4MB
# of 16-bit multipliers/tile	256+32	central eDRAM latency	~10 cycles
# of 16-bit adders/tile	256+32	Link bandwidth	6.4x4GB/s
tile eDRAM size/tile	2MB	Link latency	80ns

Table III: *Architecture characteristics.*



AI加速器架构发展——Eyeriss

- 2016年，MIT团队相继在HPCA、ISSCC、JSSC发表论文Eyeriss，对于学术界产生了重要影响
- Eyeriss在对于不同数据流进行系统比较的基础上，提出了新的row stationary数据流，并流片验证

Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks

Yu-Hsin Chen*, Joel Emer*† and Vivienne Sze*
*EECS, MIT Cambridge, MA 02139
†NVIDIA Research, NVIDIA Westford, MA 01886
*{yhchen, jsemer, sze}@mit.edu

14.5 Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks

Yu-Hsin Chen[1], Tushar Krishna[1], Joel Emer[1,2], Vivienne Sze[1]
[1]Massachusetts Institute of Technology, Cambridge, MA,
[2]Nvidia, Westford, MA

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.
IEEE JOURNAL OF SOLID-STATE CIRCUITS 1

Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks

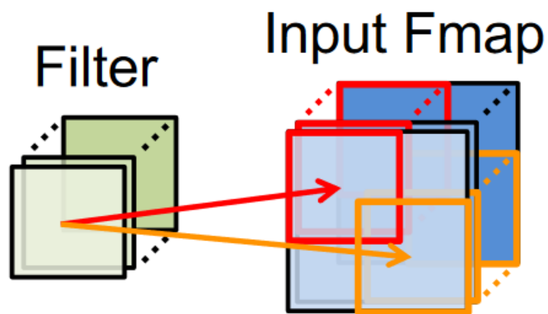
Yu-Hsin Chen, Student Member, IEEE, Tushar Krishna, Member, IEEE, Joel S. Emer, Fellow, IEEE, and Vivienne Sze, Senior Member, IEEE

AI加速器架构发展——Eyeriss

- 卷积计算中的数据复用模式
  - 通过权重和激活值的复用，可以显著降低访存需求，提升计算效率

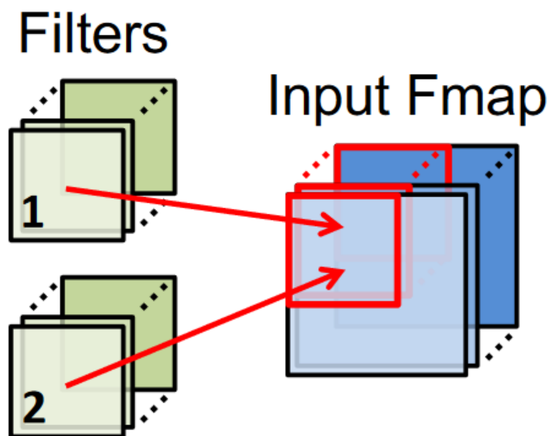
北京大学-智能硬件体系结构

卷积层



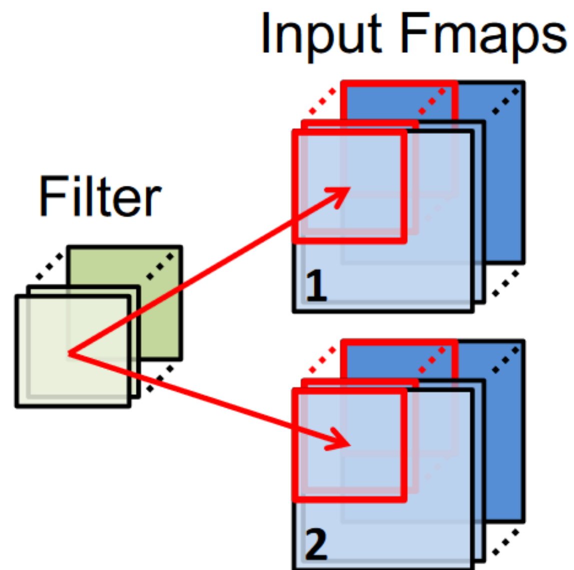
激活值和权重值的复用

卷积层和全连接层



激活值的复用

卷积层和全连接层

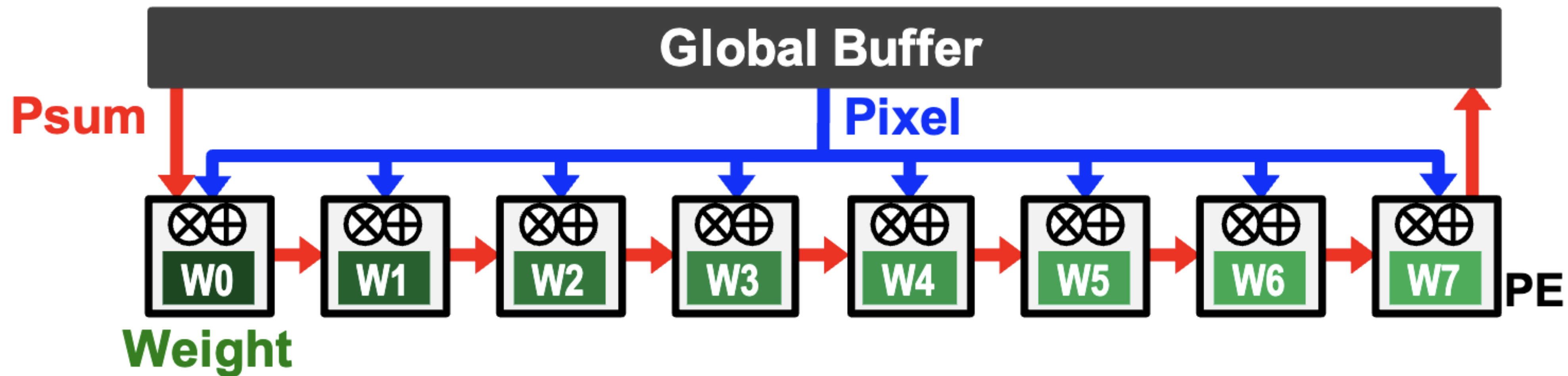


权重值的复用

AI加速器架构发展——Eyeriss

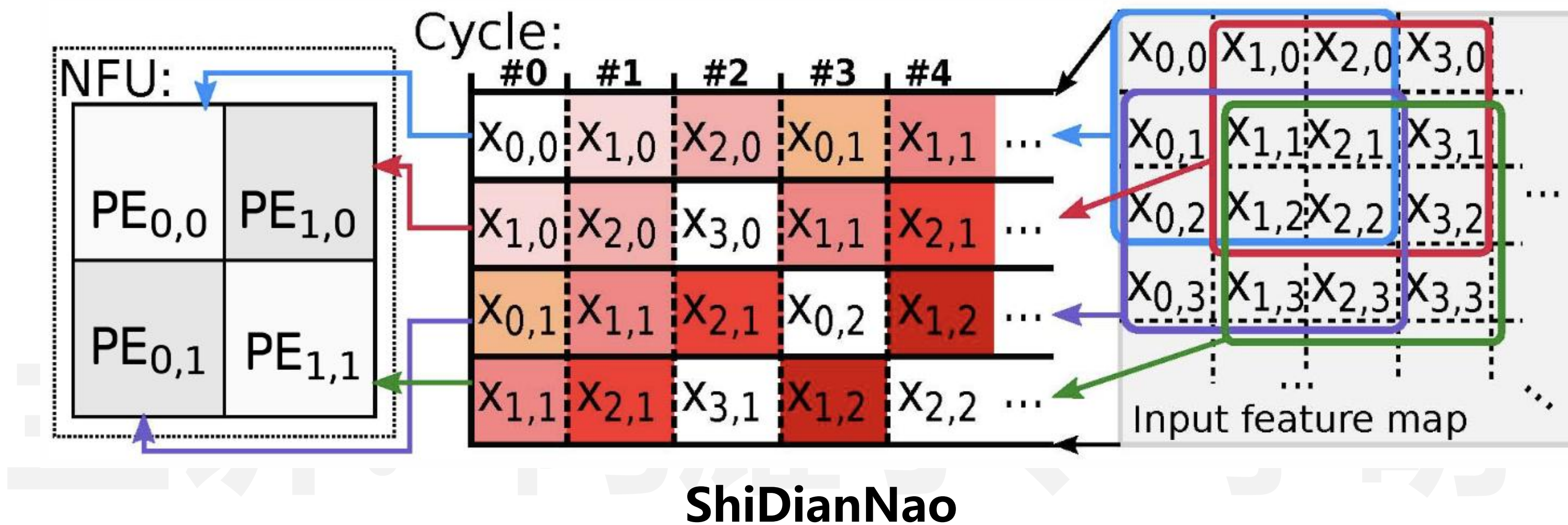
- Eyeriss把现有数据流加速器分为3类：
  - **权重静态**，代表工作包括TPU等，运算过程中，权重值被预先存储在PE中，最大化权重复用

北京大学-智能硬件体系结构



AI加速器架构发展——Eyeriss

- Eyeriss把现有数据流加速器分为3类：
  - **权重静态**，代表工作包括TPU等，权重值被预先存储在PE中，最大化权重复用
  - **输出静态**，代表工作包括ShiDianNao等，输出值停留在PE中被累加，最小化输出的读写开销

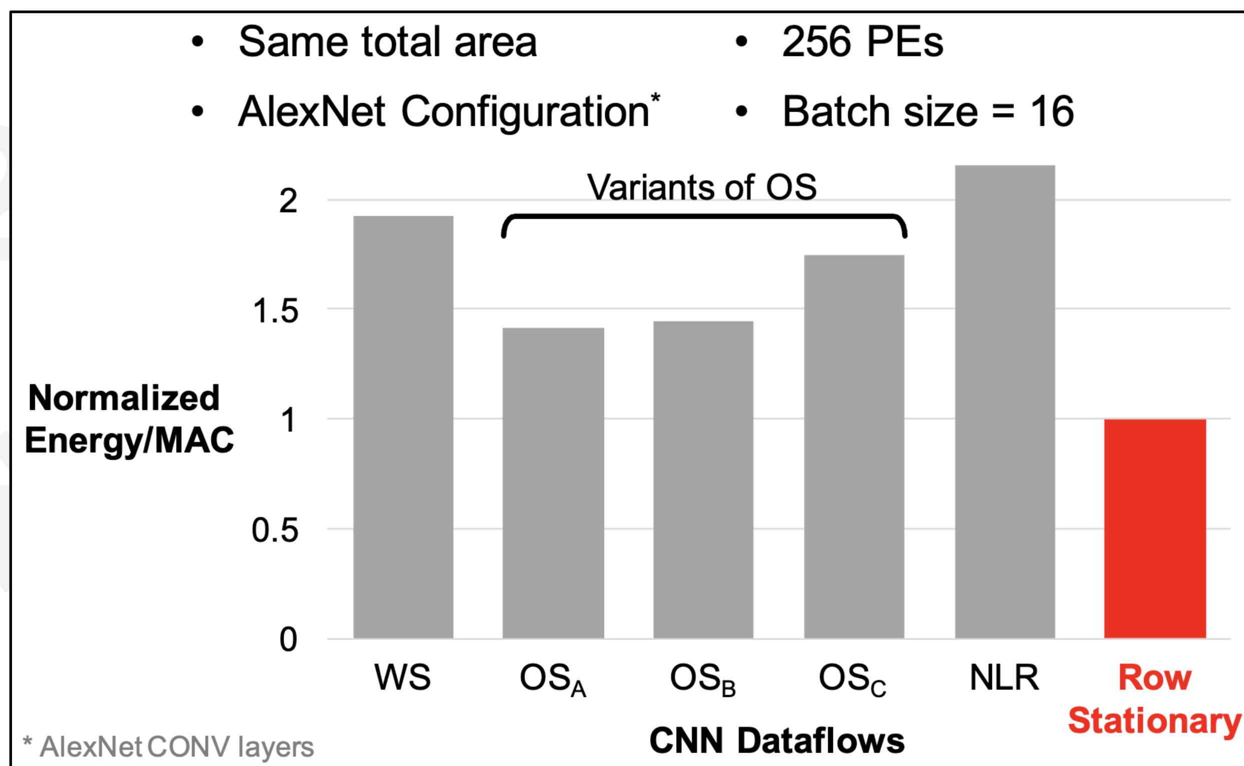


ShiDianNao

北京大学-智能硬件体系结构

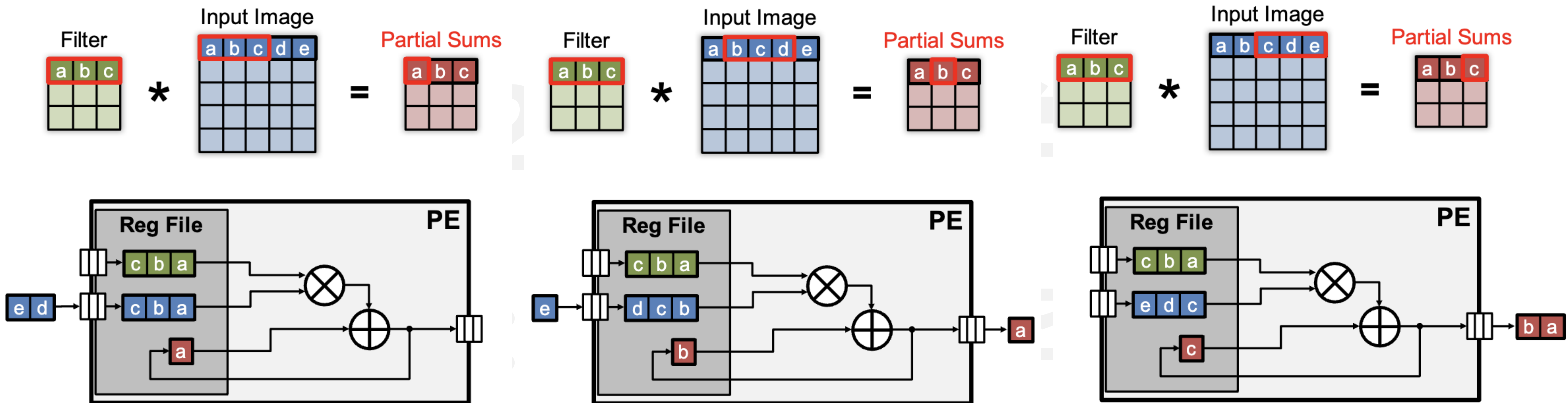
AI加速器架构发展——Eyeriss

- Eyeriss把现有数据流加速器分为3类：
  - **权重静态**，代表工作包括TPU等，权重值被预先存储在PE中，最大化权重复用
  - **输出静态**，代表工作包括ShiDianNao等，输出值停留在PE中被累加，最小化输出的读写开销
  - **No local reuse**，代表工作包括DianNao、DaDianNao等



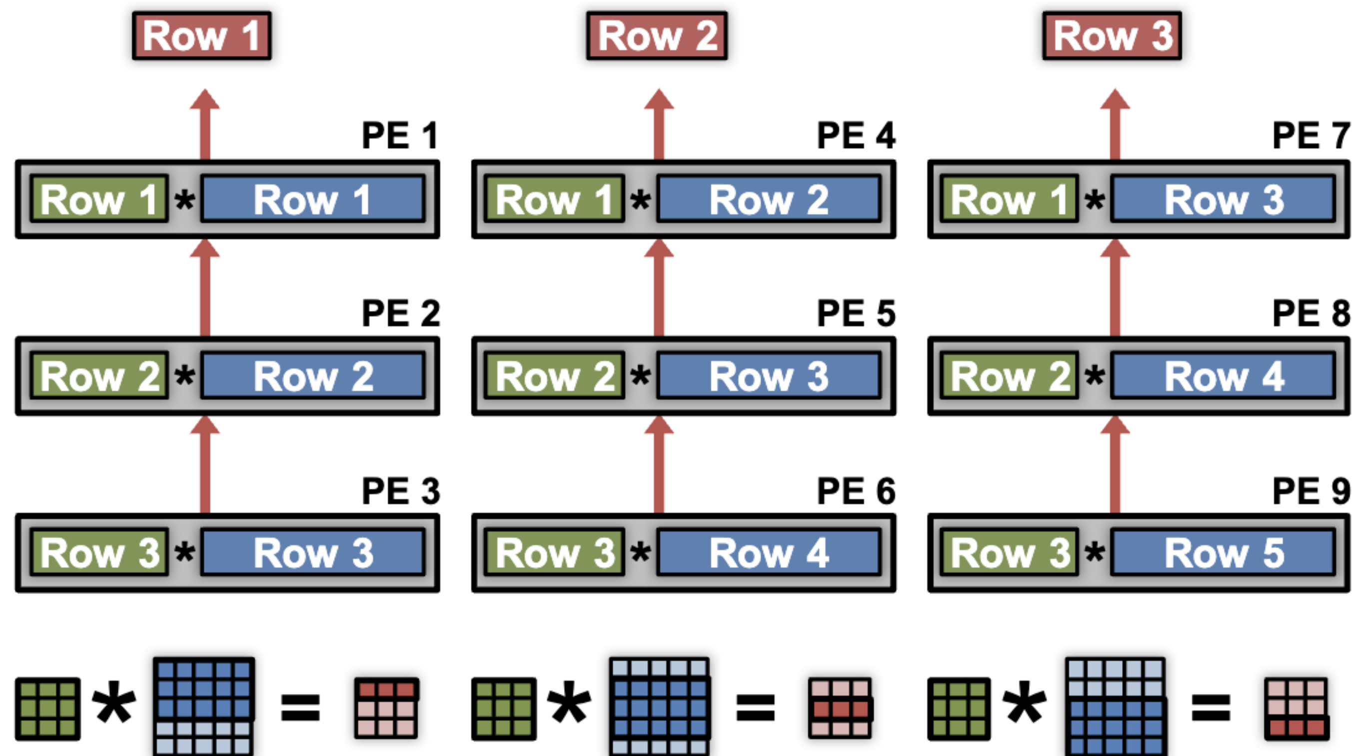
AI加速器架构发展——Eyeriss

- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用



AI加速器架构发展——Eyeriss

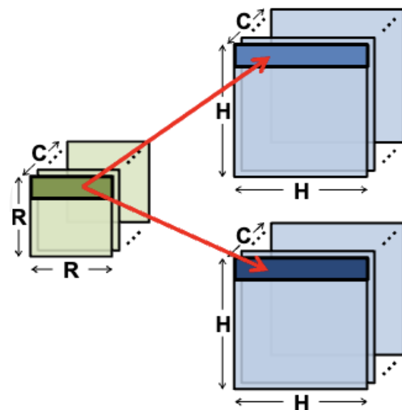
- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用
  - 不同计算单元间，进一步复用权重、输入和输出

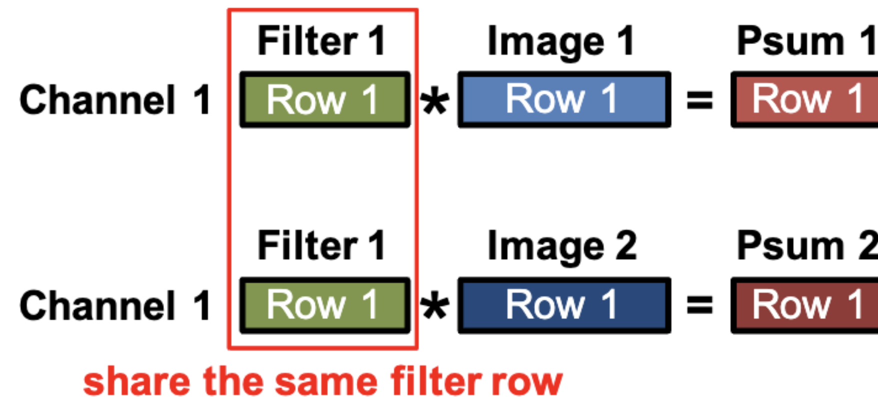


北京大学-智能硬件体系结构

AI加速器架构发展——Eyeriss

- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用
  - 不同计算单元间，进一步复用权重、输入和输出
  - 进一步拓展到多输入图像

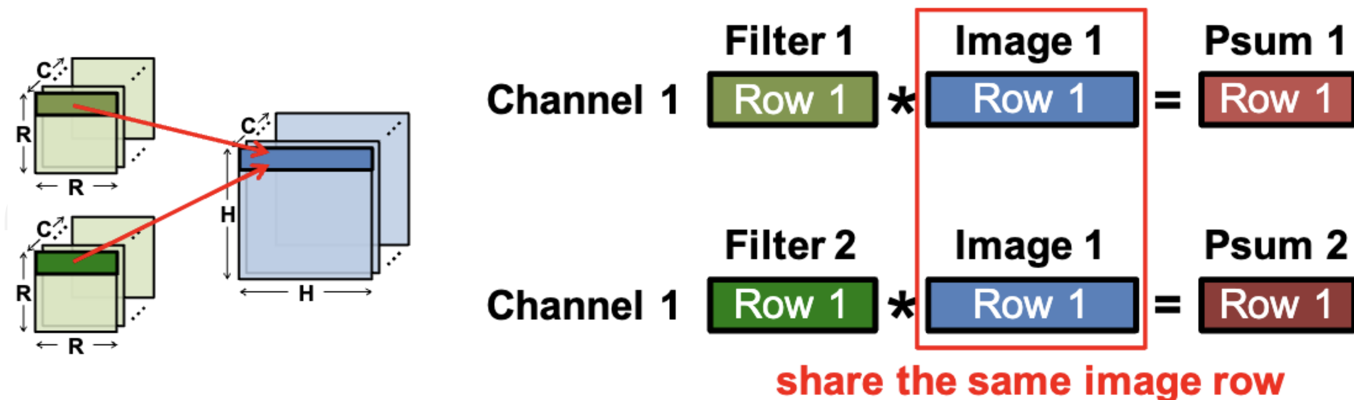




Processing in PE: concatenate image rows

AI加速器架构发展——Eyeriss

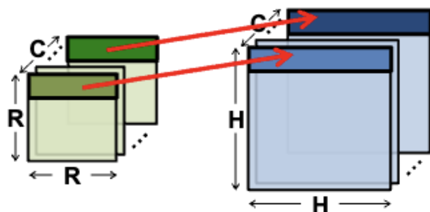
- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用
  - 不同计算单元间，进一步复用权重、输入和输出
  - 进一步拓展到多输入图像、多卷积核

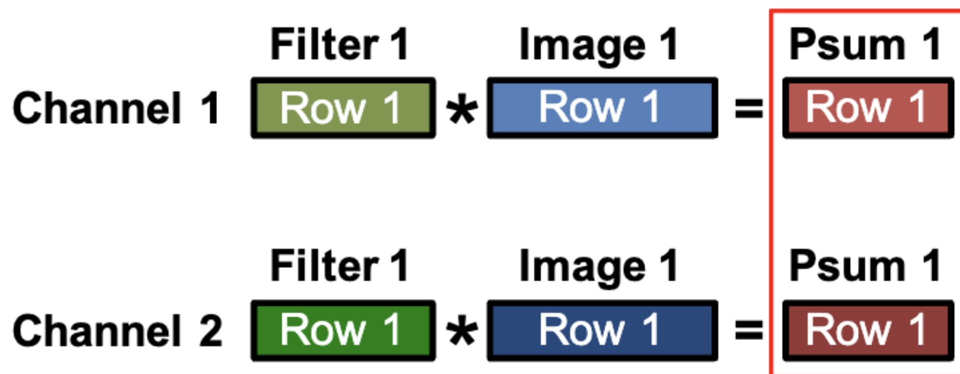


AI加速器架构发展——Eyeriss

- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用
  - 不同计算单元间，进一步复用权重、输入和输出
  - 进一步拓展到多输入图像、多卷积核，不同输入通道

北京大学-智能硬件体系结构

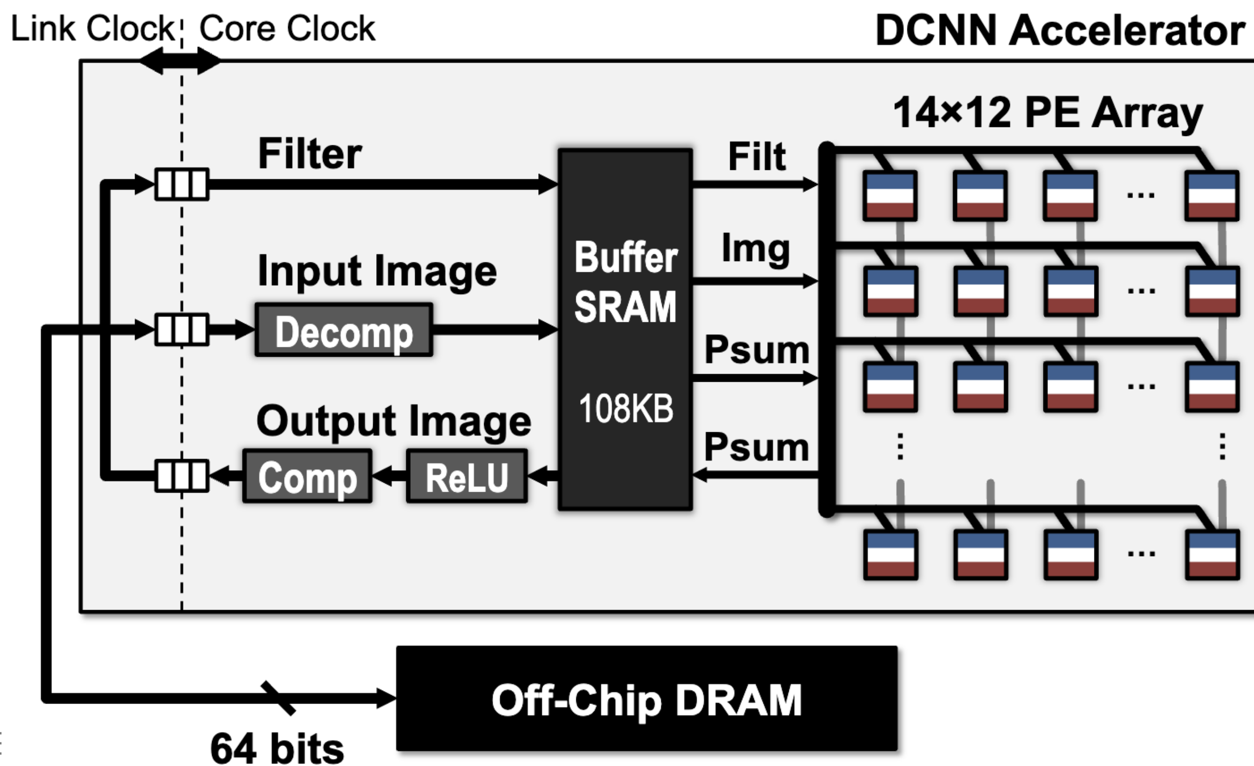




Processing in PE: interleave channels

AI加速器架构发展——Eyeriss

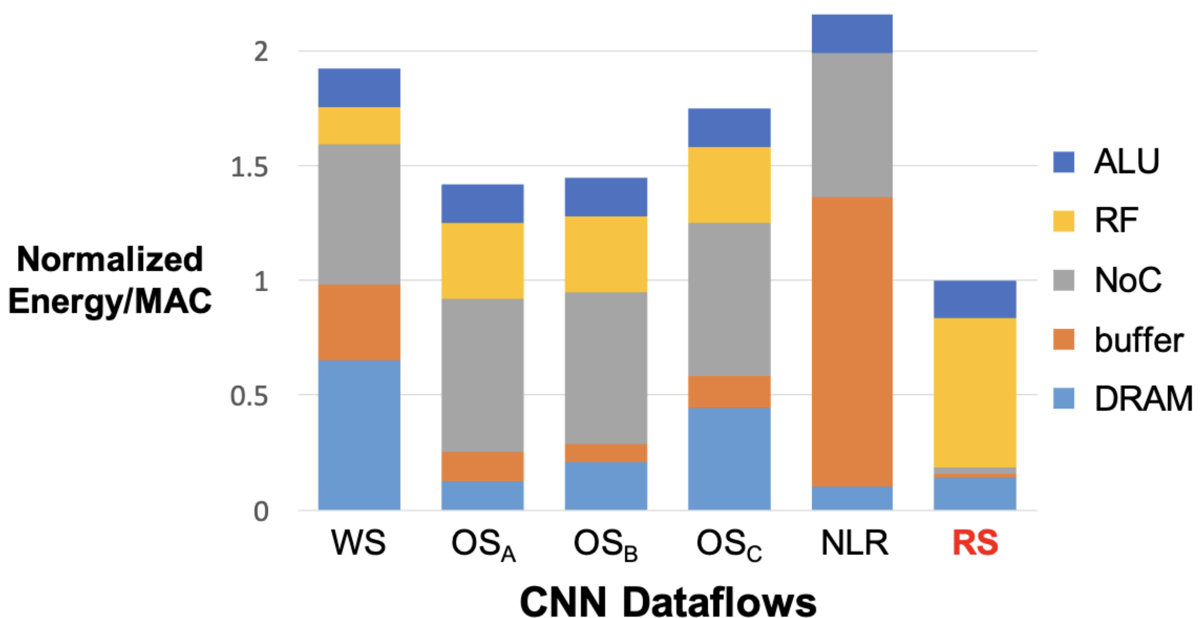
- Eyeriss的存储系统
  - 计算单元内部需要交大的便签存储器，提升数据复用
  - 采用了统一的便签存储器，**容量很小**，主要用于计算卷积层
  - 插入了压缩和解压缩模块，能够利用输入和输出结果中的0元素

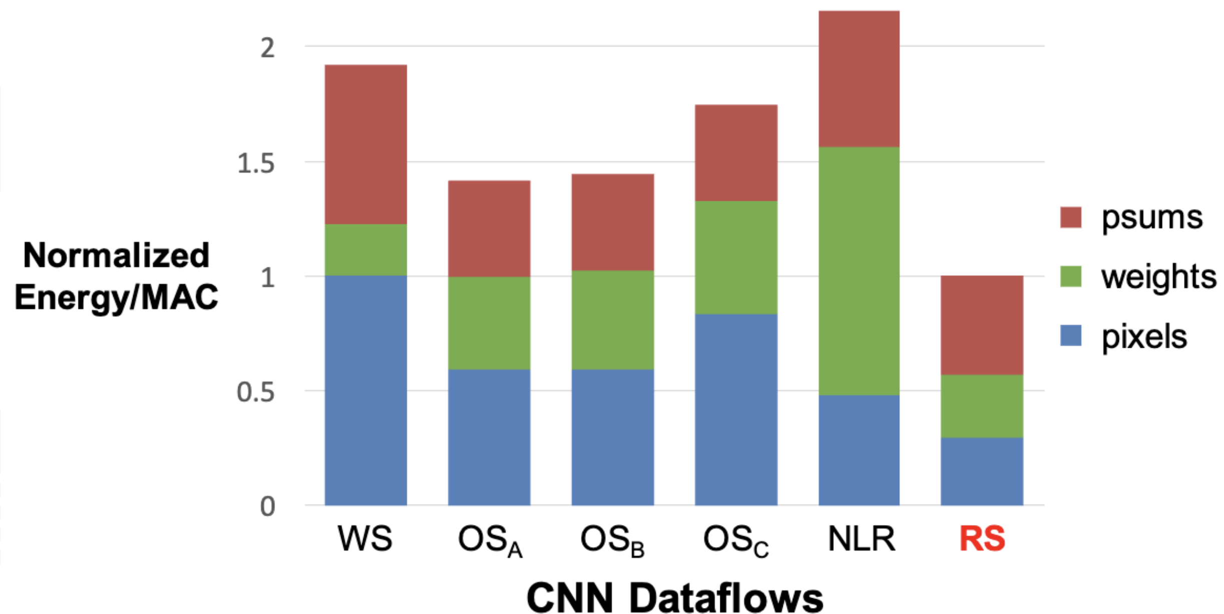


Technology	TSMC 65nm LP 1P9M
Core Area	3.5mm×3.5mm
Gate Count	1852 kGates (NAND2)
On-Chip Buffer	108 KB
# of PEs	168
Scratch Pad / PE	0.5 KB
Supply Voltage	0.82 – 1.17 V
Core Frequency	100 – 250 MHz
Peak Performance	33.6 – 84.0 GOPS (2 OP = 1 MAC)
Word Bit-width	16-bit Fixed-Point
Filter Size*	1 – 32 [width] 1 – 12 [height]
# of Filters*	1 – 1024
# of Channels*	1 – 1024
Stride Range	1–12 [horizontal] 1, 2, 4 [vertical]

AI加速器架构发展——Eyeriss

- 通过将Eyeriss的功耗与其他数据流进行比较可以发现
  - RF的功耗显著提升（每位每个PE中的存储显著提升）
  - Buffer的功耗很小，因为进一步提升了输入、权重和输出数据的复用





AI加速器架构发展——Google TPU v1

- Google TPU论文发表于ISCA 2016，从2015年开始，TPU已经集成在了Google服务器中
- 只考虑神经网络推理加速，重点关注MLP、CNN、LSTM三类神经网络
- 相较于NV K80 GPU和Intel Haswell CPU, 15-30倍延迟降低、30-80倍能效降低

Name	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% of Deployed TPUs in July 2016*
	FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	5				5	ReLU	20M	200	200	61%
MLP1	4				4	ReLU	5M	168	168	
LSTM0	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	37		19		56	sigmoid, tanh	34M	96	96	
CNN0		16			16	ReLU	8M	2888	8	5%
CNN1	4	72		13	89	ReLU	100M	1750	32	

思想自由 兼容并包

In-Datacenter Performance Analysis of a Tensor Processing Unit™

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon
Google, Inc., Mountain View, CA USA
Email: {jouppi, cliffy, nishantpatil, davidpatterson} @google.com

To appear at the 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 26, 2017.

Abstract
Many architects believe that major improvements in cost-energy-performance must now come from domain-specific hardware. This paper evaluates a custom ASIC—called a *Tensor Processing Unit (TPU)*— deployed in datacenters since 2015 that accelerates the inference phase of neural networks (NN). The heart of the TPU is a 65,536 8-bit MAC matrix multiply unit that offers a peak throughput of 92 TeraOps/second (TOPS) and a large (28 MiB) software-managed on-chip memory. The TPU's deterministic execution model is a better match to the 99th-percentile response-time requirement of our NN applications than are the time-varying optimizations of CPUs and GPUs (caches, out-of-order execution, multithreading, multiprocessing, prefetching, ...) that help average throughput more than guaranteed latency. The lack of such features helps explain why, despite having myriad MACs and a big memory, the TPU is relatively small and low power. We compare the TPU to a server-class Intel Haswell CPU and an Nvidia K80 GPU, which are contemporaries deployed in the same datacenters. Our workload, written in the high-level TensorFlow framework, uses production NN applications (MLPs, CNNs, and LSTMs) that represent 95% of our datacenters' NN inference demand. Despite low utilization for some applications, the TPU is on average about 15X - 30X faster than its contemporary GPU or CPU, with TOPS/Watt about 30X - 80X higher. Moreover, using the GPU's GDDR5 memory in the TPU would triple achieved TOPS and raise TOPS/Watt to nearly 70X the GPU and 200X the CPU.
Index terms–DNN, MLP, CNN, RNN, LSTM, neural network, domain-specific architecture, accelerator

1. Introduction to Neural Networks

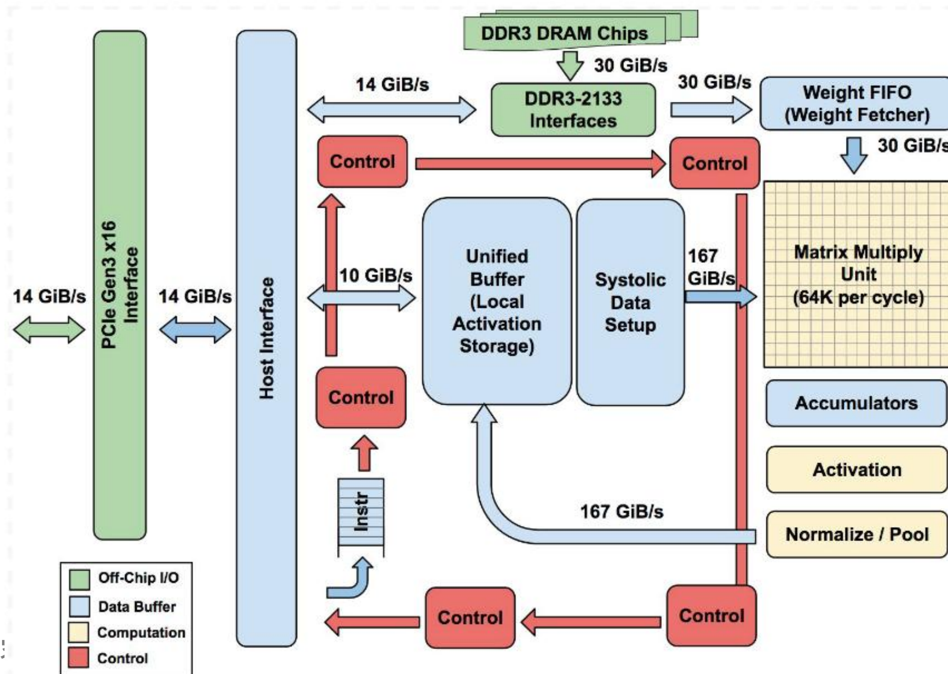
The synergy between the large data sets in the cloud and the numerous computers that power it has enabled a renaissance in machine learning. In particular, *deep neural networks* (DNNs) have led to breakthroughs such as reducing word error rates in speech recognition by 30% over traditional approaches, which was the biggest gain in 20 years [Dea16]; cutting the error rate in an image recognition competition since 2011 from 26% to 3.5% [Kri12] [Sze15] [He16]; and beating a human champion at Go [Sil16]. Unlike some hardware targets, DNNs are applicable to a wide range of problems, so we can reuse a DNN-specific ASIC for solutions in speech, vision, language, translation, search ranking, and many more.

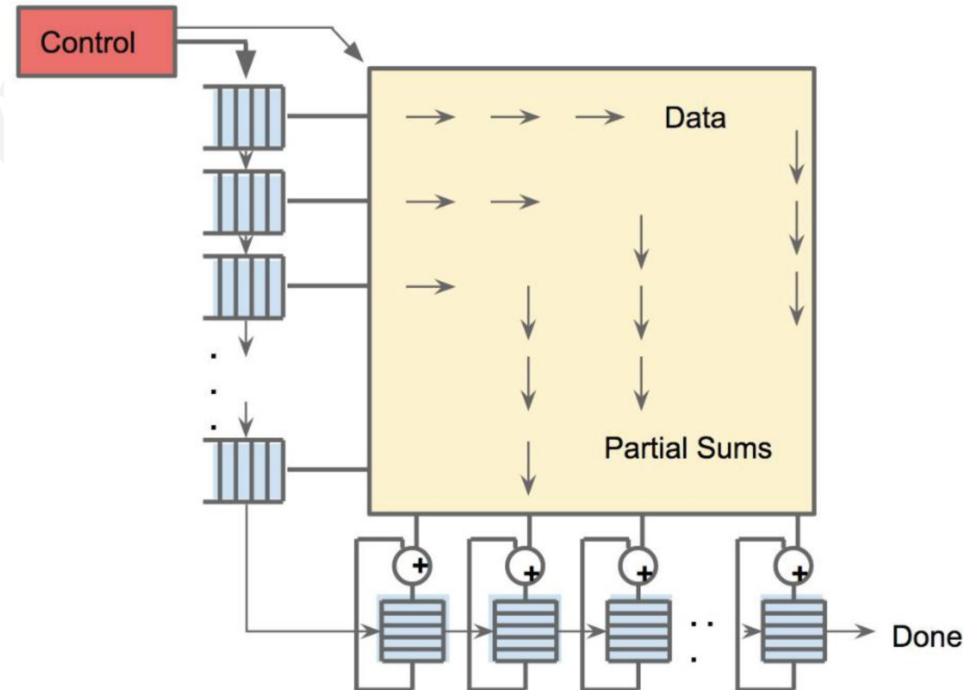
Neural networks (NN) target brain-like functionality and are based on a simple artificial neuron: a nonlinear function (such as `max(0, value)`) of a weighted sum of the inputs. These artificial neurons are collected into layers, with the outputs of one layer becoming the inputs of the next one in the sequence. The "deep" part of DNN comes from going beyond a few layers, as the large data sets in the cloud allowed more accurate models to be built by using extra and larger layers to capture higher levels of patterns or concepts, and GPUs provided enough computing to develop them.

The two phases of NN are called *training* (or learning) and *inference* (or prediction), and they refer to development versus production. The developer chooses the number of layers and the type of NN, and training determines the weights. Virtually all training today is in floating point, which is one reason GPUs have been so popular. A step called *quantization* transforms floating-point numbers into narrow integers—often just 8 bits—which are usually good enough for inference. Eight-bit integer multiplies can be 6X less energy and 6X less area than IEEE 754 16-bit floating-point multiplies, and the

AI加速器架构发展——Google TPU

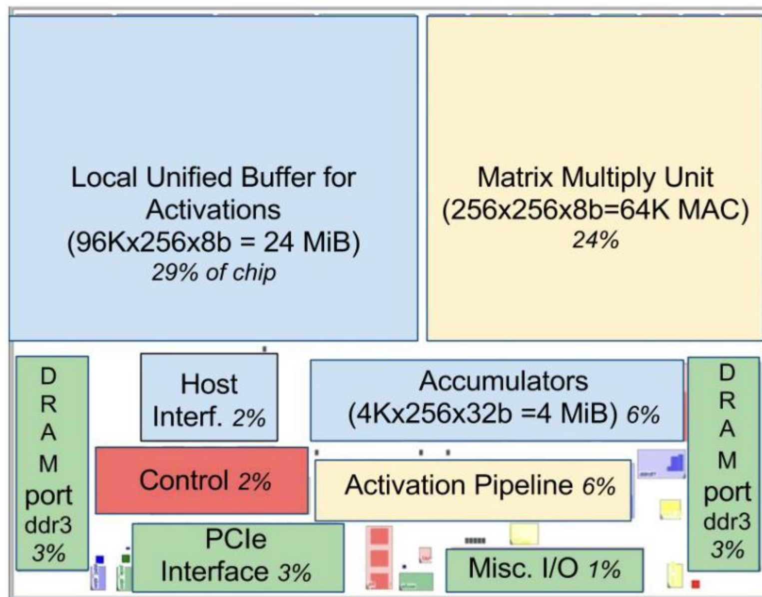
- TPU设计参数
  - 矩阵乘法单元：256 x 256，支持稠密矩阵乘法（不支持稀疏计算），double buffering，每周期产生256个partial sum，支持8比特权重，8/16比特输入
  - 累加单元：4 MB缓存（4096 x 256 x 32b），double buffering
  - 片上存储: MMU存储64KB权重，FIFO深度为4，激活存储为24MB

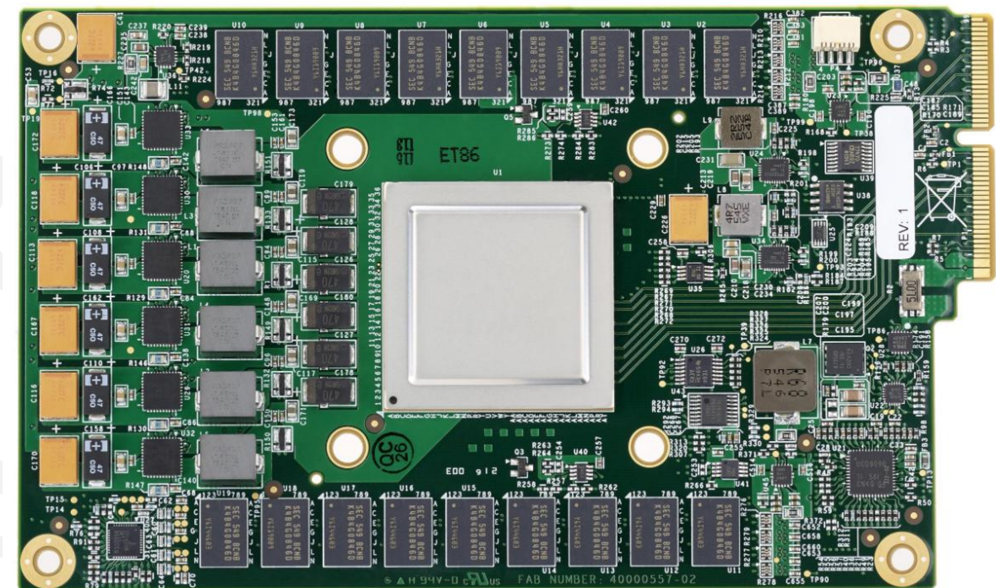




AI加速器架构发展——Google TPU

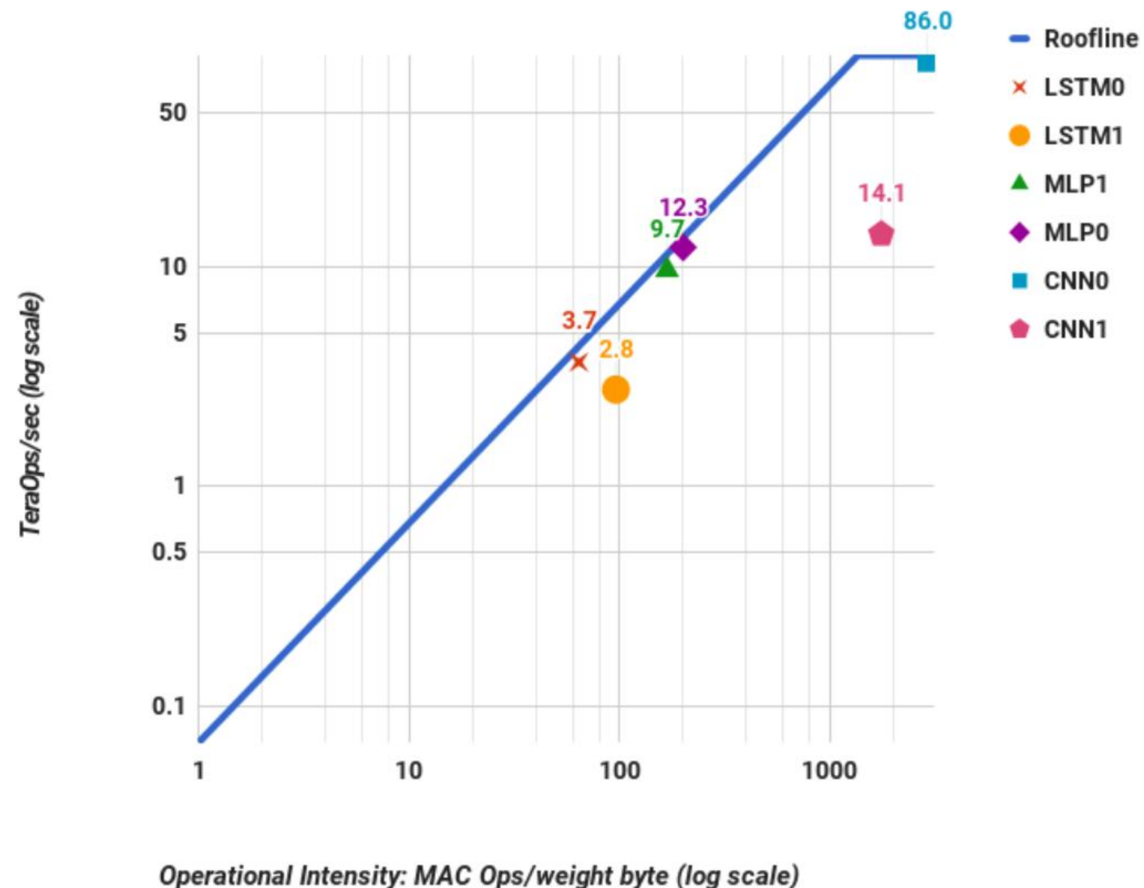
- 在TPU的版图中，数据通路面积占比为67%，I/O面积占比10%，控制占比2%
- 采用CISC指令，其中5个重要指令包括
  - Read_Host_Memory, Read_Weights, MatrixMultiply/Convolve, Activate, Write_Host_Memory





AI加速器架构发展——Google TPU

- 基于Roofline模型的系统瓶颈评估
- TPU Roofline：
  - 每Byte的权重读取需要1350乘加计算平摊，才能使得系统不会成为访存瓶颈
  - 访存瓶颈的操作：LSTM、MLP
  - 计算瓶颈的操作：卷积



AI加速器架构发展——Google TPU

- Google TPU v1的性能分析
  - 针对MLP、LSTM，模型权重的加载成为主要瓶颈
  - 实际算力与峰值算力之间存在显著差距

Name	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*
	FC	*Conv*	*Vector*	*Pool*	*Total*			
MLP0	5				5	ReLU	20M	200
MLP1	4				4	ReLU	5M	168
LSTM0	24		34		58	sigmoid, tanh	52M	64
LSTM1	37		19		56	sigmoid, tanh	34M	96
CNN0		16			16	ReLU	8M	2888
CNN1	4	72		13	89	ReLU	100M	1750

Application	*MLP0*	*MLP1*	*LSTM0*	*LSTM1*	*CNN0*	*CNN1*	*Mean*	*Row*
Array active cycles	12.7%	10.6%	8.2%	10.5%	78.2%	46.2%	28%	1
Useful MACs in 64K matrix (% peak)	12.5%	9.4%	8.2%	6.3%	78.2%	22.5%	23%	2
Unused MACs	0.3%	1.2%	0.0%	4.2%	0.0%	23.7%	5%	3
Weight stall cycles	53.9%	44.2%	58.1%	62.1%	0.0%	28.1%	43%	4
Weight shift cycles	15.9%	13.4%	15.8%	17.1%	0.0%	7.0%	12%	5
Non-matrix cycles	17.5%	31.9%	17.9%	10.3%	21.8%	18.7%	20%	6
RAW stalls	3.3%	8.4%	14.6%	10.6%	3.5%	22.8%	11%	7
Input data stalls	6.1%	8.8%	5.1%	2.4%	3.4%	0.6%	4%	8
TeraOps/sec (92 Peak)	12.3	9.7	3.7	2.8	86.0	14.1	21.4	9

Table 3. Factors limiting TPU performance of the NN workload based on hardware performance counters. Rows 1, 4, 5, and 6 total 100% and are based on measurements of activity of the matrix unit. Rows 2 and 3 further break down the fraction of 64K weights in the matrix unit that hold useful weights on active cycles. Our counters cannot exactly explain the time when the matrix unit is idle in row 6; rows 7 and 8 show counters for two possible reasons, including RAW pipeline hazards and PCIe input stalls. Row 9 (TOPS) is based on measurements of production code while the other rows are based on performance-counter measurements, so they are not perfectly consistent. Host server overhead is excluded here. CNN1 results are explained in the text.

AI加速器架构发展——Google TPU v2

- 不同于TPU v1只支持推理，TPU v2需要同时支持训练
- 相较于推理，支持训练的难度更大：
  - 更多计算（和类型）：反向传播、转置、求导等操作
  - 更多存储：中间计算需要保存，反向传播中会用到
  - 更高精度：INT8无法满足训练需求
  - 更高的可编程性：不同的优化器等
  - 更难并行计算：

Build it quickly

Achieve high performance...

...at scale

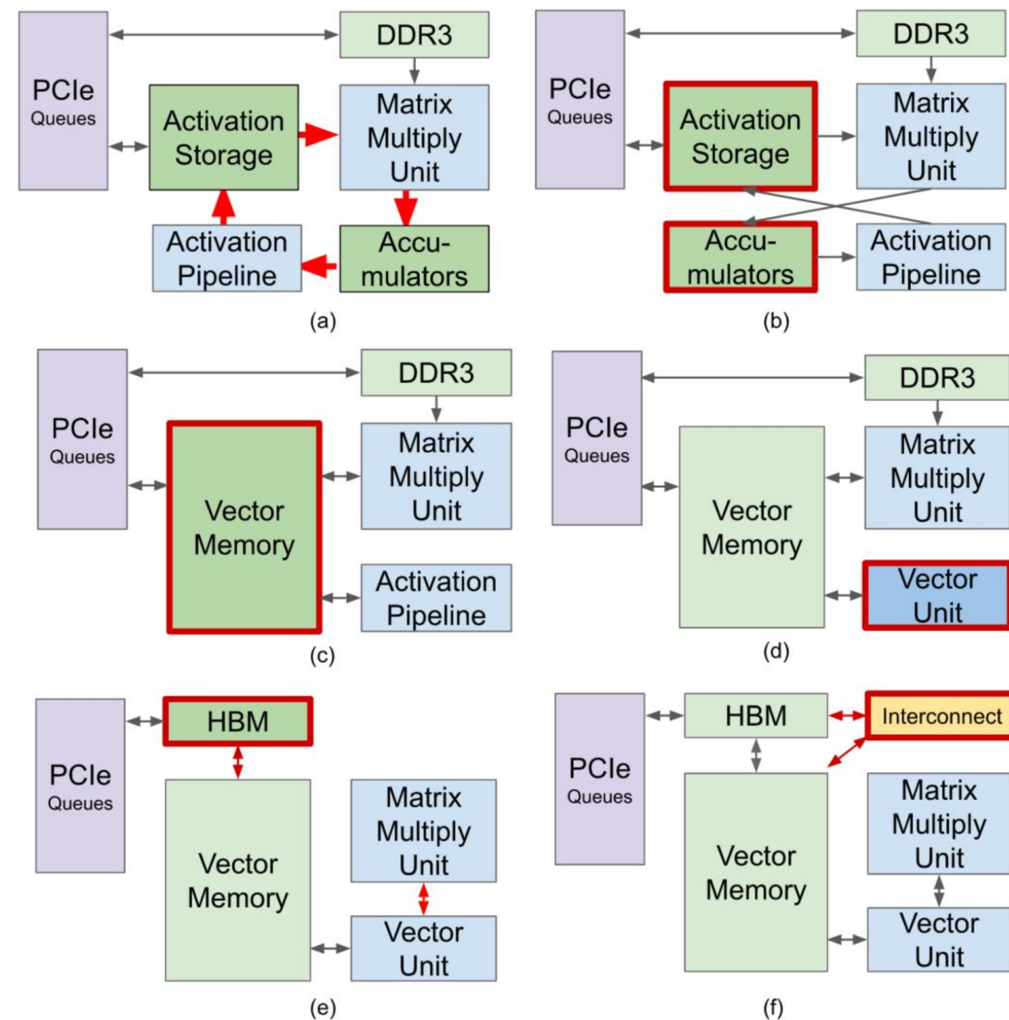
...for new workloads out-of-the-box

...all while being cost effective

北京大学-智能硬件体系结构 2024年秋季 主讲：陶耀宇、李萌

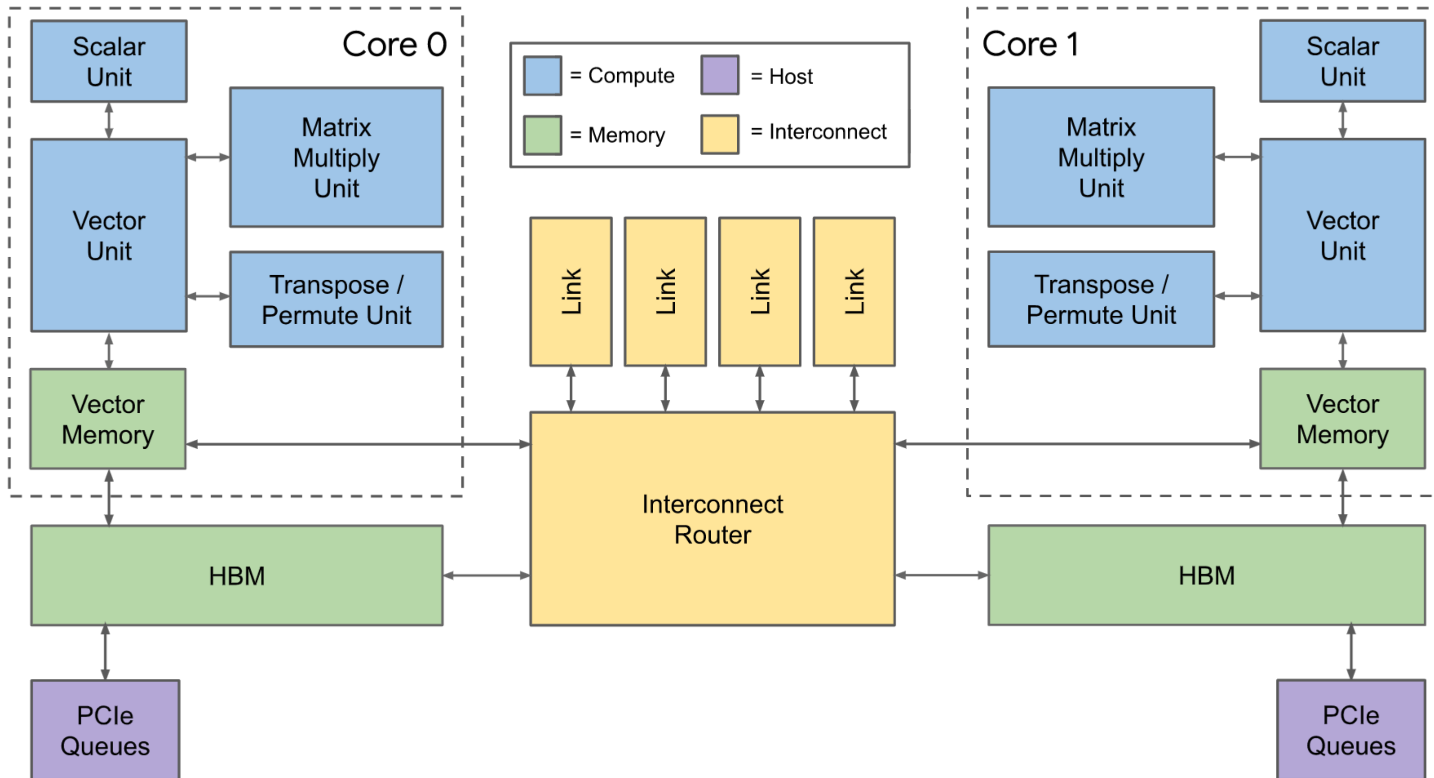
AI加速器架构发展——Google TPU v2

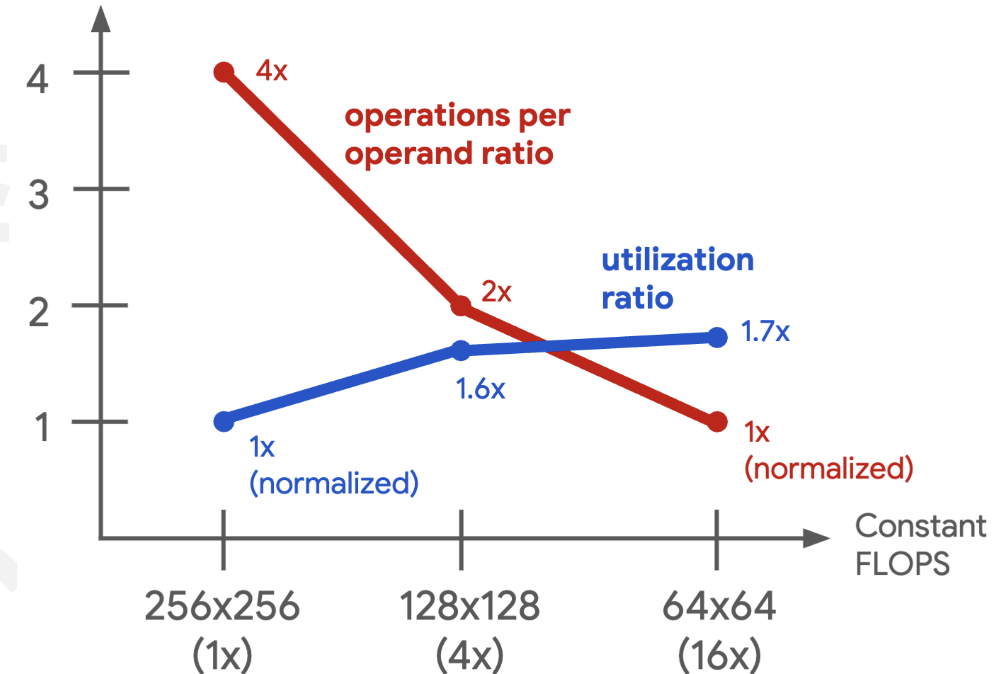
- 相较于TPU v1，TPU v2的**核心改动包括**
  - 将累加便签存储器、激活值便签存储器合并
  - 将累加器改为更为灵活、可编程的向量处理单元
  - 将MMU改为向量单元的协处理器，简化访存需求
  - 将DDR3改为HBM，提供更高访存带宽
  - 增加多卡互联，提升多卡扩展效率



AI加速器架构发展——Google TPU v2

- 采用双核架构，ISA支持矩阵、向量和标量计算，注重通用性
- 矩阵计算单元采用128 x 128的脉动阵列，支持bf16（s1e8m7）乘法和FP32累加
- 为什么要选择128 x 128的脉动阵列，而不是TPU v1中的256 x 256?

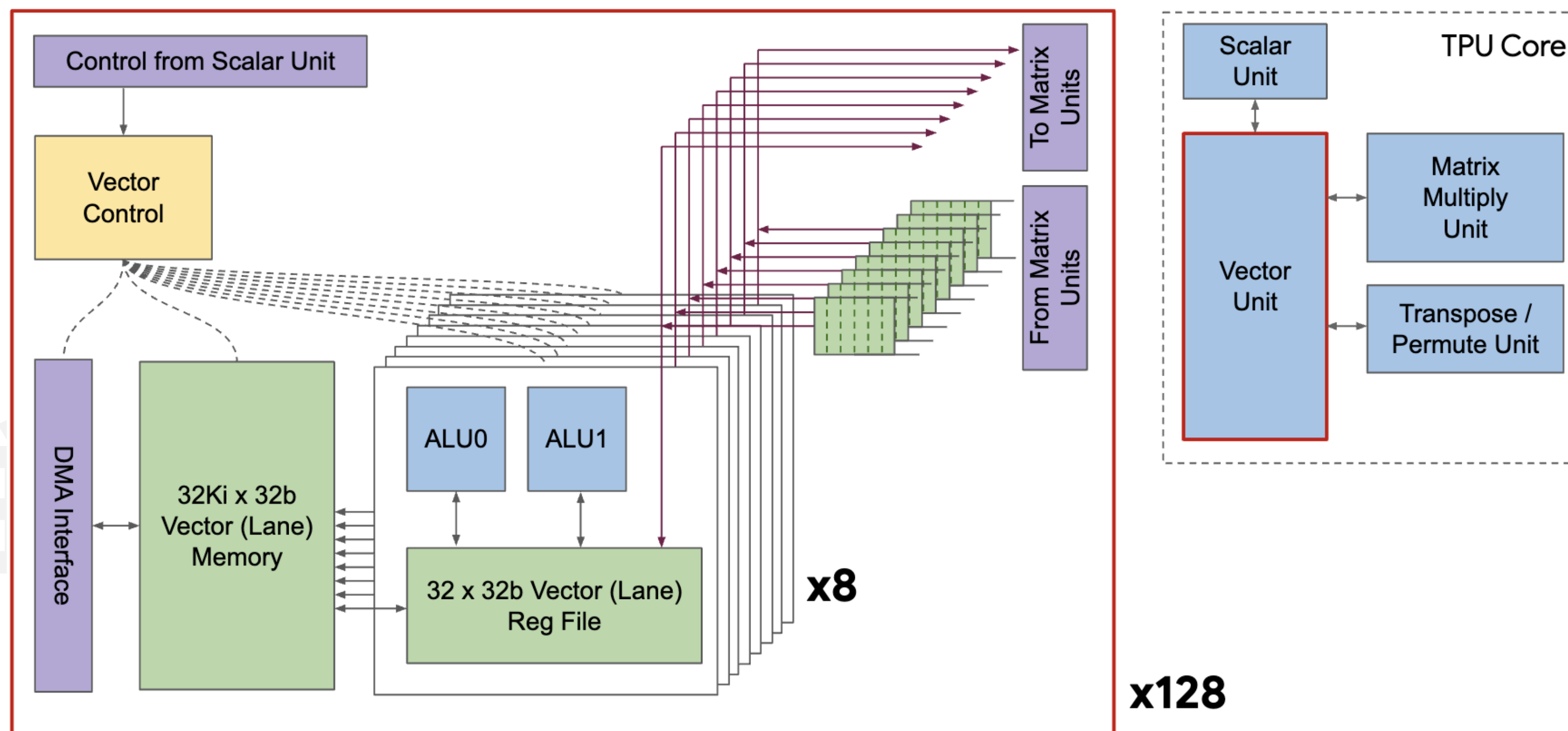




AI加速器架构发展——Google TPU v2

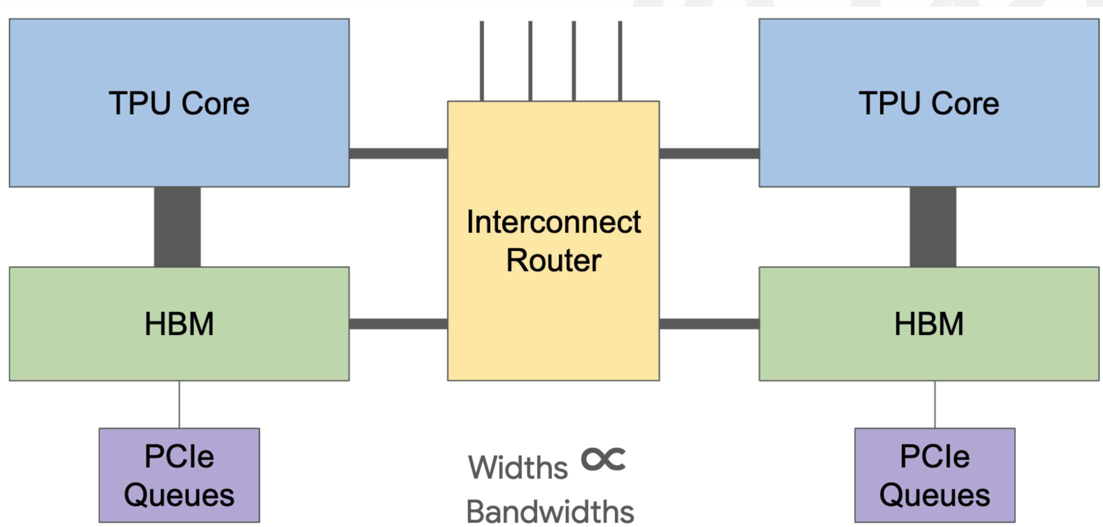
- 向量计算单元的功能：
  - 8个向量计算核，每个核每个周期处理128个输入
  - 为矩阵计算单元提供输入

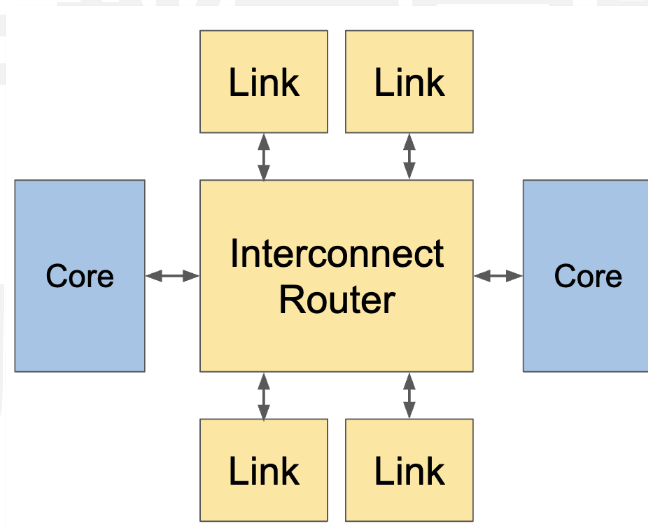
北京大学-智能硬件体系结构

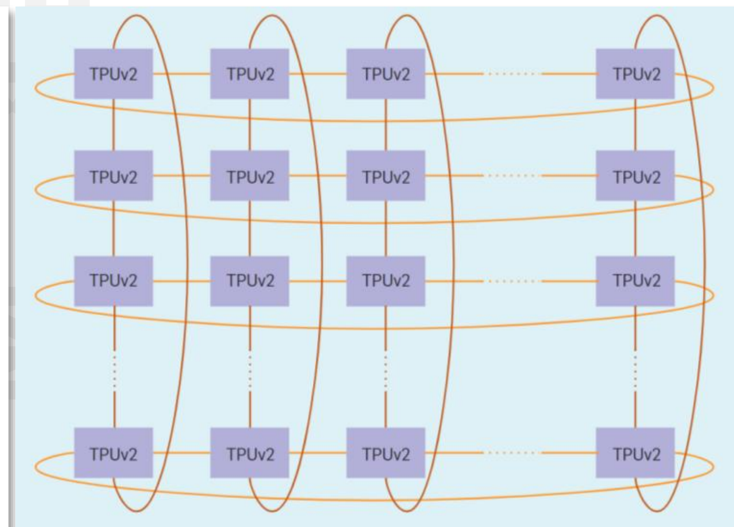


AI加速器架构发展——Google TPU v2

- TPU v2的存储系统
  - 片上存储约为32MB
  - 16GB HBM，带宽约为600 GB/s（TPU v1采用DDR3，带宽约为30 GB/s）
  - 片上路由有4个链路，每个链路带宽为500 Gbps，采用2维圆环（Torus）的拓扑结构
  - 最多支持256个TPU的拓展

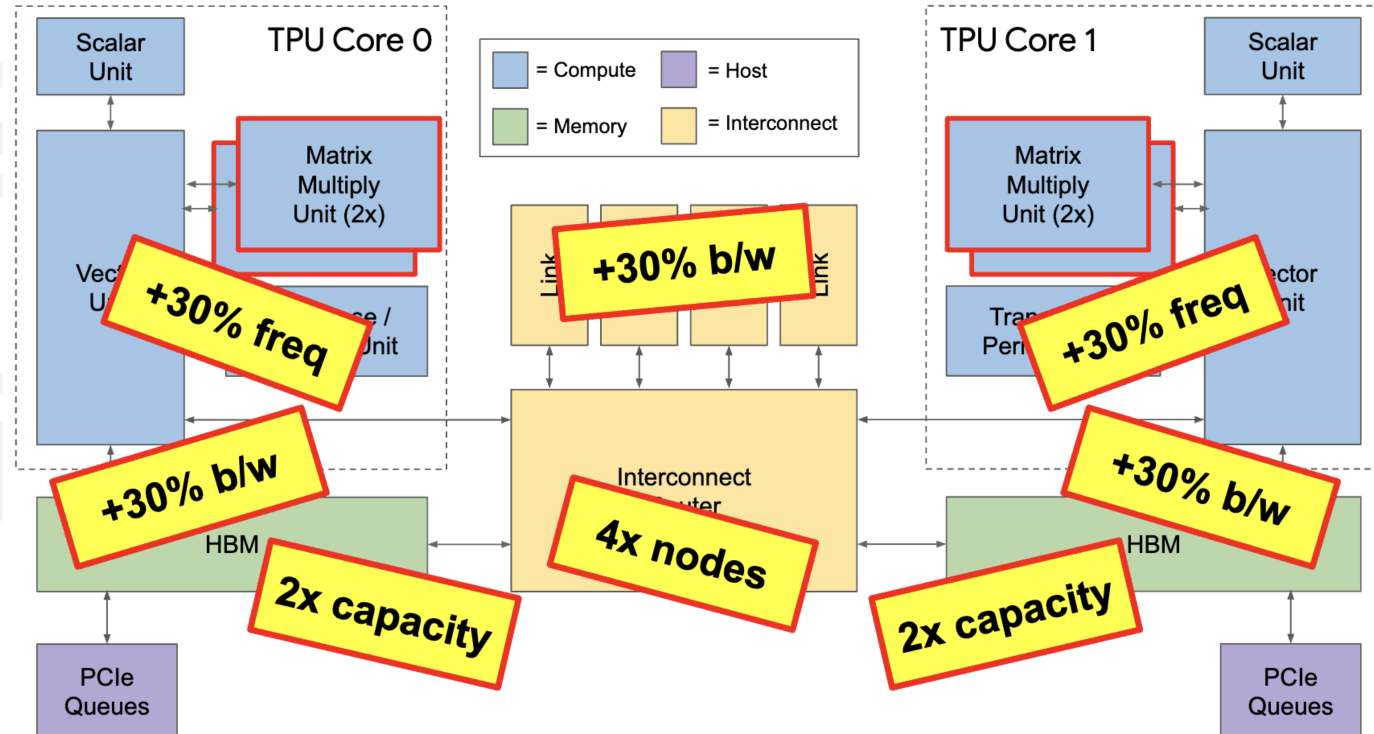






AI加速器架构发展——Google TPU v3

- 在TPU v2的基础上，TPU v3进行了进一步的改进了提升，包括
  - 矩阵乘法单元扩大2倍
  - 通过工程优化，将时钟频率从700 MHz提升到了940 MHz
  - 提升HBM带宽30%，扩大HBM容量2倍，提升互连带宽30%，达到每个链路650 Gb/s
  - 最大支持1024个TPU的扩展



目 录

CONTENTS

01. 典型AI芯片架构

02. AI芯片软硬件协同设计

03. AI大模型芯片设计

04. 存算一体AI芯片架构

人工智能处理器芯片

- 我们介绍了人工智能处理器芯片的基本组成和代表性架构
  - 基本组成：计算单元、访存、互连通信
  - 代表性架构：DianNao系列、Eyeriss系列、Google TPU系列
- 接下来**我们将介绍如何通过神经网络/处理器架构的协同设计和优化**，进一步提升处理器芯片效率
  - 神经网络模型**量化**与混合精度人工智能处理器
  - 神经网络模型**稀疏化**与人工智能处理器架构

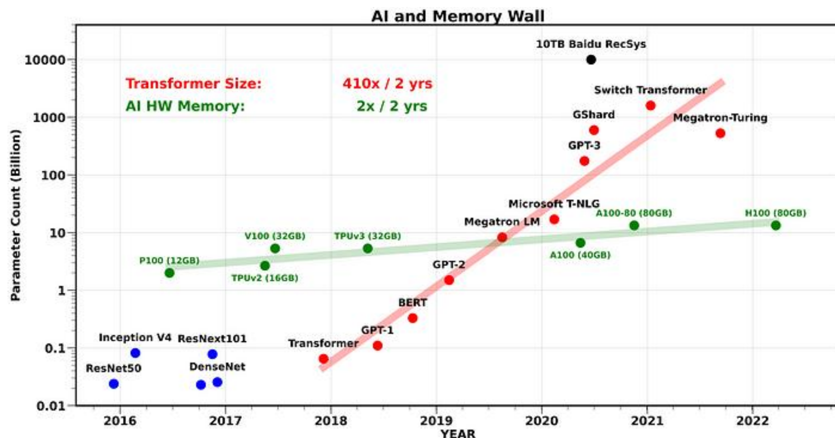
北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

人工智能模型的发展与挑战

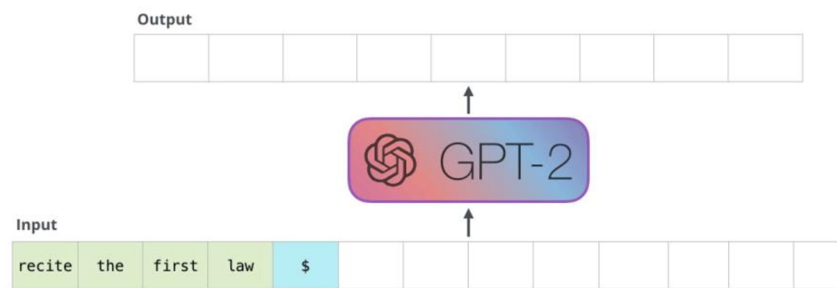
北京大学 PEKING UNIVERSITY

- 高效人工智能模型计算面临高存储、高带宽、动态计算图、部署平台复杂多样的挑战

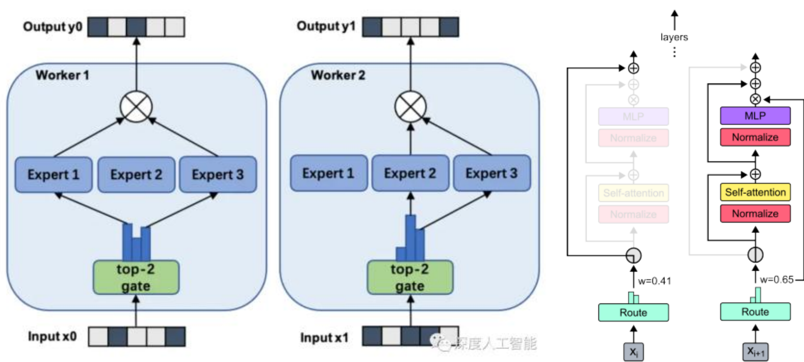
模型参数指数级增长，造成单模型**存储需求>1 TB**



大模型自回归解码，导致平均访存**带宽需求>1 TB/s**



动态计算图导致传统静态数据流优化失效



部署平台的**计算、存储**能力存在显著差距

	Cloud AI	Mobile AI	Tiny AI
Memory	24 GB	4 GB	500 KB
Storage	~ TB/PB	~100s GB	~1s MB
# params	>70 B	3 – 13 B	~1s M

人工智能模型的发展与挑战

北京大学 PEKING UNIVERSITY

- 为了进一步提升人工智能模型的推理效率，采用算法和AI处理器协同设计和优化方法
  - 神经网络模型**量化**、**稀疏化**、**神经网络架构搜索**、**知识蒸馏**等
- 本次课我们将以神经网络模型量化、稀疏化为例，介绍算法和加速器的协同设计

北京大学-智能硬件体系结构

模型量化

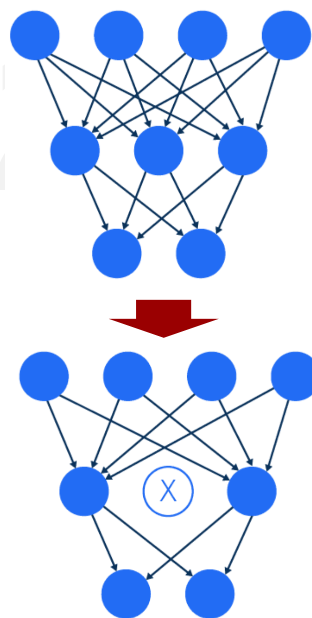
FP32

0.34	3.75	5.64
1.12	2.7	-0.9
-4.7	0.68	1.43

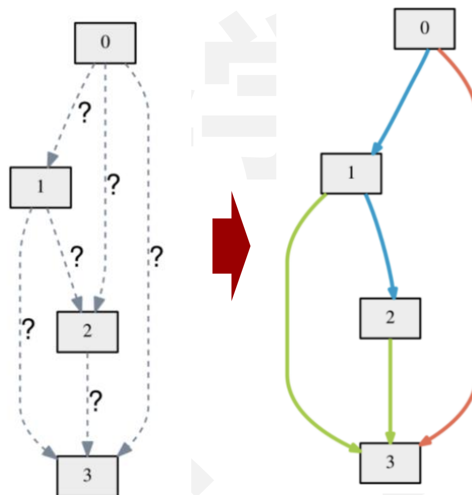
INT8

64	134	217
76	119	21
3	81	99

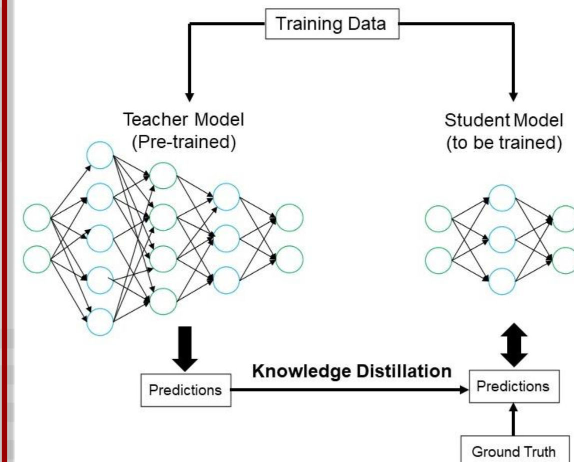
模型稀疏化



神经网络架构搜索



知识蒸馏



基于模型量化的软硬件协同设计

- 量化是将数据从**连续值集**映射为**离散值集**的过程
- 模型量化能够利用神经网络对于量化噪声的容错能力，显著提升模型推理计算效率
  - 无论是输入图像，还是神经网络本身都对于量化噪声具有较高的容忍能力

北京大学 智能硬件体系结构

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

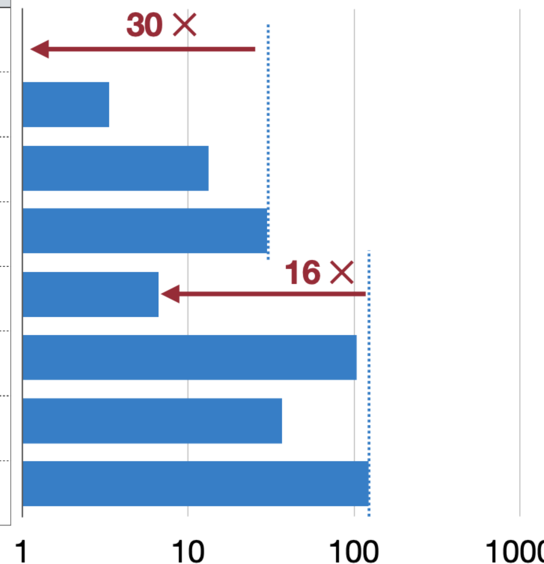
(

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

− -1)×1.07

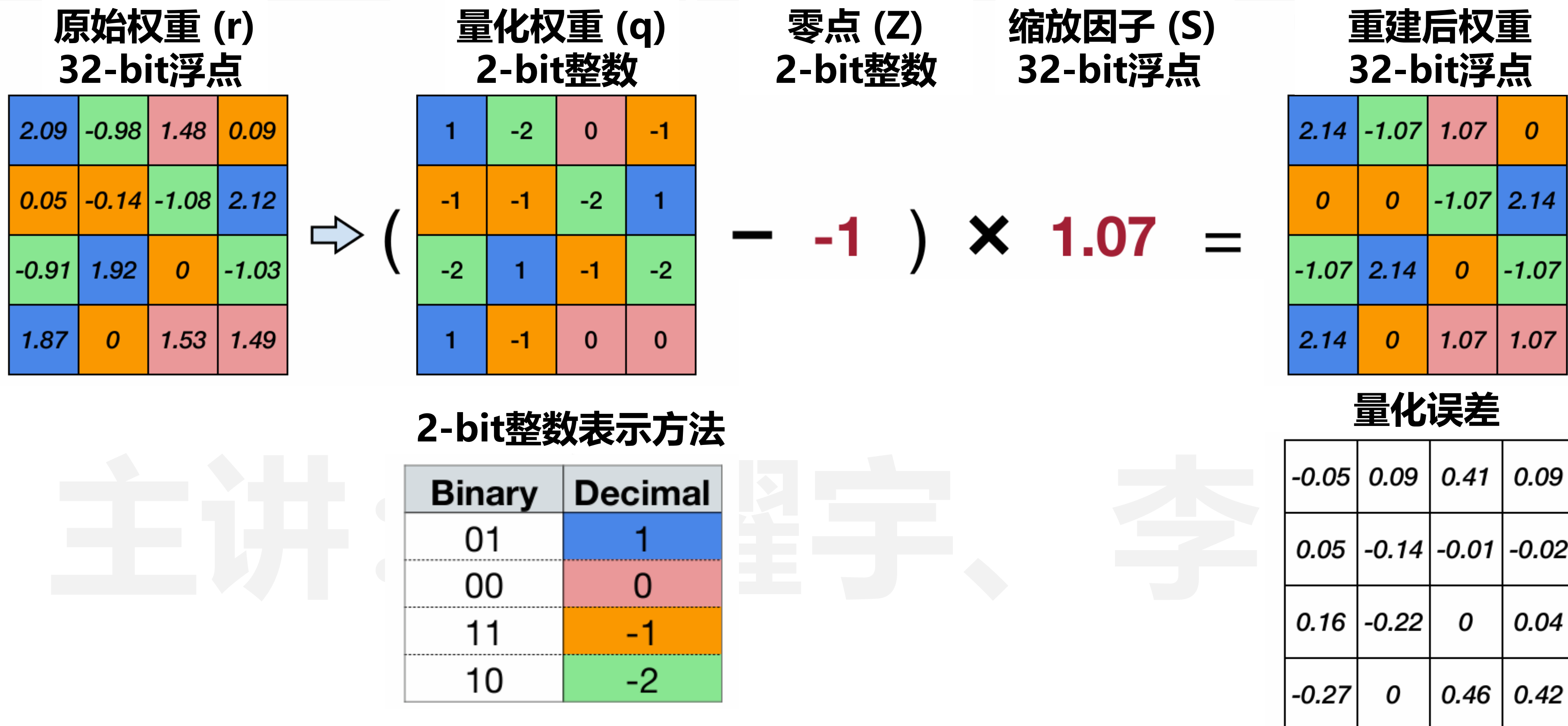
Operation	Energy [pJ]
8 bit int ADD	0.03
32 bit int ADD	0.1
16 bit float ADD	0.4
32 bit float ADD	0.9
8 bit int MULT	0.2
32 bit int MULT	3.1
16 bit float MULT	1.1
32 bit float MULT	3.7

Rough Energy Cost For Various Operations in 45nm 0.9V



量化神经网络基础

- 量化过程中的主要参数：缩放因子、零点、量化后数值
  - **静态**量化：量化参数在推理前确定，推理时保持不变
  - **动态**量化：量化参数在推理时计算得到



Binary	Decimal
01	1
00	0
11	-1
10	-2

- 权重的量化与激活值的量化对于计算的影响不同

北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

$$Y = WX$$

$$S_Y\,(q_Y - Z_Y) = S_W(q_W - Z_W)\cdot S_X(q_X - Z_X)$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W - Z_W)(q_X - Z_X) + Z_Y$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

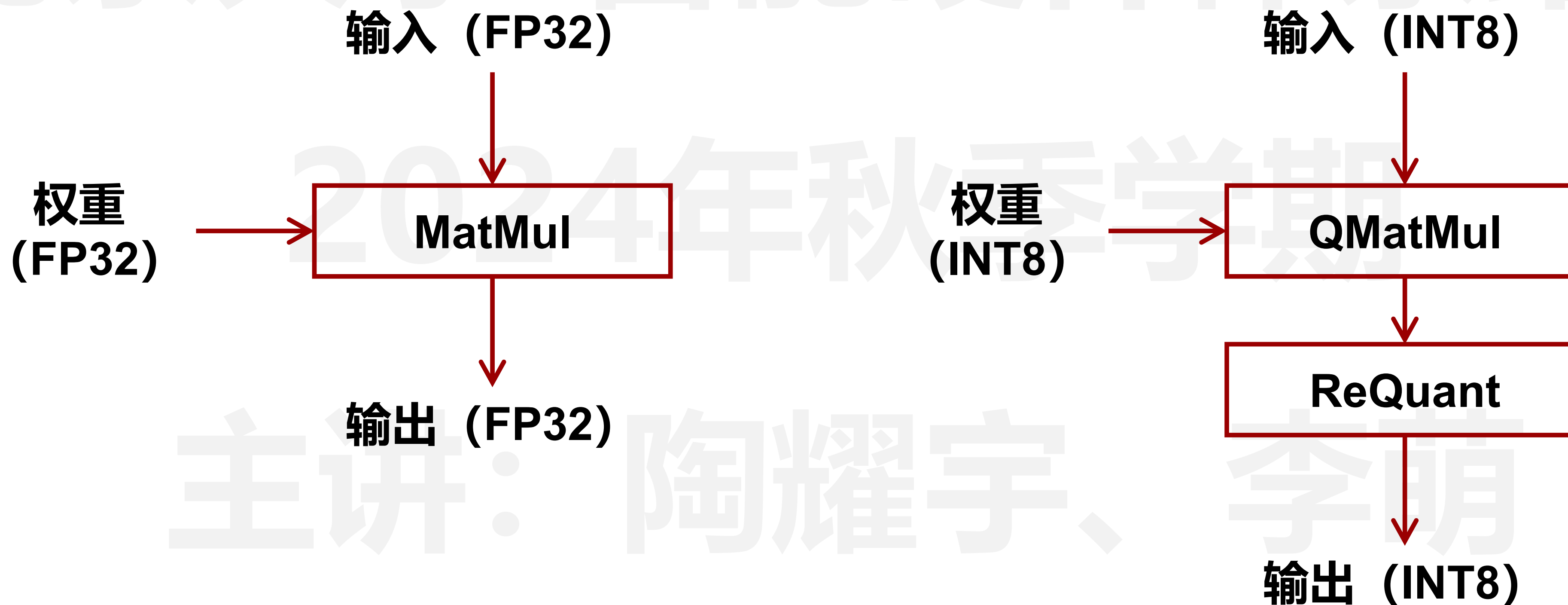
QMatMul

Z_W通常为0

提前计算

量化神经网络基础

- 权重的量化与激活值的量化对于计算的影响不同
- 量化神经网络推理：引入新的**重量化**计算，**将高比特精度计算结果重量化为低比特精度输出**
- 思考：QMatMul和MatMul的计算有什么区别?

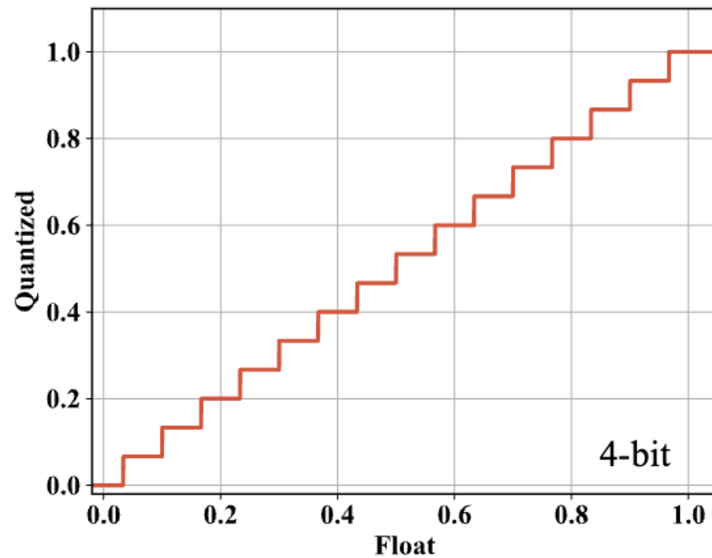


北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

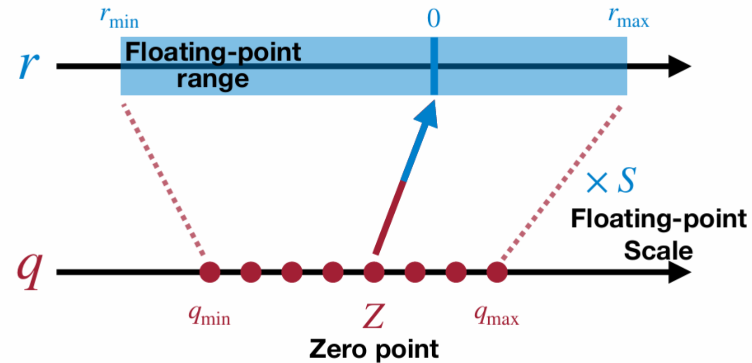
均匀量化与非均匀量化

- 均匀量化：输入到输出的映射为线性函数，因此均匀间隔的输入产生均匀间隔的输出
  - 针对均匀量化，QMatMul和MatMul的计算，除比特精度外，基本相同

北京大学-智能硬件体系结构

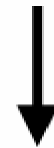


(a) Uniform Quantization



$$r_{max} = S\,(q_{max} - Z)$$

$$r_{min} = S\,(q_{min} - Z)$$

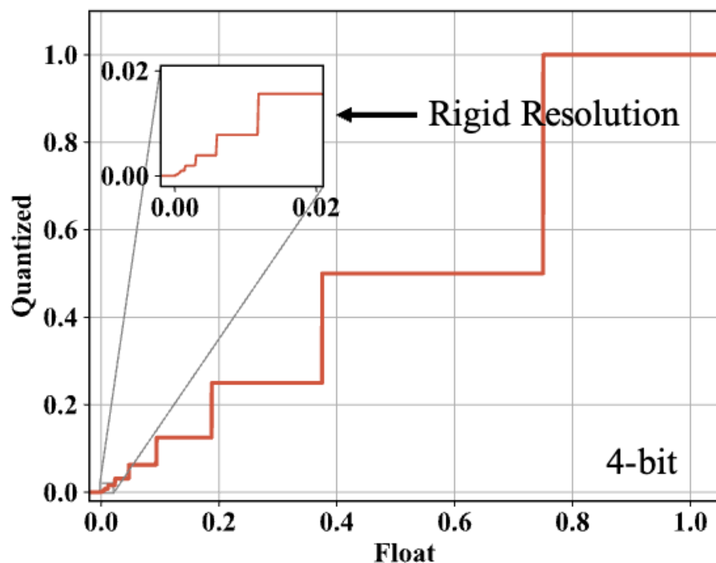


$$r_{max} - r_{min} = S\,(q_{max} - q_{min})$$

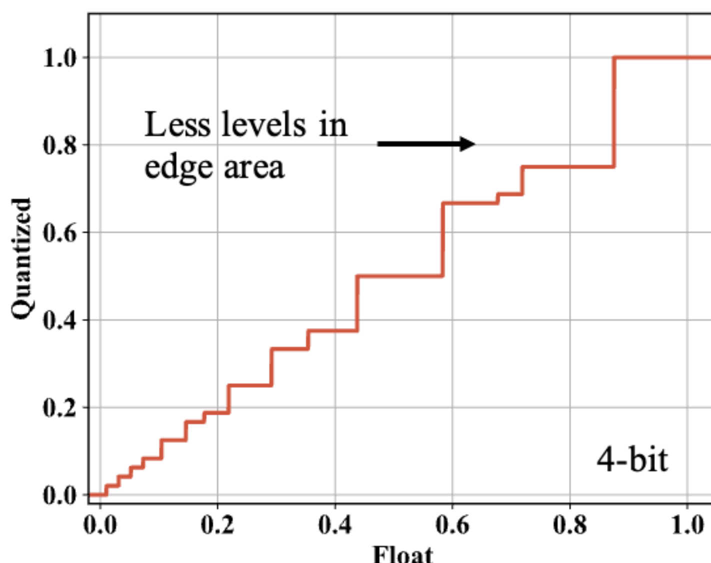
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

均匀量化与非均匀量化

- 均匀量化：输入到输出的映射为线性函数，因此均匀间隔的输入产生均匀间隔的输出
- **非均匀量化：输入到输出的映射为非线性函数，常见算法包括Power of Two（PoT）、APoT等**
- 思考：均匀和非均匀量化还有哪些类别？



(b) Power-of-Two Quantization



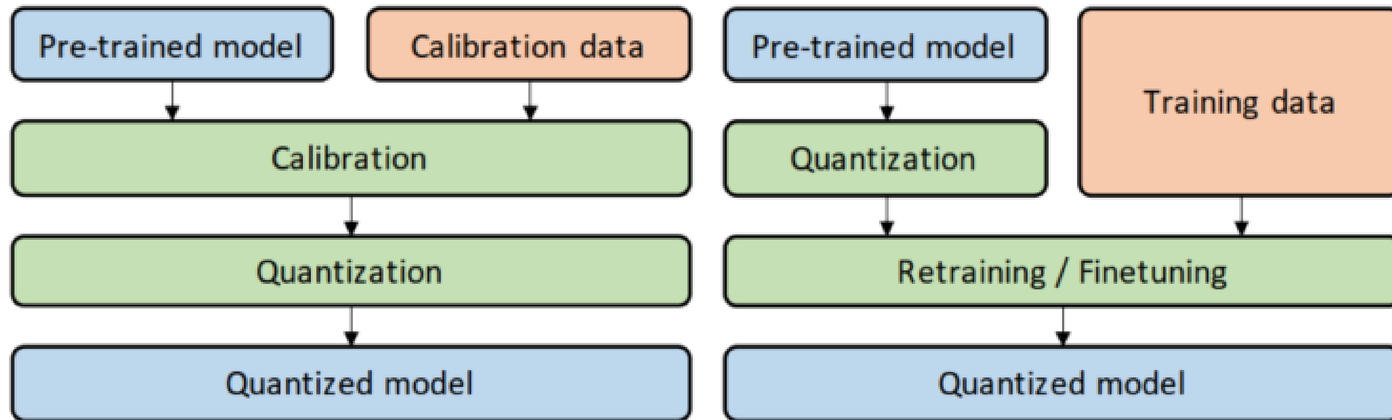
(c) Additive Power-of-Two Quantization

PoT量化：乘法可以用移位进行计算，显著降低计算复杂度，但是，对于较高比特数，PoT量化提升有限

APoT量化：乘法通过移位和加法实现，计算效率受APoT的项数决定

量化方法比较

- **训练后量化（post training quantization）：模型训练完成后对权重、激活值进行量化，其中，激活值的量化，需要借助校准数据**
- **量化感知训练（quantization-aware training）：将训练过的模型量化后又再进行重训练，或者直接训练量化模型**



PTQ运行时间短，数据需求小，但是模型推理准确率下降较大

QAT准确率高，但是训练时间长，数据要求高，在特定任务上可能过拟合

北京大学 智能硬件体系结构

神经网络量化与AI处理器

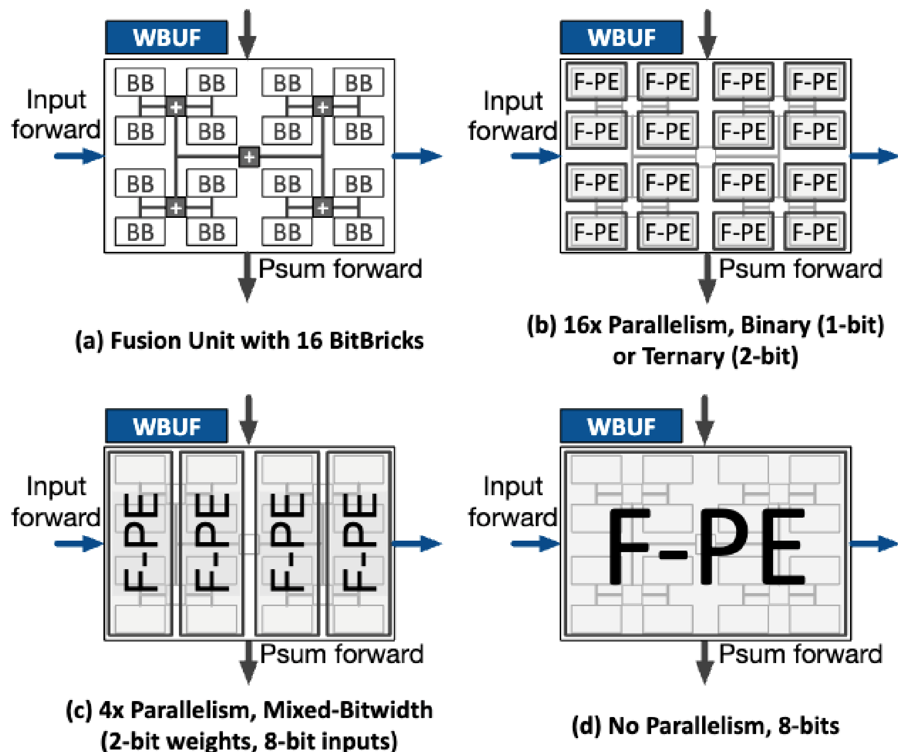
北京大学 PEKING UNIVERSITY

- 量化神经网络想要真正实现计算能效、吞吐等的提升，离不开**AI处理器**的支持
  - 例如支持低比特计算的计算单元（乘法器、累加器等）、支持低比特数据的存储方式等等
- 因此，神经网络量化往往需要配合专用硬件（或核函数）设计，形成软硬件协同设计和优化
- 本节课，我们重点介绍3个例子
  - BitFusion：支持混合精度计算的AI处理器
  - OliVe：面向低比特大模型的AI处理器
  - ANT：基于定制化数据类型的AI处理器

北京大学-智能硬件体系结构
2024年秋季学期
主讲：陶耀宇、李萌

神经网络量化与AI处理器 —— BitFusion

- Bit Fusion文章发表于ISCA 2018，核心在于高效支持混合精度神经网络推理
- 核心观察：不同神经网络模型以及同一神经网络模型中的不同层，对于量化噪声的容错能力不同
  - 采用混合精度量化可以在保持模型精度的同时，最大化压缩模型
- 文章提出Bit Fusion架构，重点通过可重构的计算单元，实现不同计算精度的灵活支持



(a) Fusion Unit with 16 BitBricks

(b) 16x Parallelism, Binary (1-bit) or Ternary (2-bit)

(c) 4x Parallelism, Mixed-Bitwidth (2-bit weights, 8-bit inputs)

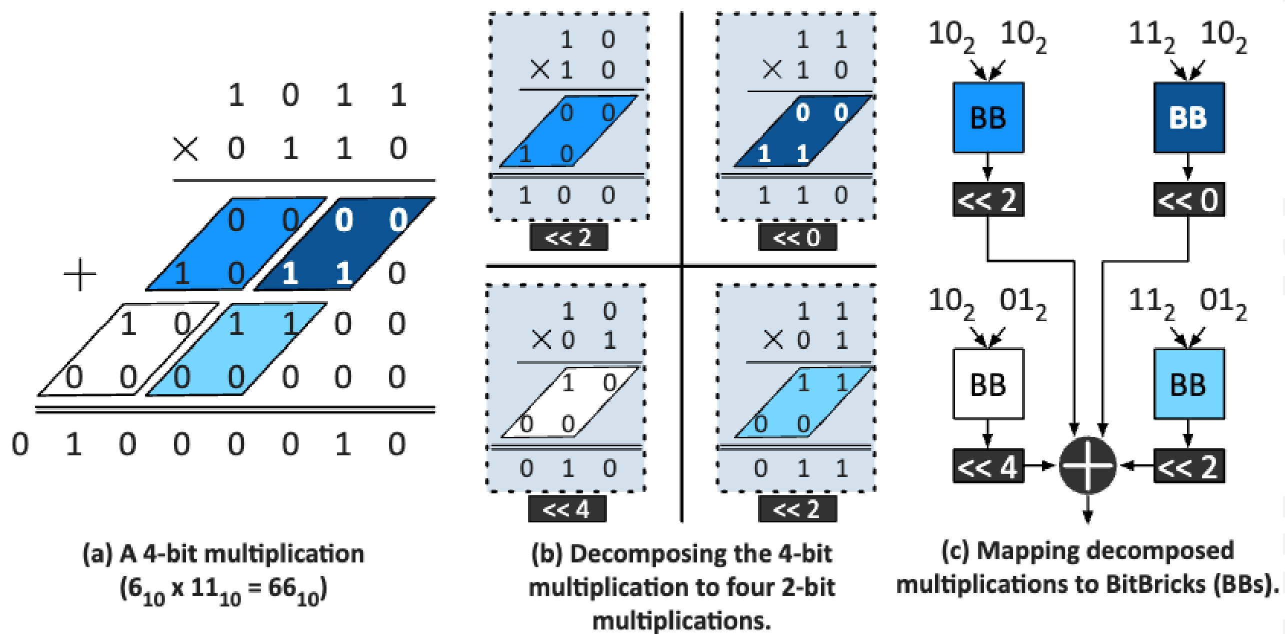
(d) No Parallelism, 8-bits

- Fusion单元由16个BitBricks组成，每个BitBrick每周期可以支持2比特计算
- 多个BitBricks组合，能够实现W4A4、W2A8、W8A2以及W8A8的计算

Hardik Sharma et al., *Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network*, ISCA 2018

神经网络量化与AI处理器 —— BitFusion

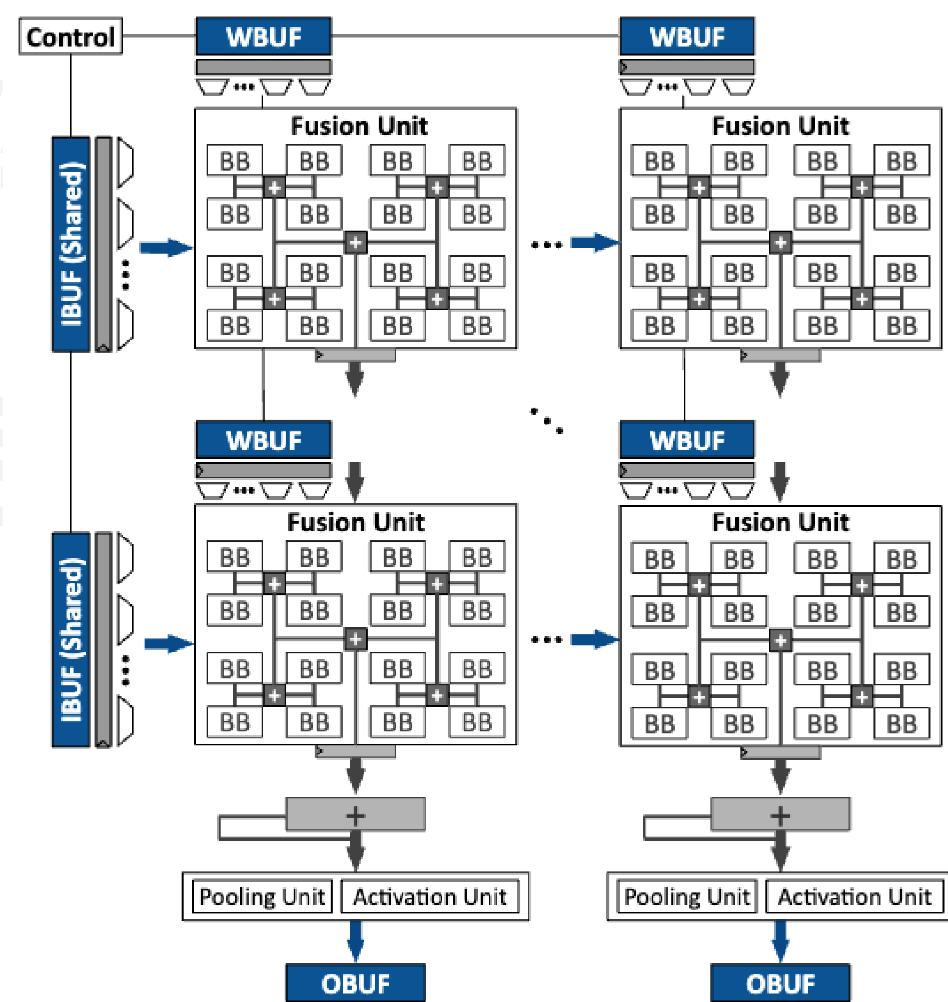
- 每个BitBrick里面需要灵活的累加器设计，才能有效实现不同比特精度的累加
- BitFusion采用脉动阵列的设计，进一步降低访存开销



(a) A 4-bit multiplication (6_{10} x 11_{10} = 66_{10})

(b) Decomposing the 4-bit multiplication to four 2-bit multiplications.

(c) Mapping decomposed multiplications to BitBricks (BBs).



神经网络量化与AI处理器——ANT

- ANT发表于MICRO 2022，提出了新的**数据类型**，适应神经网络推理需求
- 核心观察：1. 神经网络中不同的tensor分布各不相同；2.同一个tensor内部对于接近0的值或特别大的值不需要使用高精度表示；使用现有的INT或Float格式量化难以适应这两种变化
- 提出了一种新的数据类型**flint**，适合表示**高斯分布**的数据

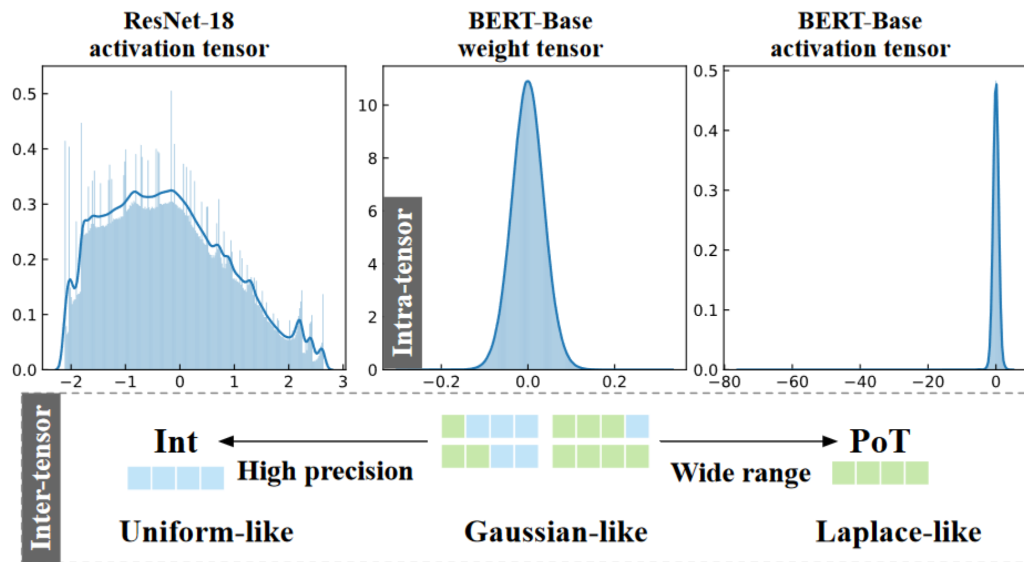


Figure 1: Intra-tensor and inter-tensor adaptivity.

Bits	Exponent Value	Fraction Value	Value in Decimal
0000	-	0	0
0001	$1-1=0$	1	$2^0 \times 1 = 1$
001x	$2-1=1$	1, 1.5	2, 3
01xx	$3-1=2$	1, 1.25, 1.5, 1.75	4, 5, 6, 7
11xx	$4-1=3$	1, 1.25, 1.5, 1.75	8, 10, 12, 14
101x	$5-1=4$	1, 1.5	16, 24
1001	$6-1=5$	1	32
1000	$7-1=6$	1	$2^6 \times 1 = 64$

Table II: The value table of 4-bit unsigned `flint` with the exponent bias of -1. The blue numbers are the first-one-encoded exponent and “x” is mantissa with value of 0 or 1.

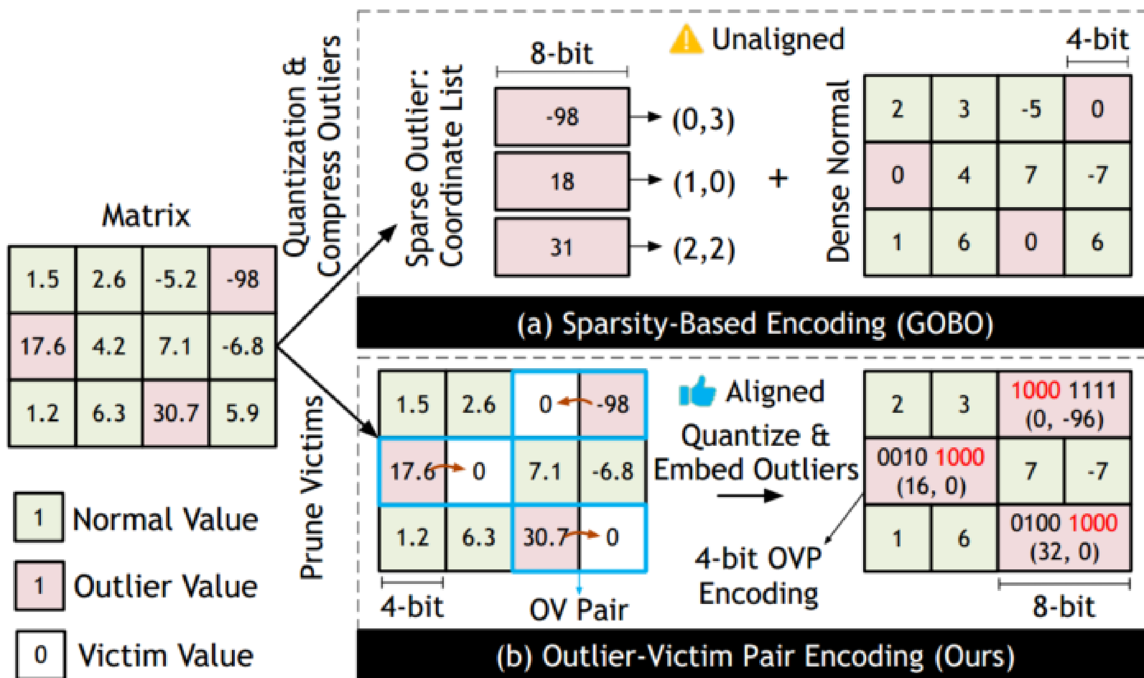
Cong Guo et al., *ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization*, MICRO 2022

神经网络量化与AI处理器——ANT

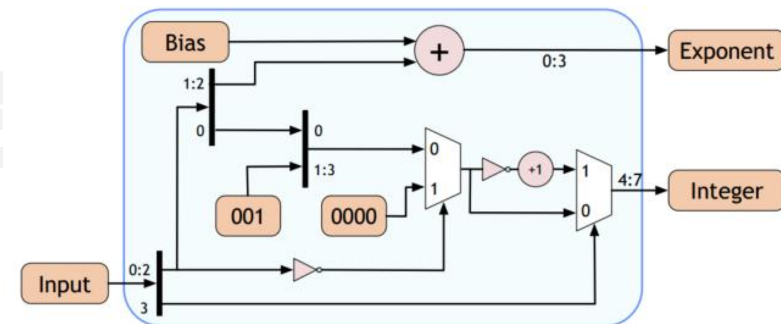
- 为了解决tensor之间数据分布不同的问题，通过校准数据集选择量化后均方误差最小的数据类型
  - 直觉上int类型适合均匀分布，Float/PoT类型适合拉普拉斯分布，Flint类型适合高斯分布
- 设计了可以支持以上四种格式的处理单元（PE），主要包括解码器，乘法器，加法器和累加器
- 将设计的PE集成到了脉动阵列形成新的架构

神经网络量化与AI处理器——OliVe

- OliVe发表于ISCA 2023，针对大模型中的离群值（outliers）进行设计
- 核心观察：大语言模型中存在一些ouliers, 这些outliers很重要，但旁边的正常值（称为victims）不重要，因此可以牺牲victims进而用更多比特表示outliers
- 对于outliers, 提出了新的数据格式Abfloat, 通过自适应的bias使编码的值表示比正常值更大的数



Binary	Exponent	Integer	Real Value
000	0	0	0
001	0	3	$3 \times 2^0 = 3$
01x	1	2, 3	$2 \times 2^1 = 4, 3 \times 2^1 = 6$
10x	2	2, 3	$2 \times 2^2 = 8, 3 \times 2^2 = 12$
11x	3	2, 3	$2 \times 2^3 = 16, 3 \times 2^3 = 24$



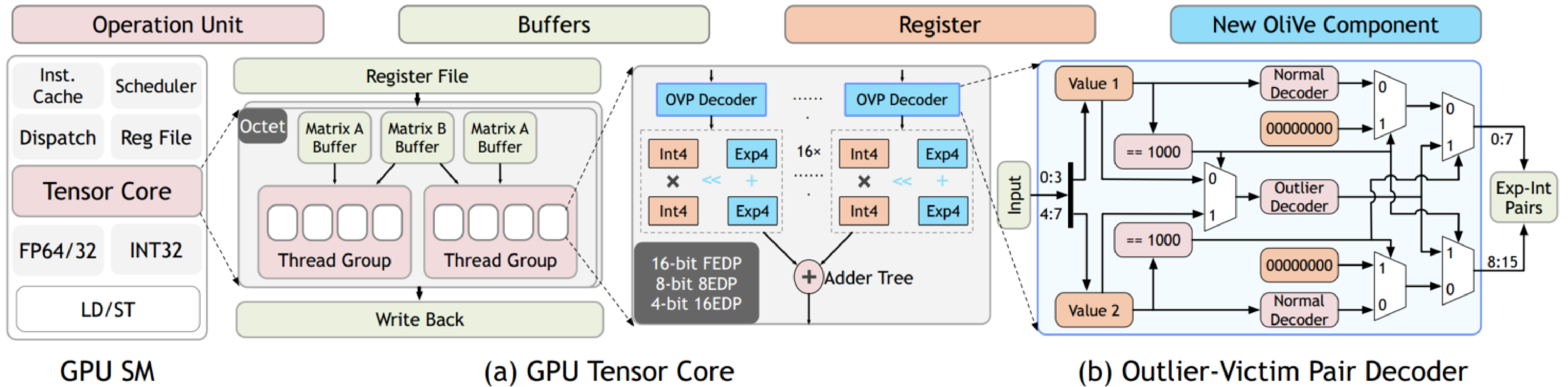
Cong Guo et al., *OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization*, ISCA 2023

神经网络量化与AI处理器——OliVe

北京大学 PEKING UNIVERSITY

- 解码完成后，文章设计了乘累加单元电路（MAC Unit），并将他们集成到了GPU tensor core以及脉动阵列中

北京大学-智能硬件体系结构



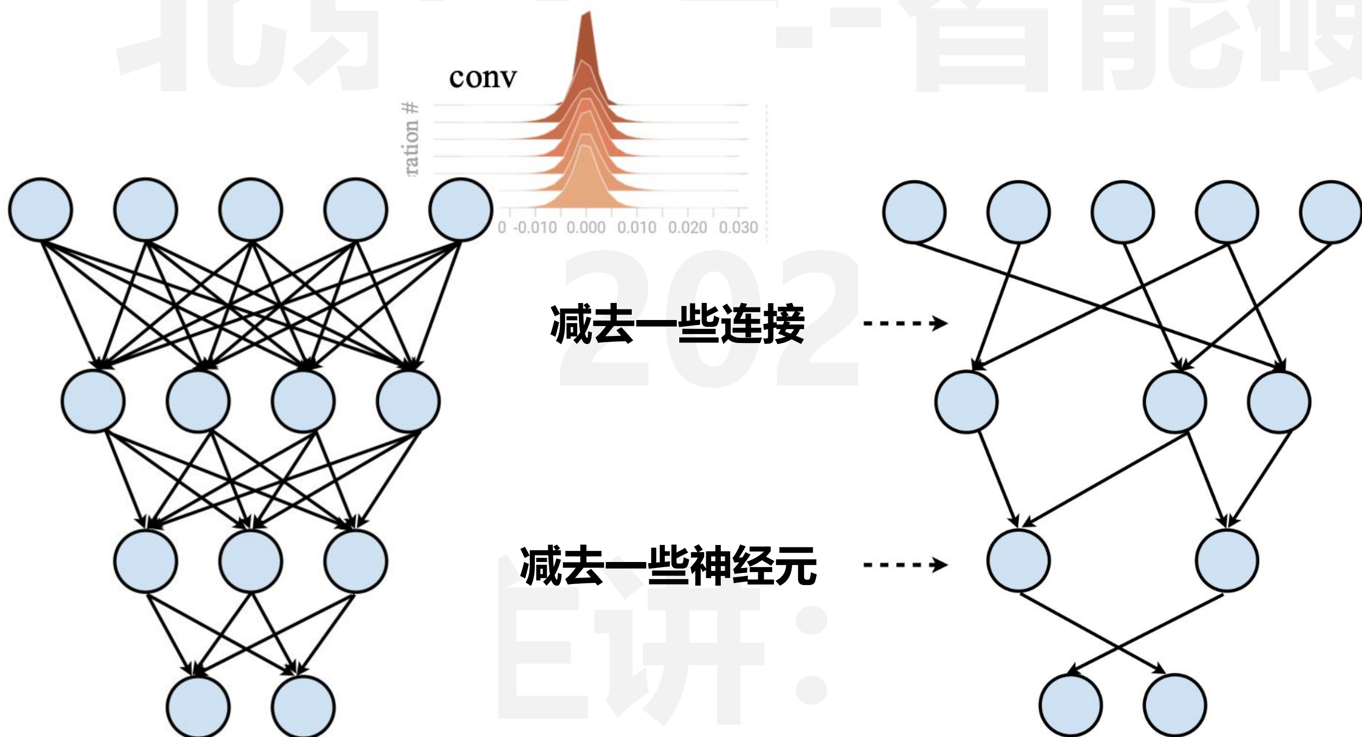
GPU SM　　(a) GPU Tensor Core　　(b) Outlier-Victim Pair Decoder

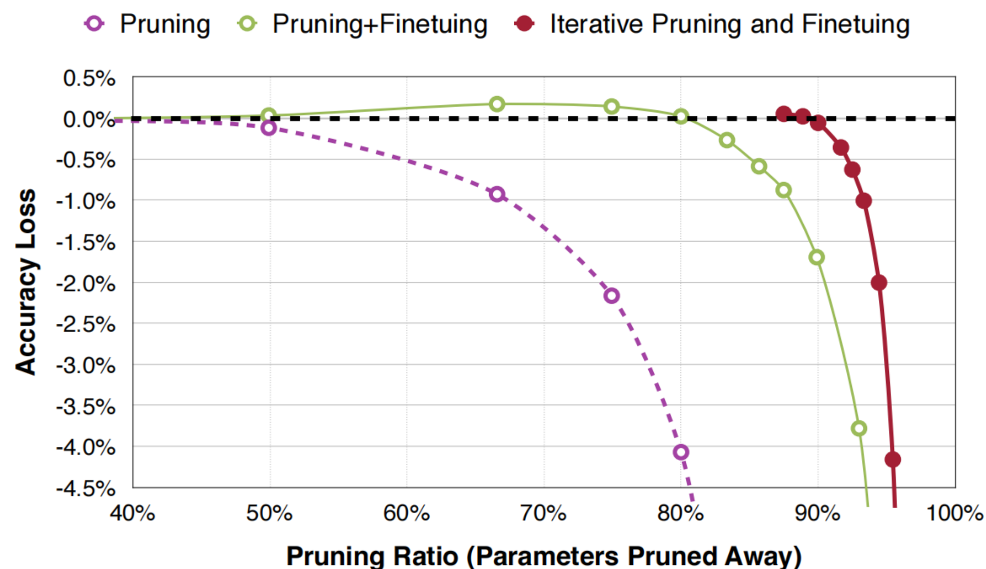
王昕、阿雁宁、李明

基于模型稀疏化的软硬件协同设计

- 除了对于量化噪声的容错性，神经网络往往呈现显著的**稀疏性**，即对特定神经元或者模型权重进行剪枝，不会影响其推理准确率

北京大学-智能硬件体系结构



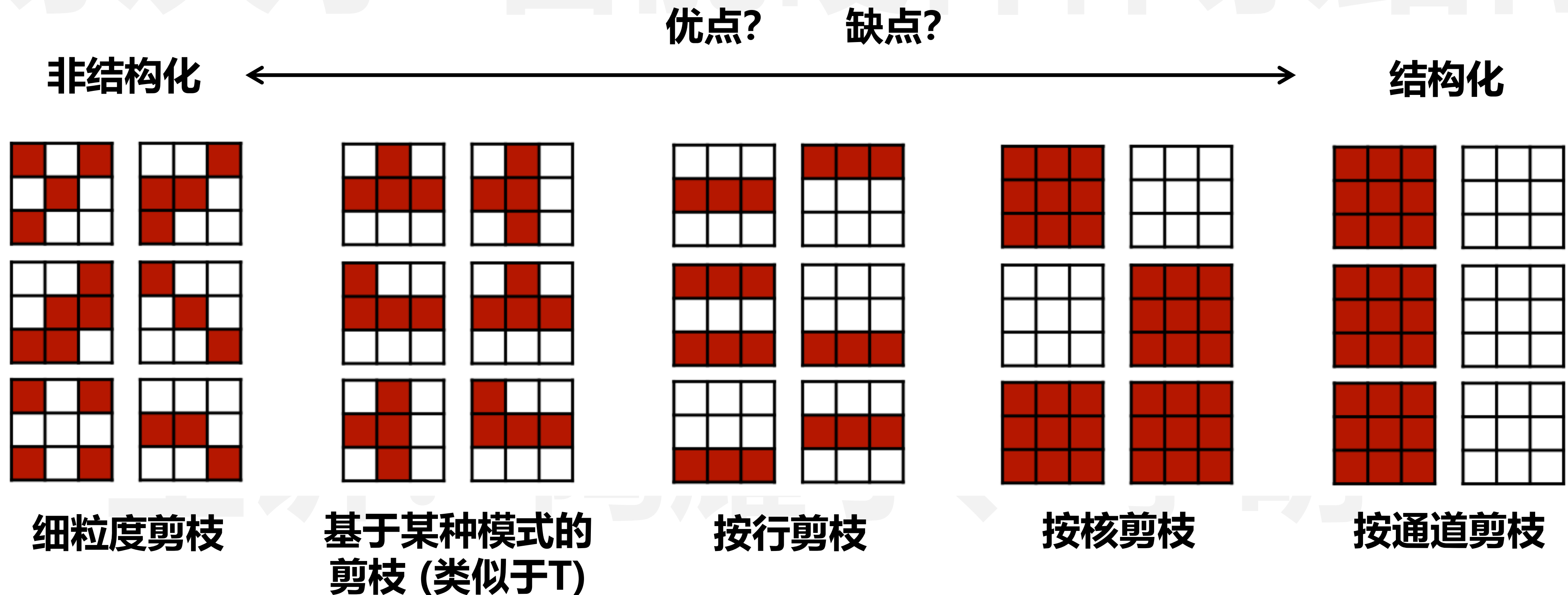


模型稀疏化的Taxonomy

北京大学 PEKING UNIVERSITY

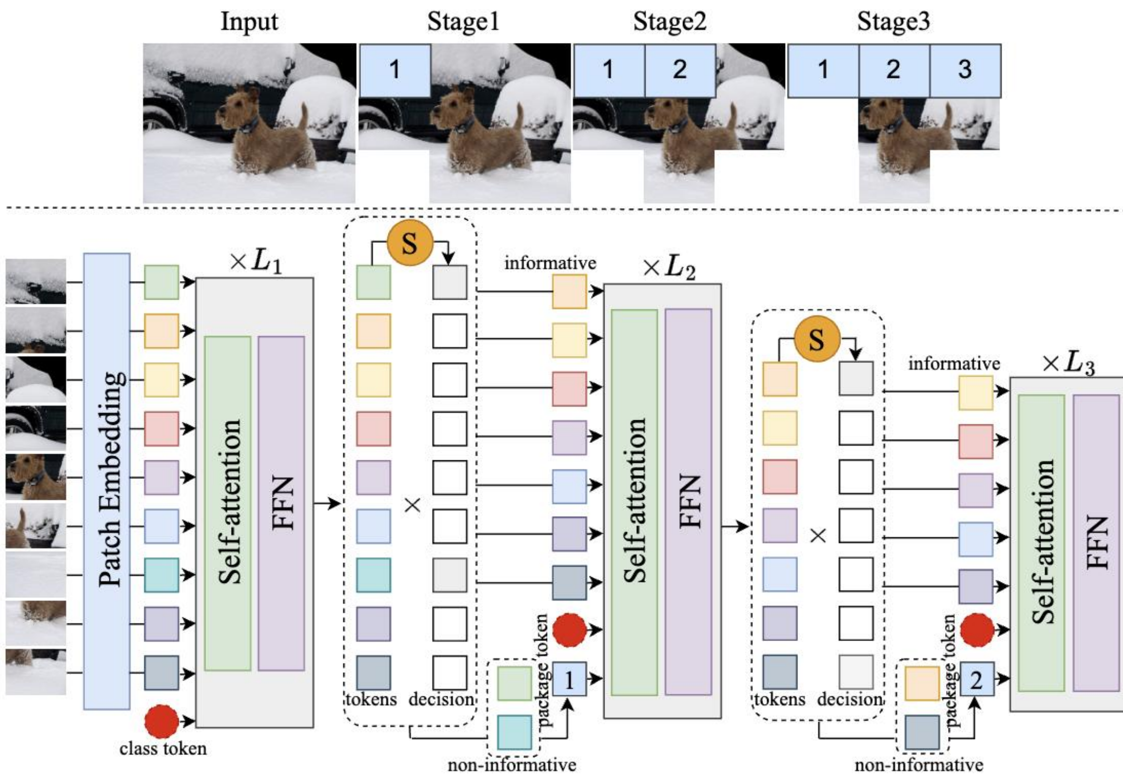
- 依据剪枝粒度不同，神经网络剪枝可以分为**结构化**和**非结构化剪枝**
- 细粒度非结构化剪枝更加灵活，往往准确率更高（或稀疏度更高），但是硬件加速更加困难
- 粗粒度结构化剪枝则更加硬件友好，但是灵活性受限

北京大学-智能硬件体系结构



模型稀疏化的Taxonomy

- 依据剪枝对象不同，神经网络剪枝可以分为**权重剪枝**与**激活值剪枝**
- 权重剪枝适用于CNN、RNN、Transformer等，通常为**静态**剪枝
- 激活值剪枝则更常见于Transformer模型或基于ReLU的CNN模型中，通常为**动态**剪枝



Peiyan Dong et al., *HeatViT: Hardware-Efficient Adaptive Token Pruning for Vision Transformers*, HPCA 2023

北京大学-智能硬件体系结构 秋季学期 李萌

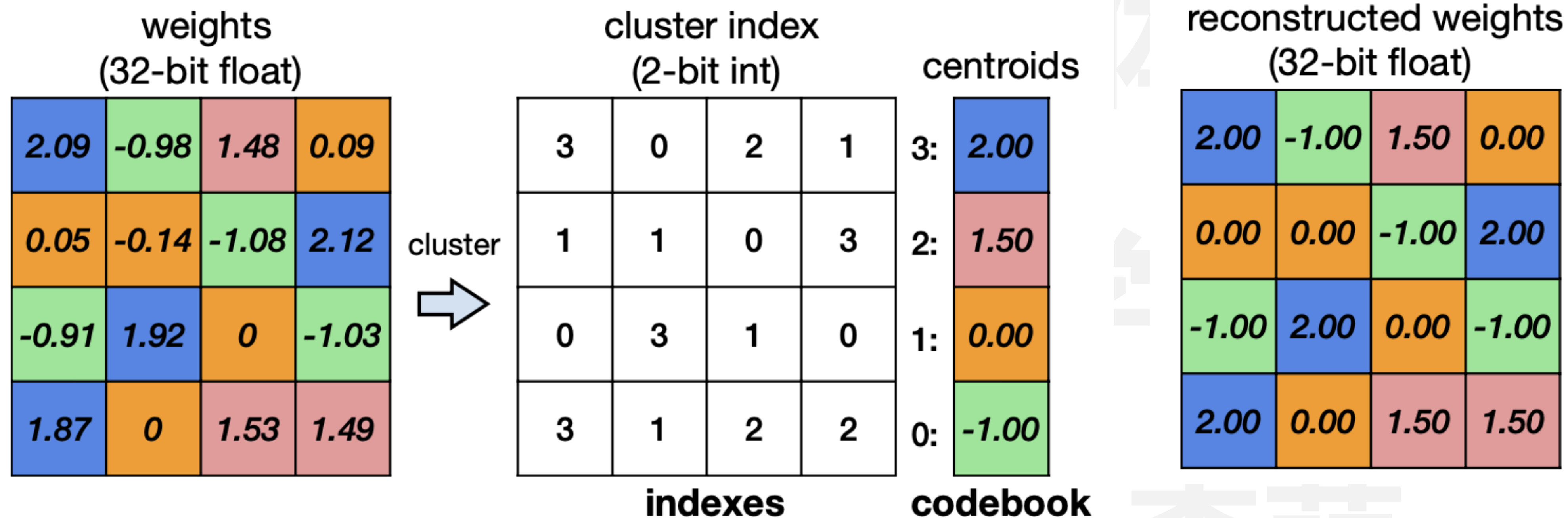
神经网络稀疏化与AI处理器

- 稀疏神经网络想要真正实现计算能效、吞吐等的提升，同样需要底层AI处理器的支持
  - 特别是对于稀疏激活值，导致计算呈现显著动态性
  - 例如动态剪枝单元、动态计算单元等
- 因此，神经网络量化往往需要配合专用硬件（或核函数）设计，形成软硬件协同设计和优化
- 本节课，我们重点介绍3个例子
  - EIE、SNAP：同时考虑权重和激活稀疏的AI处理器架构
  - Nvidia sparse tensor core：商用稀疏AI处理器架构
- 一个很好的Tutorial：Sparse Tensor Accelerator Modeling Tutorial @ ISCA 2021

北京大学-智能硬件体系结构 2024年秋季学期 主讲：陶耀宇、李萌

北京大学 PEKING UNIVERSITY

- 文章发表于ISCA 2016，针对基于量化和剪枝的压缩模型，提出了高效推理引擎EIE



Song Han et al., *EIE: efficient inference engine on compressed deep neural network*, ISCA 2016

神经网络量化与AI处理器 —— EIE

- 文章发表于ISCA 2016，针对基于量化和剪枝的压缩模型，提出了高效推理引擎EIE
- 核心观察：缺少有效处理不规则存储的稀疏数据的硬件单元
  - 静态的权重稀疏+动态的激活稀疏
- 文章提出高效推理引擎EIE，实现了高效的稀疏矩阵向量乘法

北京大学 智能硬件体系结构
年秋季学期
陶耀宇、李萌

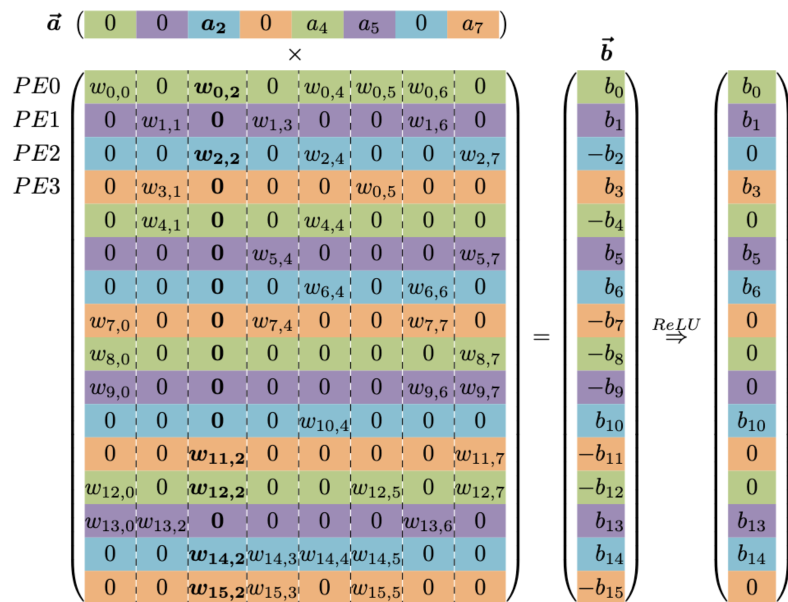


Figure 2. Matrix W and vectors a and b are interleaved over 4 PEs. Elements of the same color are stored in the same PE.

- 权重稀疏通过 Index + value 存储为 Compressed sparse column (CSC) 格式
- 激活稀疏通过只向PE广播非零激活值实现
- 权重共享通过将权重固定为16个值并用4位索引查找实现

神经网络量化与AI处理器 —— EIE

- **Act Queue平衡负载**：非零激活值对应的一列权重中的非零数量不同导致不同PE间负载不平衡
- Leading Non-zero Detection Node：检测非零激活值并广播到所有PE

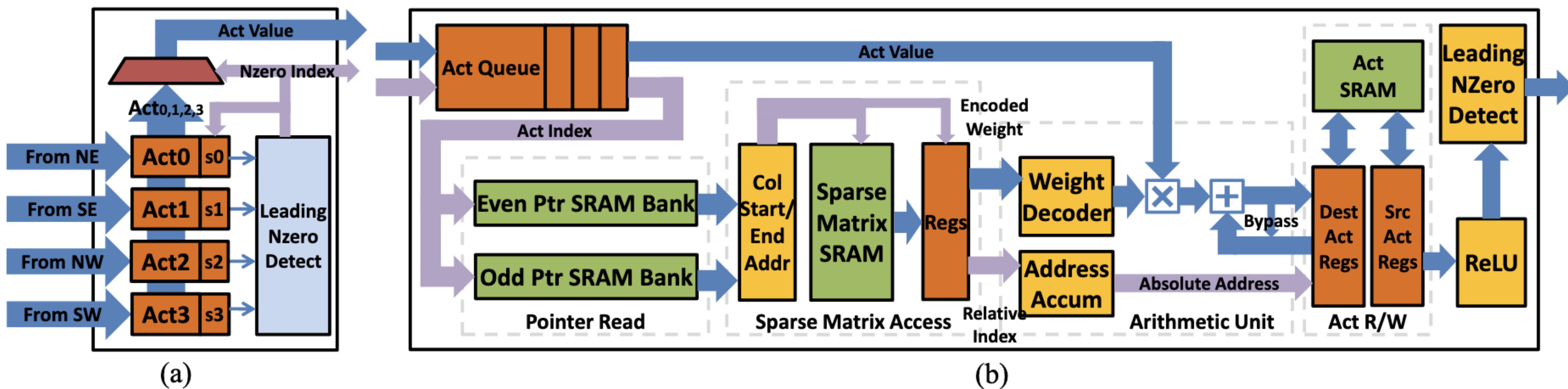
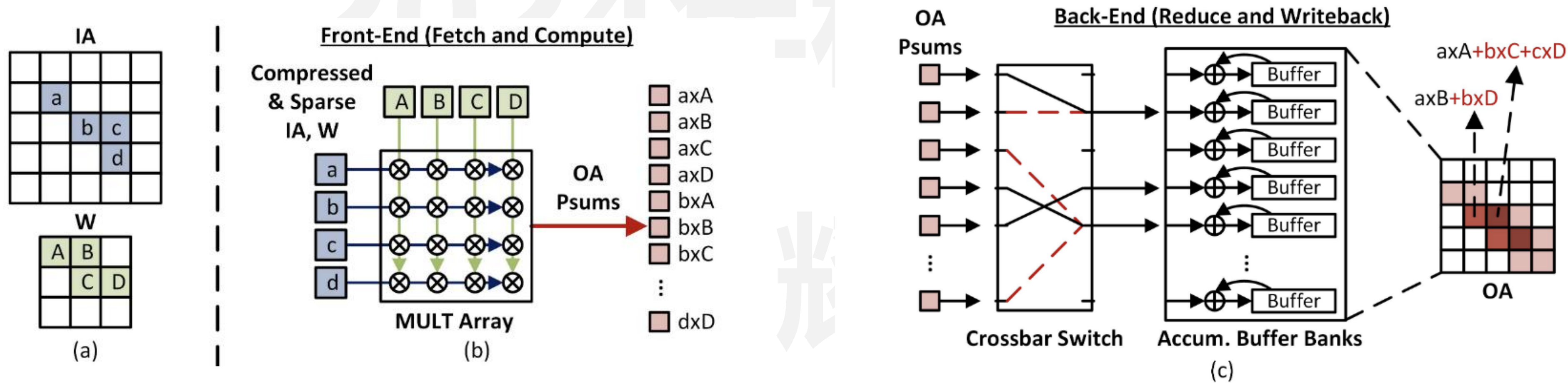


Figure 4. (a) The architecture of Leading Non-zero Detection Node. (b) The architecture of Processing Element.

神经网络量化与AI处理器 —— SNAP

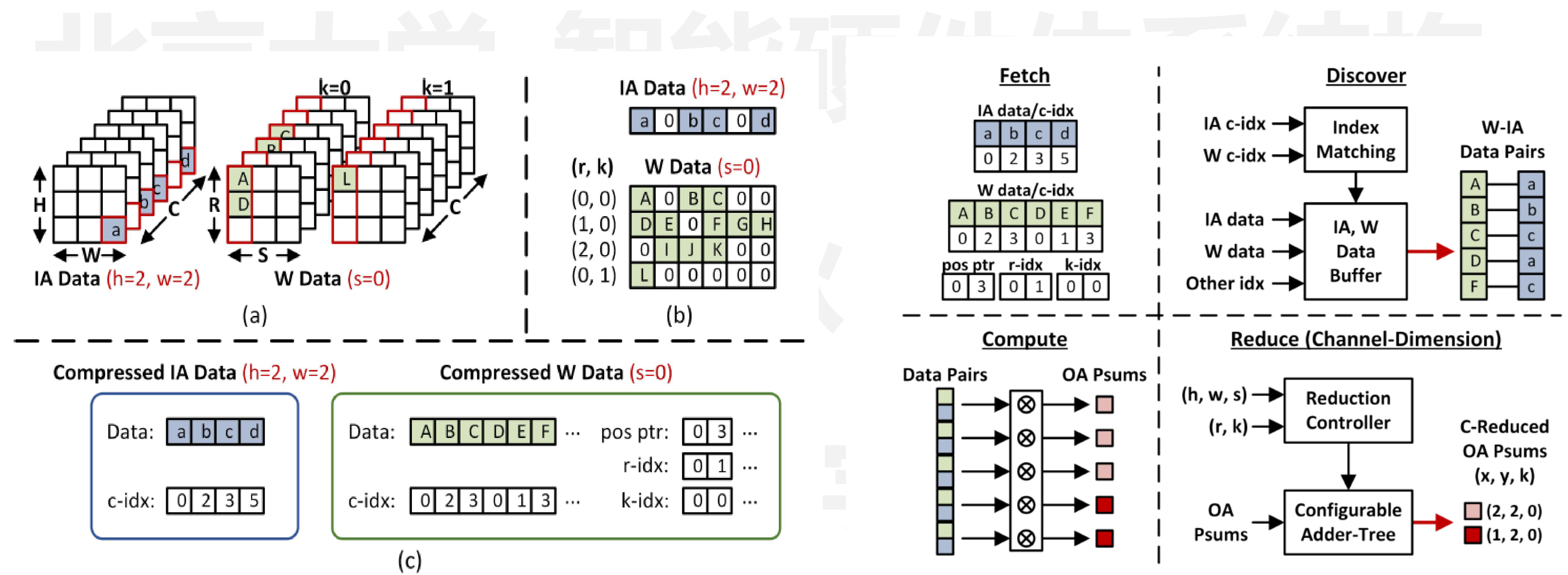
- 文章发表于JSSC 2021，通过channel-first数据流对稀疏网络模型进行了硬件加速
- 核心观察：数据稀疏使得网络推理更加高效，但现有稀疏计算的数据流仍面临以下挑战
  - Front-end：读取的W-IA pairs数量不足导致计算单元利用率低
  - Back-end：计算结果写回的地址冲突
  - 灵活性：对于不同kernel大小和层类型的支持性不足



Jie-Fang Zhang et al., *SNAP: An Efficient Sparse Neural Acceleration Processor for Unstructured Sparse Deep Neural Network Inference*, JSSC 21

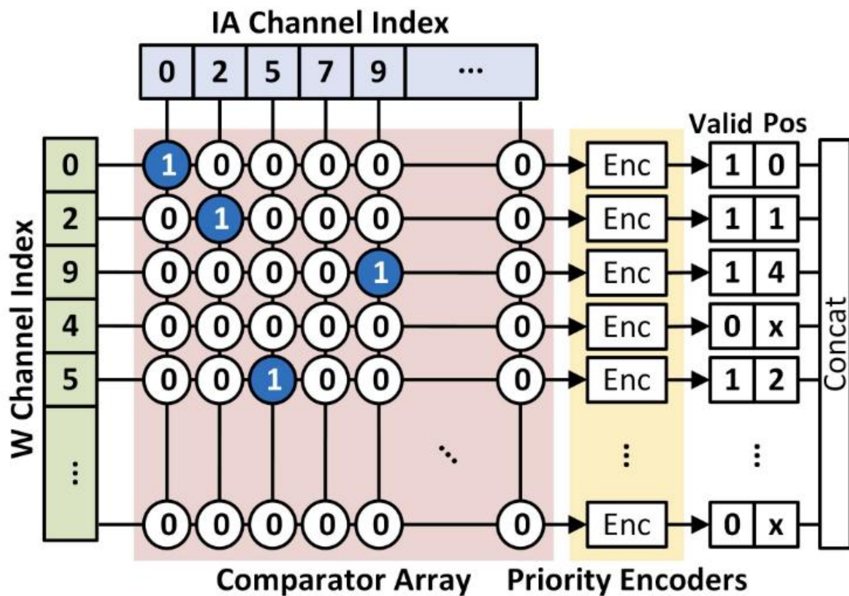
北京大学 PEKING UNIVERSITY

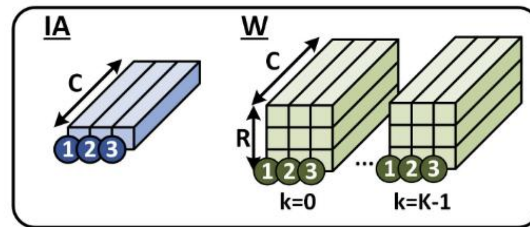
- Channel-first数据流：同一个channel的可以任意配对，产生的乘法结果都是有用的
- 通过可配置加法树实现先**归约加和**，然后更新Psum，从而减少Psum更新时的地址冲突

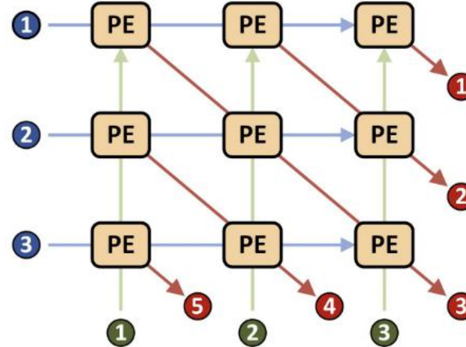


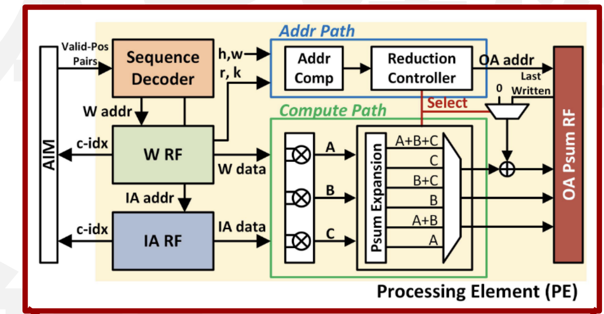
神经网络量化与AI处理器 —— SNAP

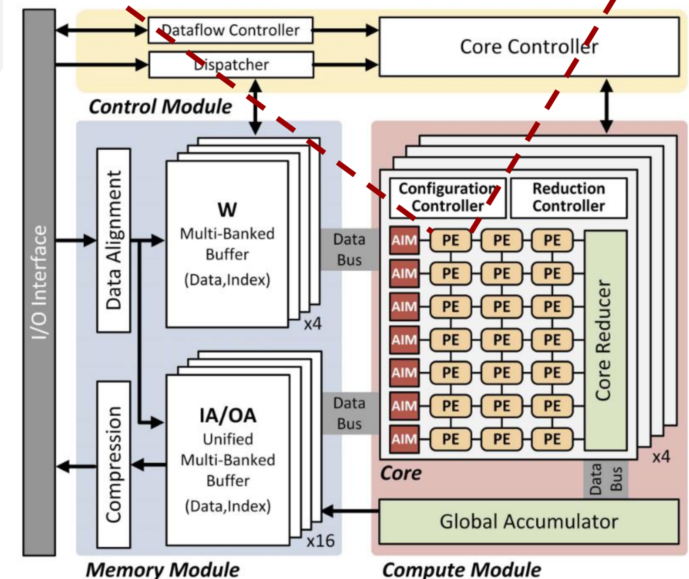
- Front-end：Index Matching单元提取足够数量的 W-IA pairs，提高乘法阵列利用率
- Back-end：两级 partial sum 压缩，PE-level 和 core-level，减少内存访问冲突
- 灵活性：core-level的压缩可以支持不同层的计算





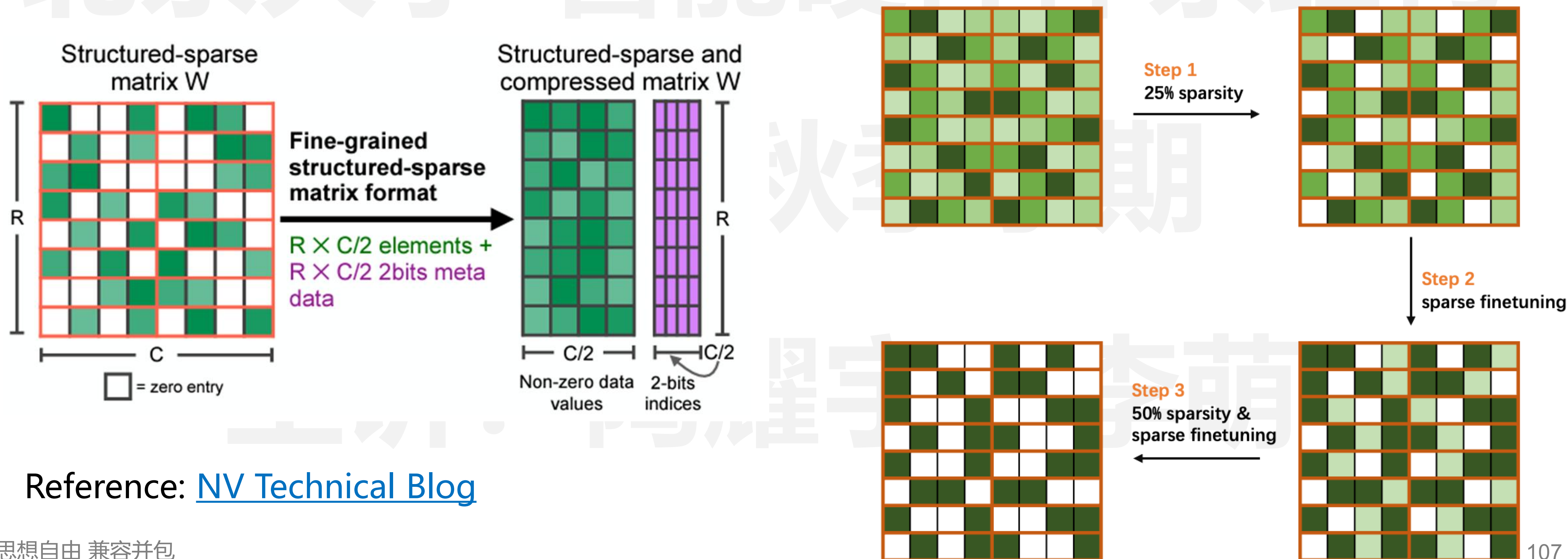






神经网络量化与AI处理器——NV Sparse TensorCore

- 2020年，英伟达推出Ampere架构，支持稀疏张量计算
- Ampere架构支持固定的2:4稀疏模式，即每4个元素中，2个为0，并且能够实现2倍的加速比
- 该稀疏模式可以拓展为更加普适的N:M稀疏，即每M个元素中，N个为0



Reference: NV Technical Blog