北京大学
PEKING UNIVERSITY

人工智能的硬件基石
从物理器件到计算架构

第十一讲：AI芯片设计-III

主讲：陶耀宇

2026年春季

注意事项

北京大学 PEKING UNIVERSITY

- 课程作业情况

- **作业时间**
  - 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28日（因CLAB故障延期）**
- **第2次lab时间：5月23日-6月18日（因CLAB故障延期，已适当简化）**
- **文献调研与报告（6月1日、6月8日）（形式待定，可以录视频）**

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）

目 录

CONTENTS

AI加速器架构发展——Google TPU v1

北京大学 PEKING UNIVERSITY

- 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	

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

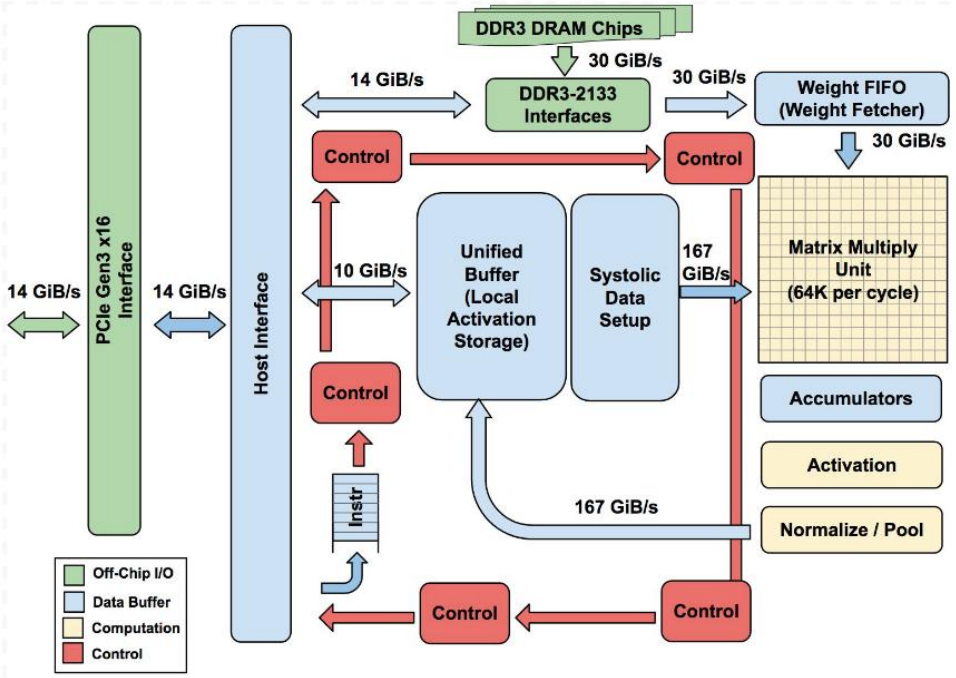
The synergy between the large data sets in the cloud and the numerous computers that power it has enabled a renaissance in machine learning. In particular, *deep neural networks* (DNNs) have led to breakthroughs such as reducing word error rates in speech recognition by 30% over traditional approaches, 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.

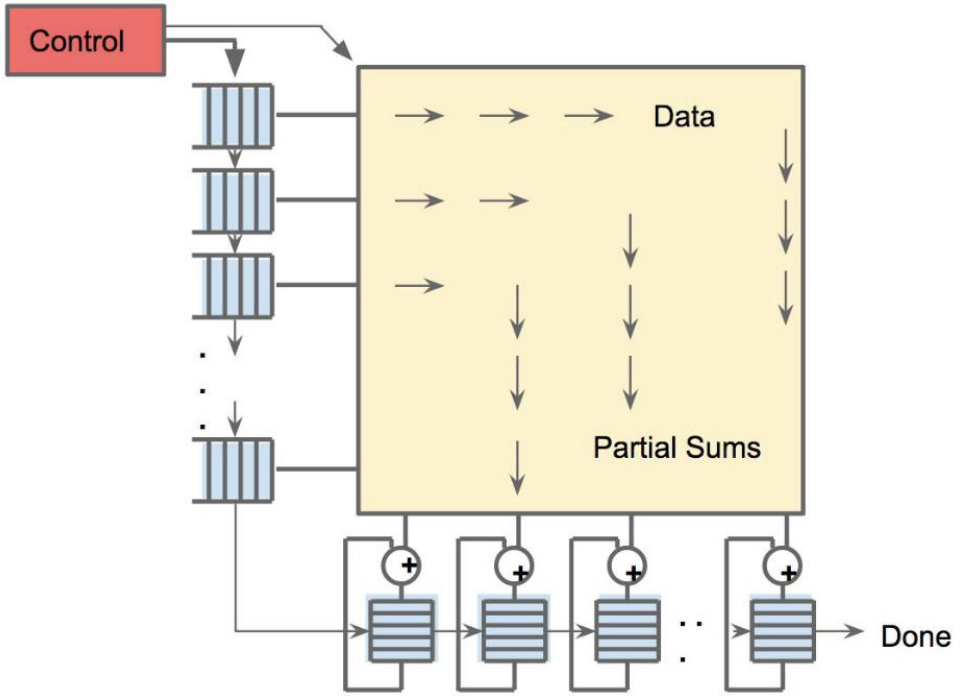
Neural networks (NN) target brain-like functionality and are based on a simple artificial neuron: a nonlinear function (such as `max(0, value)`) of a weighted sum of the inputs. These artificial neurons are collected into layers, with the outputs of one layer becoming the inputs of the next one in the 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

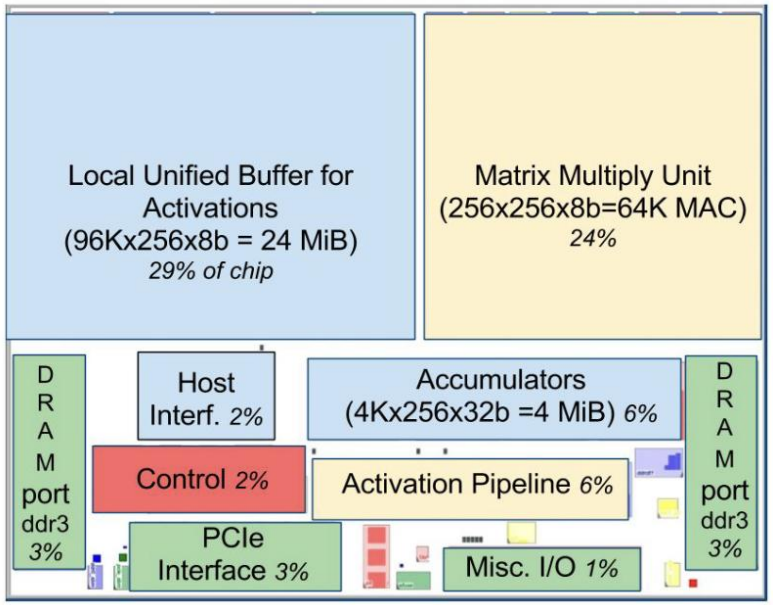
- TPU设计参数
  - 矩阵乘法单元：256 x 256，支持稠密矩阵乘法（不支持稀疏计算），double buffering，每周期产生256个partial sum，支持8比特权重，8/16比特输入
  - 累加单元：4 MB缓存（4096 x 256 x 32b），double buffering
  - 片上存储: MMU存储64KB权重，FIFO深度为4，激活存储为24MB

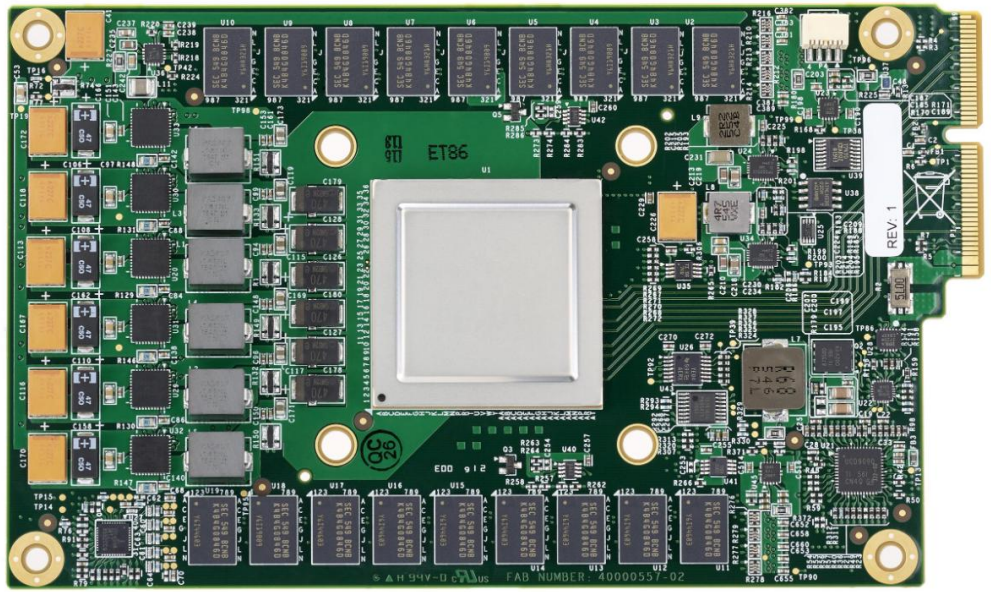




AI加速器架构发展——Google TPU

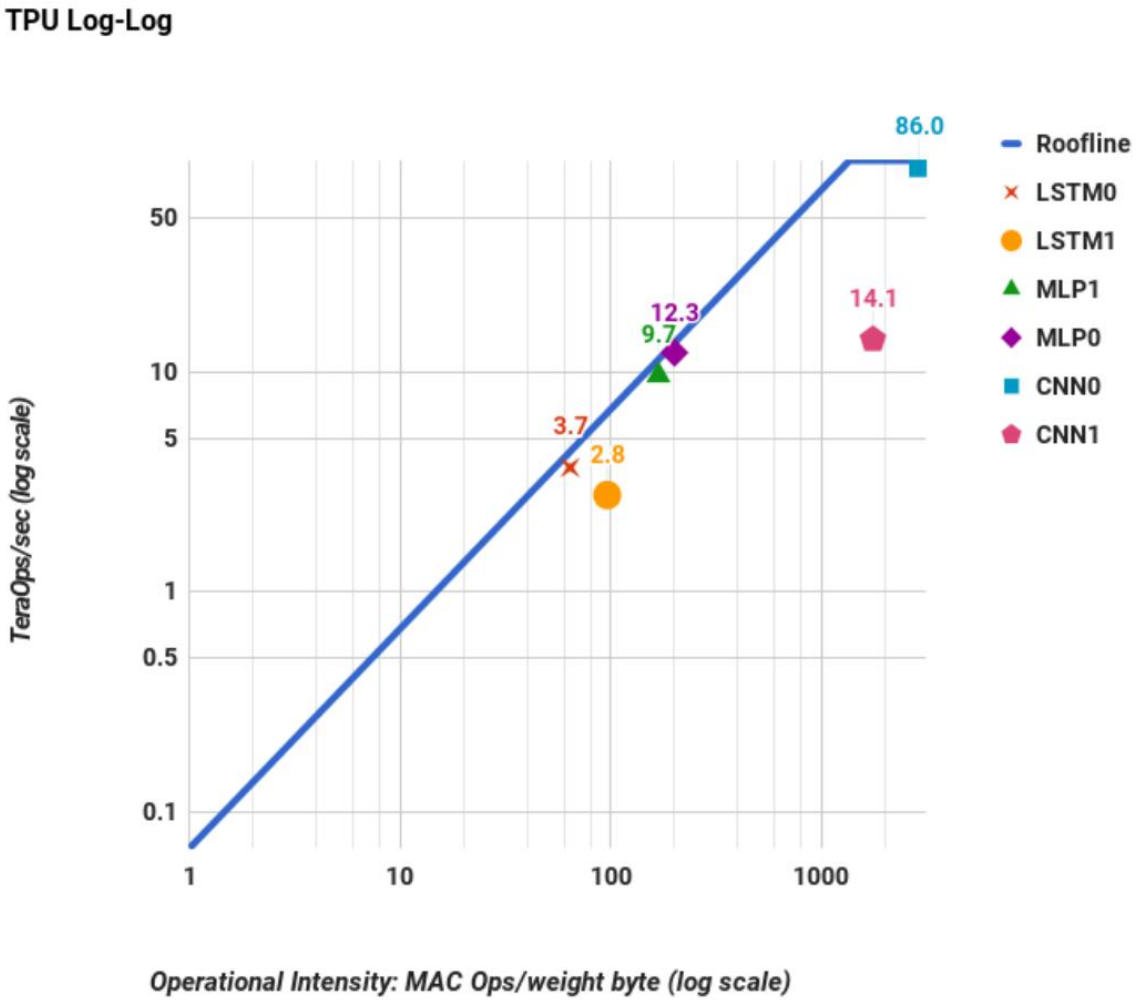
- 在TPU的版图中，数据通路面积占比为67%，I/O面积占比10%，控制占比2%
- 采用CISC指令，其中5个重要指令包括
  - Read_Host_Memory, Read_Weights, MatrixMultiply/Convolve, Activate, Write_Host_Memory





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

北京大学 PEKING UNIVERSITY

- 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...

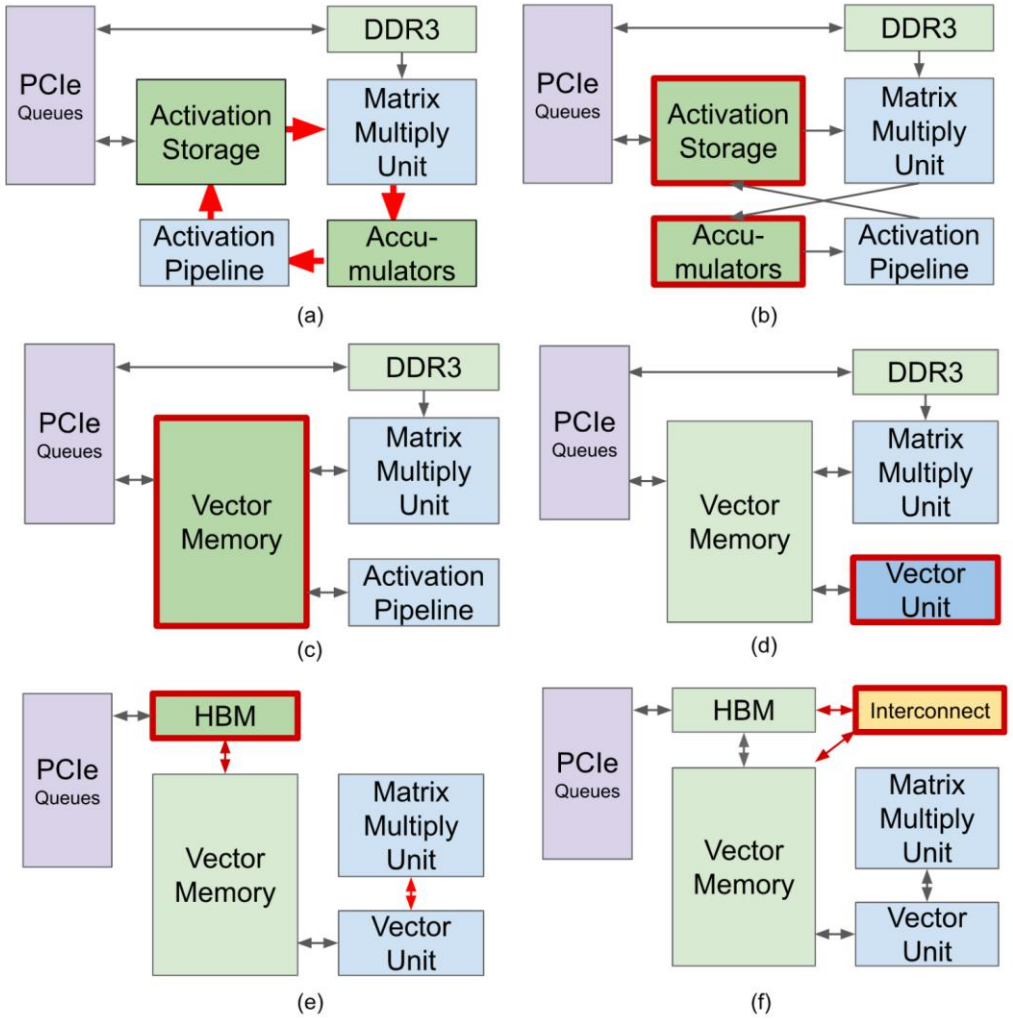
...at scale

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

...all while being cost effective

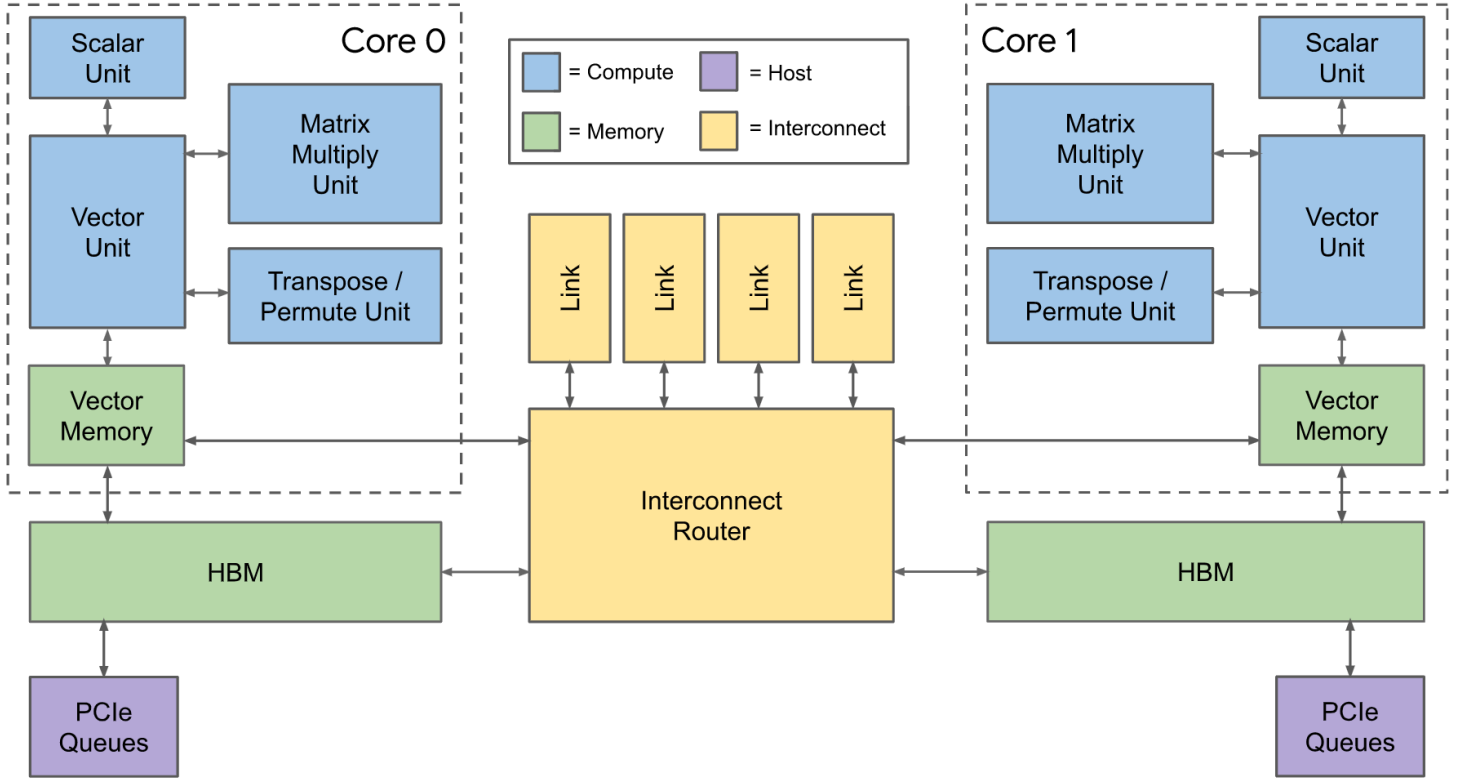
AI加速器架构发展——Google TPU v2

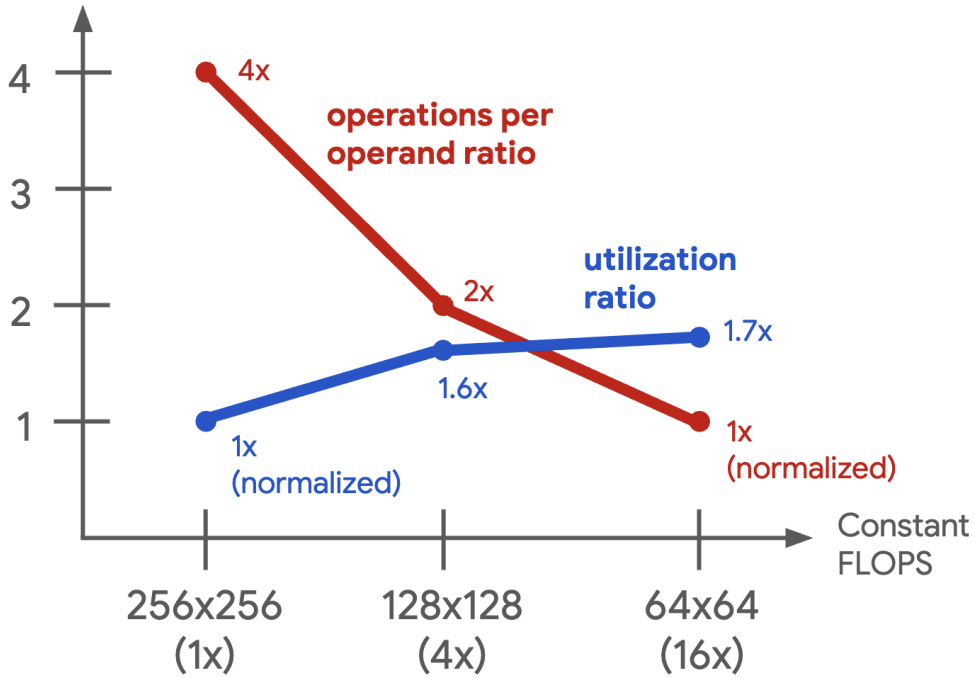
- 相较于TPU v1，TPU v2的**核心改动包括**
  - 将累加便签存储器、激活值便签存储器合并
  - 将累加器改为更为灵活、可编程的向量处理单元
  - 将MMU改为向量单元的协处理器，简化访存需求
  - 将DDR3改为HBM，提供更高访存带宽
  - 增加多卡互联，提升多卡扩展效率



AI加速器架构发展——Google TPU v2

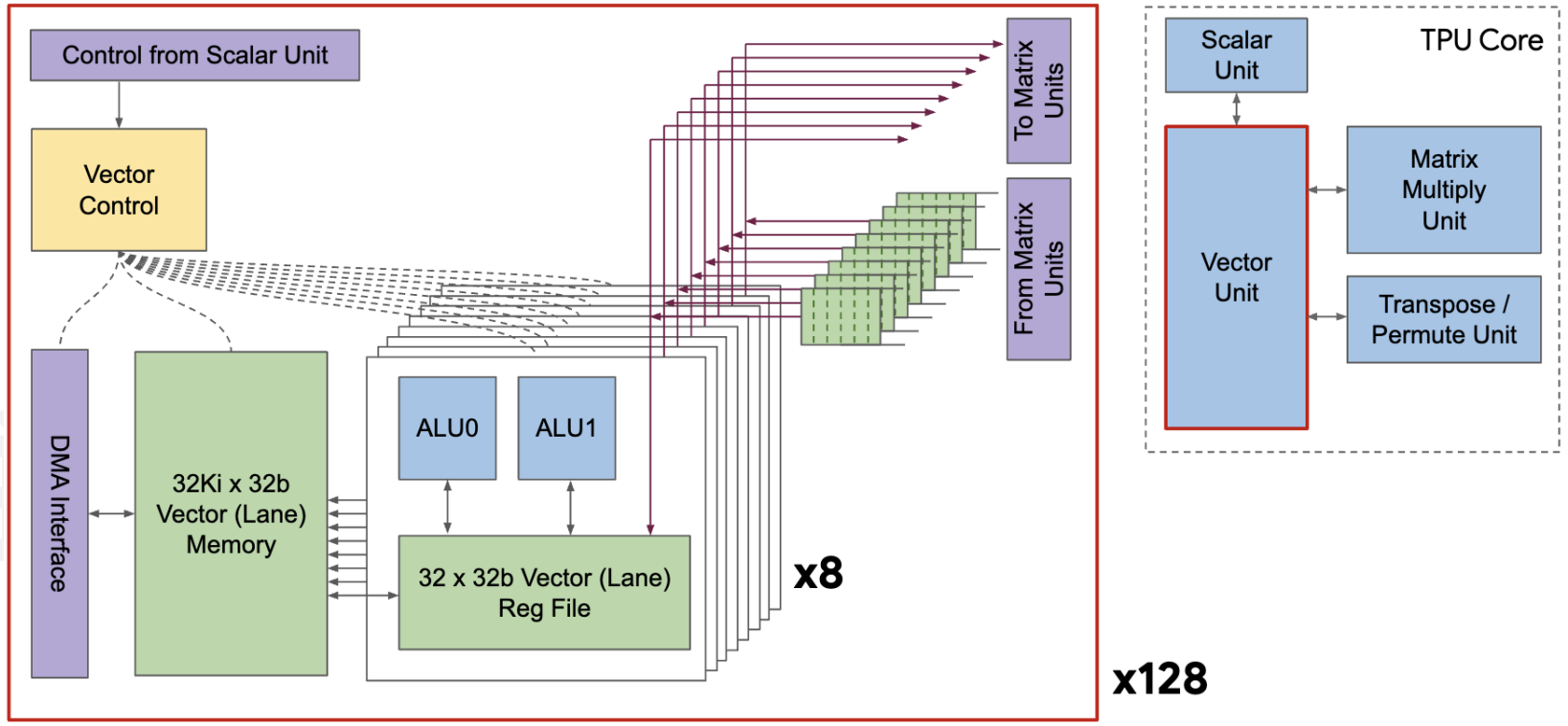
- 采用双核架构，ISA支持矩阵、向量和标量计算，注重通用性
- 矩阵计算单元采用128 x 128的脉动阵列，支持bf16（s1e8m7）乘法和FP32累加
- 为什么要选择128 x 128的脉动阵列，而不是TPU v1中的256 x 256?





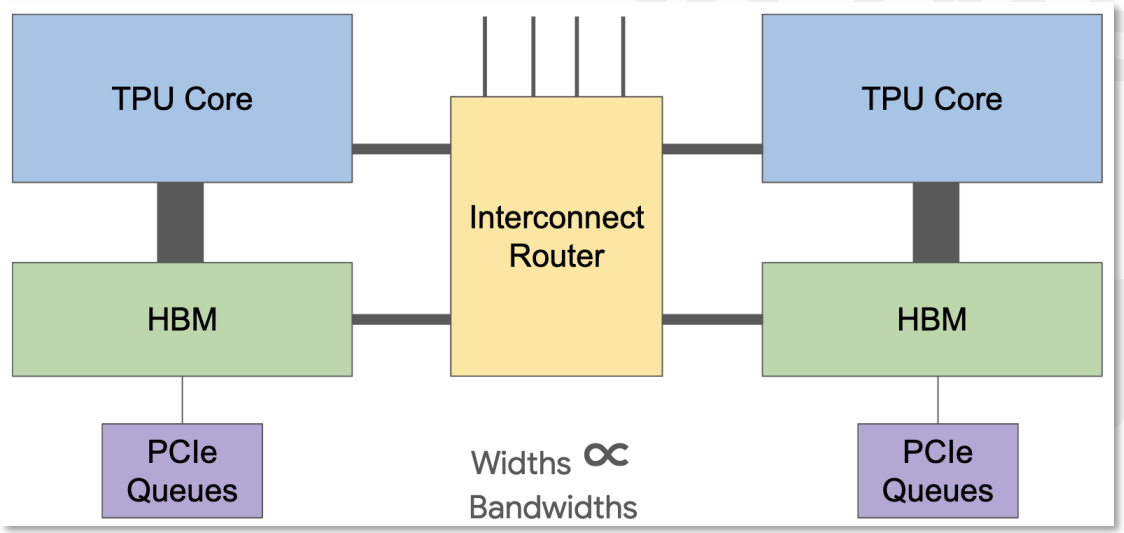
AI加速器架构发展——Google TPU v2

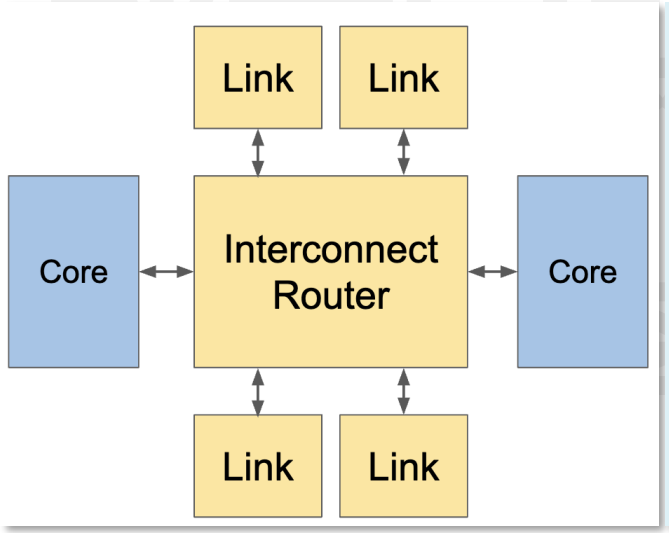
- 向量计算单元的功能：
  - 8个向量计算核，每个核每个周期处理128个输入
  - 为矩阵计算单元提供输入

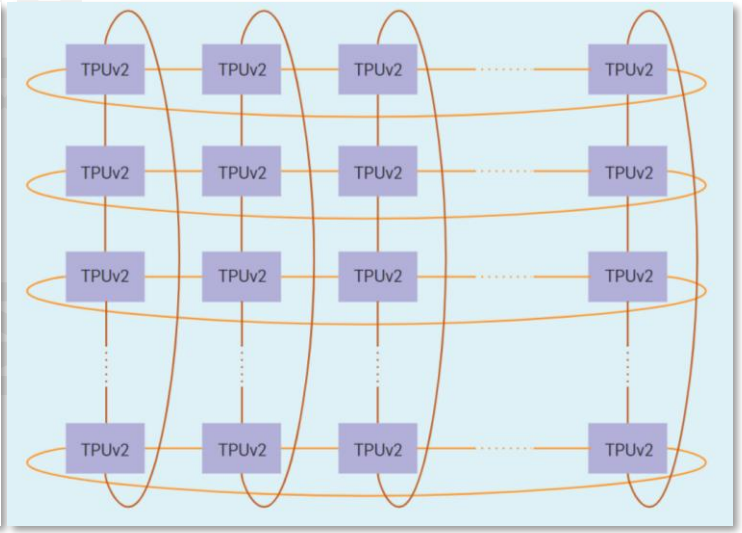


AI加速器架构发展——Google TPU v2

- 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

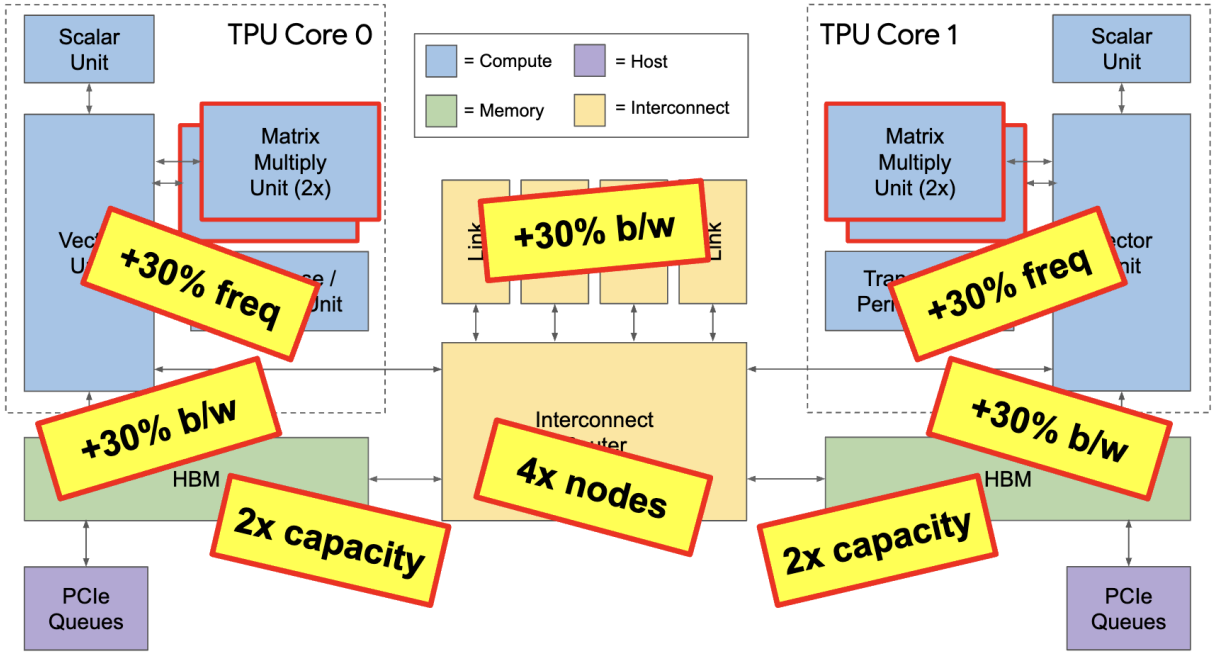






AI加速器架构发展——Google TPU v3

- 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

人工智能处理器芯片

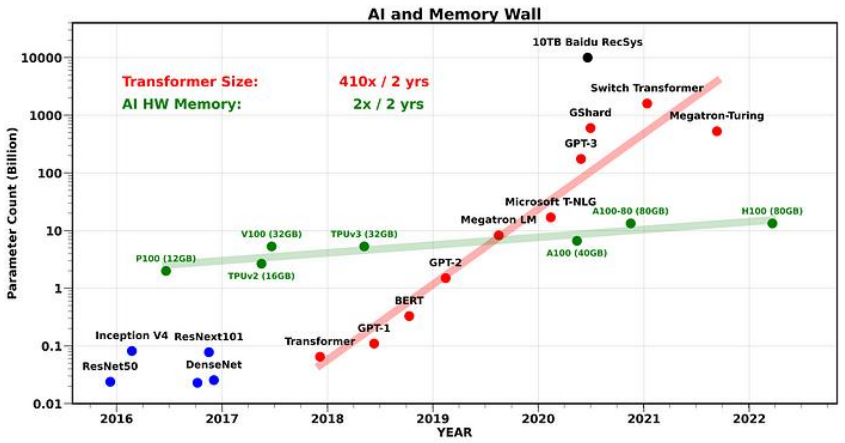
北京大学 PEKING UNIVERSITY

- 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

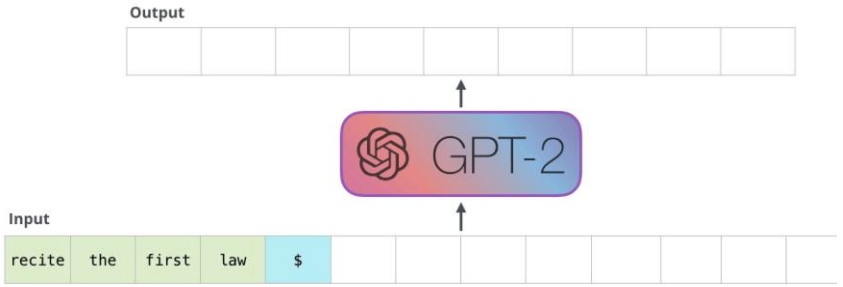
人工智能模型的发展与挑战

- 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

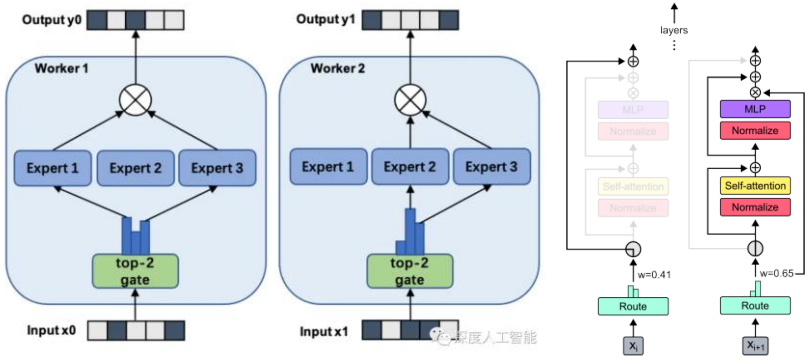
模型参数指数级增长，造成单模型存储需求>1 TB



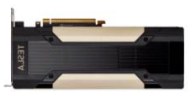
大模型自回归解码，导致平均访存带宽需求>1 TB/s



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

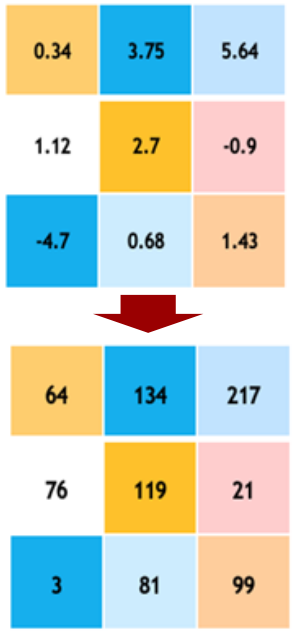


	Cloud AI	Mobile AI	Tiny AI
Memory	24 GB	4 GB	500 KB
Storage	~ TB/PB	~100s GB	~1s MB
# params	>70 B	3 – 13 B	~1s M

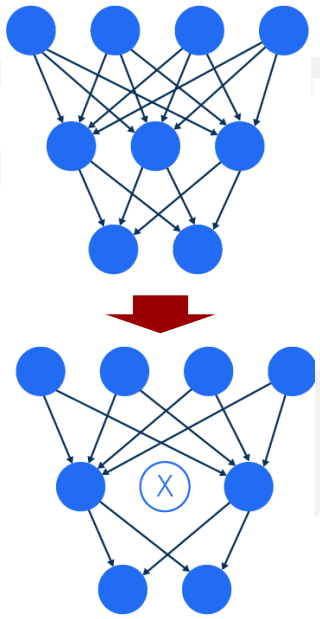
人工智能模型的发展与挑战

- 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

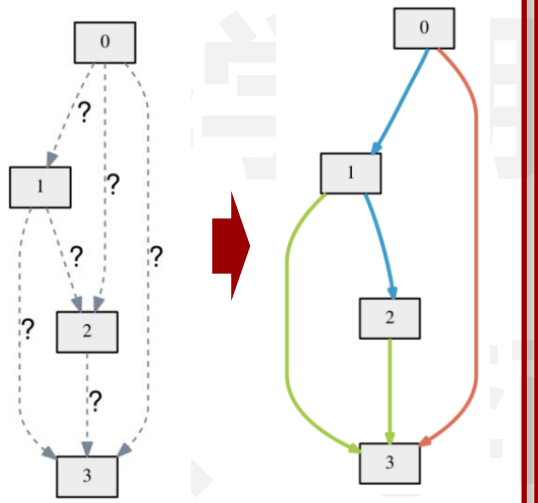
模型量化



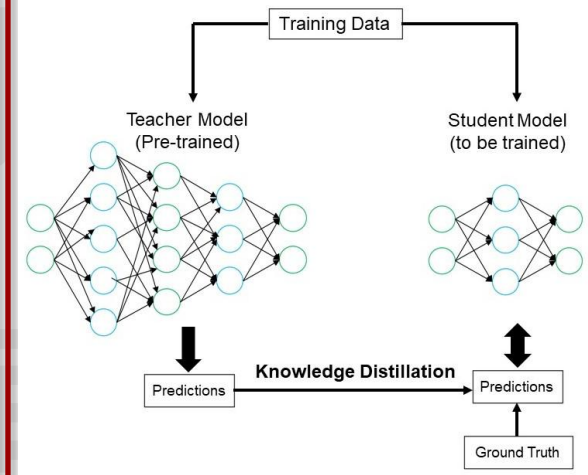
模型稀疏化



神经网络架构搜索

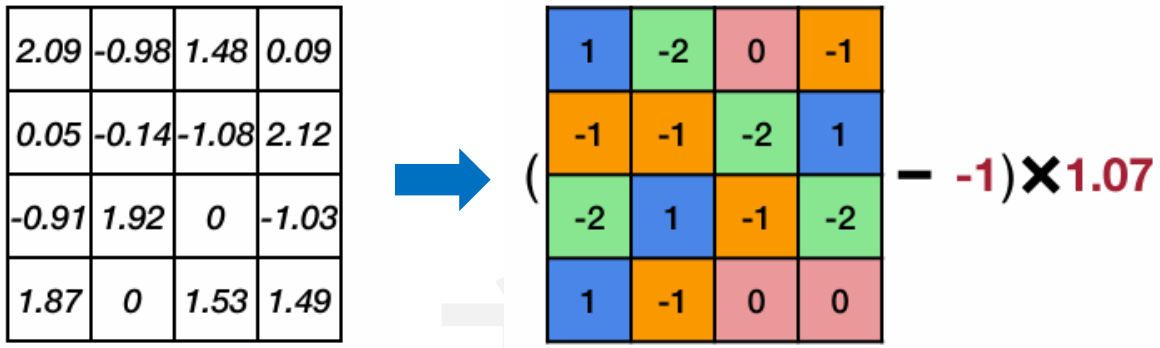


知识蒸馏



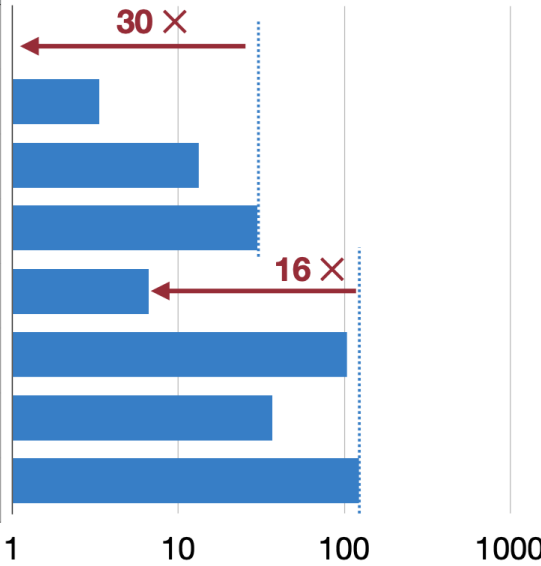
基于模型量化的软硬件协同设计

- 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



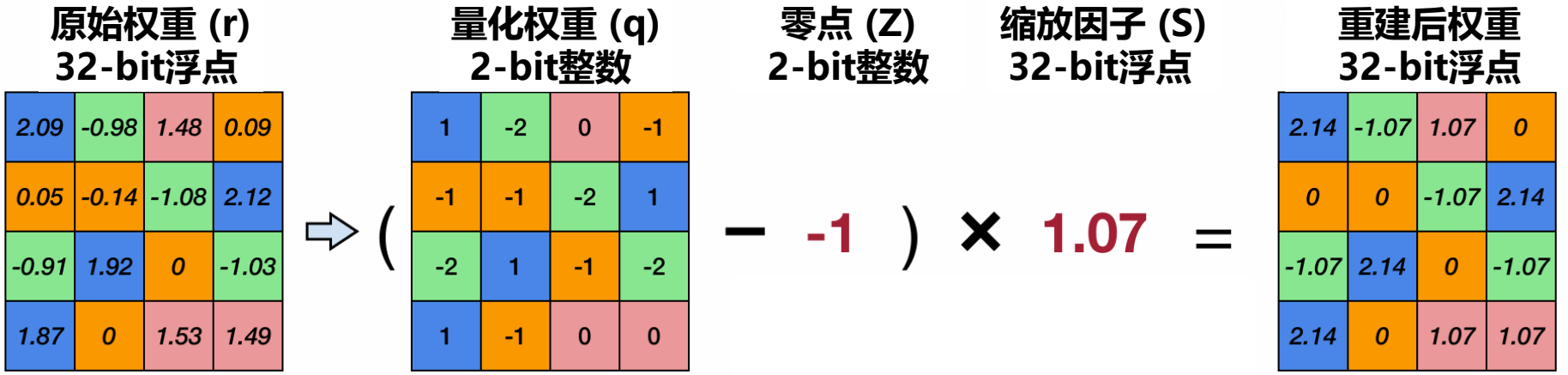
Operation	Energy [pJ]
8 bit int ADD	0.03
32 bit int ADD	0.1
16 bit float ADD	0.4
32 bit float ADD	0.9
8 bit int MULT	0.2
32 bit int MULT	3.1
16 bit float MULT	1.1
32 bit float MULT	3.7

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



2-bit整数表示方法

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

量化误差

-0.05	0.09	0.41	0.09
0.05	-0.14	-0.01	-0.02
0.16	-0.22	0	0.04
-0.27	0	0.46	0.42

- 权重的量化与激活值的量化对于计算的影响不同

$$Y = WX$$

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

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

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

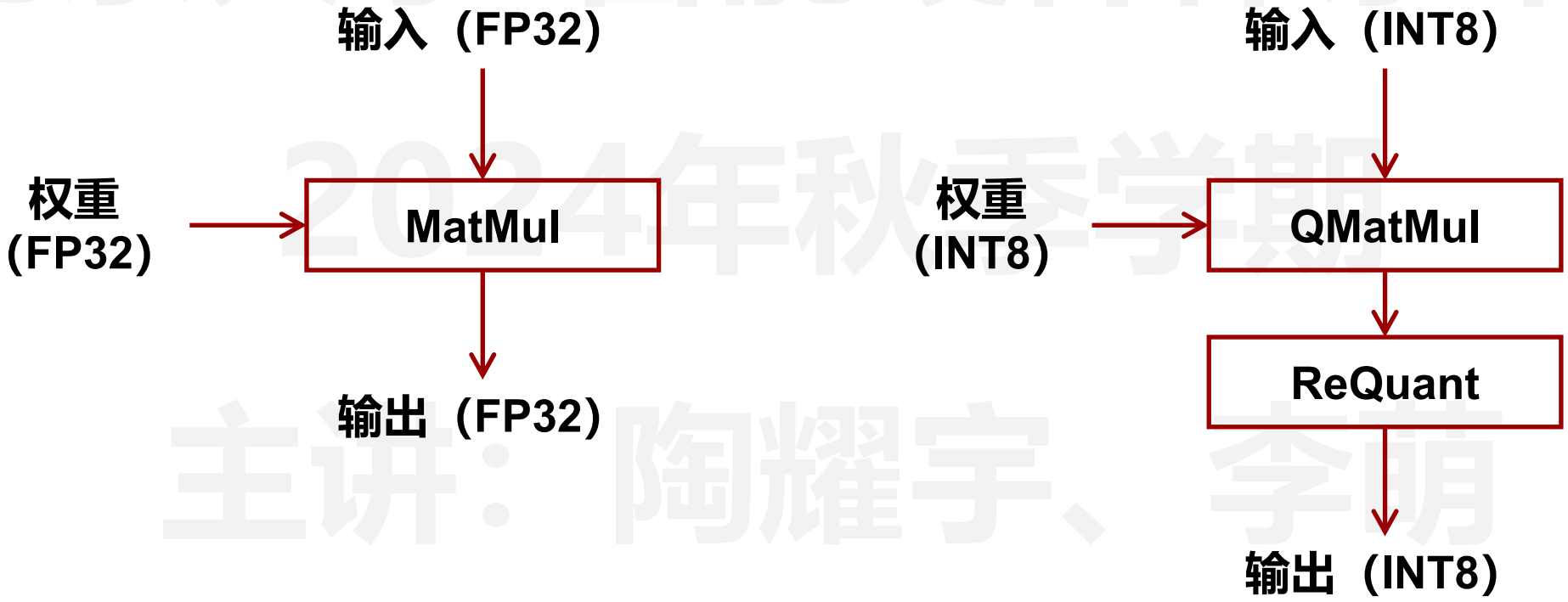
QMatMul

Z_w通常为0

提前计算

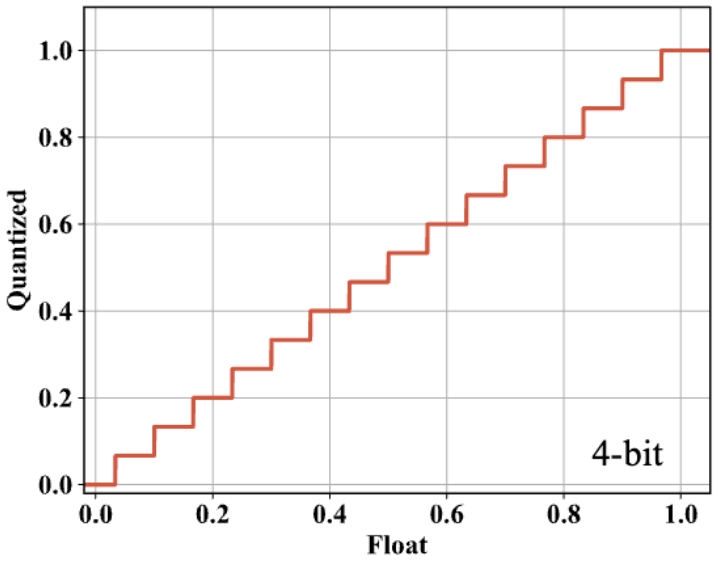
量化神经网络基础

- 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?

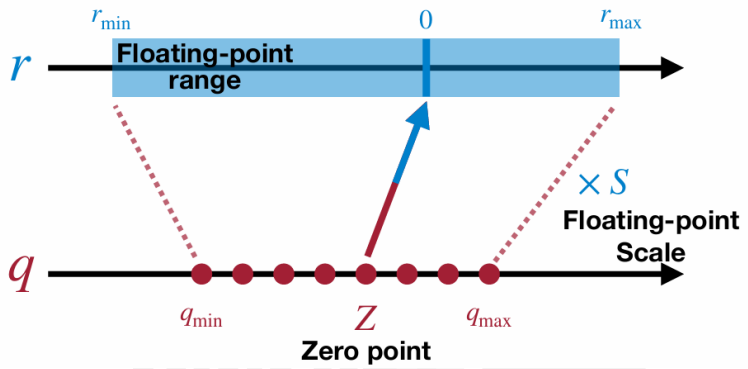


均匀量化与非均匀量化

- **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**



(a) Uniform Quantization



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

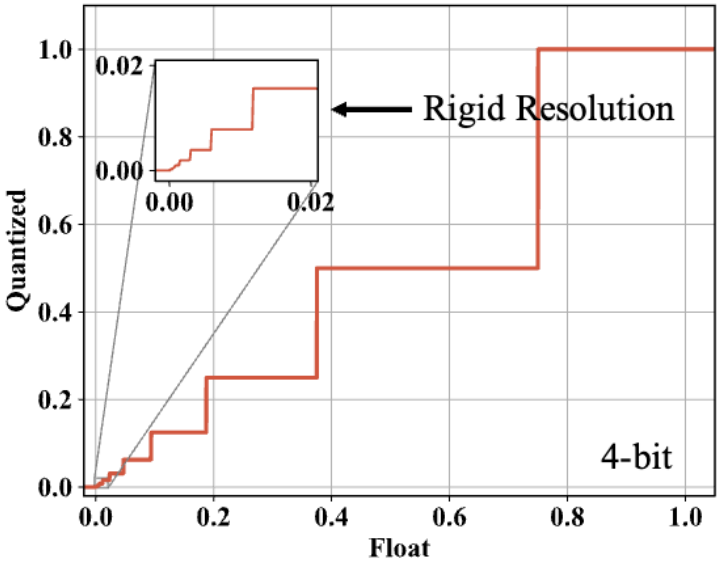
$$r_{min} = S\,(q_{min} - Z)$$

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

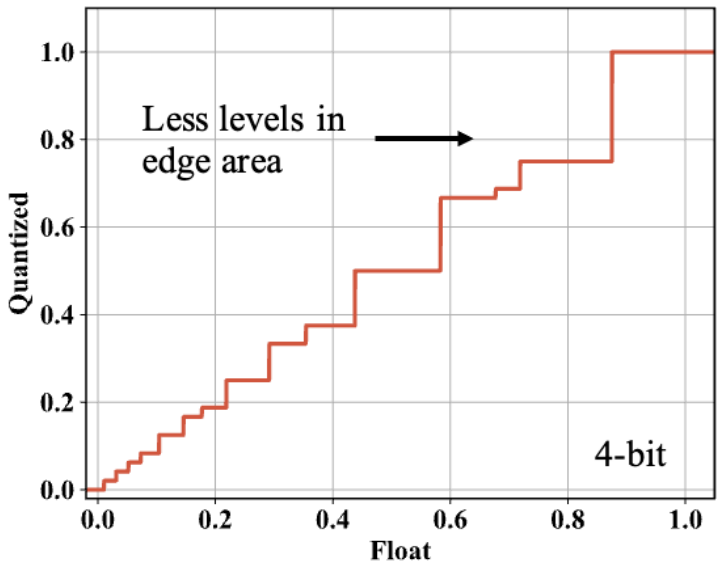
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

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



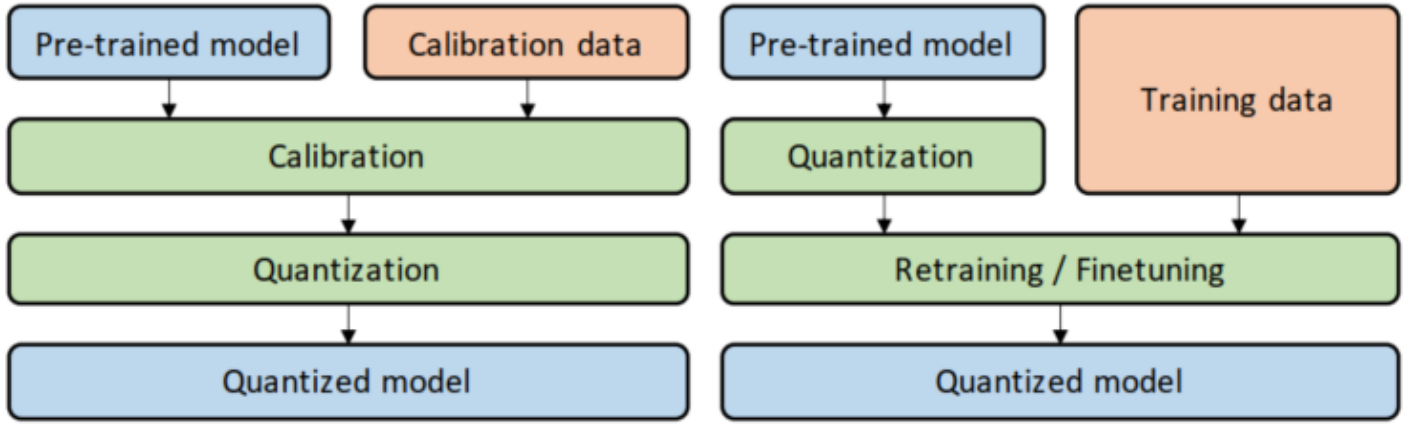
(c) Additive Power-of-Two Quantization

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

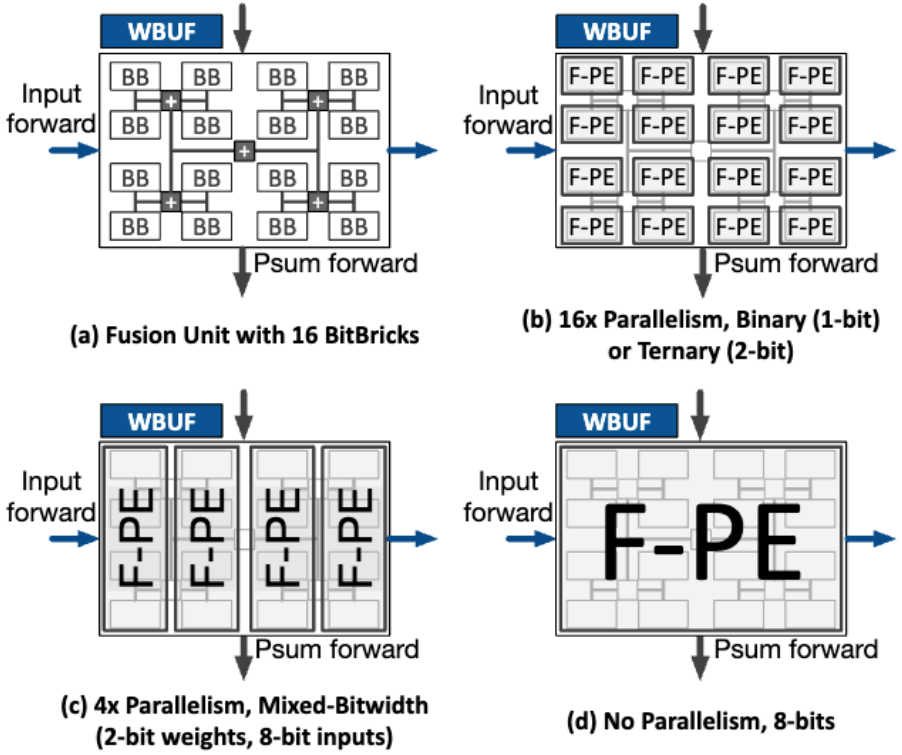
神经网络量化与AI处理器

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

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

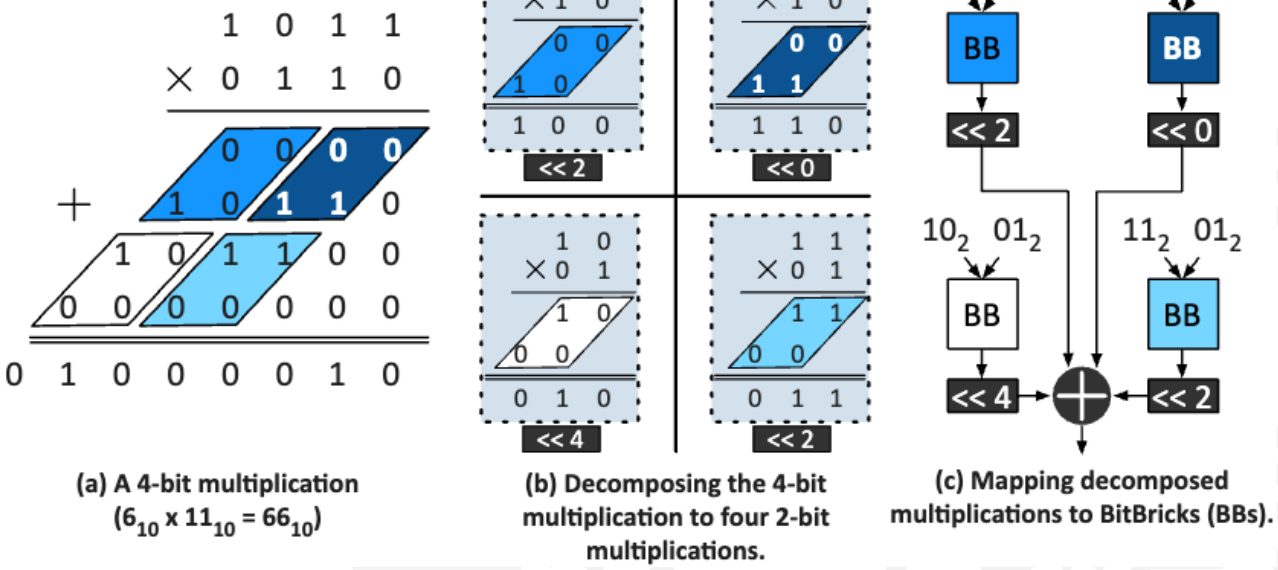


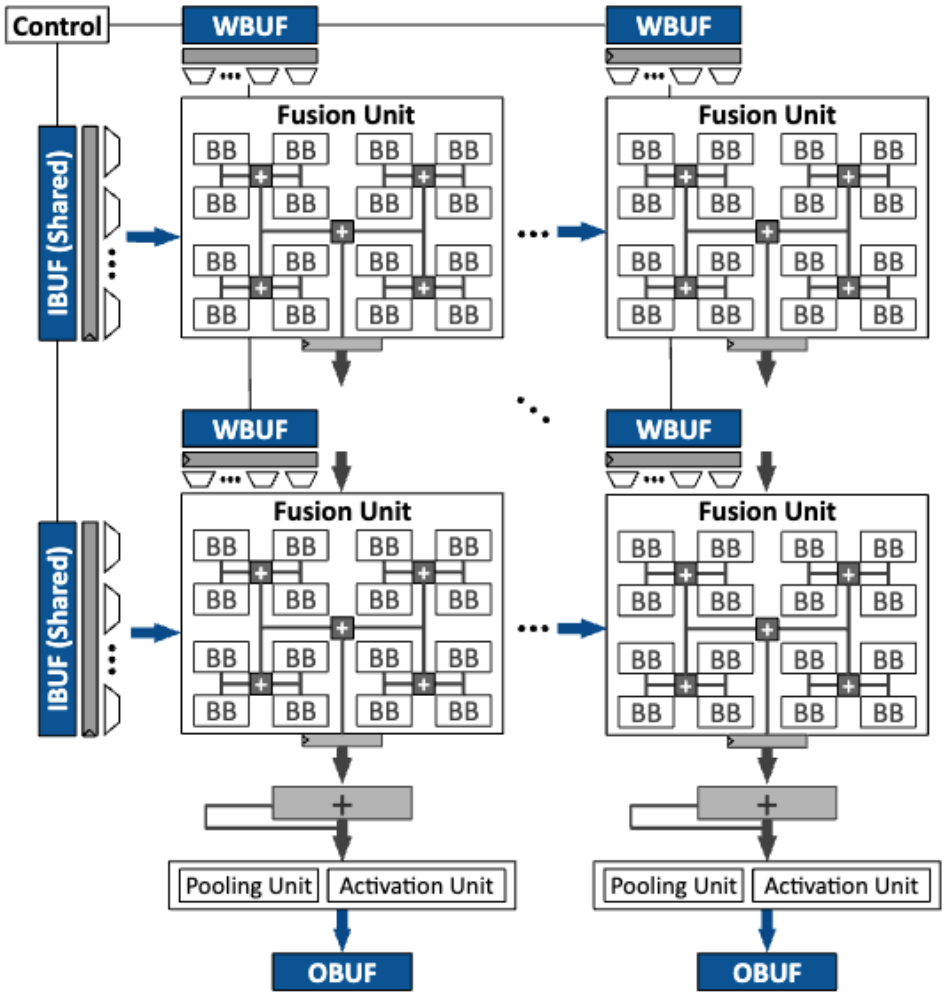
- Fusion单元由16个BitBricks组成，每个BitBrick每周期可以支持2比特计算
- 多个BitBricks组合，能够实现W4A4、W2A8、W8A2以及W8A8的计算

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





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

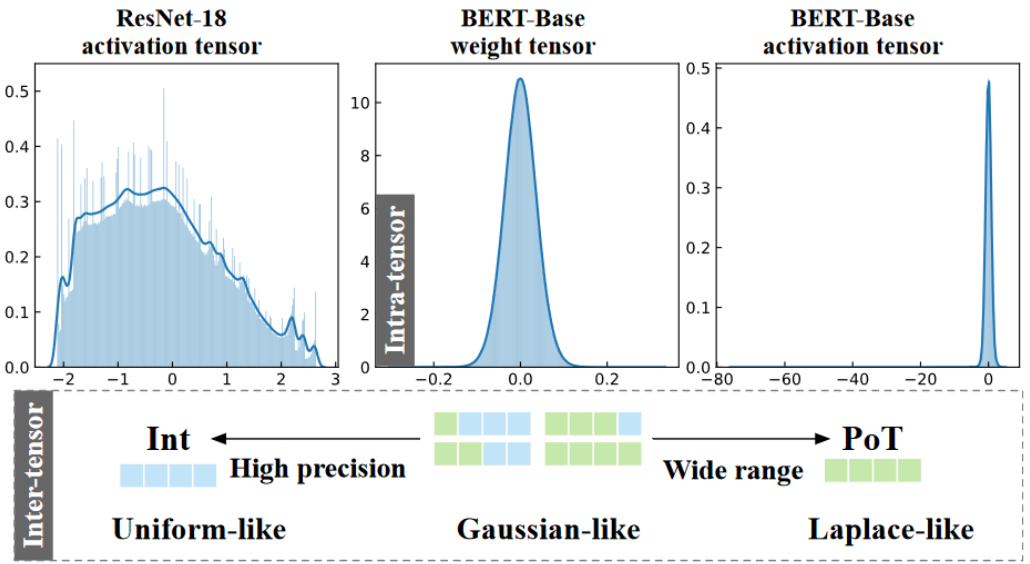


Figure 1: Intra-tensor and inter-tensor adaptivity.

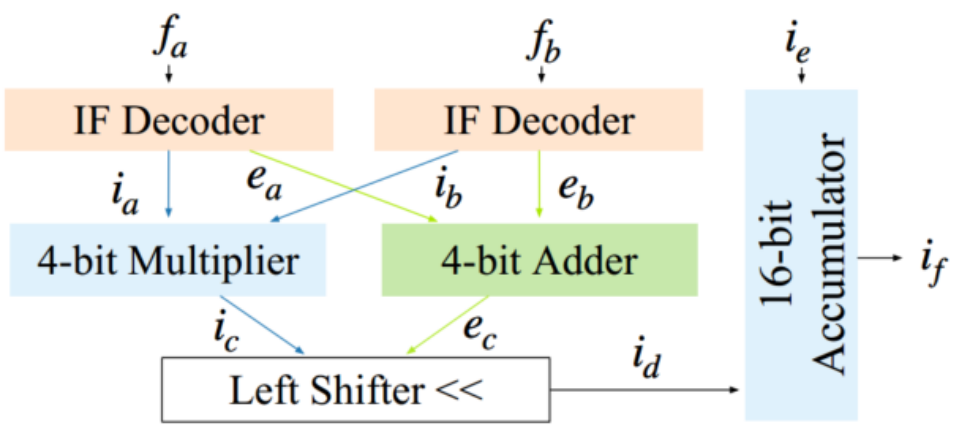
Bits	Exponent Value	Fraction Value	Value in Decimal
0000	-	0	0
0001	$1-1=0$	1	$2^0 \times 1 = 1$
001x	$2-1=1$	1, 1.5	2, 3
01xx	$3-1=2$	1, 1.25, 1.5, 1.75	4, 5, 6, 7
11xx	$4-1=3$	1, 1.25, 1.5, 1.75	8, 10, 12, 14
101x	$5-1=4$	1, 1.5	16, 24
1001	$6-1=5$	1	32
1000	$7-1=6$	1	$2^6 \times 1 = 64$

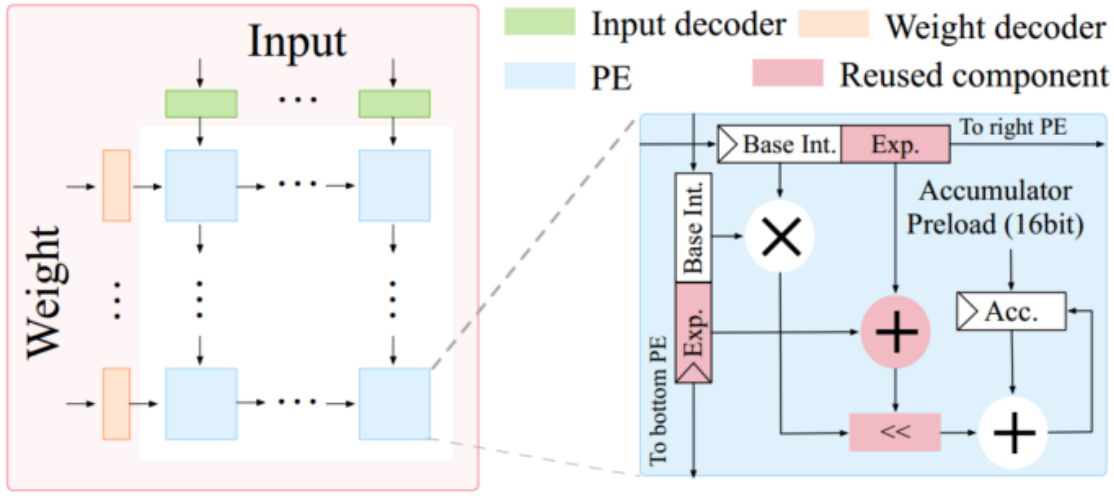
Table II: The value table of 4-bit unsigned `flint` with the exponent bias of -1. The blue numbers are the first-one-encoded exponent and "x" is mantissa with value of 0 or 1.

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

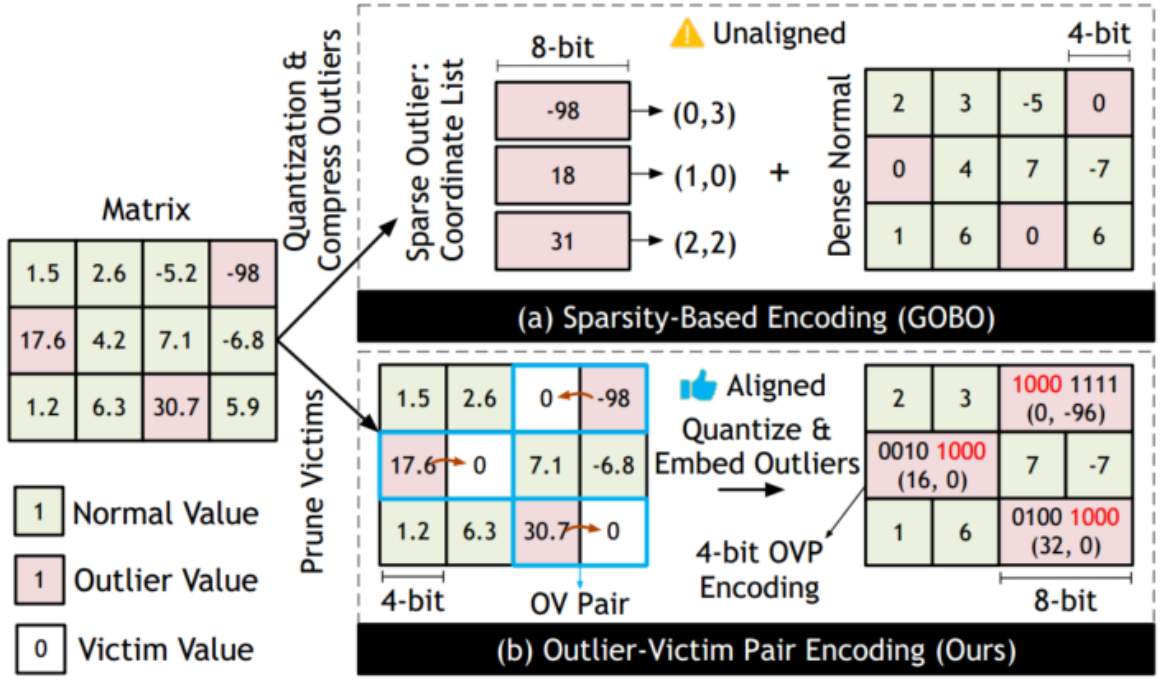
- 为了解决tensor之间数据分布不同的问题，通过校准数据集选择量化后均方误差最小的数据类型
  - 直觉上int类型适合均匀分布，Float/PoT类型适合拉普拉斯分布，Flint类型适合高斯分布
- 设计了可以支持以上四种格式的处理单元（PE），主要包括解码器，乘法器，加法器和累加器
- 将设计的PE集成到了脉动阵列形成新的架构



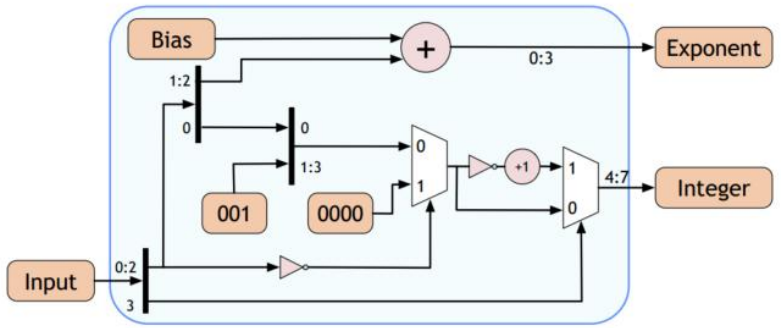


神经网络量化与AI处理器——OliVe

- 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



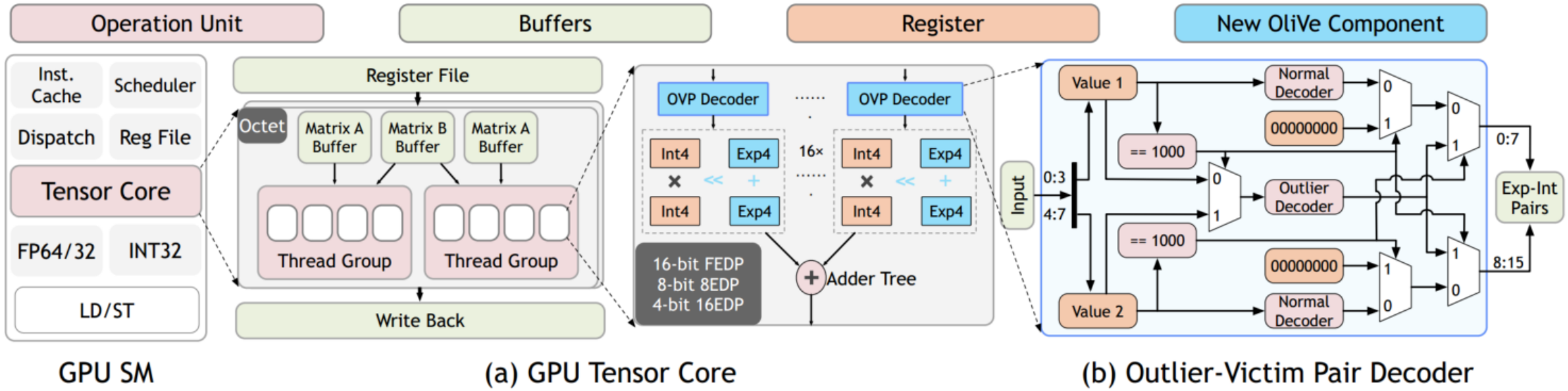
Binary	Exponent	Integer	Real Value
000	0	0	0
001	0	3	$3 \times 2^0 = 3$
01x	1	2, 3	$2 \times 2^1 = 4, 3 \times 2^1 = 6$
10x	2	2, 3	$2 \times 2^2 = 8, 3 \times 2^2 = 12$
11x	3	2, 3	$2 \times 2^3 = 16, 3 \times 2^3 = 24$



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

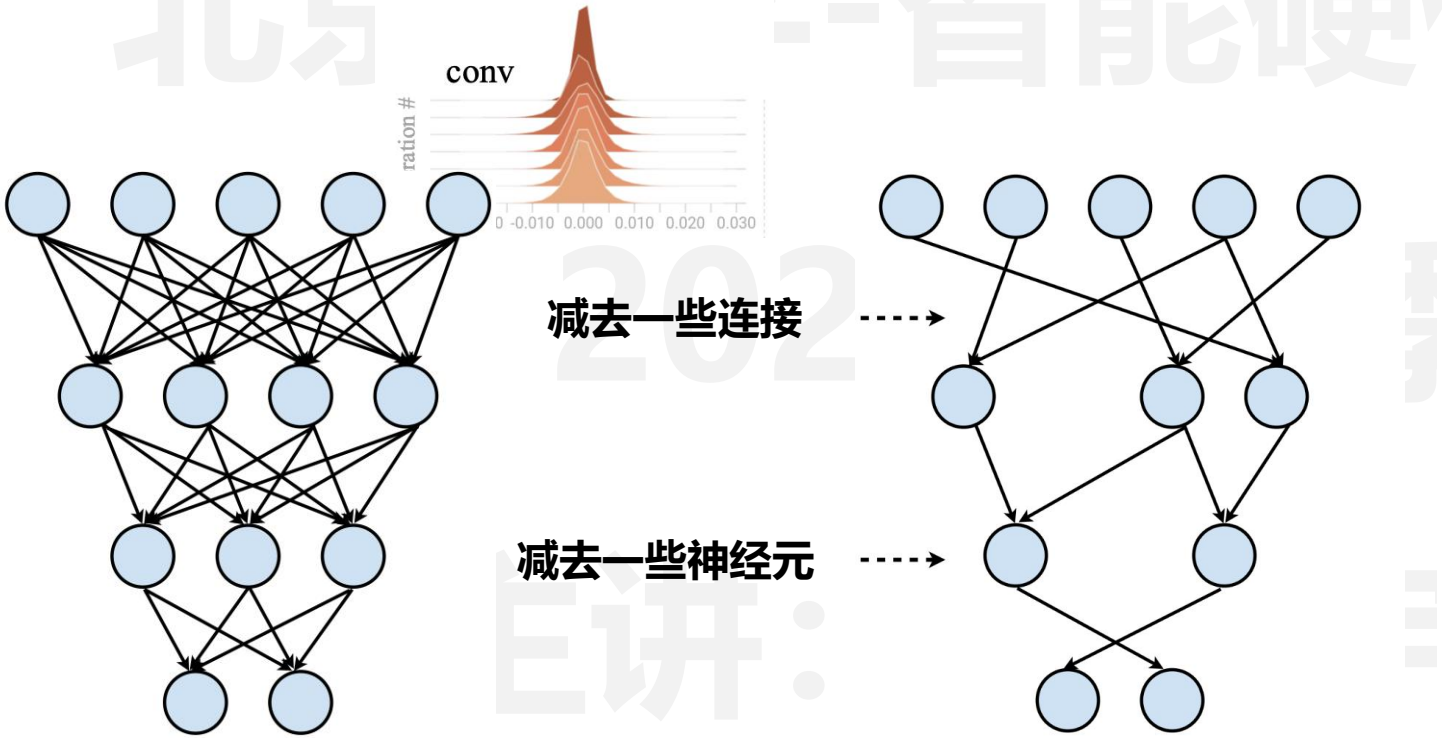
神经网络量化与AI处理器——OliVe

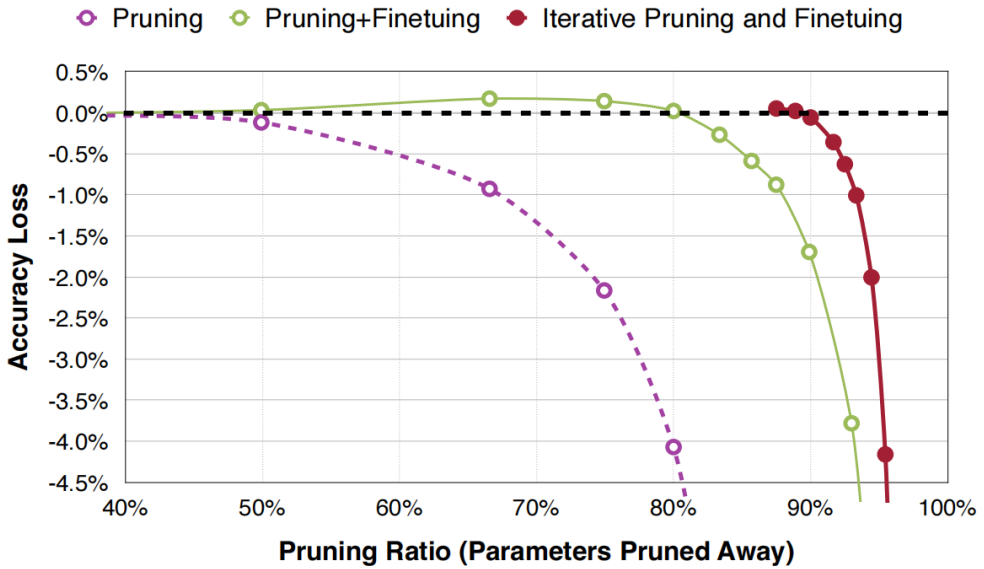
- 解码完成后，文章设计了乘累加单元电路（MAC Unit），并将他们集成到了GPU tensor core以及脉动阵列中



基于模型稀疏化的软硬件协同设计

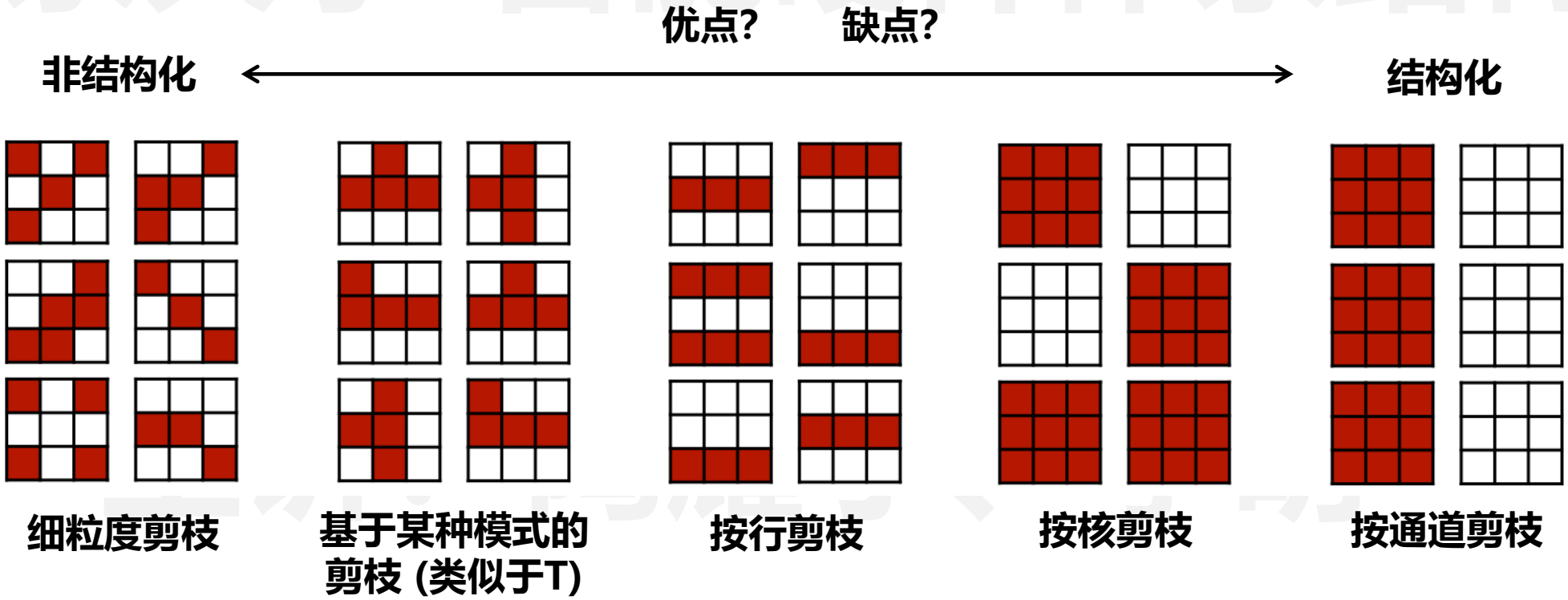
- 除了对于量化噪声的容错性，神经网络往往呈现显著的**稀疏性**，即对特定神经元或者模型权重进行剪枝，不会影响其推理准确率





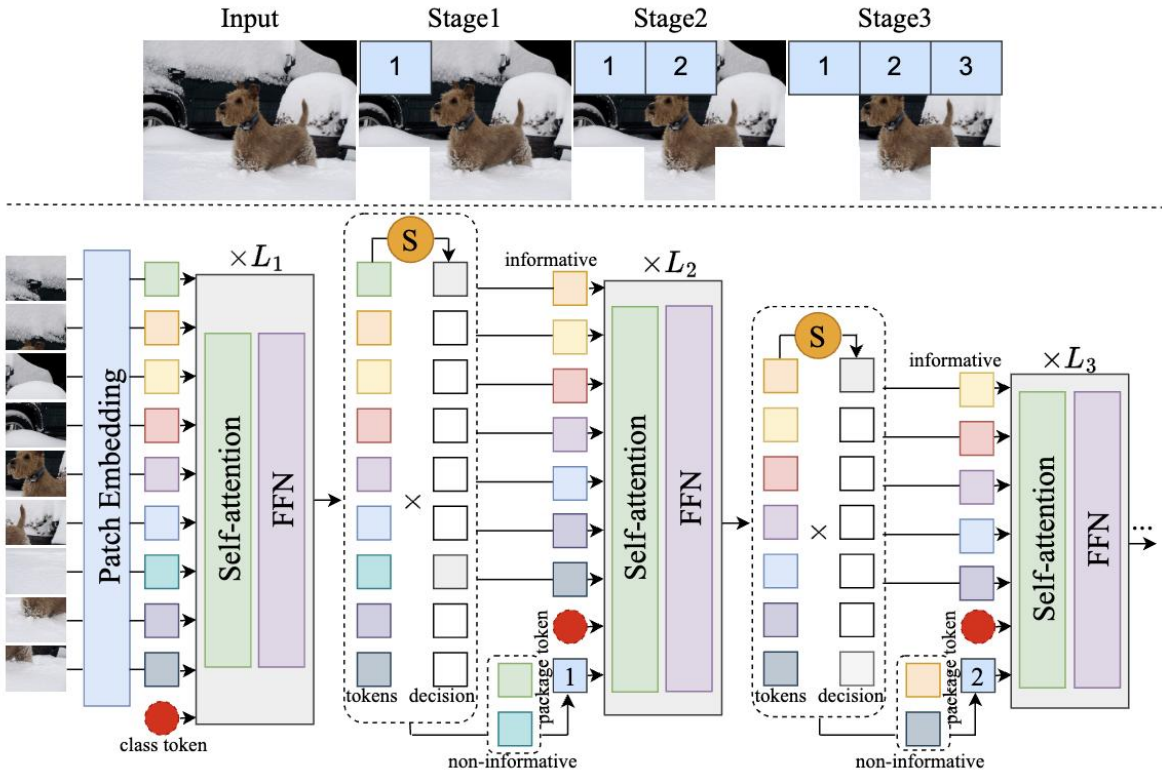
模型稀疏化的Taxonomy

- 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



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

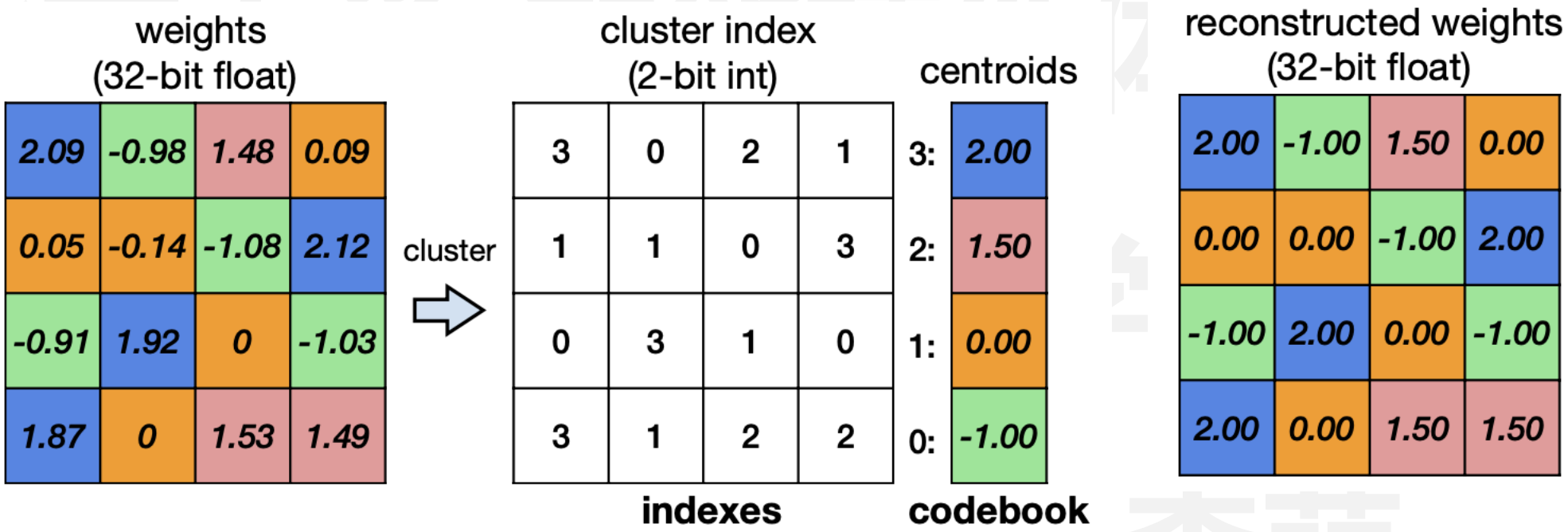
神经网络稀疏化与AI处理器

- 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

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



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

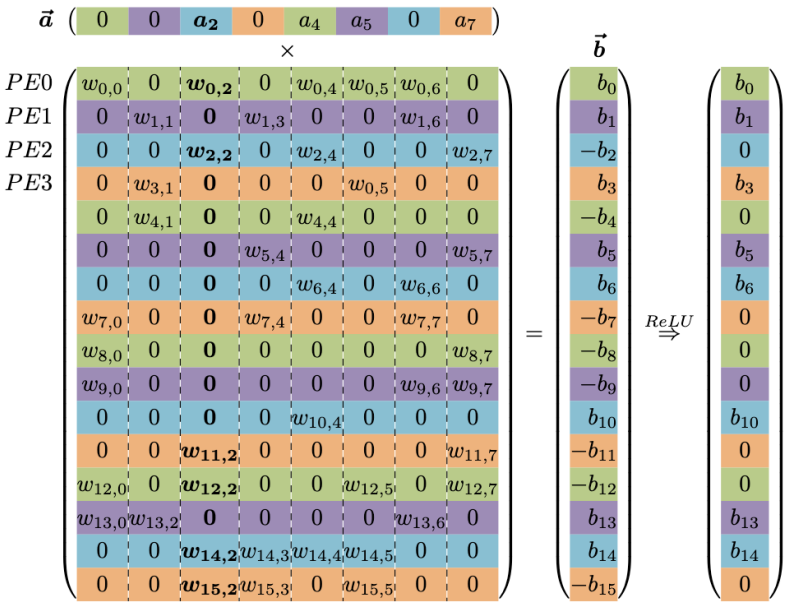


Figure 2. Matrix W and vectors a and b are interleaved over 4 PEs. Elements of the same color are stored in the same PE.

- 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

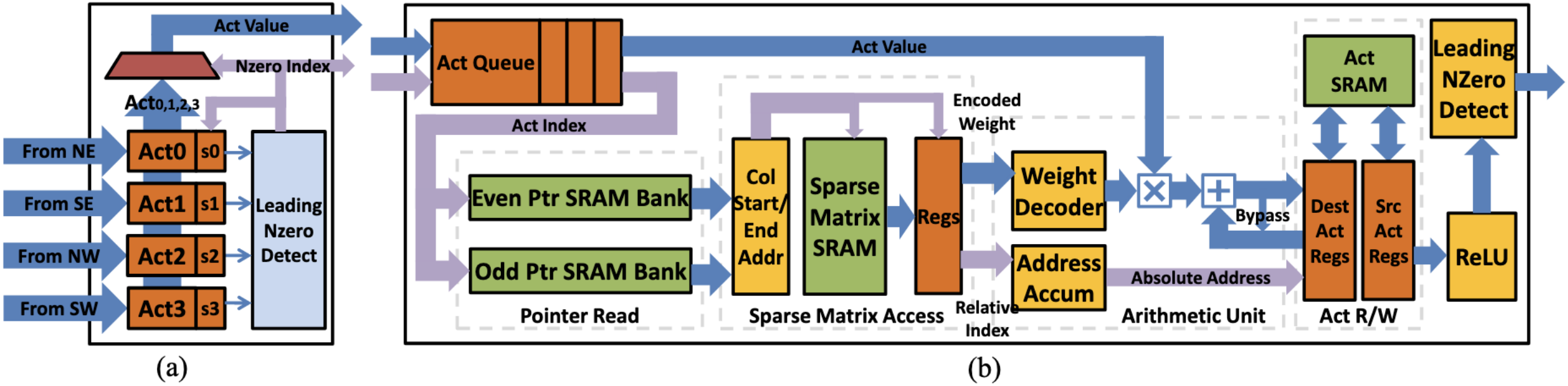
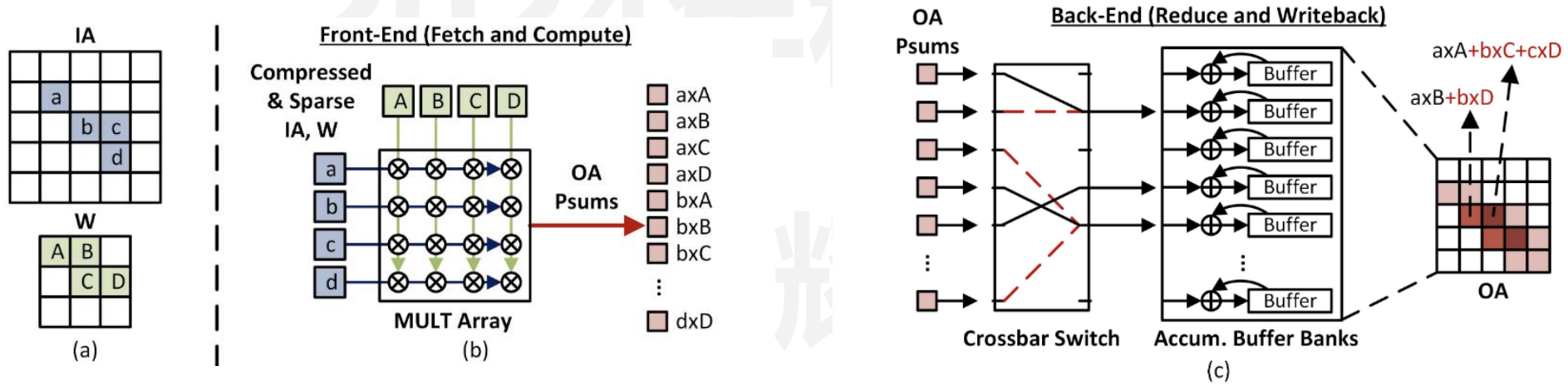


Figure 4. (a) The architecture of Leading Non-zero Detection Node. (b) The architecture of Processing Element.

神经网络量化与AI处理器 —— SNAP

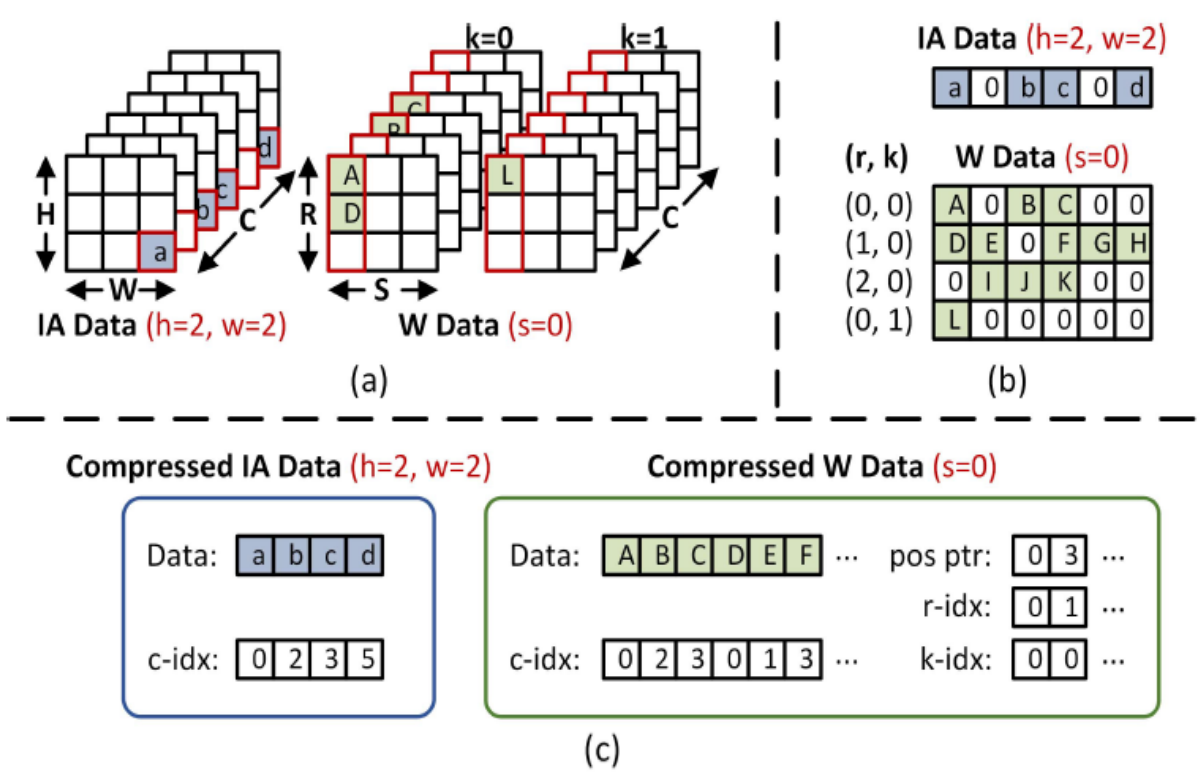
- 文章发表于JSSC 2021，通过channel-first数据流对稀疏网络模型进行了硬件加速
- 核心观察：数据稀疏使得网络推理更加高效，但现有稀疏计算的数据流仍面临以下挑战
  - Front-end：读取的W-IA pairs数量不足导致计算单元利用率低
  - Back-end：计算结果写回的地址冲突
  - 灵活性：对于不同kernel大小和层类型的支持性不足

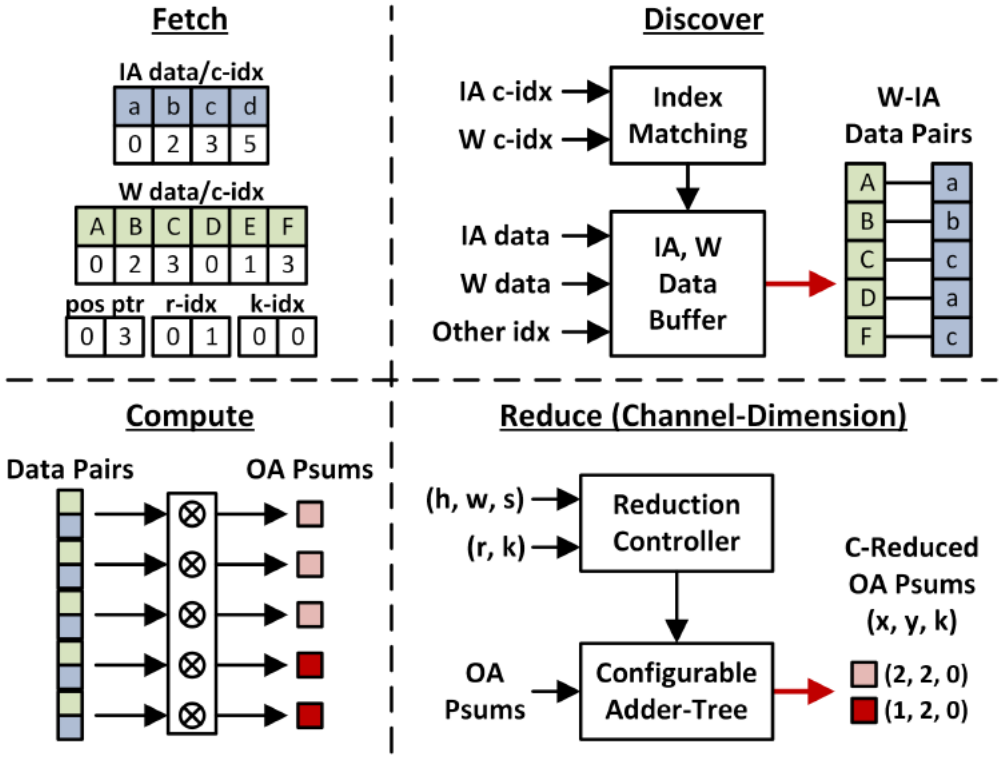


Jie-Fang Zhang et al., *SNAP: An Efficient Sparse Neural Acceleration Processor for Unstructured Sparse Deep Neural Network Inference*, JSSC 21

神经网络量化与AI处理器 —— SNAP

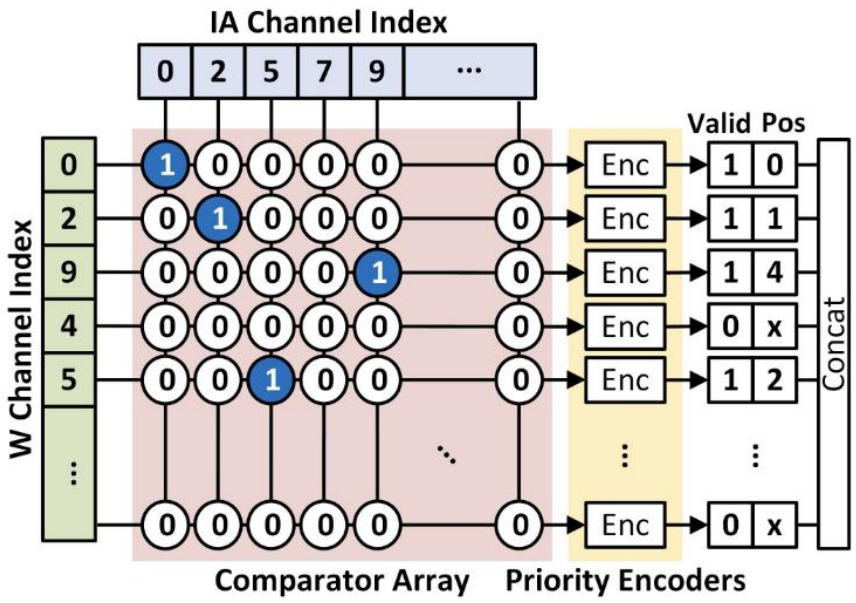
- Channel-first数据流：同一个channel的可以任意配对，产生的乘法结果都是有用的
- 通过可配置加法树实现先**归约加和**，然后更新Psum，从而减少Psum更新时的地址冲突

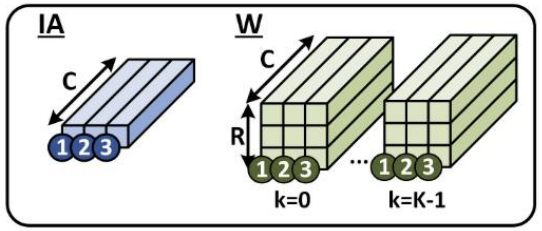


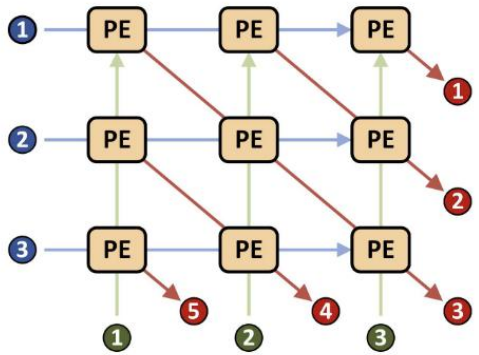


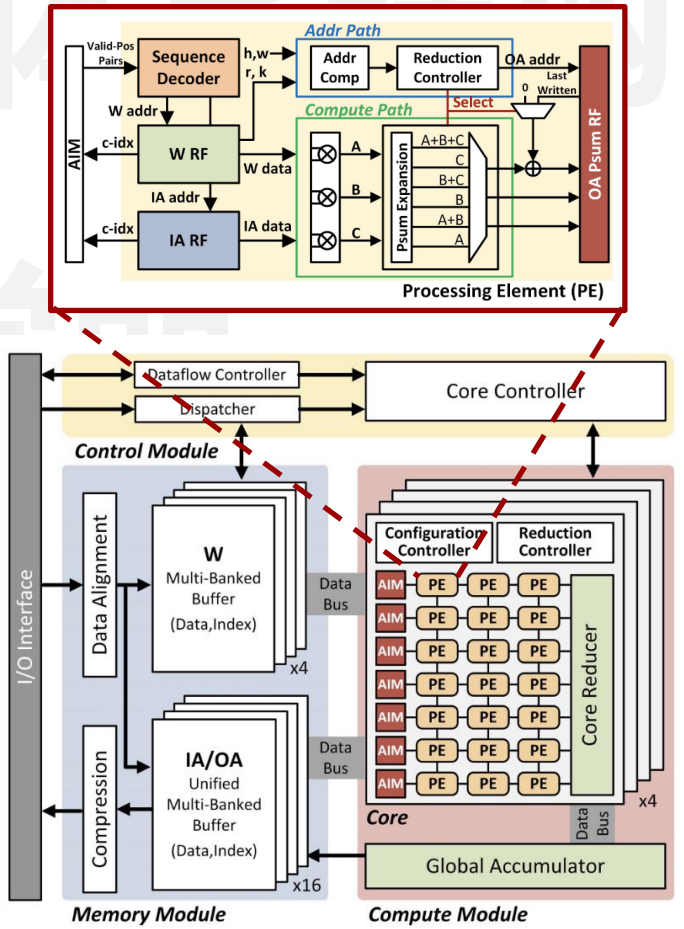
神经网络量化与AI处理器 —— SNAP

- 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



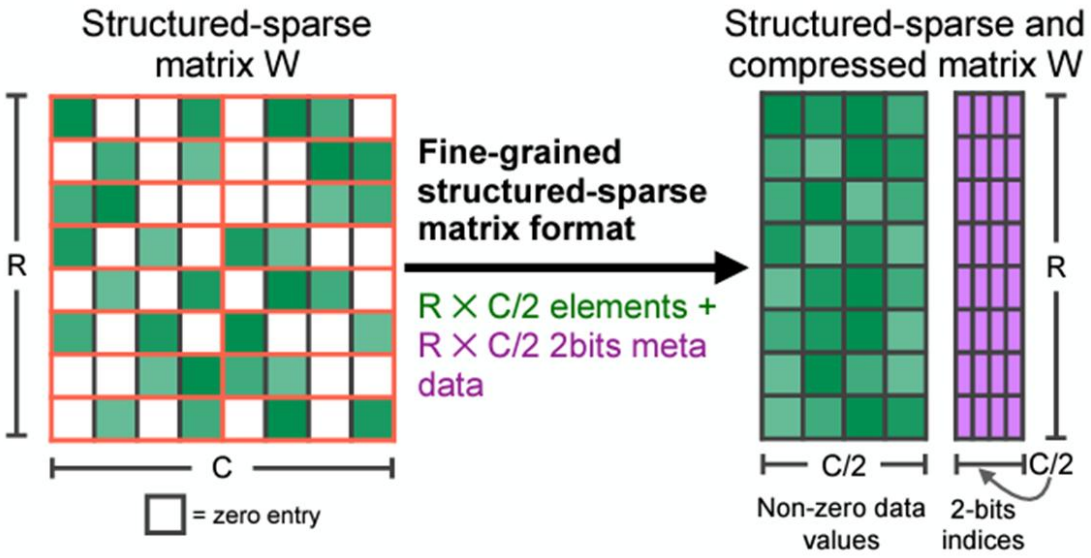


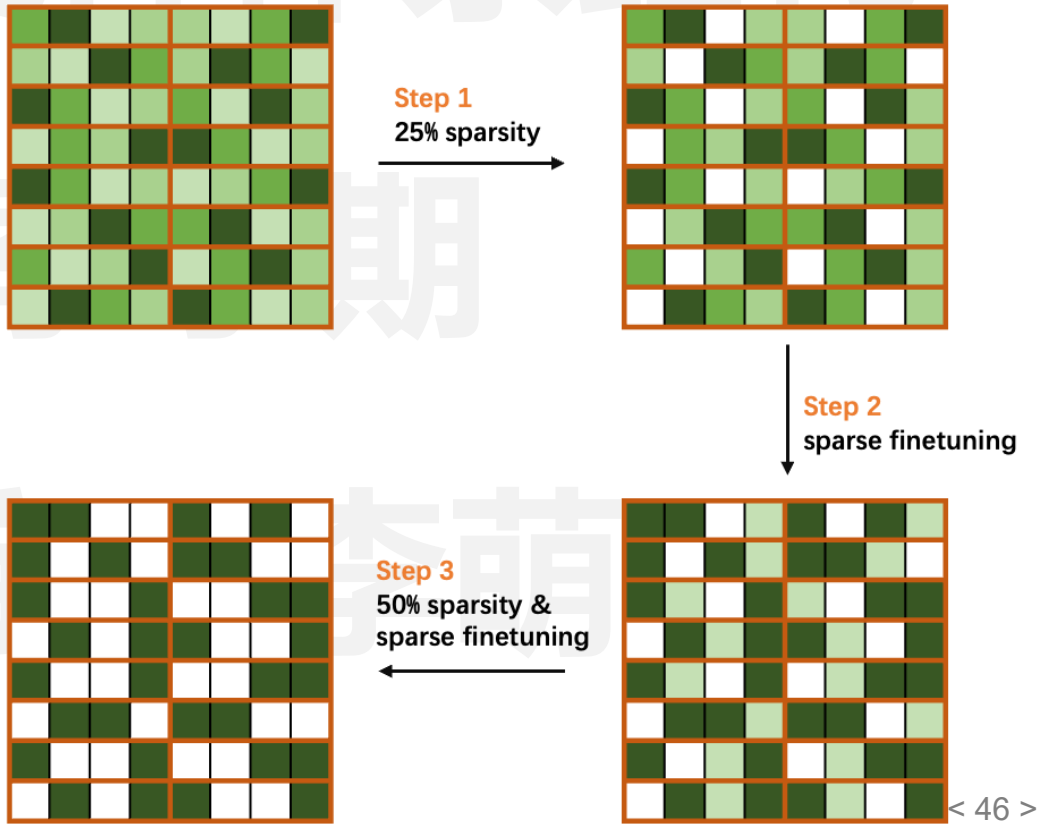




神经网络量化与AI处理器——NV Sparse TensorCore

- 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

目 录

CONTENTS

研究背景

主流大模型

主干网络

Transformer

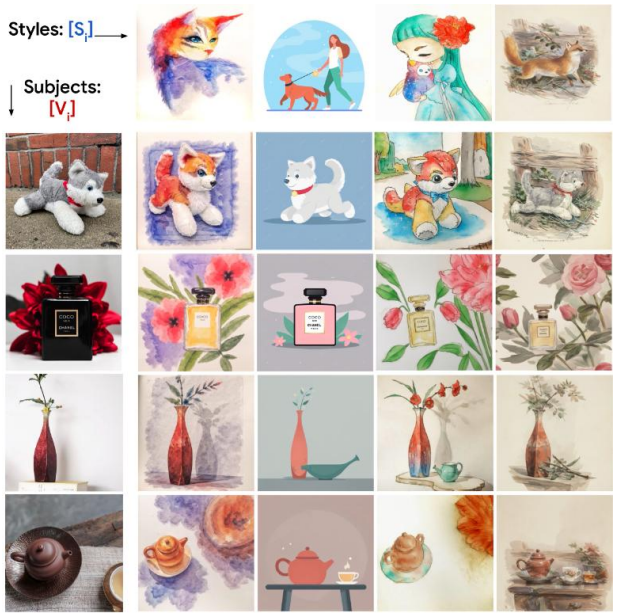
聊天机器人

实时聊天 + 性格设定

https://ai.baidu.com/unit/v2/static/socialbot

图像合成

更改图片风格



Shah, Viraj, et al. arXiv:2311.13600 (2023).

文本视频关联

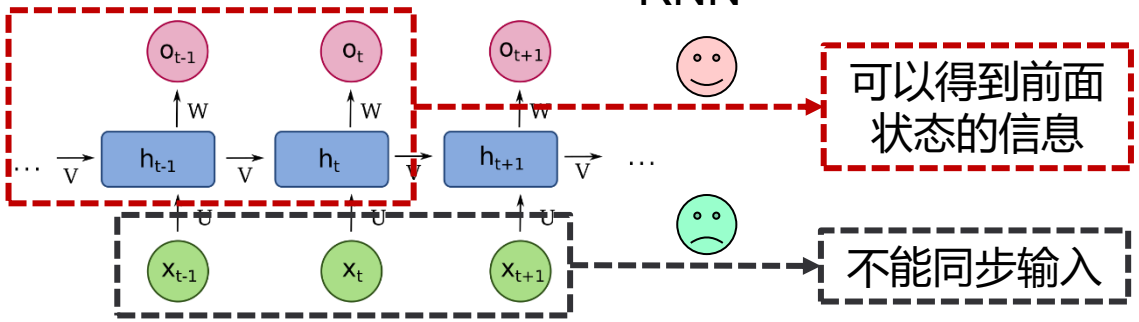
根据文字更改视频内容

https://pika.art

Transformer模型优势

指标	RNN	CNN	Transformer
数据特征	有序序列	多维网格	一组向量
长序列处理	✓	X	✓
可并行性	X	✓	✓

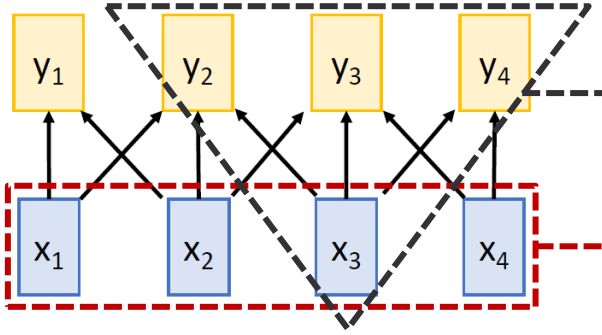
RNN



可以得到前面状态的信息

不能同步输入

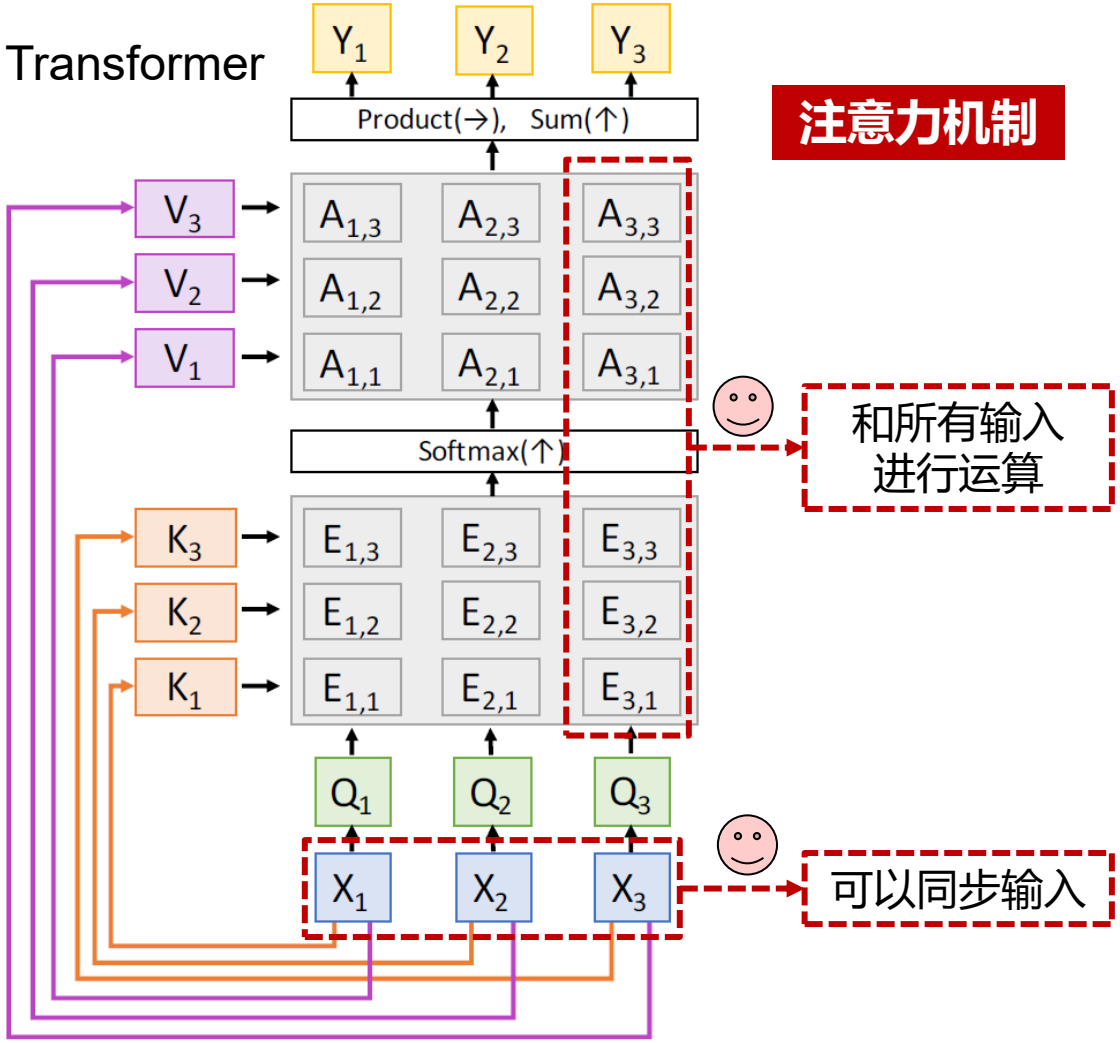
CNN



不能得到全局信息

可以同步输入

Transformer

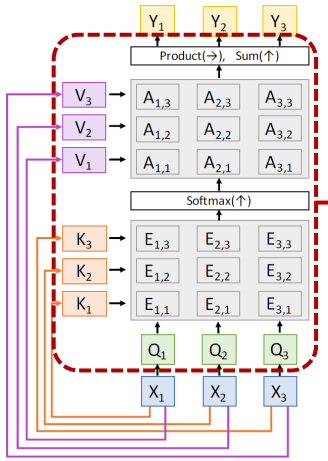


注意力机制

和所有输入进行运算

可以同步输入

Transformer硬件部署问题



时间复杂度

$O(n^2 \times d)$

n: 序列长度
d: 维度

序列长度(32K)增大 → 注意力模块延时**增加**

任务种类增多 → 字典维度增大 → 注意力模块延时**增加**

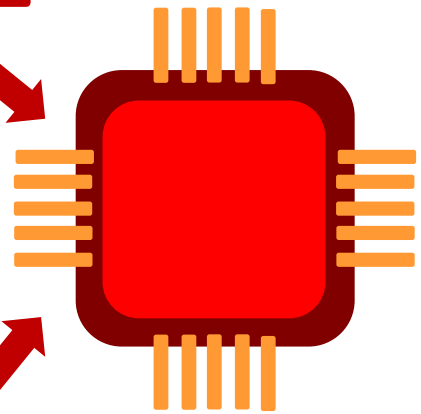
字典维度增大 → 输入维度增大导致能耗**增加**

字典维度增大 → 训练延时随模型增大**增加**

字典维度增大 → 能耗随模型增大**增加**

高延时

高能耗



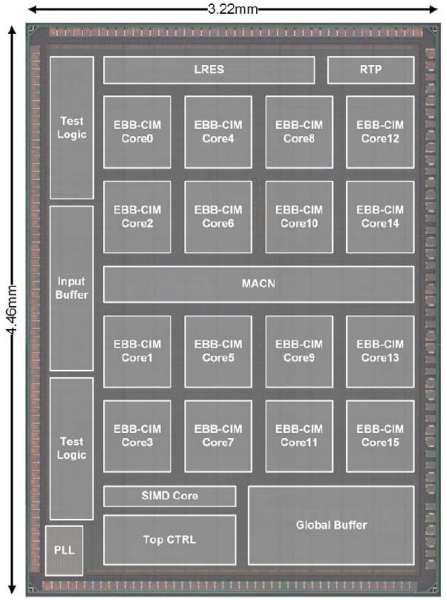
ASIC/专用加速器

模型名称		训练时间(GPU-hours)	总能耗(MWh)	碳排放量(tCO_2eq)
OPT-175B		809472	365	137
BLOOM-175B		1082880	475	183
LLaMA	7B	82432	36	14
	13B	135168	59	23
	33B	530432	233	90
	65B	1022362	449	173
LLaMA-2	7B	184320	83	31
	13B	368640	162	62
	34B	1038336	396	154
	70B	1720320	748	291

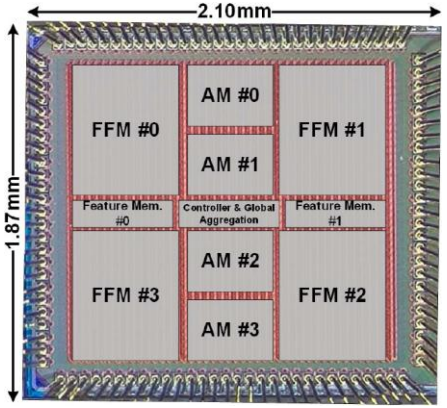
From LLaMA/LLaMA 2, 数据收集于A100-80GB

Transformer专用加速器

工作	单位	存储介质	存算模式	加速模型	流片/仿真
ReTransformer	杜克大学	ReRAM	存内计算	Transformer	仿真
ISSCC'2023	复旦大学	SRAM	近存计算	Transformer	流片
ReBERT	KAIST	ReRAM	存内计算	BERT	仿真
TranCIM	清华大学	SRAM	近存计算	BERT	流片
X-Former	普渡大学	SRAM/RRAM	近存/存内计算	BERT	仿真
P3ViT	澳门大学	SRAM	存内计算	ViT	流片
H3DAtten	佐治亚理工	SRAM/RRAM	近存/存内计算	Swin Transformer	仿真
MulTCIM	清华大学	SRAM	近存计算	MML Transformer	流片
TransPIM	加利福尼亚大学	DRAM	近存计算	BERT+GPT	仿真



Liu, Shiwei, et al. *ISSCC* 2023



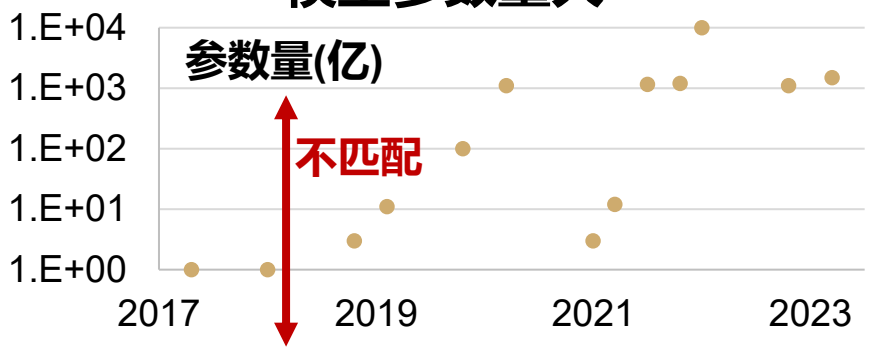
Tu, Fengbin, et al. *ISSCC* 2023

Transformer专用加速器挑战

高延时&高能耗 → ASIC/专用加速器

挑战1

模型参数量大



存算一体芯片最大容量

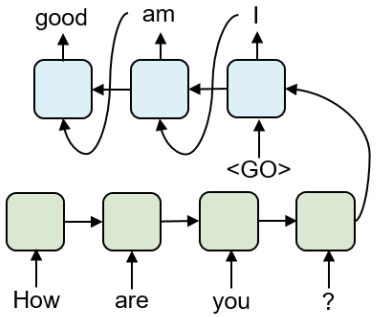
存储介质	SRAM	DRAM	RRAM	PCM	Flash
容量	1MB	160GB	4MB	21.25MB	80MB

模型级加速
减少计算量

挑战2

数据调度和传统深度神经网络不同

1、串行输入



2、注意力结构

模块级加速
高效数据调度

挑战3

缺乏对专有算子的支持

主要线性算子	非线性算子
矩阵向量乘法(MVM)	GELU
矩阵矩阵乘法(MatMul)	LayerNorm
残差相加	Softmax

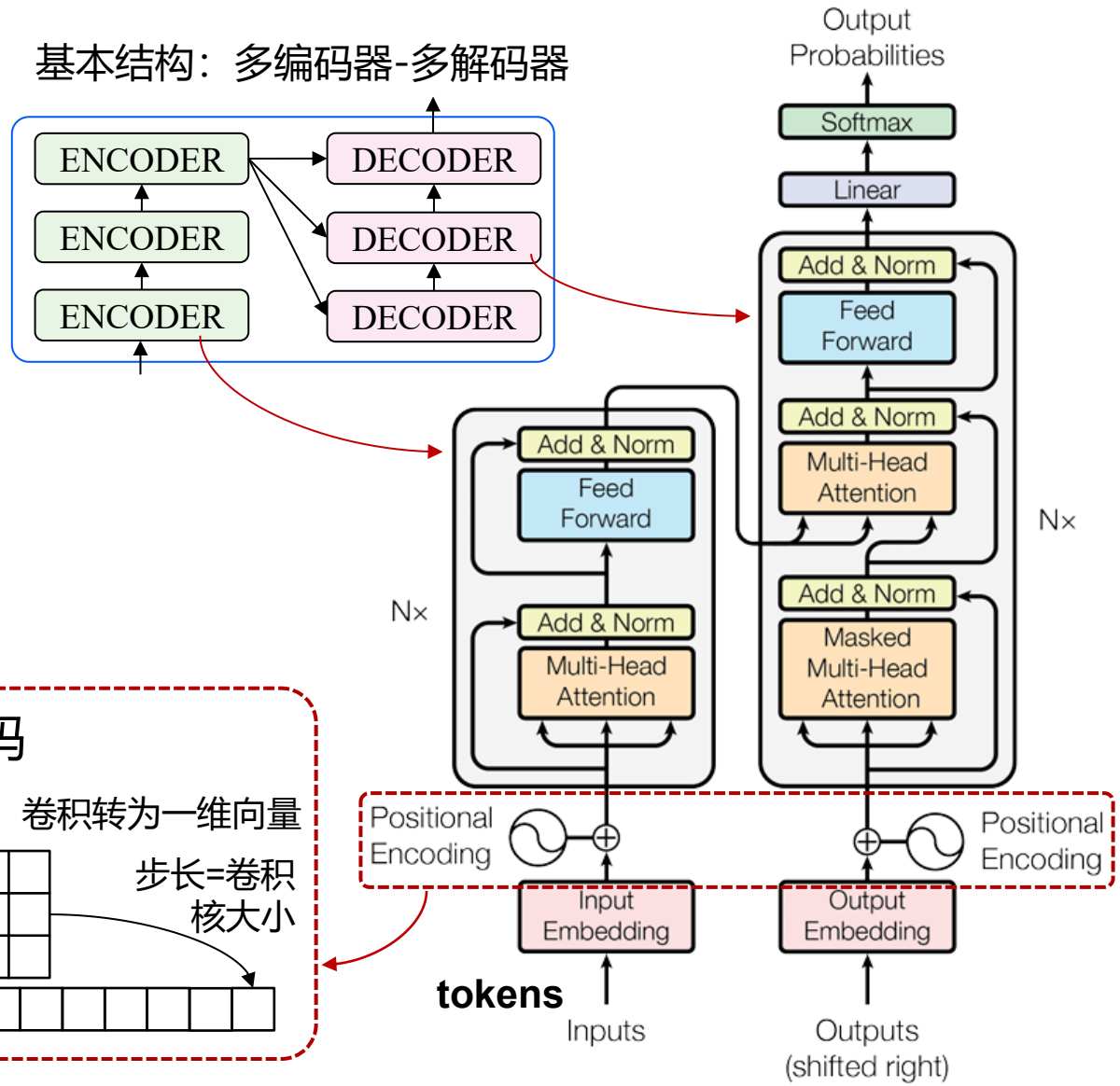
输入和权重不固定

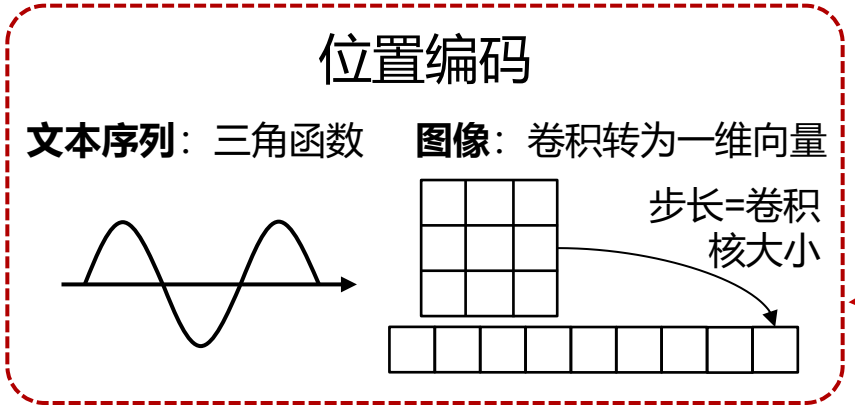
动态归一化

算子级加速
加速专有算子

Transformer网络结构

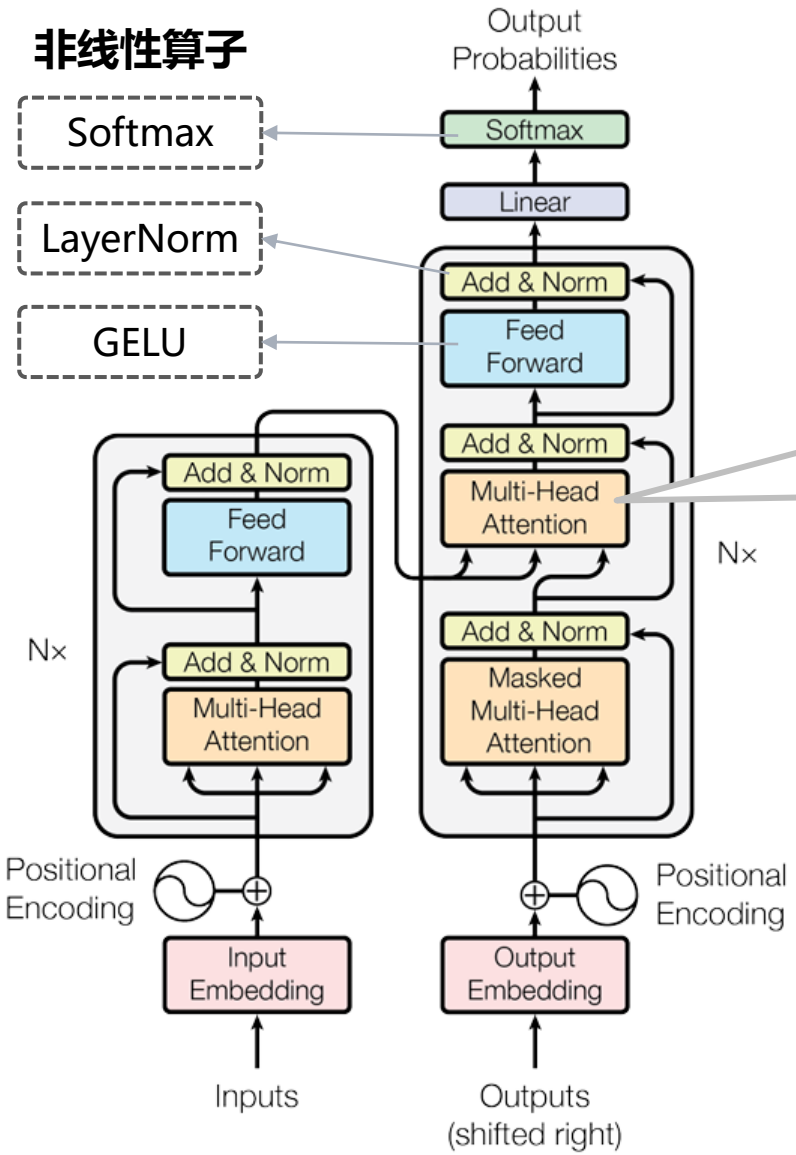
- **基本结构**：
  - 多个编码器层-多个解码器层
- **输入序列**：
  - 由一组token组成
  - 每个token为一个向量表示一个词根
  - 序列开头[CLS]——句子向量表示
  - 输入模型前经过embedding层
- **embedding**：
  - 输入编码：字典信息
  - 位置编码：位置信息

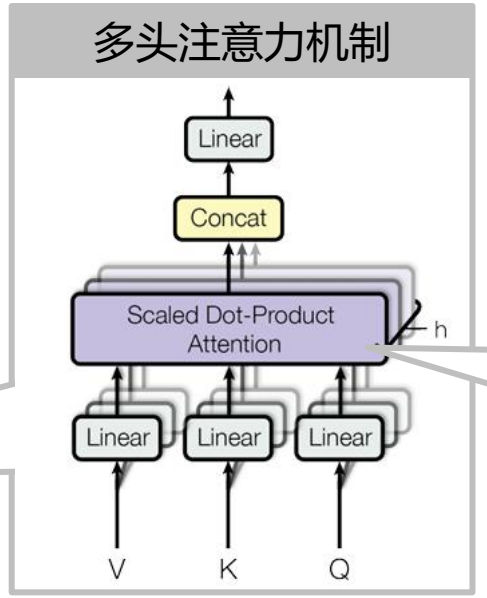


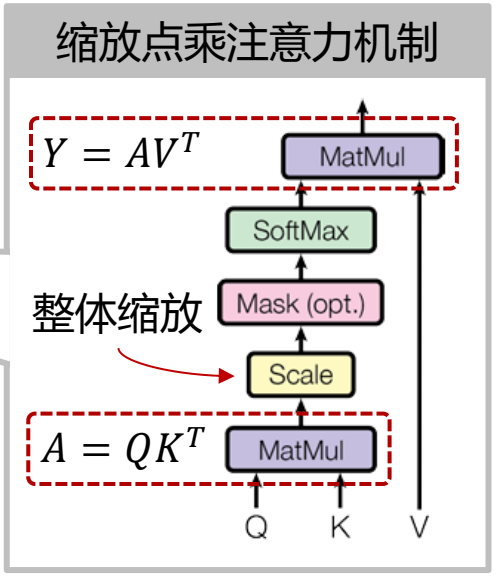


Vaswani, Ashish, et al. *NIPS* 2017

Transformer网络结构







层级	模块级	主要线性算子	非线性算子
embedding	注意力模块	矩阵向量乘法(MVM)	GELU
编码器	前馈神经网络	矩阵矩阵乘法(MatMul)	LayerNorm
解码器	残差模块	残差相加	Softmax

Transformer种类

- **根据网络结构可分为**：
  - Encoder-only；Decoder-only；Encoder-Decoder

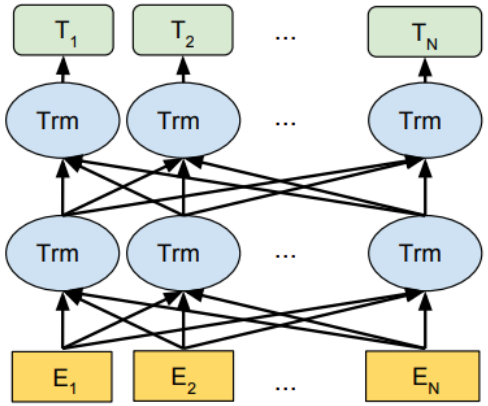
低延时要求

输入依赖于上一次的输出

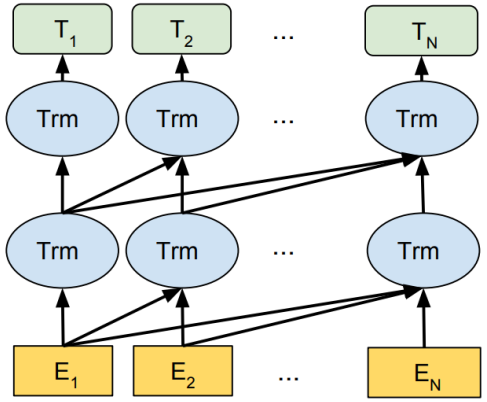
流水设计

分阶段并行设计

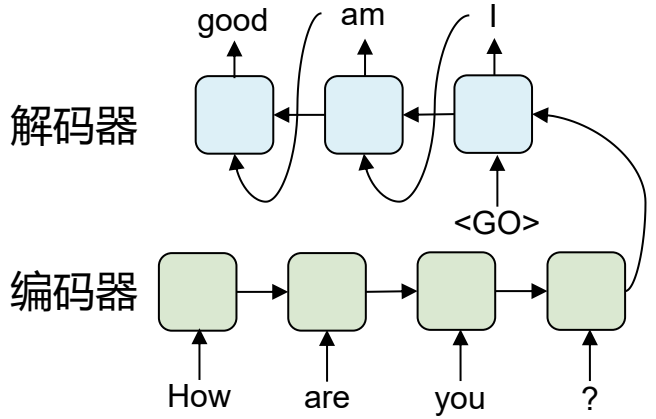
Encoder-only



Decoder-only



Encoder-Decoder



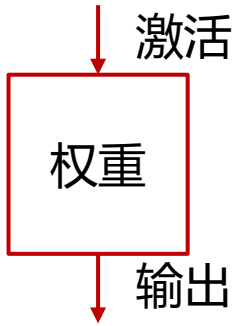
常见模型	BERT，RoBERTa	GPT，LLAMA	T5，BART
适合任务	分类任务	问答任务	翻译任务
输入输出特征	所有token并行输入	输入依赖于上一次的输出	输出强依赖于输入
适配加速器特征	**高吞吐**	**低延时**	**低延时**
适配并行设计	**流水并行设计**	**分阶段并行设计**	**分阶段并行设计**

目录

模型级加速：模型压缩

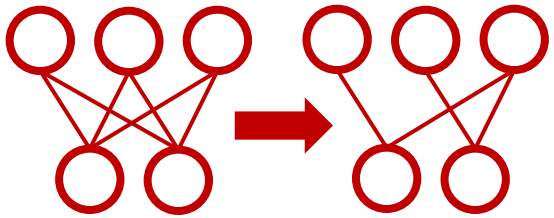
- **模型压缩**：硬件部署前**算法**优化

① **模型量化**



- 对**激活/权重/输出**进行量化
- 降低模型**权重存储**需求
- 降低硬件**计算精度**要求

② **静态剪枝**

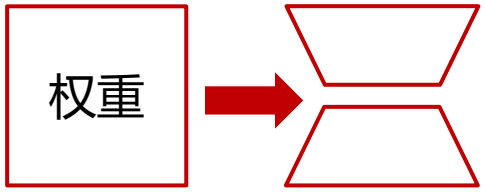


- 删去**不重要**的注意力维度/头
- 降低硬件**计算量**

③ **权重复用**

- **重复利用**网络中的某一部分
- 降低模型**权重存储**需求
- 降低**硬件设计**需求

④ **低秩分解**



- 将权重矩阵拆分为几个低秩矩阵的组合
- 降低模型**训练存储**需求

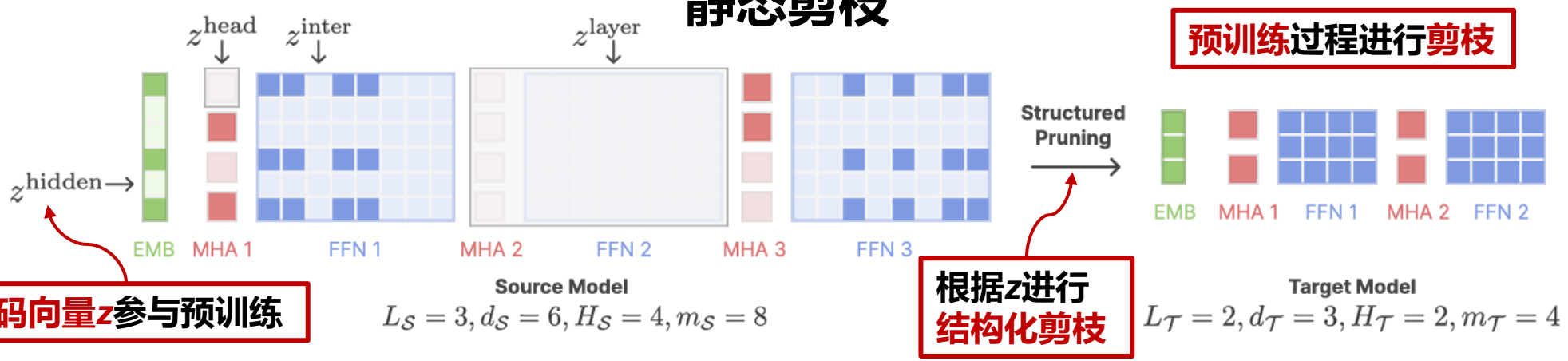
模型级加速：模型压缩

模型量化

Q-BERT	TernaryBERT	I-BERT	BinaryBERT	PTQ4ViT	FQ-ViT	BiT
INT8	三值	INT8	二值	INT8	INT4	二值

目前部分运算比如全连接层可实现三值/二值精度下降5%以内，INT8无损

静态剪枝



Xia, Mengzhou, et al. arXiv:2310.06694 (2023)

模型级加速：模型压缩

- **权重复用**

注意力模块权重相似

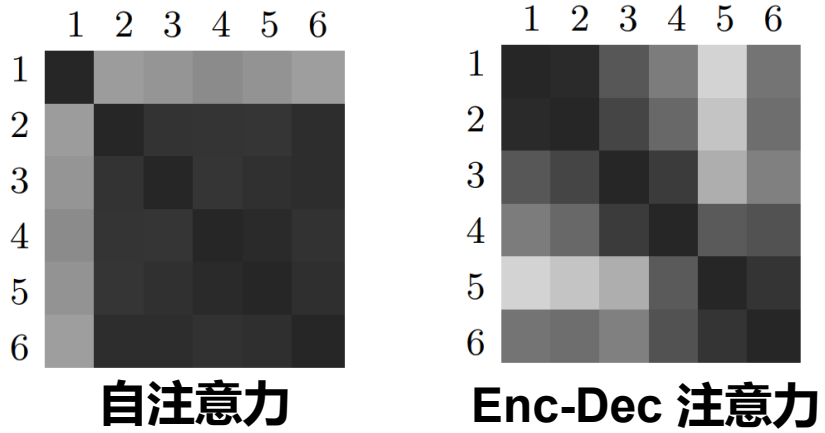
1、不同维度之间权重相似

多个维度学习同样的东西

2、不同层之间权重相似

不同层的权重的JS散度

颜色越深代表相似度越高

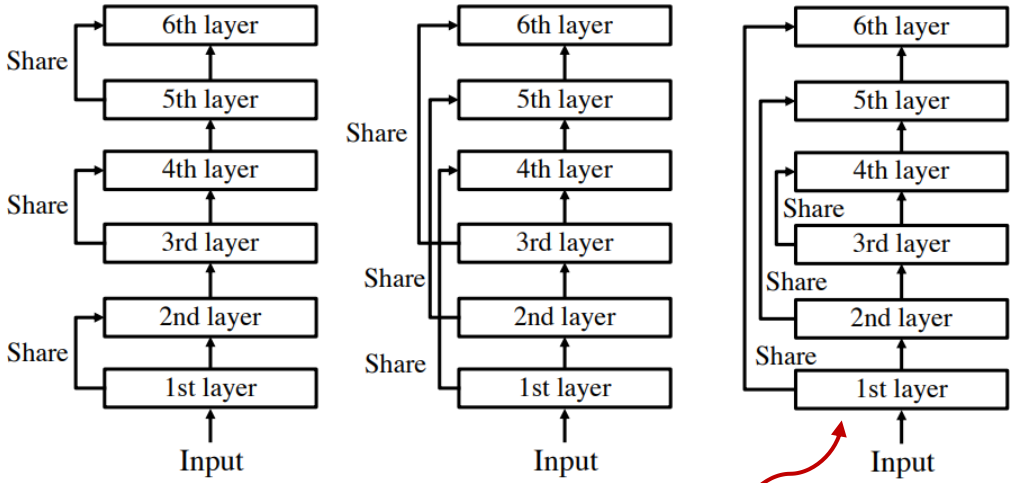


自注意力

Enc-Dec 注意力

层间/维度间权重复用

根据权重相似性进行复用



ALBERT
——全层复用

低层有更多参数冗余

Lin, Zhouhan, et al. *ICLR* 2017

Xiao, Tong, et al. *IJCAI* 2019

Takase, Sho, and Shun Kiyono. *SustaiNLP* 2023

模型级加速：模型压缩

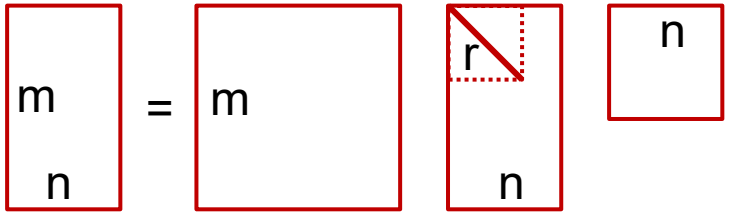
- **低秩分解**
  - 适用于**微调训练**过程
  - 训练中的权重更新为**低秩**矩阵，可以用两个更小矩阵来表示
  - 拆分更新权重降低**训练显存**

矩阵低秩分解

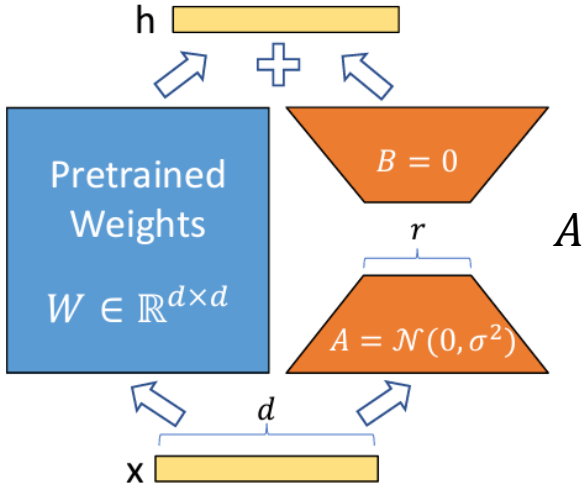
- 奇异值分解SVD

$$A = U\Sigma V^T$$

仅有r宽度的对角线上为特征值



微调训练应用



$$h = W_0x + \Delta Wx$$
$$= W_0x + BAx$$

$$W_0 \in R^{d\times k}, B \in R^{d\times r}$$
$$A \in R^{r\times k}, r \ll \min(d, k)$$

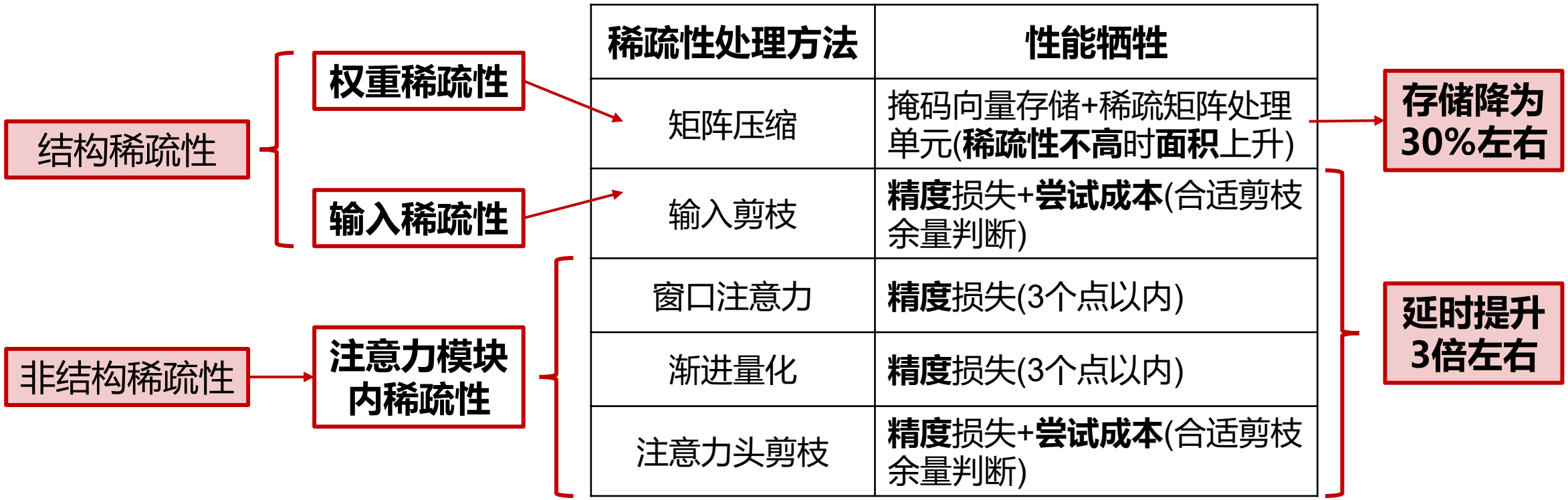
W_0为原权重
ΔW为更新权重

Hu, Edward J., et al. *ICLR* 2022

目录

模型级加速：稀疏性处理

- **稀疏性处理：**硬件部署后**电路**优化
  - 针对**不同稀疏性**优化

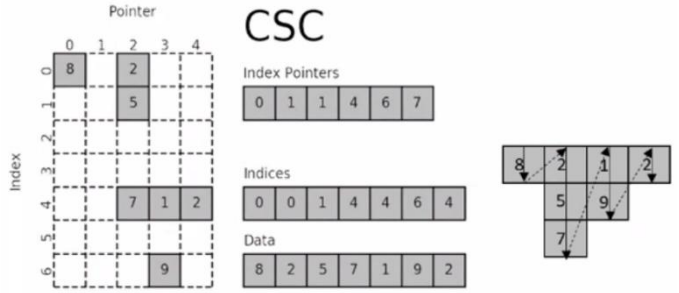


模型级加速：稀疏性处理

- **权重稀疏性**
  - 根据稀疏向量直接对输入进行裁剪

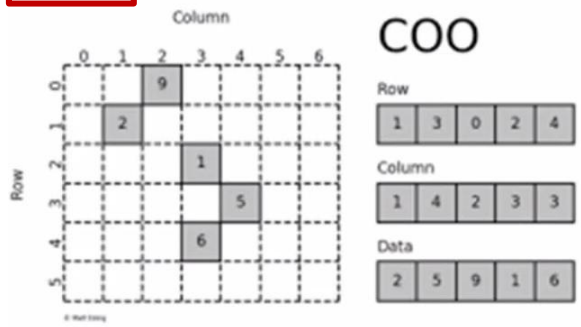
稀疏矩阵

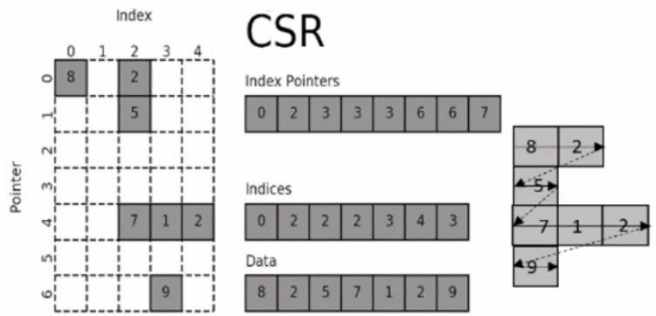
- **CSC**：按列压缩
- **CSR**：按行压缩
- **COO**：存储坐标



非零值的行的索引+行的起始位置指针

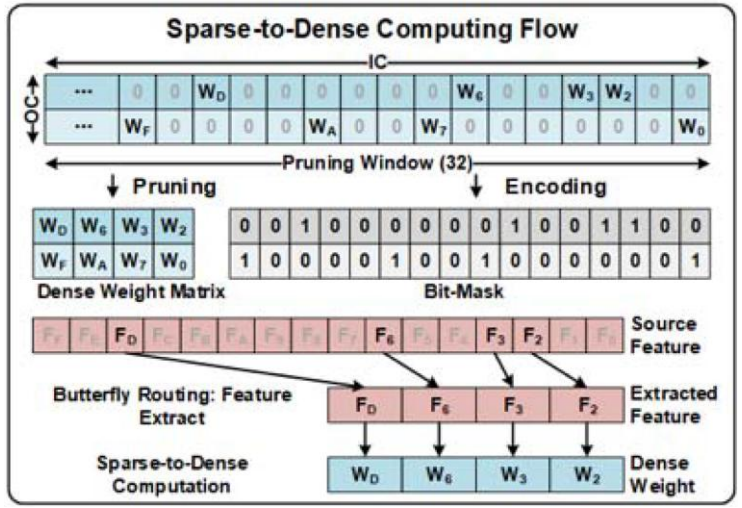
坐标





稀疏性矩阵应用

- 稀疏矩阵->压缩矩阵+掩码向量
- 根据掩码向量对**输入**进行**裁剪**



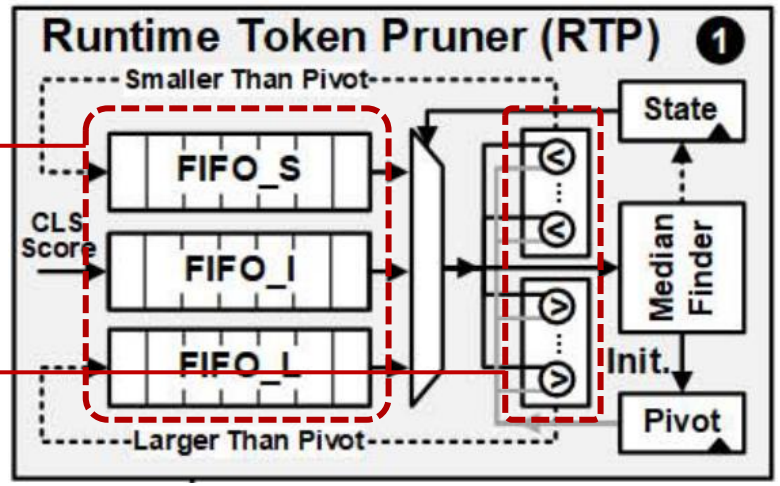
Liu, Shiwei, et al. *ISSCC* 2023

模型级加速：稀疏性处理

- **输入稀疏性**
  - 删去相对不重要的输入

三块FIFO
- FIFO_L: 大于阈值的token
- FIFO_I: 等于阈值的token
- FIFO_S: 小于阈值的token

一组比较器
- 比较对象固定为阈值
- 一组token进行并行比较



操作流程
- 循环实现
- 循环中止目标：
  - FIFO_S/FIFO_L内元素个数和设定值相当

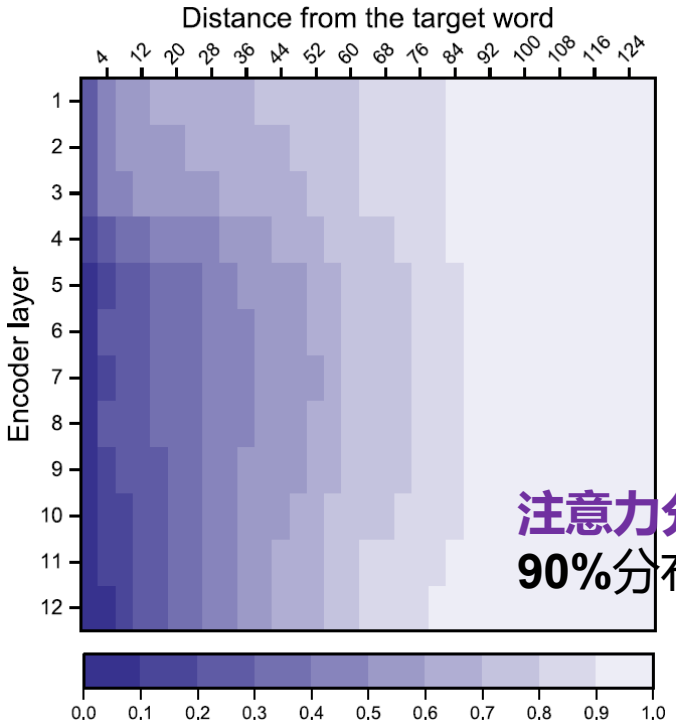
循环内容
- **初始化**：
  - 阈值为[CLS]
  - 所有token放入FIFO_I
  - 选定FIFO_I
- **重复**：
  - 阈值比较并放入FIFO
- **循环判断**：
  - 满足要求则输出
  - 不满则选定FIFO_S/L
  - 阈值为其中随机一个token

Tu, Fengbin, et al. *ISSCC* 2023

模型级加速：稀疏性处理

- **注意力模块内稀疏性**
  - **利用注意力机制更关注于关键词的特征**

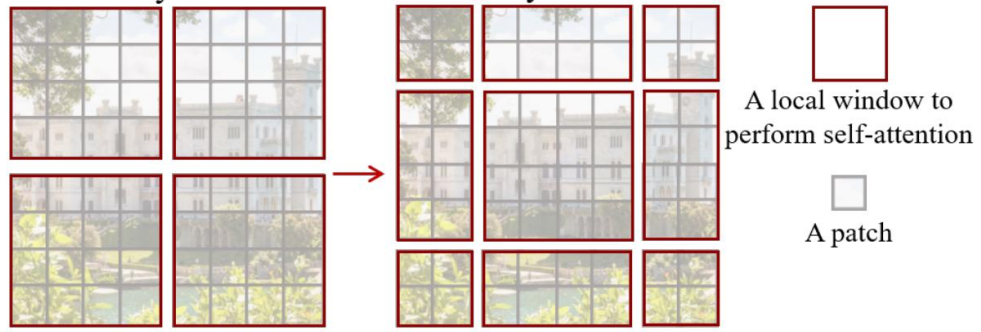
窗口注意力



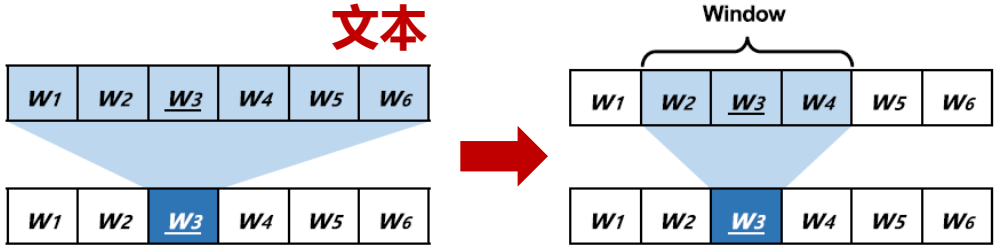
- 按注意力模块更关注目标词**附近**
- 目标词可以只和**附近token**进行注意力计算

注意力分数的累和
90%分布在**84**token以内

图像



文本



Liu, Ze, et al. *CVPR* 2021

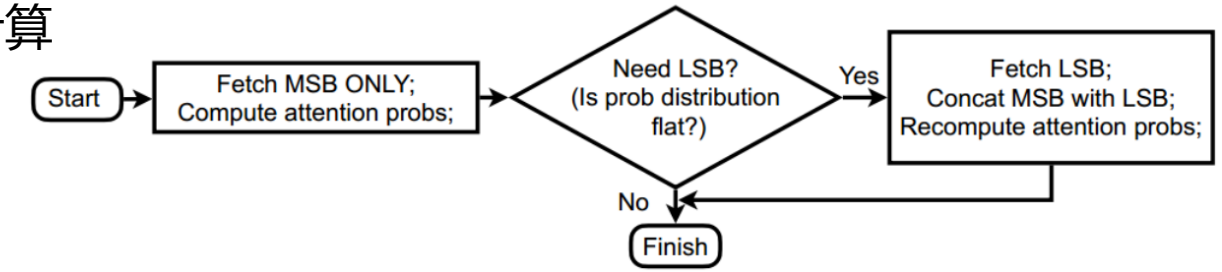
Kang, Myeonggu, Hyein Shin, and Lee-Sup Kim, *TCAD* 2021

模型级加速：稀疏性处理

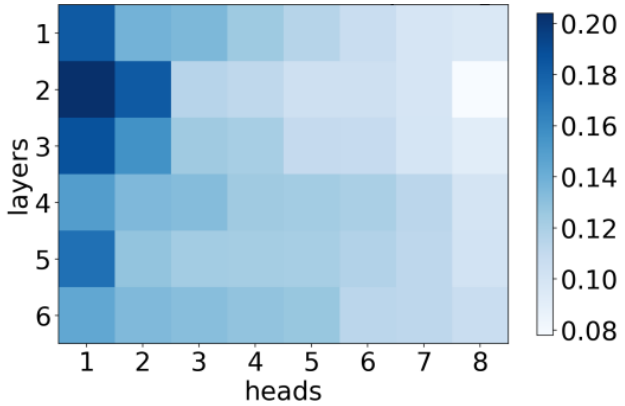
- **注意力模块内稀疏性**
  - **利用注意力机制更关注于关键词的特征**

渐进量化

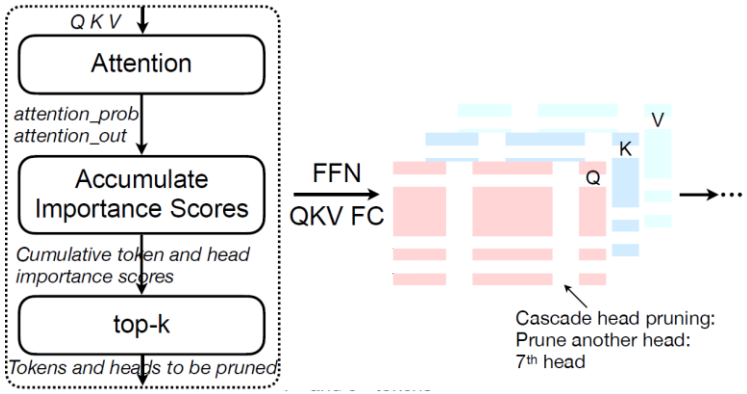
- 按**位**进行计算，而不是按**token**计算
- 每次计算**输入/权重**的一个比特
- 判断输出是否**平整**
- 不平整——已经实现注意力功能



注意力头剪枝



- 各层各头注意力分数**累加**
  - 各头累加结果**相差很大**
  - 大小关系在**各层统一**
- 提前进行注意力头剪枝



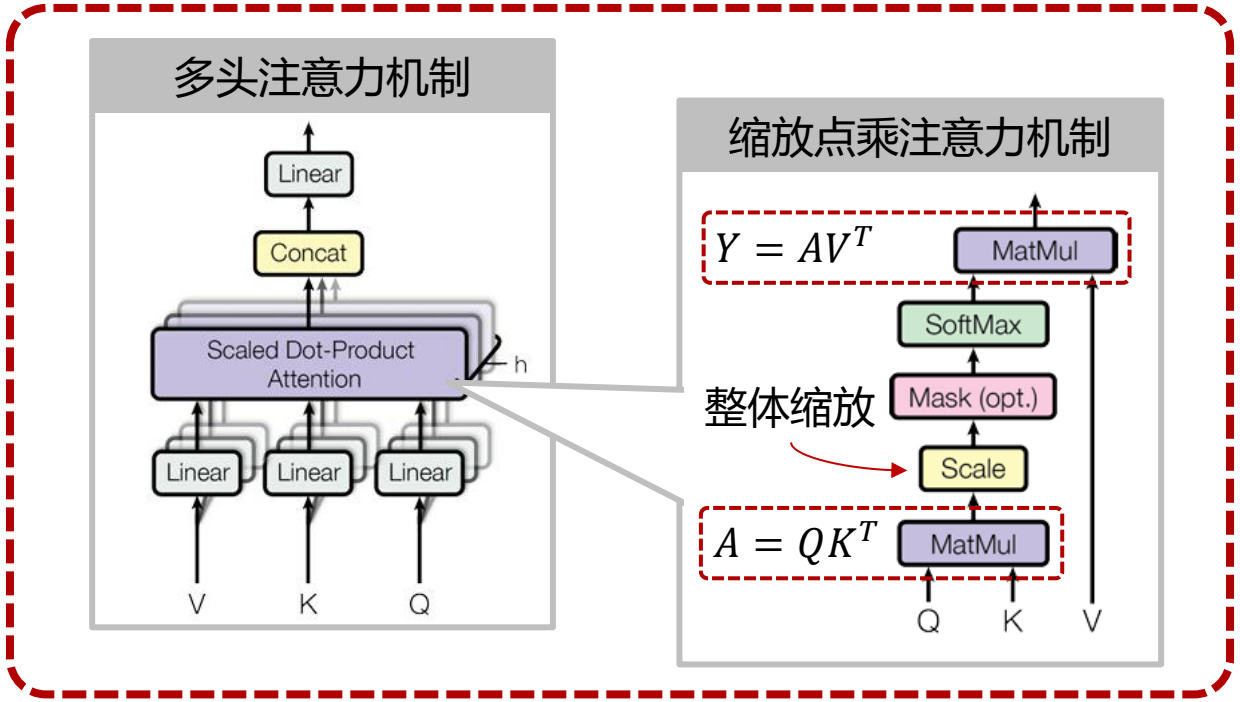
Voita, Elena, et al. *ACL* 2019

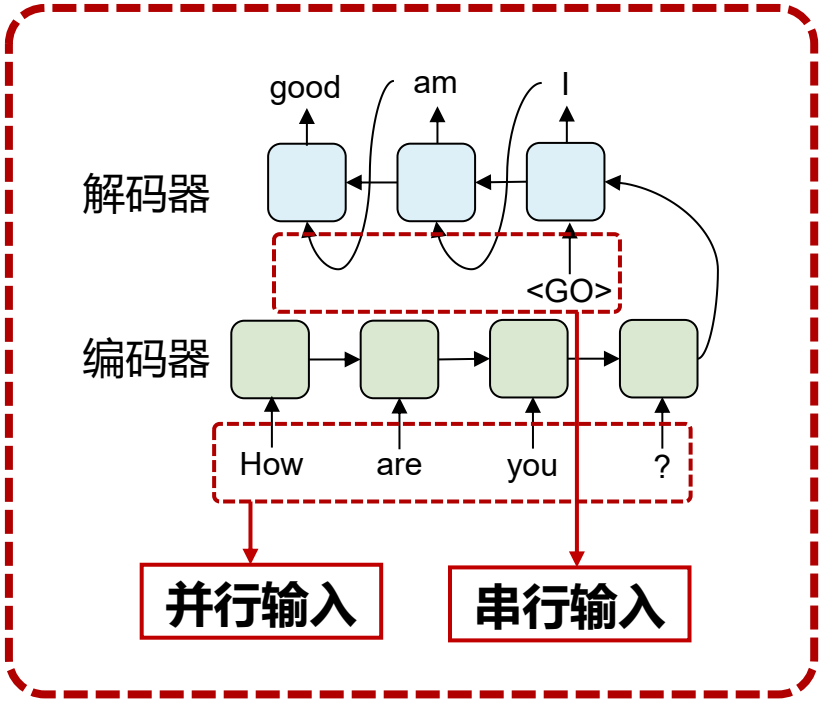
Wang, Hanrui, Zhekai Zhang, and Song Han. *HPCA* 2021

目录

模块级加速：注意力模块加速

- **注意力模块结构分析**
  - 数据调度主要难点：MatMul运算需要等待所有token的线性层运算结束
- **架构加速需求：**
  - 流水并行设计、分阶段并行设计

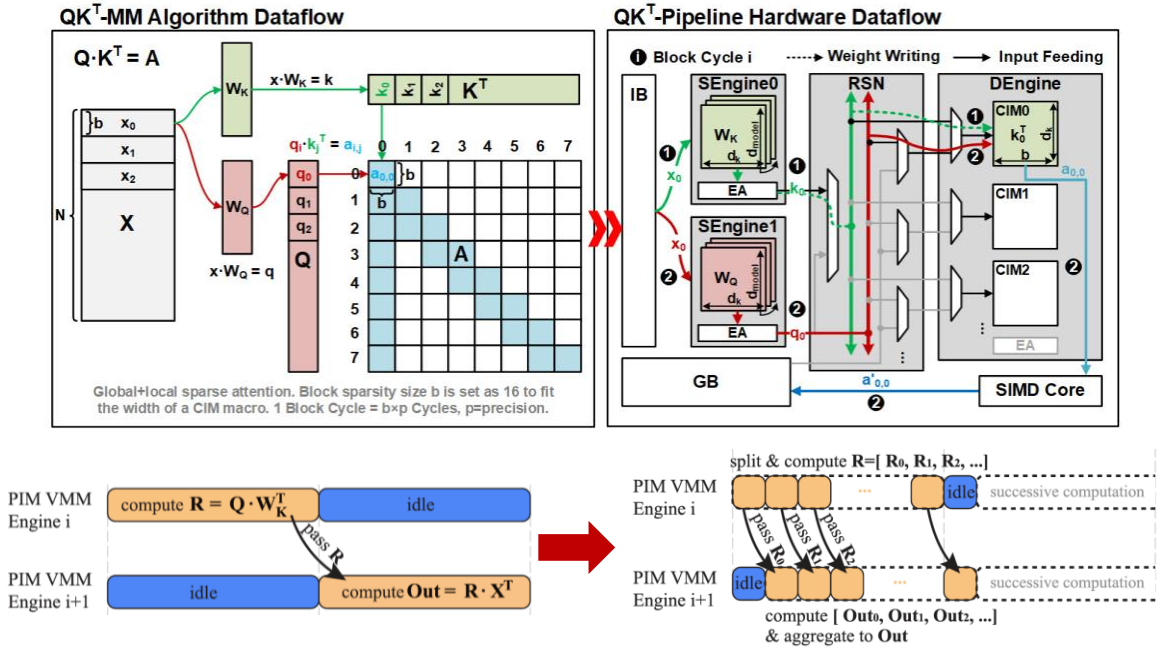




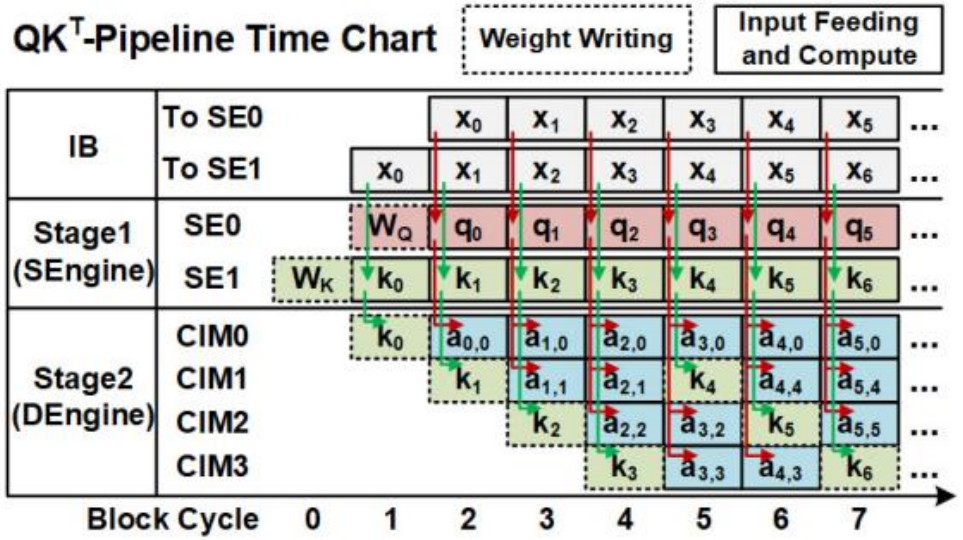
模块级加速：注意力模块加速

- **流水并行设计**
  - 将输入拆分为**多块切片**提高计算核心利用率
  - Q/K/V线性层**运算错位**以并行进行写入权重和计算

输入切片



流水设计

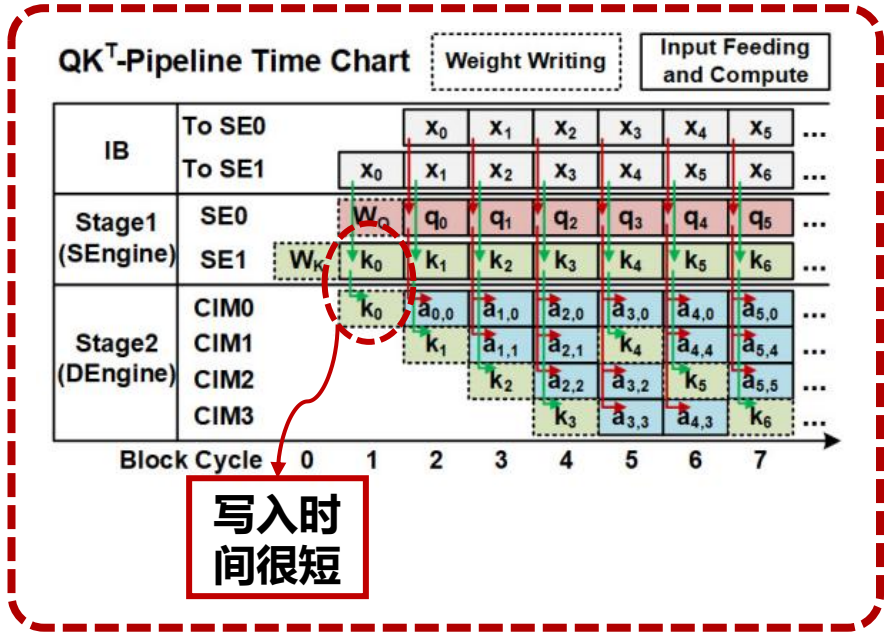


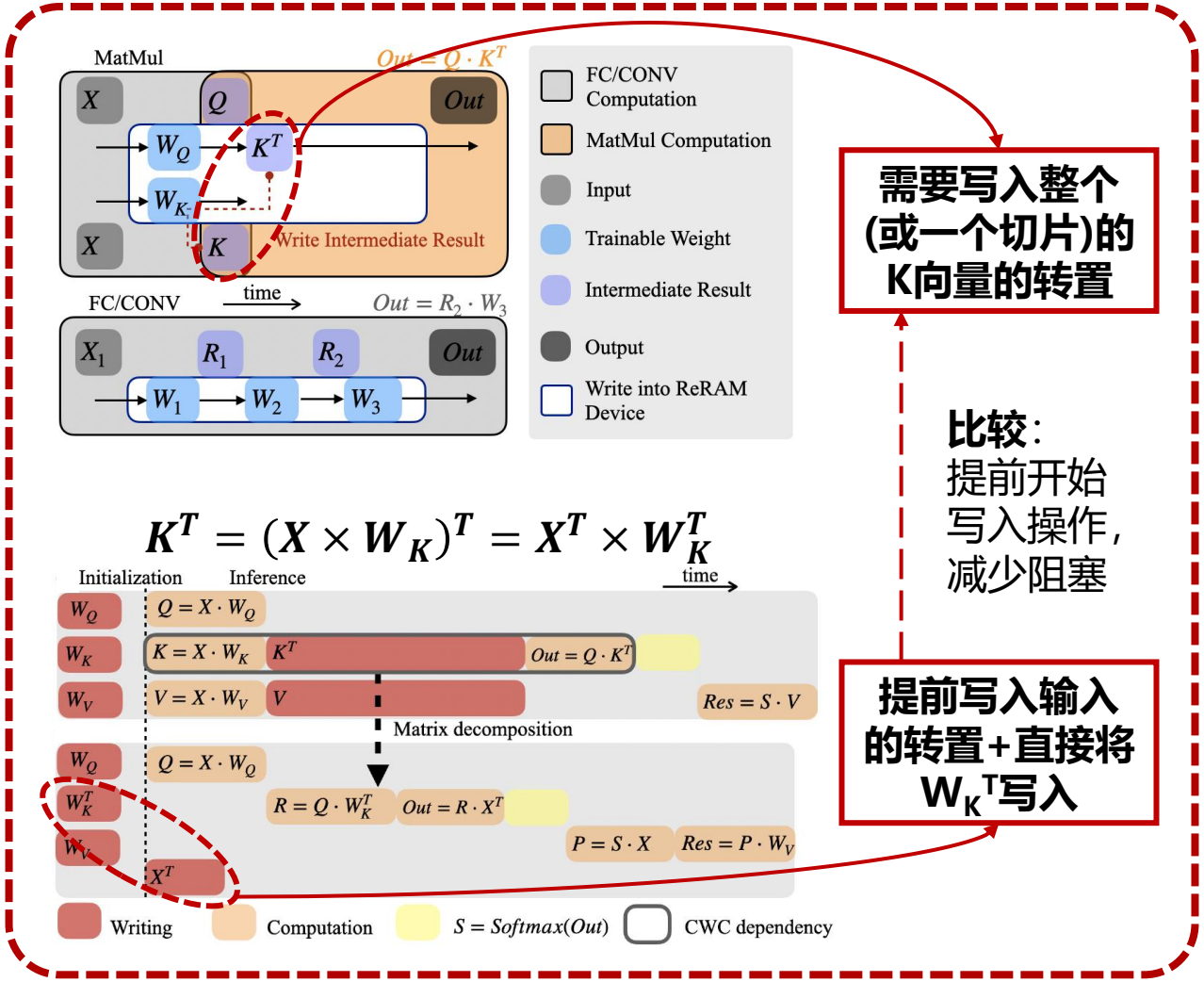
Tu, Fengbin, et al. *ISSCC* 2022

Yang, Xiaoxuan, et al. *ICCAD* 2020

模块级加速：注意力模块加速

- **流水并行设计**
  - 通过拆分运算以提前写入

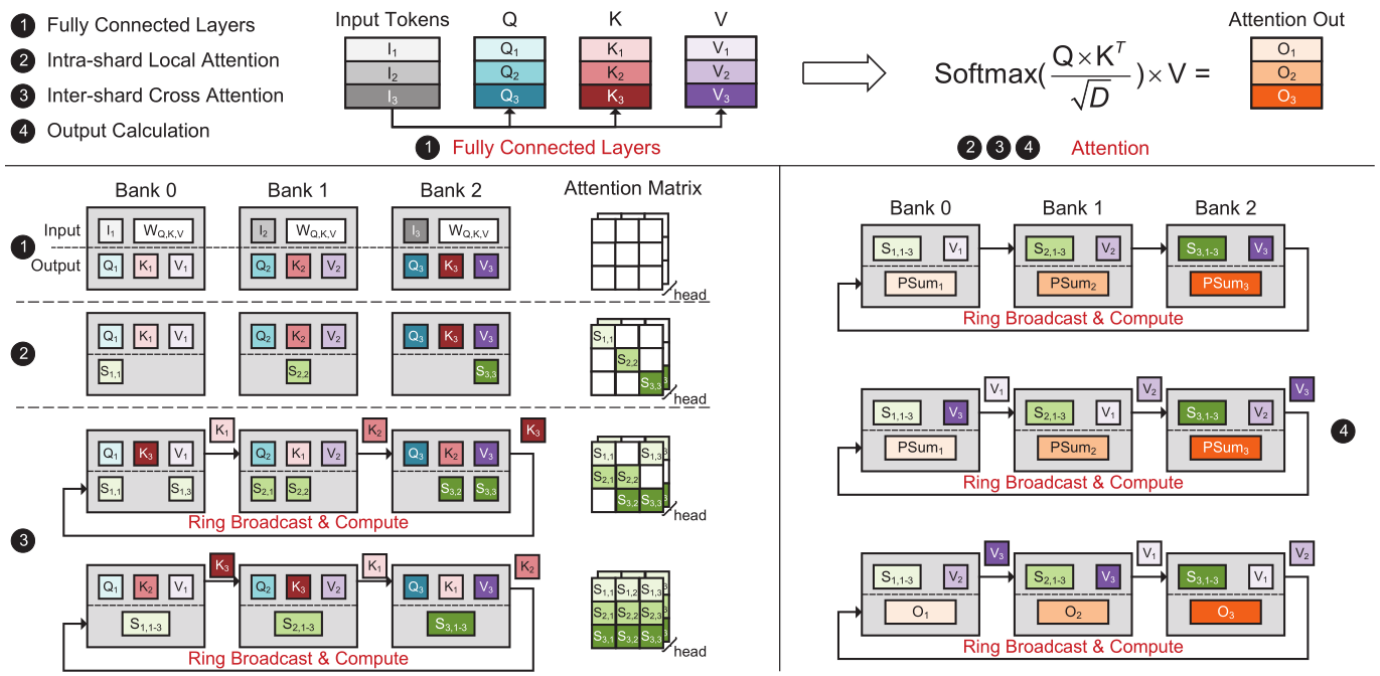




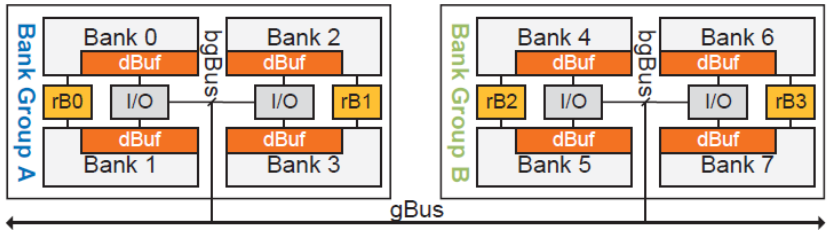
Yang, Xiaoxuan, et al. *ICCAD* 2020

模块级加速：注意力模块加速

- **分阶段并行设计**
  - 通过**环形广播**和所有token进行注意力计算



环形广播



总线传输 + 环形广播缓存传输

Time Step	1	2	3
Bank Group Bus A	3→4	7→0	1→2
Bank Group Bus B	3→4	7→0	5→6
Ring Broadcast Buffer	0→1, 6→7	2→3, 4→5	*idle*

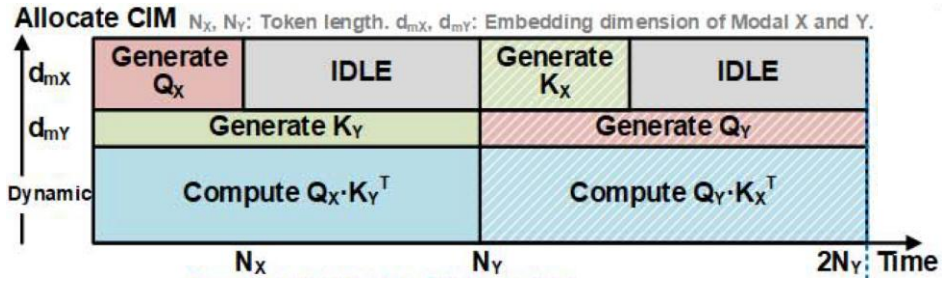
Zhou, Minxuan, et al. *HPCA* 2022

模块级加速：注意力模块加速

- **分阶段并行设计**
  - 针对**多模态**输入规划权重片上分布
  - 提高**计算核心利用率**，降低推断延时

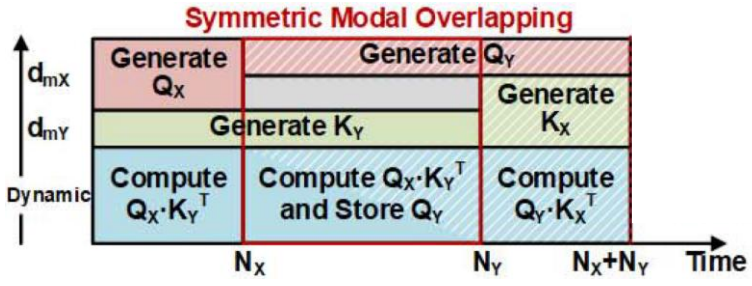
多模态输入的问题

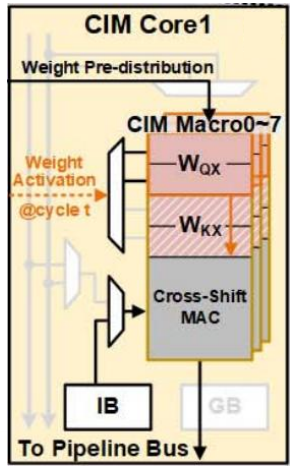
- **传统权重分布**：不同步骤在不同计算核心实现
- **多模态输入**：不同阶段运算延时不同
- 导致多模态输入不匹配时产生**额外延时**



解决方案

- 计算核心**存储多份权重**矩阵
- 利用**模态对称性**重叠运算
- 分配负载提高硬件利用率





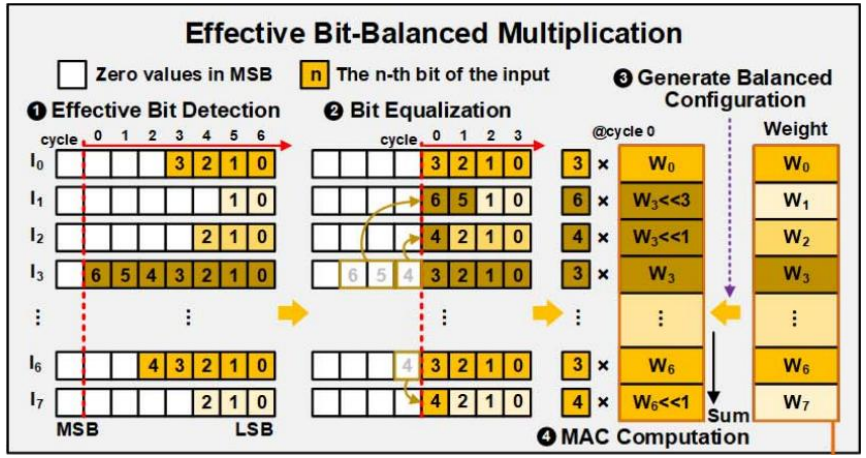
Tu, Fengbin, et al. *ISSCC* 2023

目录

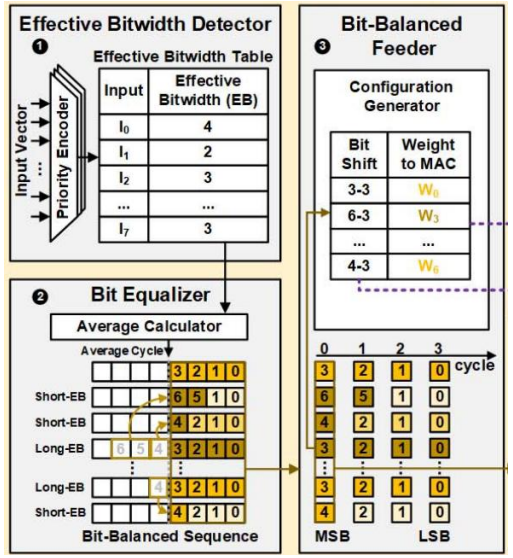
模块级加速：前馈神经网络加速

- **输入特征**——token之间的值差距较大
  - **注意力机制**本身设计目的是使模型更关注关键位置
  - **softmax**函数会放大较大输出，负值等较小输出会接近于0
- 输入前进行**有效位宽均衡**操作

有效位宽均衡



- 输入检测**有效位宽**
- 均衡有效位宽
- **等有效位宽序列**输入
- 根据**均衡表格**配置权重
- 根据均衡表格接收输出



Tu, Fengbin, et al. *ISSCC* 2023

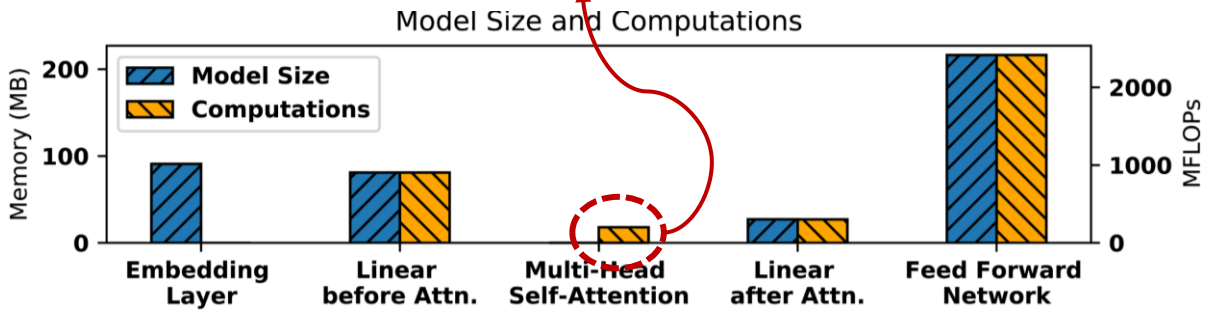
目录

算子级加速：线性算子加速

- **线性算子加速**
  - 全连接层、注意力机制、ViT的卷积运算、残差等涉及常规乘加操作
  - 以**MVM**算子和**MatMul**算子为主
  - MVM算子：**权重矩阵固定**，输入向量可变
  - MatMul算子：两个输入矩阵都不固定，每次计算前需要**加载权重矩阵**

算子计算量分布

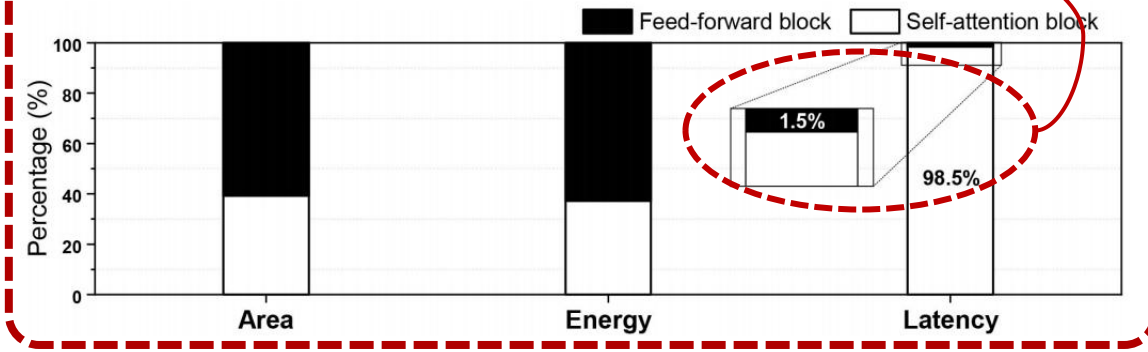
MatMul算子仅占**0.5%**左右



Ganesh, Prakhar, et al. *TACL* 2021

算子计算量分布

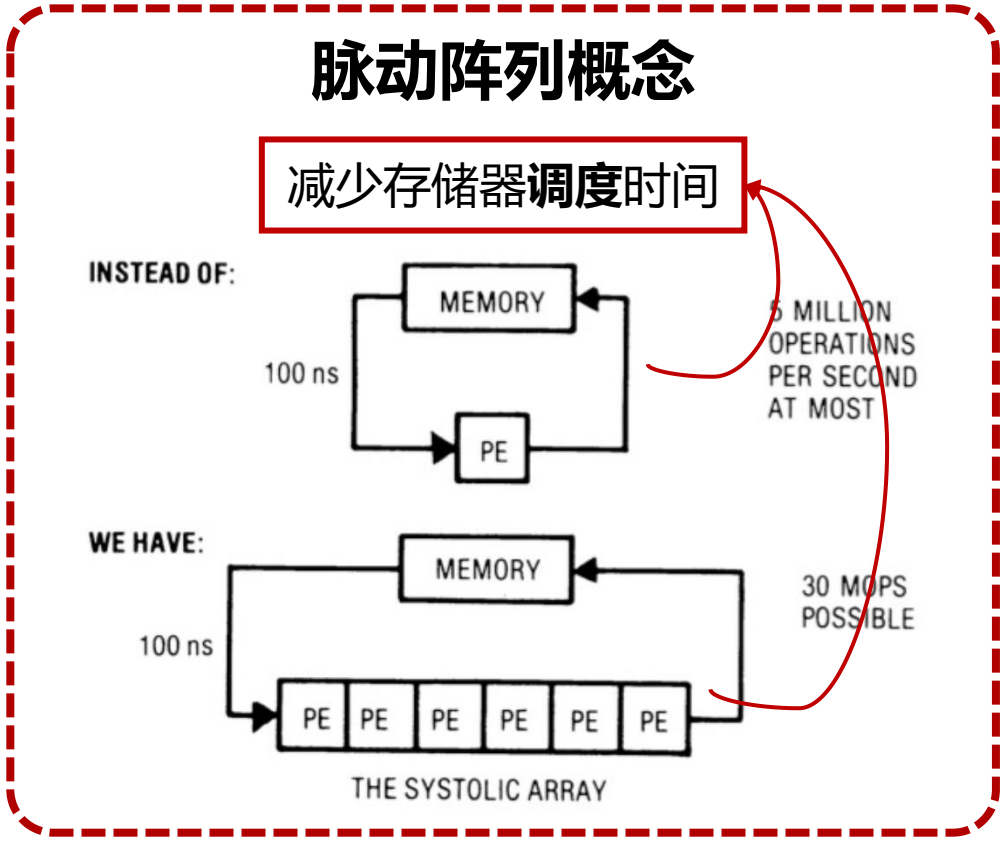
矩阵加载可能占据很高的延时

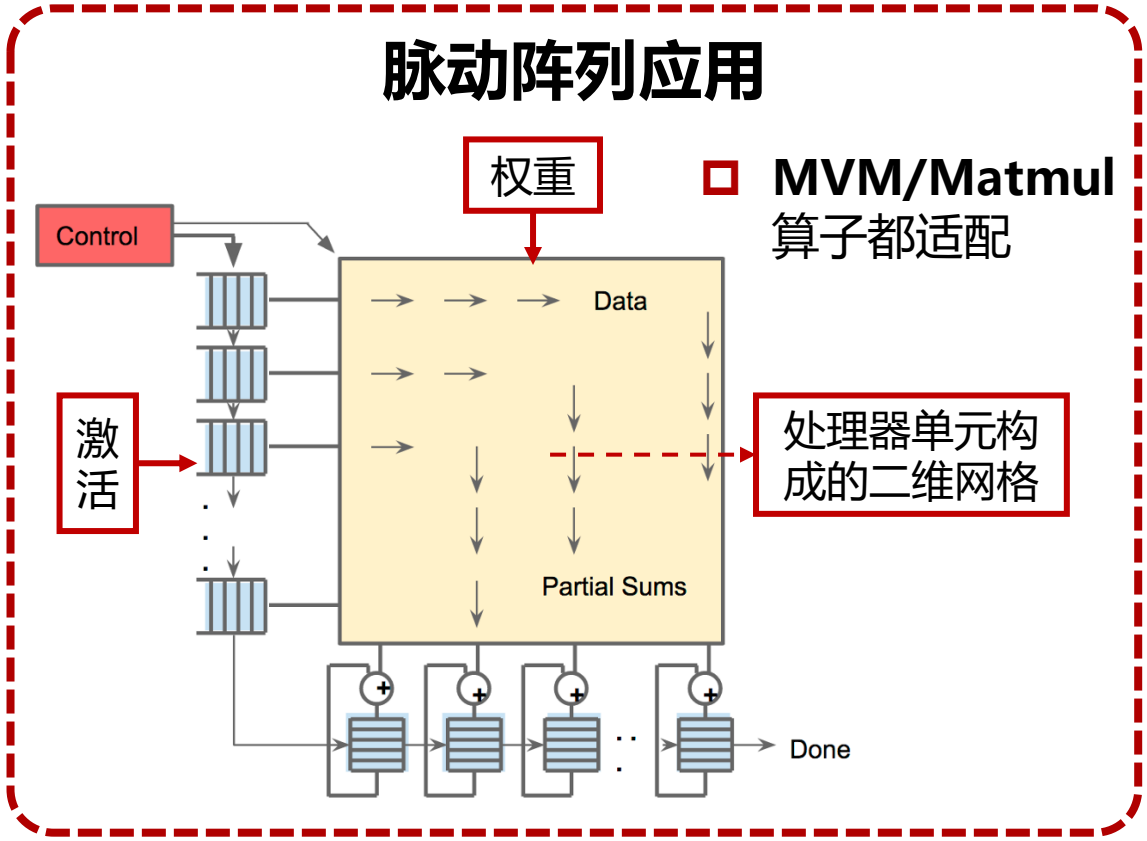


Kang, Myeonggu, Hyein Shin, and Lee-Sup Kim. *TCAD* 2021

算子级加速：线性算子加速

- **存算分离**
  - **脉动阵列**加速乘法累加运算



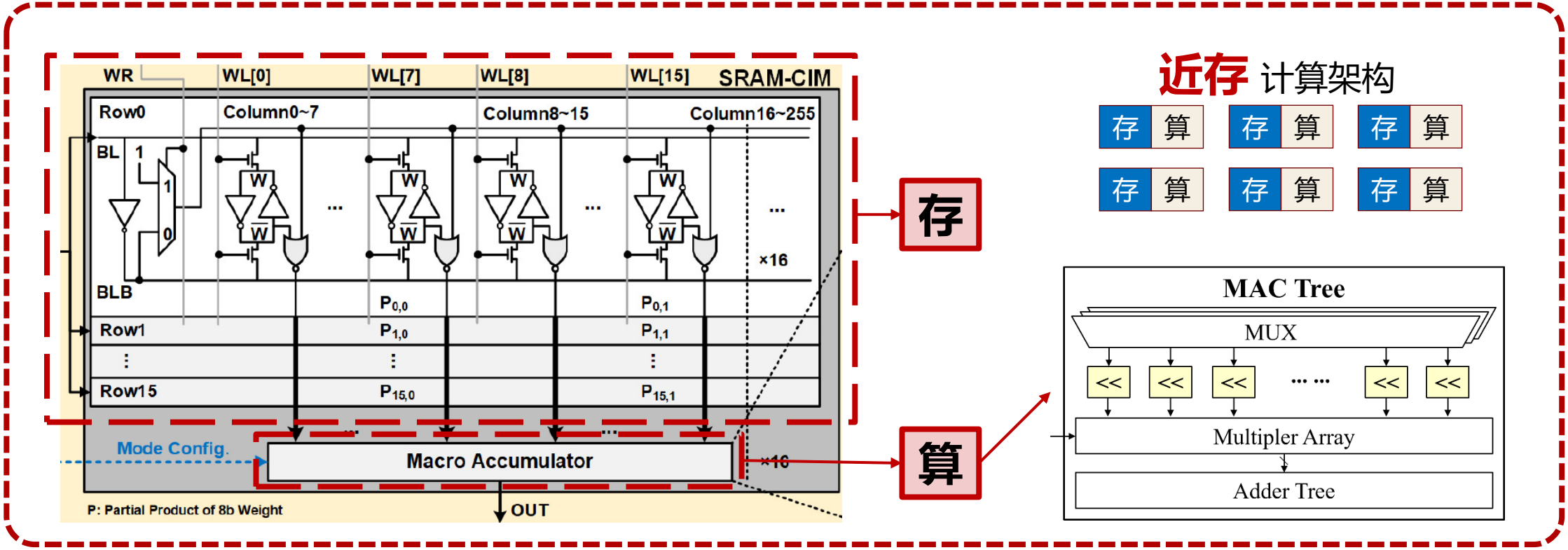


Kung, Hsiang-Tsung. *Computer* 1982

Jouppi, Norman P., et al. *ISCA* 2017

算子级加速：线性算子加速

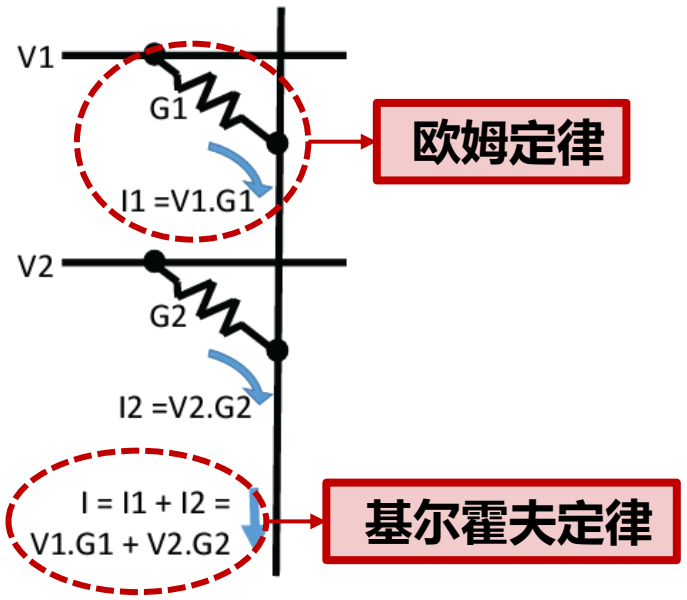
- **近存计算**
  - 存储阵列输出处增加逻辑单元，**减少传输**；DRAM、SRAM等存储介质
  - 乘法累加树——写入时间较长时影响MatMul算子实现



Liu, Shiwei, et al. *ISSCC* 2023

算子级加速：线性算子加速

- **存内计算**
  - 存储阵列内完成乘法累加运算，提高**吞吐**和**能效**；忆阻器、SRAM等存储介质
  - 利用交叉阵列实现——写入时间较长时影响MatMul算子实现



存内 计算架构

- ① **模拟存内计算：**
  - 开启多行直接实现累加计算
- ② **数字存内计算：**
  - 单行开启，在阵列内完成乘法运算
- **算子实现**：
  - MVM算子实现能效较高
  - 部分介质不适合MatMul算子

Shafiee, Ali, et al. *ACM SIGARCH Computer Architecture News* 2016

目录

算子级加速：非线性算子加速

- **非线性算子加速**
  - 分为需要/不需要统计输入特征的部分

层级	模块级	主要线性算子	非线性算子
embedding	注意力模块	矩阵向量乘法(MVM)	GELU
编码器	前馈神经网络	矩阵矩阵乘法(MatMul)	LayerNorm
解码器	残差模块	残差相加	Softmax

输入特征统计加速

非线性函数加速

GELU

不需要统计输入特征

$$GELU(x) = 0.5x\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

$$erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x exp(-t^2)\,dt$$

LayerNorm

需要统计输入特征

$$LayerNorm(x) = \frac{x-\mu}{\sigma},$$

$$\mu = \frac{1}{C}\sum_{i=1}^{C} x_i\,,\sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i-\mu)^2}$$

Softmax

需要统计输入特征

$$Softmax(x) = \frac{\exp(x_i)}{\sum_{j=1}^{k}\exp(x_j)},$$

$$x = [x_1, \ldots, x_k]$$

算子级加速：非线性算子加速

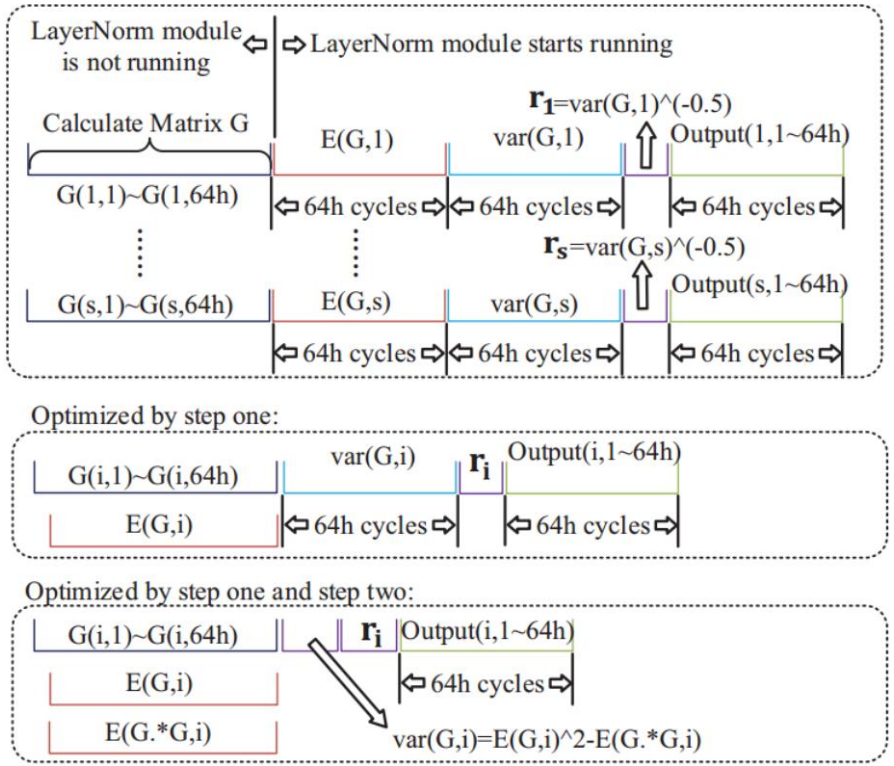
- **非线性算子加速：输入特征统计**
  - 均值统计/方差统计/指数统计

均值统计
- 记录累和值和数量
- 每次数据流经过时更新

方差统计
- 数学重构为 $var = \mu^2 - \frac{1}{C}\sum_{i=1}^{C} x_i^2$
- 记录平方值和数量
- 每次数据流经过时更新

指数统计
- 需要记录所有数据的指数
- 在数据输出时完成

数据流上进行加速

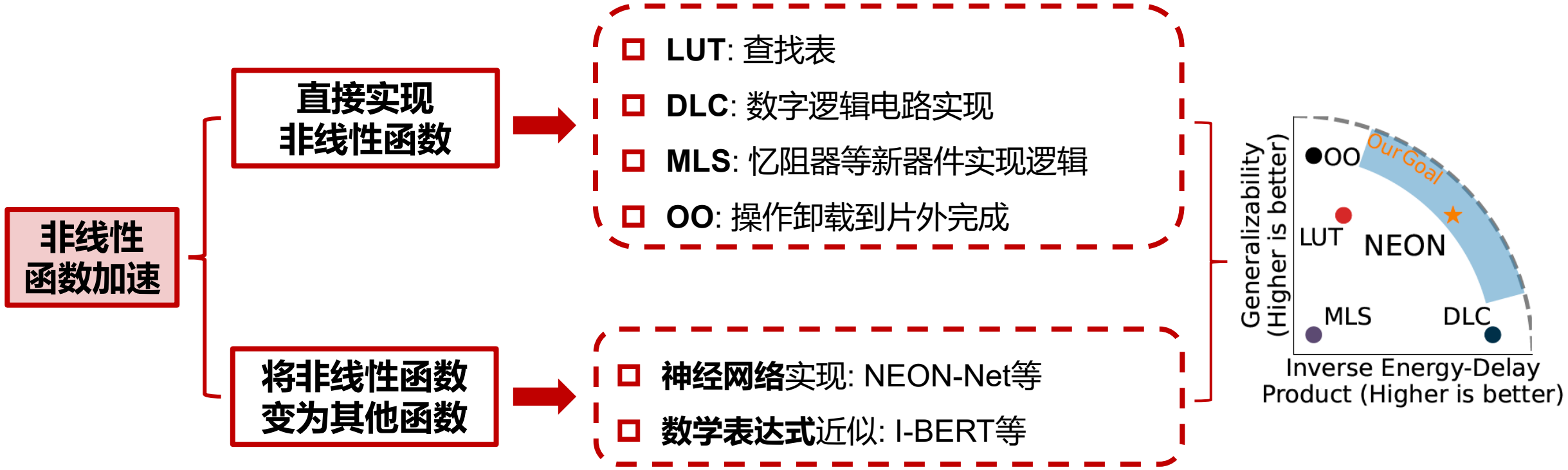


Kim, Sehoon, et al. *PMLR* 2021

Wang, Meiqi, et al. *APCCAS*, 2018.

算子级加速：非线性算子加速

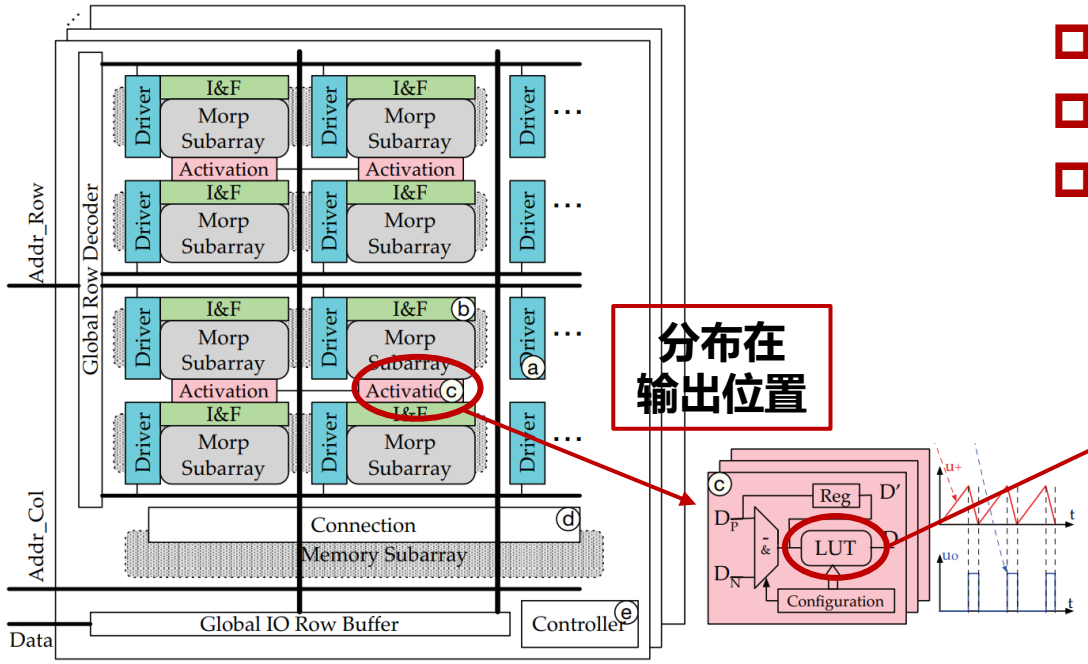
- **非线性算子加速：非线性函数加速**
  - 两种思路：将非线性函数变为线性函数/直接实现非线性函数



Manglik, Aditya, et al. arXiv:2211.05730 (2022)

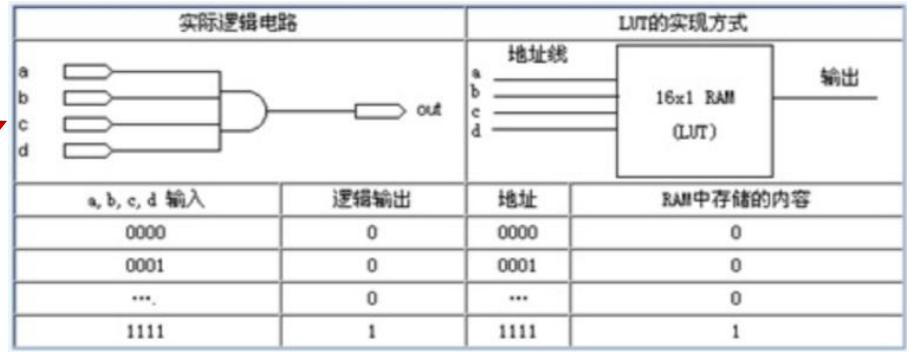
算子级加速：非线性算子加速

- **非线性函数实现：LUT(查找表)**
  - **优点**：同样的电路结构可以用于实现不同的非线性函数，具有一定的**灵活性**
  - **缺点**：大范围输入需要**大容量存储阵列**



分布在输出位置

- 通过**提前存储**结果加速计算
- **可扩展性**限制：大范围输入需要大容量
- **阵列大小**限制：存储阵列大小限制计算范围
  - 拆分多个**子阵列**会导致**高延迟**

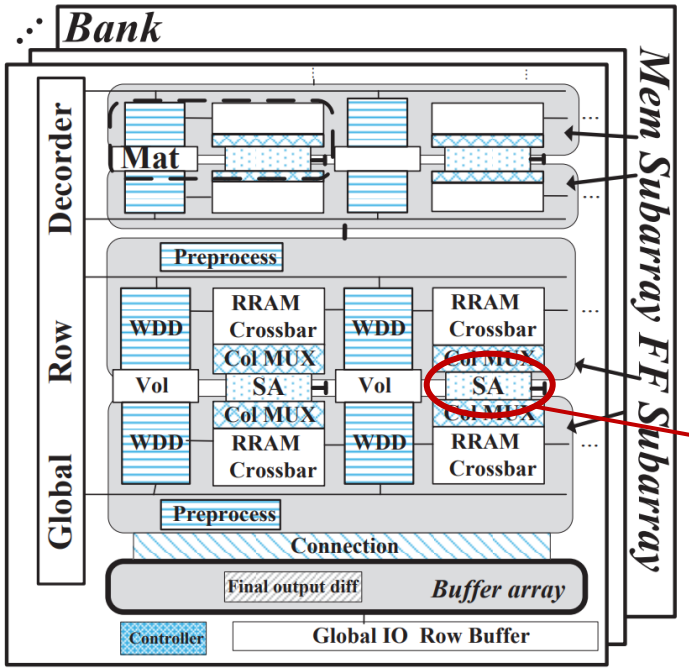


a, b, c, d 输入	逻辑输出	地址	RAM中存储的内容
0000	0	0000	0
0001	0	0001	0
….	0	…	0
1111	1	1111	1

Song, Linghao, et al. *HPCA* 2017

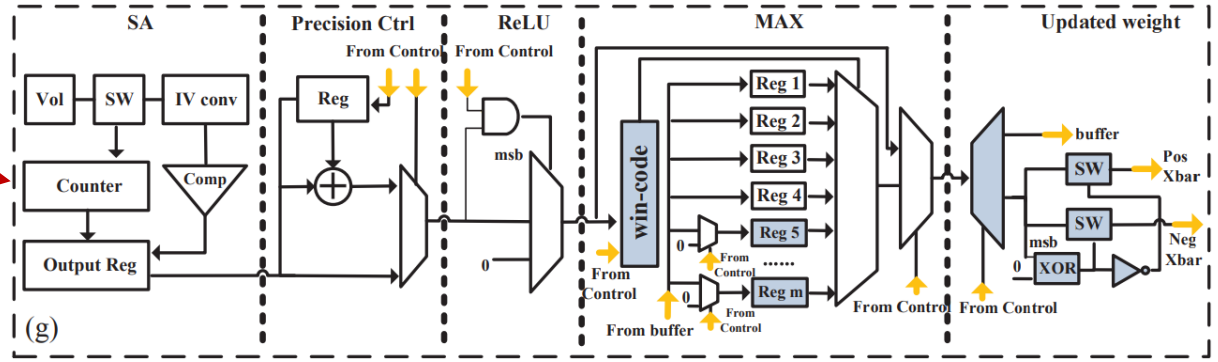
算子级加速：非线性算子加速

- **非线性函数实现：DLC(数字逻辑电路实现)**
  - **优点**：计算过程**延迟**低+使用成熟CMOS工艺对**面积**要求较低
  - **缺点**：固化数字电路导致**灵活度**低+数字设计需要考虑**静态功耗**



分布在输出位置

- **固化了数字逻辑电路**
- **定制化**：对面积要求低并且计算延时低
- **不定制化**：通用计算核心或者泰勒级数实现
  - 前者电路复杂；后者引入乘法器，增加了面积



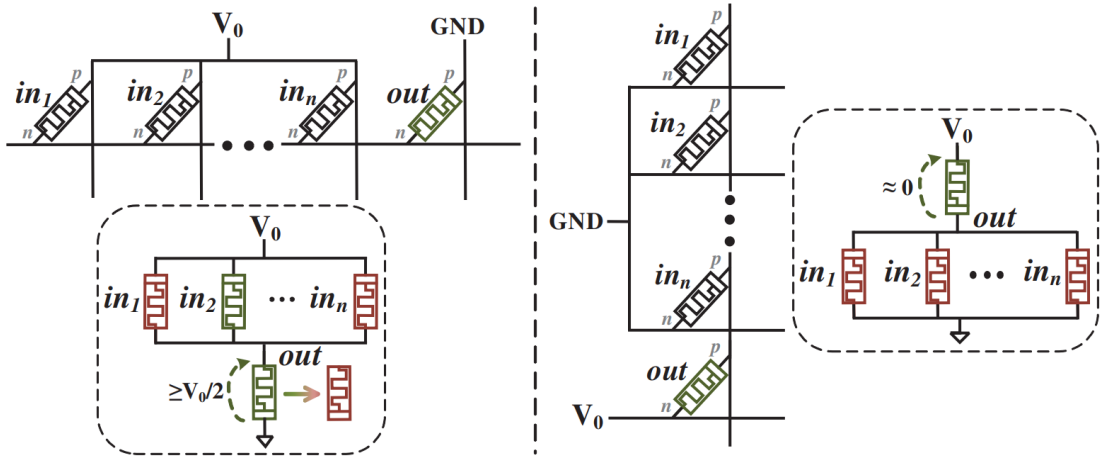
Cheng, Ming, et al. *DAC* 2017

算子级加速：非线性算子加速

- **非线性函数实现：MLS&OO**

MLS(新器件实现逻辑)

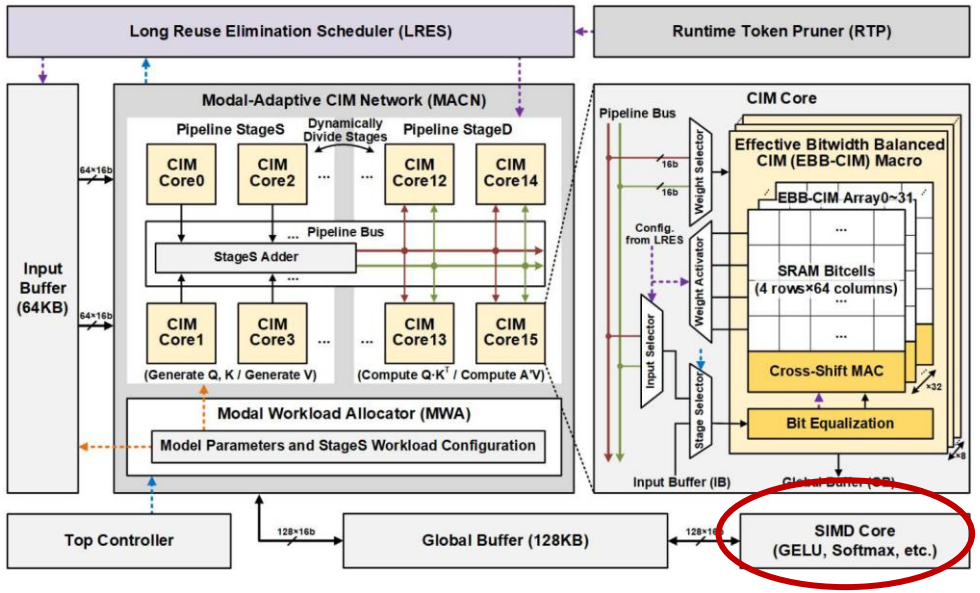
- 通过**器件分压**执行**写入**操作
- **优点：静态功耗**低+计算单元**面积**小
- **缺点：写入能耗**高+定制化**灵活度**低+需要很多器件单元**成本**高



Gupta, Saransh, Mohsen Imani, and Tajana Rosing. *ICCAD* 2018

OO(操作片外卸载)

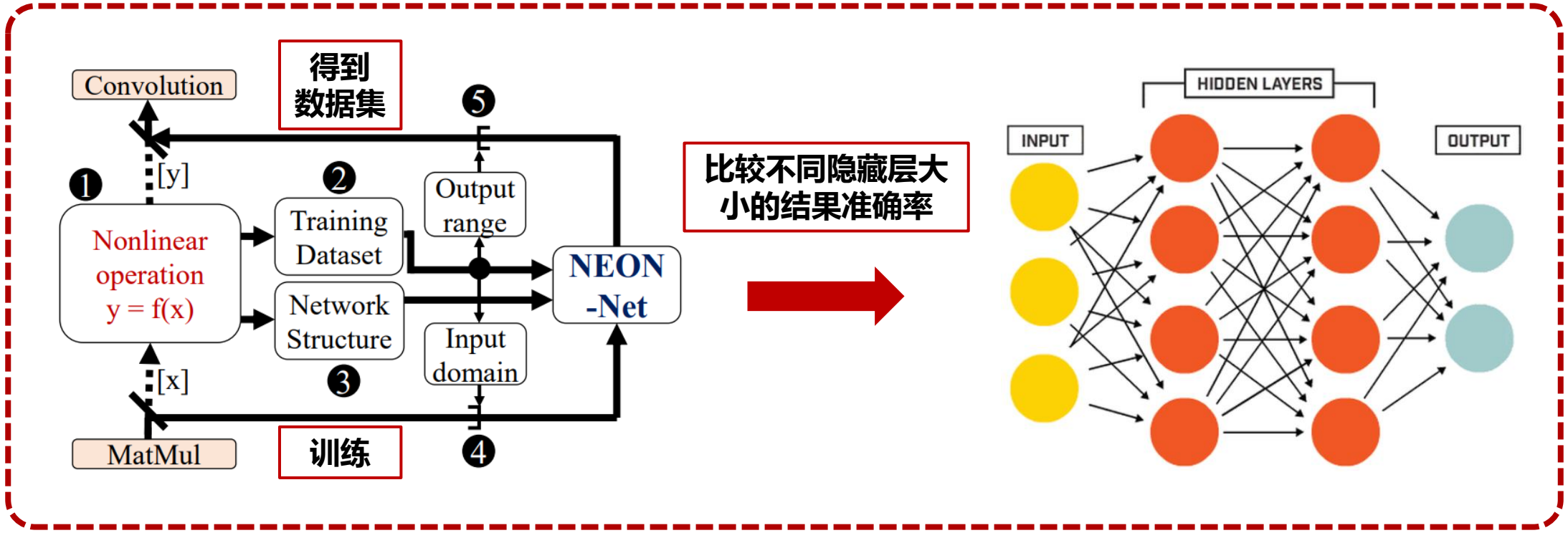
- 数据传输到片外完成
- **优点：**直接利用通用CPU
- **缺点：数据移动**过多影响能耗和延时



Tu, Fengbin, et al. *ISSCC* 2023

算子级加速：非线性算子加速

- **非线性函数转换：神经网络实现**
  - 利用存算一体等对神经网络加速比较高的实现方式
  - 通过泛函数逼近定理精确地模拟非线性函数



Manglik, Aditya, et al. arXiv:2211.05730 (2022)

算子级加速：非线性算子加速

- **非线性函数转换：数学表达式近似**

GELU

- 误差函数近似
- 根据输入范围近似

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x exp(-t^2)\, dt$$

$$= sgn(x)[a(clip(|x|, max = -b) + b)^2 + 1]$$

LayerNorm

- 迭代实现开根号效果
- 重复 $x_{i+1} = \left\lfloor (x_i + \frac{n}{x_i})/2 \right\rfloor$

Softmax

- 和最大值进行归一化

缩小输入范围

$$Softmax(x) = \frac{\exp(x_i)}{\sum_{j=1}^{k} \exp(x_j)} = \frac{\exp(x_i - x_{max})}{\sum_{j=1}^{k} \exp(x_j - x_{max})}$$

$$= \exp[x_i - x_{max} - \ln(\sum_{j=1}^{k} \exp(x_j - x_{max}))],$$

避免除法器

- 指数函数的多项式近似

$$exp(x) = 0.3585(x + 1.353)^2 + 0.344$$

- 指数函数/对数函数拆分2的基数

小数运算+移位

$$e^y = 2^{u+v} = 2^u \times 2^v = \begin{cases} 2^v \ll u & u > 0 \\ 2^v \gg u & u \le 0 \end{cases}, u = \lfloor y \times \log_2 e \rfloor, v = y \times \log_2 e - u$$

$$lnF = ln2 \times \log_2 F = ln2(w + \log_2 k) = ln2(k - 1 + w), F = 2^w + k$$

Kim, Sehoon, et al. *PMLR* 2021

Wang, Meiqi, et al. *APCCAS* 2018

目录

总结及展望


高延时&高能耗
ASIC/专用加速器
模型参数量大
数据调度和传统深度神经网络不同
缺乏对专用算子的支持
模型级加速
模型压缩
稀疏性处理
模型量化
+
一系列结构简化
对不同的稀疏性的动态剪枝
部分最低可量化到二值
结构简化降低存储需求
动态剪枝减少计算量
模块级加速
注意力
前馈神经网络
流水并行
+
分阶段并行设计
对输入/权重特征进行分析
不同的网络种类适配不同的并行设计
分析输入/权重特征
算子级加速
线性算子
非线性算子
不同存算模式/不同存储介质
非线性函数变为线性
+
不同非线性函数实现方式
近似或迭代实现非线性算子
分析线性算子特征和实现