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

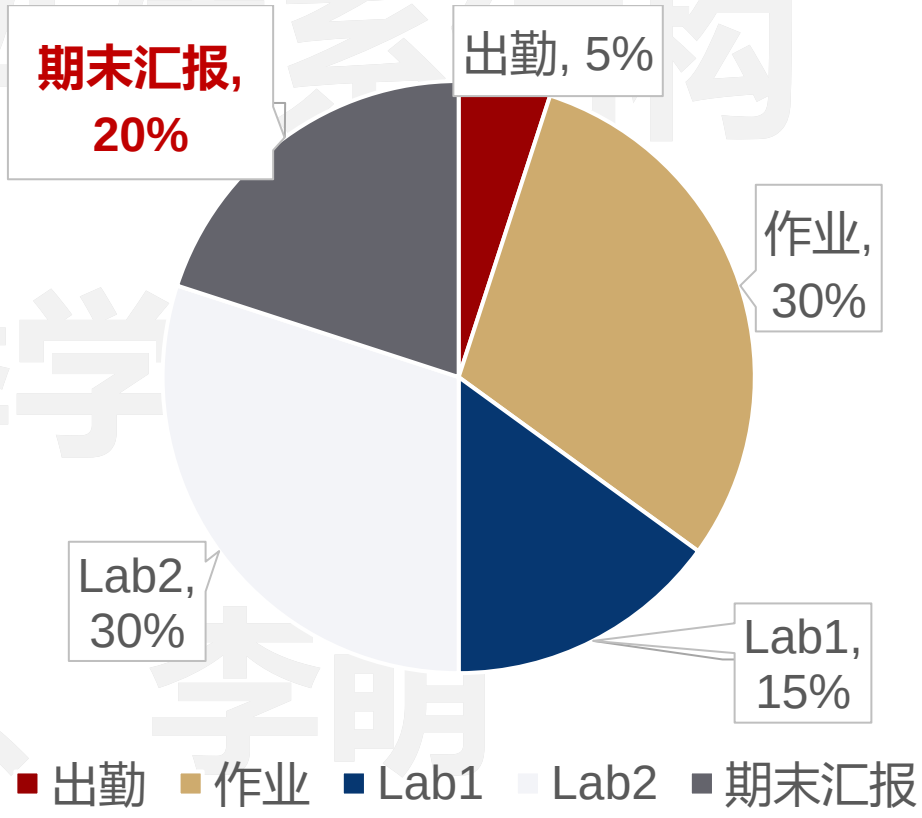
主讲：陶耀宇、李萌

主讲：陶耀宇

2025年春季

注意事项

- 课程作业情况
  - 第5次作业时间：5月13号-5月27号（2道题）
  - 第6次作业时间：5月27号~6月10号（1道题）
  - 第2次lab时间：4月17日 – 6月15日
    - 基础任务（多个test programs通过率）
    - Bonus（占比Lab2的50%，简单AI卷积加速核设计，开放式、无强制要求）
  - 期末汇报
    - 请自行挑选并阅读近5年内的芯片/架构等相关论文
    - 请同学们准备10页左右的PPT，**每位同学12分钟演讲 + 3分钟答疑**
    - **每位同学均为汇报同学打分，互评成绩占期末汇报成绩50%，请于5月19日前将汇报题目发给助教**



目 录

CONTENTS

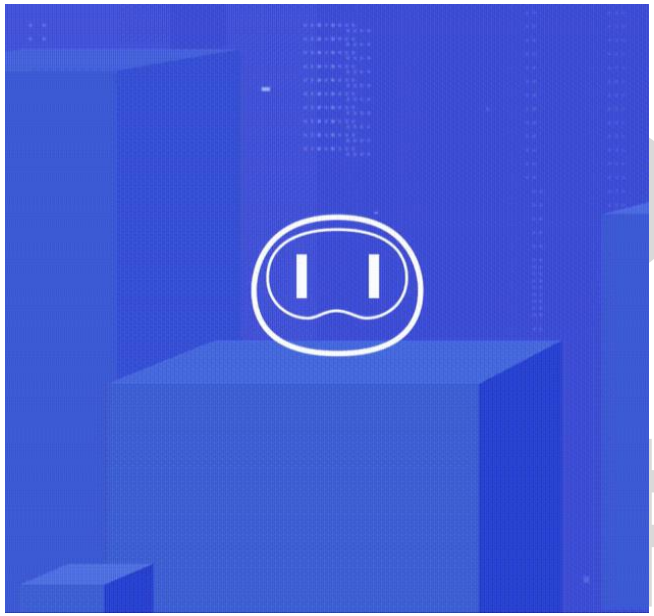
研究背景

主流大模型 ← 主干网络 ← Transformer

主流大模型 → 聊天机器人
主流大模型 → 图像合成
主流大模型 → 文本视频关联

聊天机器人

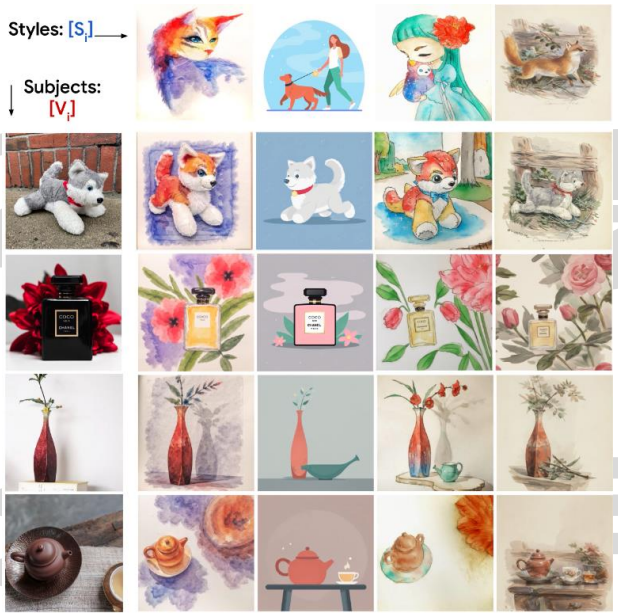
实时聊天 + 性格设定



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

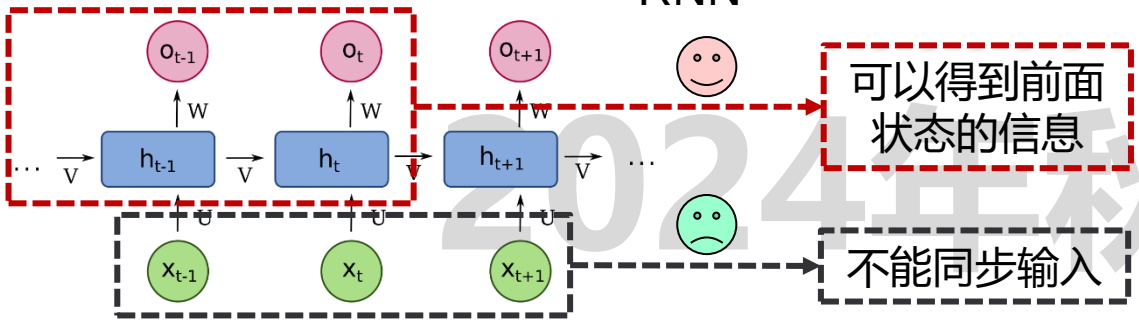
根据文字更改视频内容

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	✓	✓

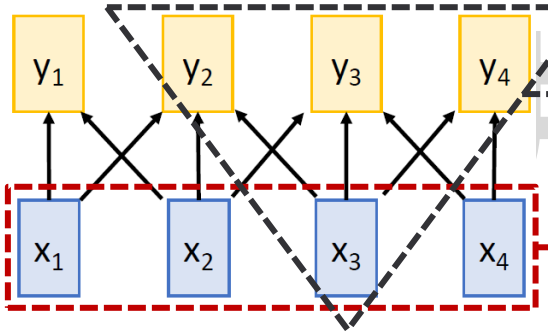
RNN



可以得到前面状态的信息

不能同步输入

CNN

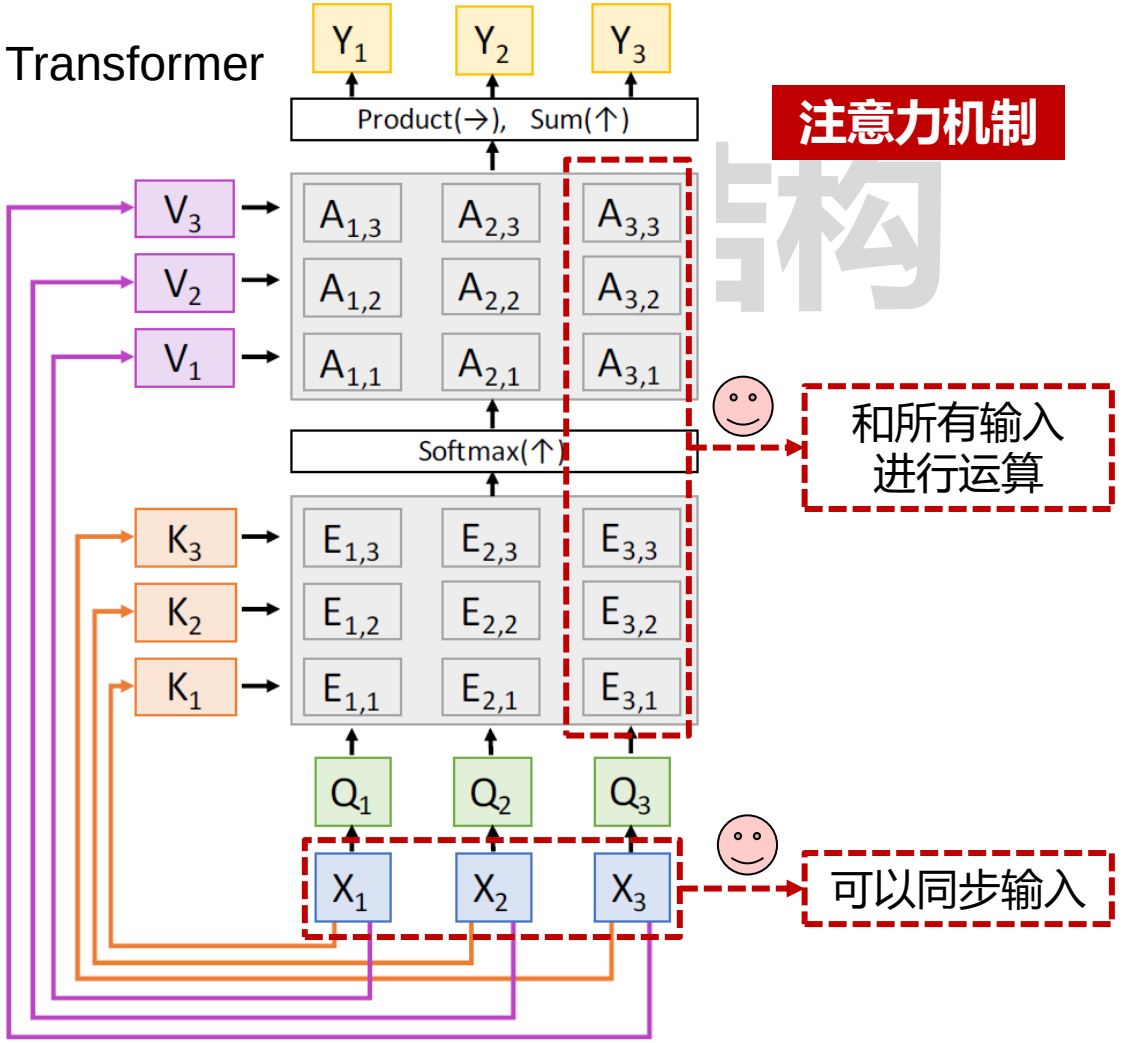


不能得到全局信息

可以同步输入

Transformer

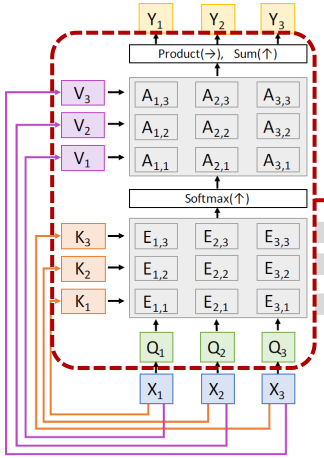
注意力机制



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**

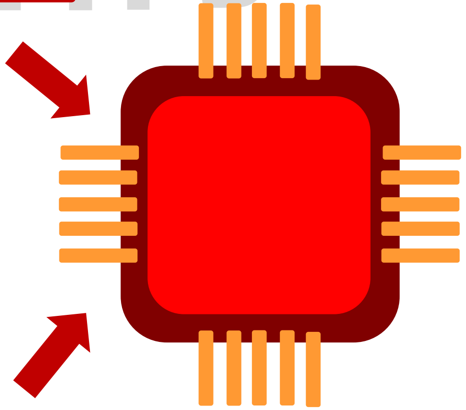
任务种类增多 → 字典维度增大 → **注意力模块延时增加**；**输入维度增大导致能耗增加**

训练延时随模型增大**增加**

能耗随模型增大**增加**

注意力模块延时**增加**、训练延时随模型增大**增加** → **高延时**

输入维度增大导致能耗**增加**、能耗随模型增大**增加** → **高能耗**



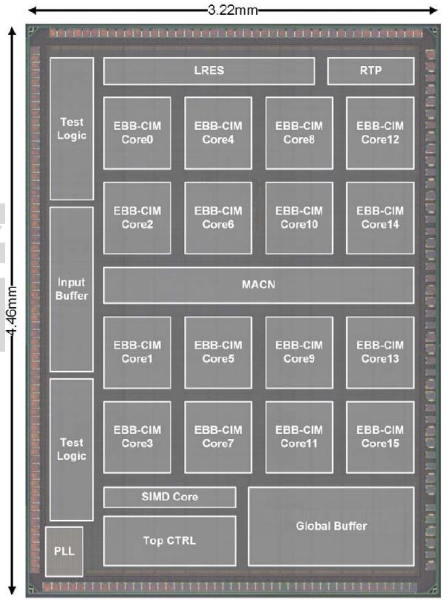
ASIC/专用加速器

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

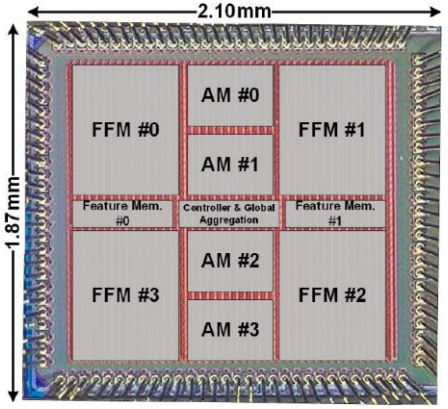
From LLaMA/LLaMA 2, 数据收集于A100-80GB

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



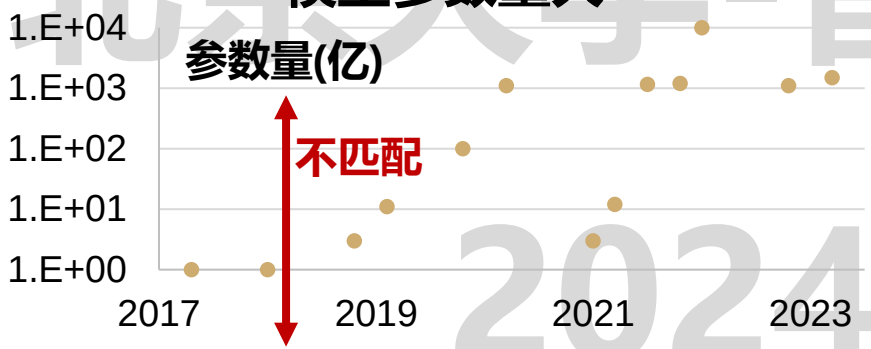
Tu, Fengbin, et al. *ISSCC* 2023

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



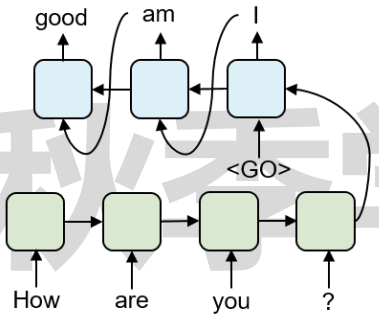
存储介质	SRAM	DRAM	RRAM	PCM	Flash
容量	1MB	160GB	4MB	21.25MB	80MB

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

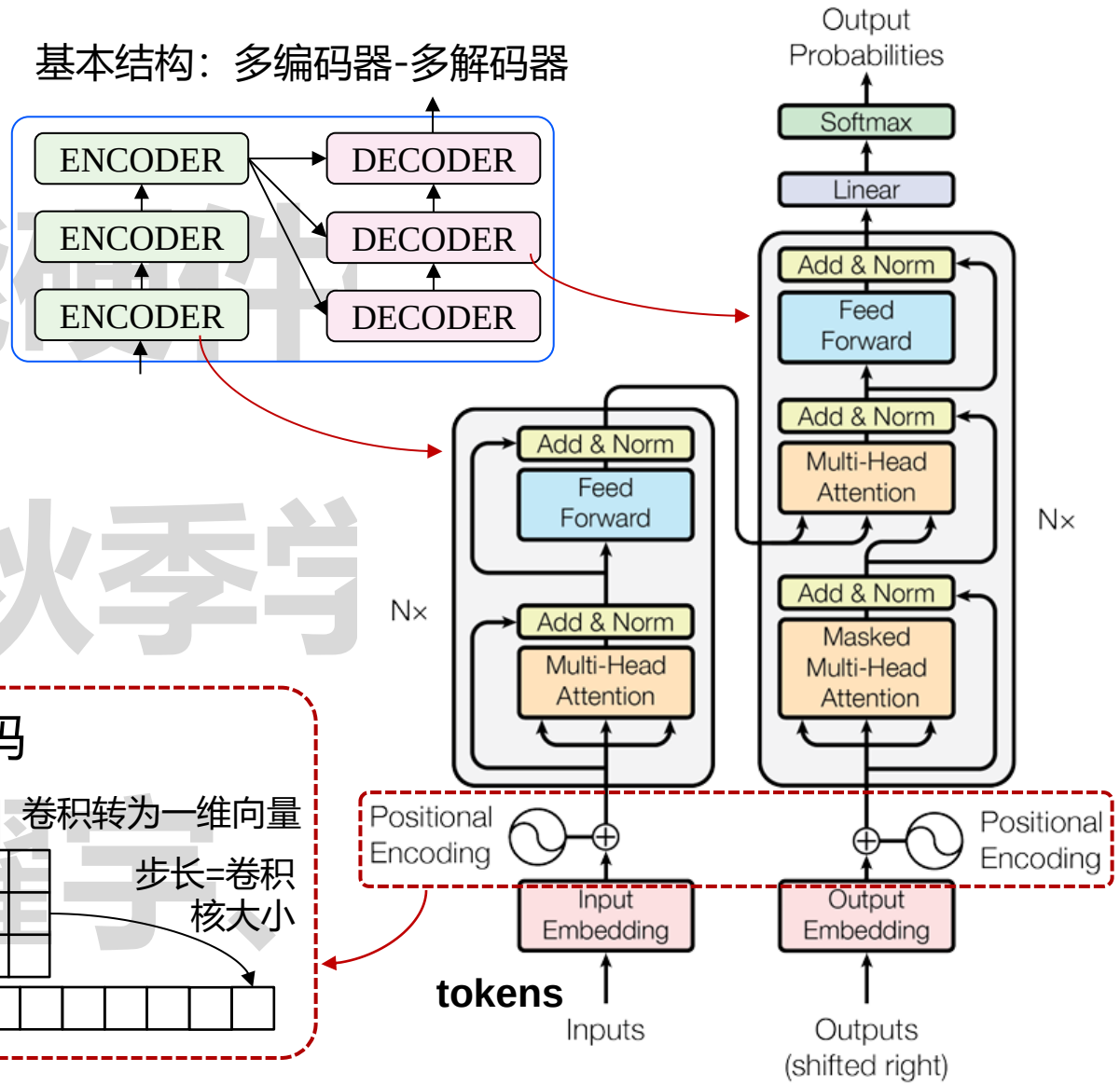
主要线性算子	非线性算子
矩阵向量乘法(MVM)	GELU
矩阵矩阵乘法(MatMul)	LayerNorm
残差相加	Softmax

输入和权重不固定　　**动态归一化**

算子级加速
加速专有算子

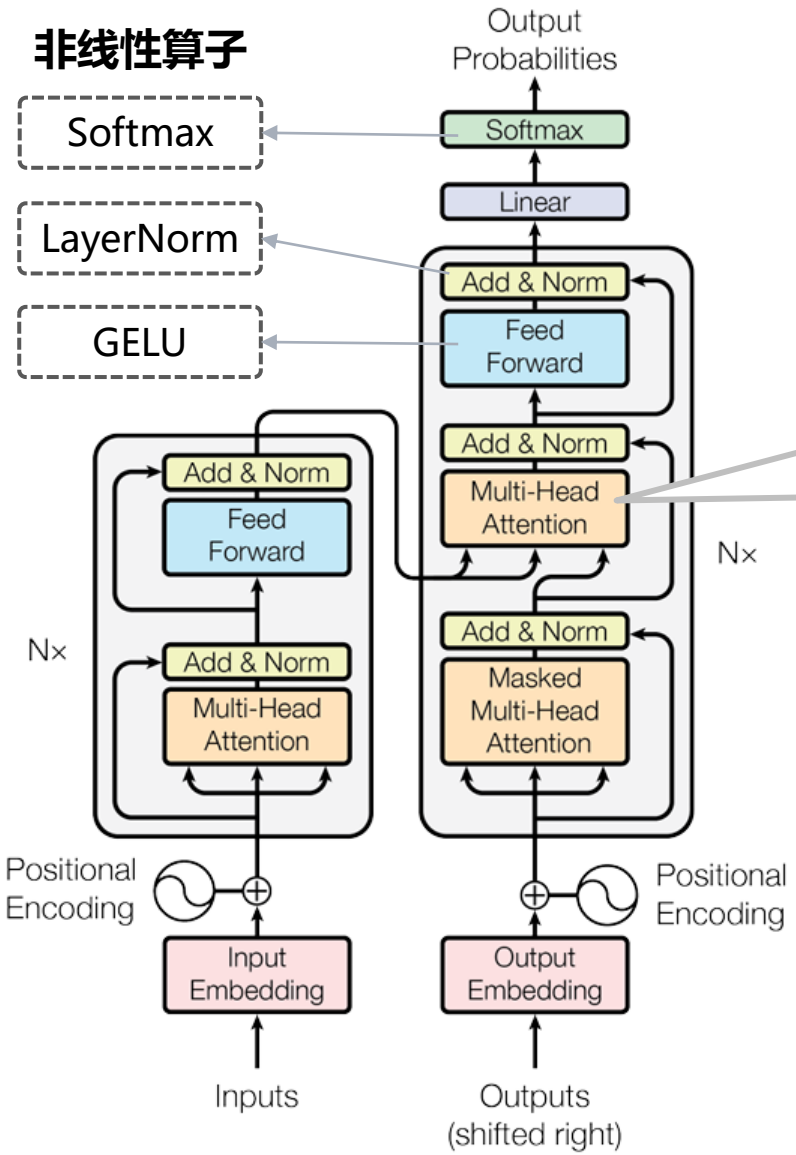
Transformer网络结构

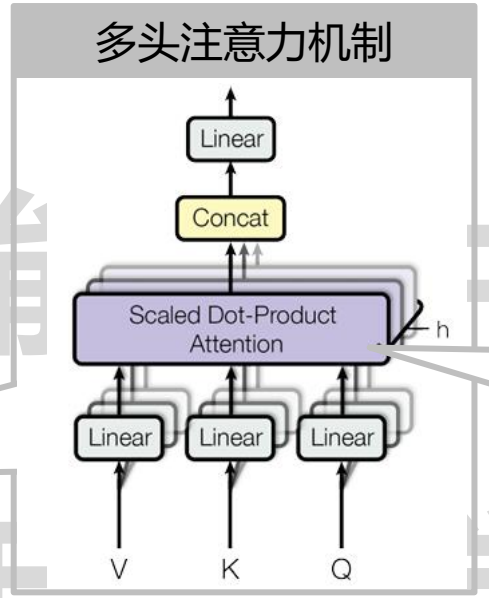
- **基本结构**：
  - 多个编码器层-多个解码器层
- **输入序列**：
  - 由一组token组成
  - 每个token为一个向量表示一个词根
  - 序列开头[CLS]——句子向量表示
  - 输入模型前经过embedding层
- **embedding**：
  - 输入编码：字典信息
  - 位置编码：位置信息

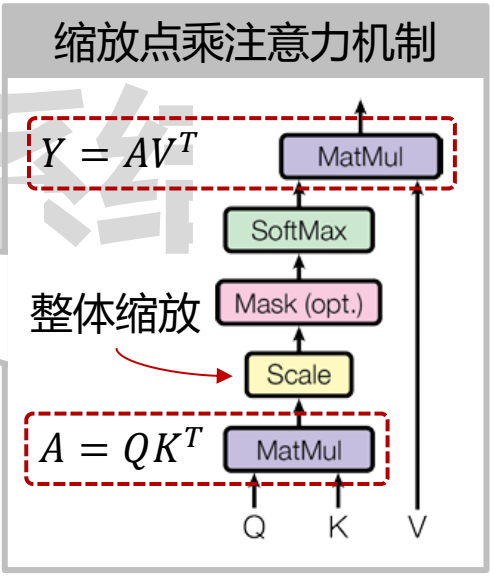


Vaswani, Ashish, et al. *NIPS* 2017

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

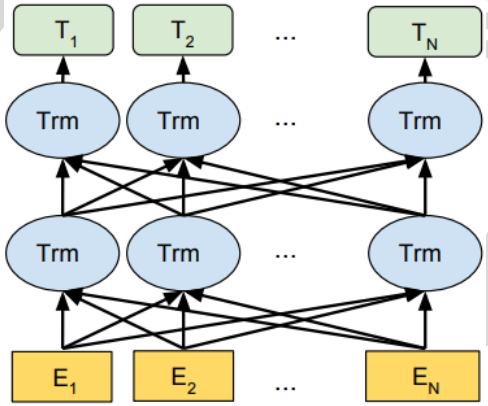
低延时要求

输入依赖于上一次的输出

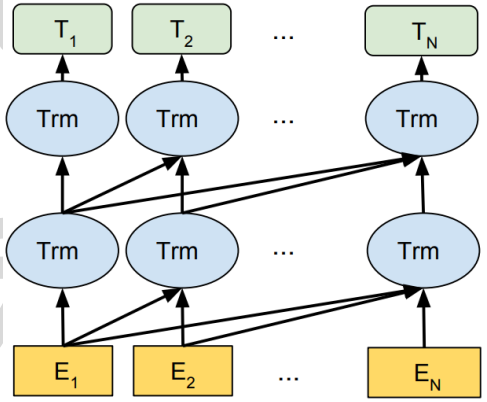
流水设计

分阶段并行设计

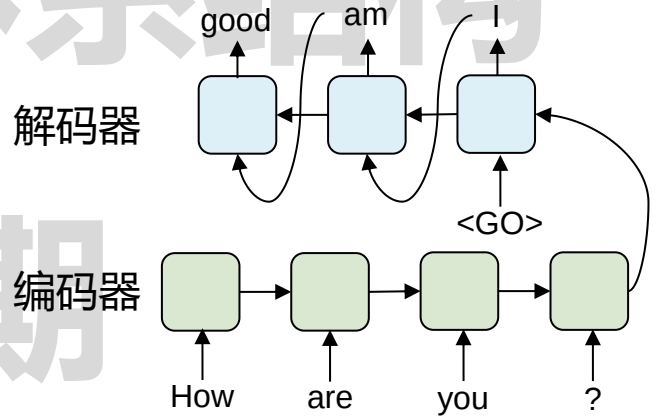
Encoder-only



Decoder-only



Encoder-Decoder



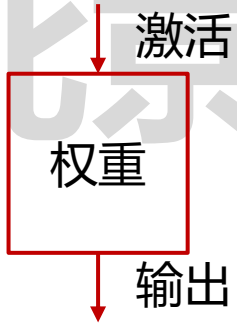
常见模型	BERT，RoBERTa	GPT，LLAMA	T5，BART
适合任务	分类任务	问答任务	翻译任务
输入输出特征	所有token并行输入	输入依赖于上一次的输出	输出强依赖于输入
适配加速器特征	**高吞吐**	**低延时**	**低延时**
适配并行设计	**流水并行设计**	**分阶段并行设计**	**分阶段并行设计**

目录

模型级加速：模型压缩

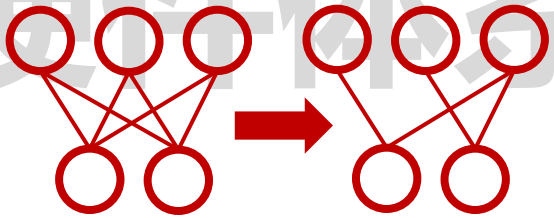
- **模型压缩**：硬件部署前**算法**优化

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**

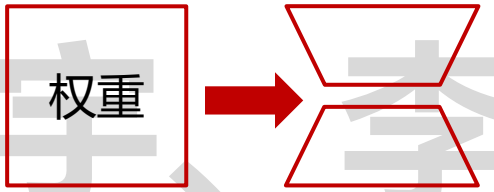


- 删去**不重要**的注意力维度/头
- 降低硬件**计算量**

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

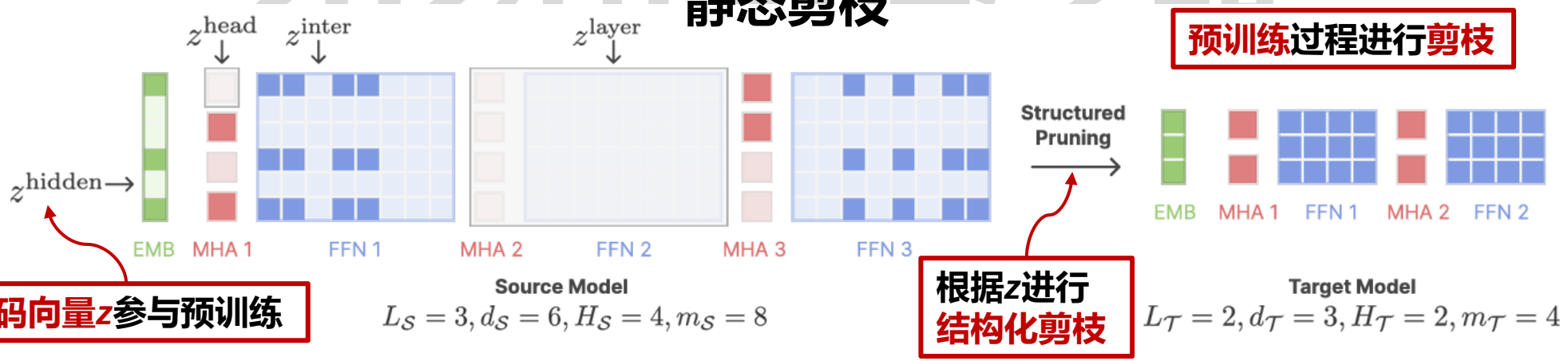
模型级加速：模型压缩

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

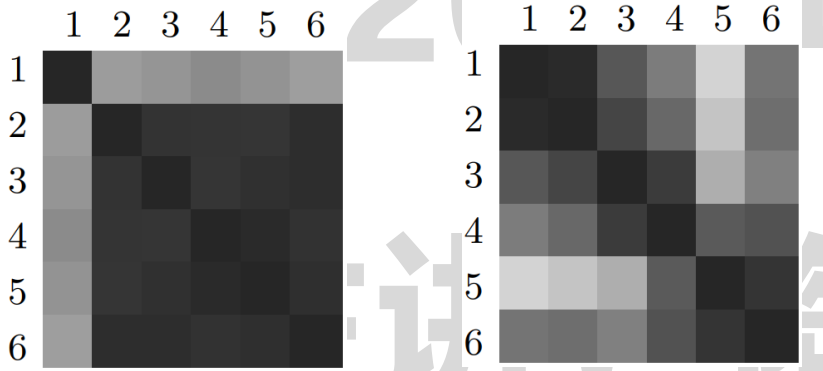
1、不同维度之间权重相似

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

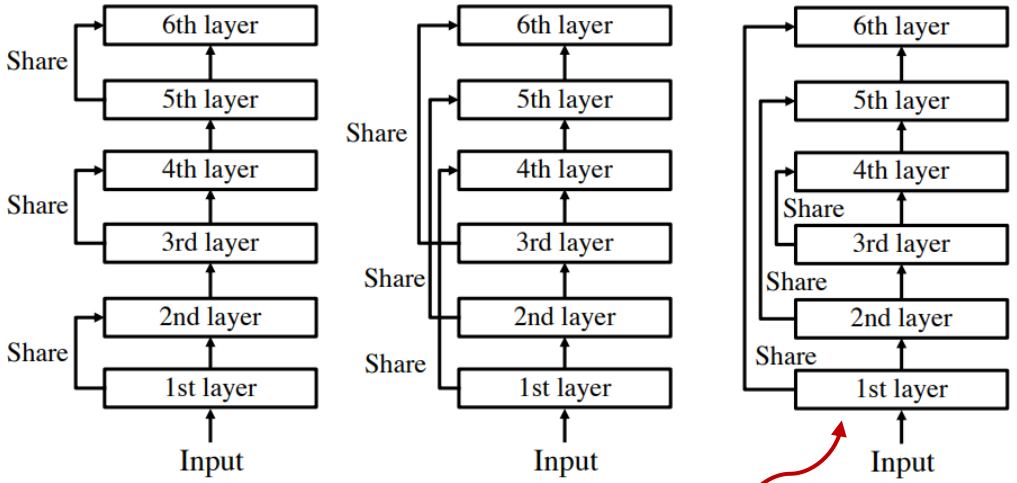


自注意力

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**

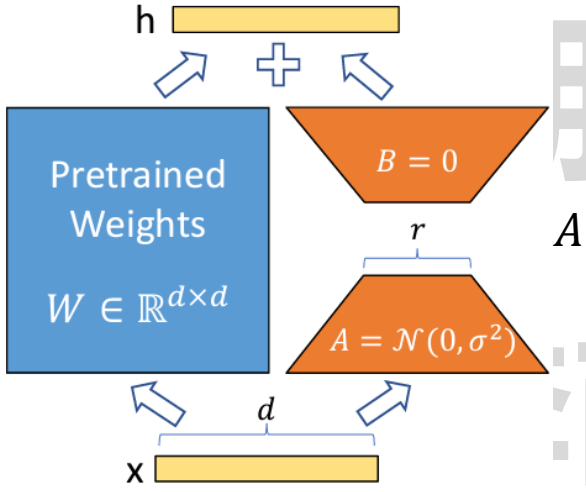
矩阵低秩分解

- 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

结构稀疏性
- **权重稀疏性**
- **输入稀疏性**

非结构稀疏性
- **注意力模块内稀疏性**

稀疏性处理方法	性能牺牲
矩阵压缩	掩码向量存储+稀疏矩阵处理单元(**稀疏性不高**时**面积**上升)
输入剪枝	**精度**损失+**尝试成本**(合适剪枝余量判断)
窗口注意力	**精度**损失(3个点以内)
渐进量化	**精度**损失(3个点以内)
注意力头剪枝	**精度**损失+**尝试成本**(合适剪枝余量判断)

存储降为30%左右

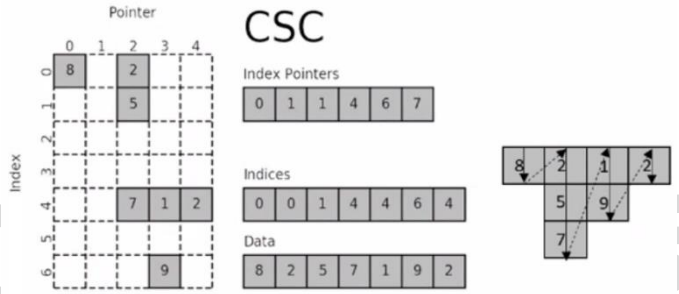
延时提升3倍左右

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

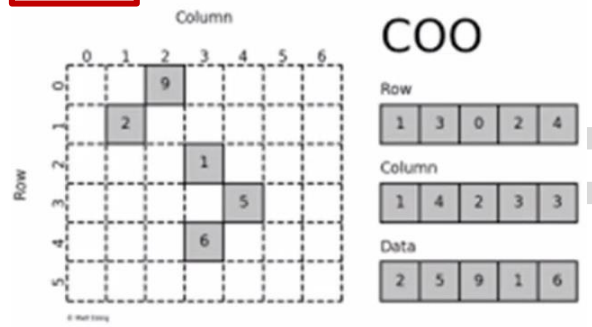
稀疏矩阵

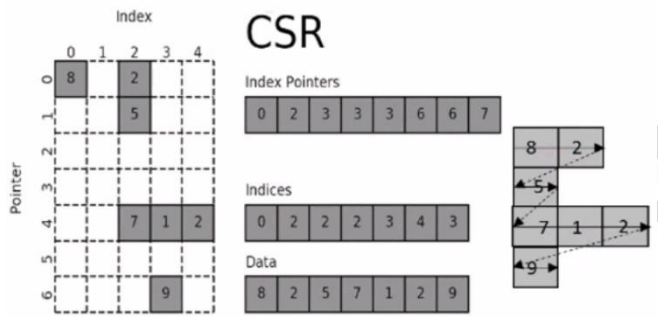
- **CSC**：按列压缩
- **CSR**：按行压缩
- **COO**：存储坐标



非零值的行的索引+行的起始位置指针

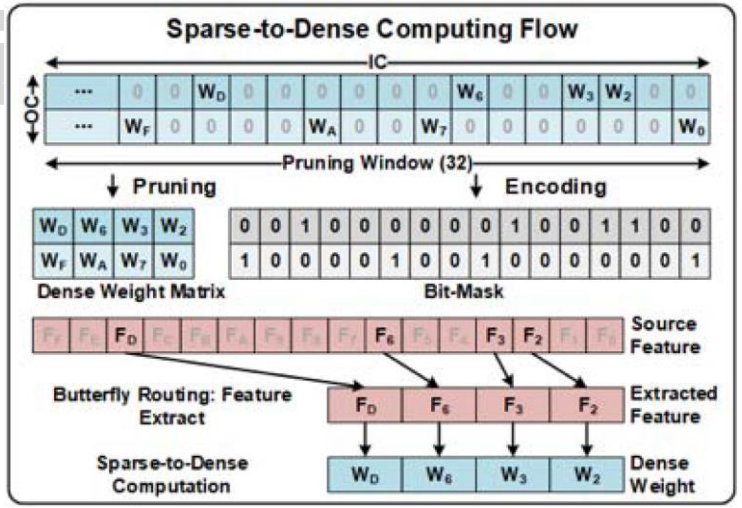
坐标





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**



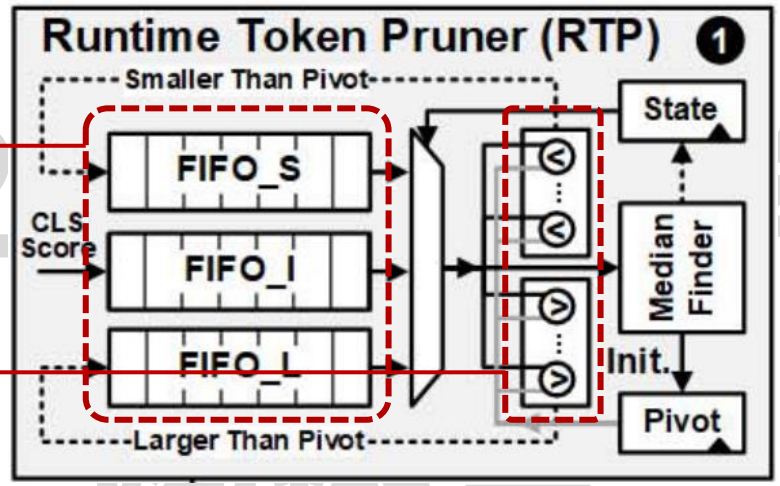
Liu, Shiwei, et al. *ISSCC* 2023

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

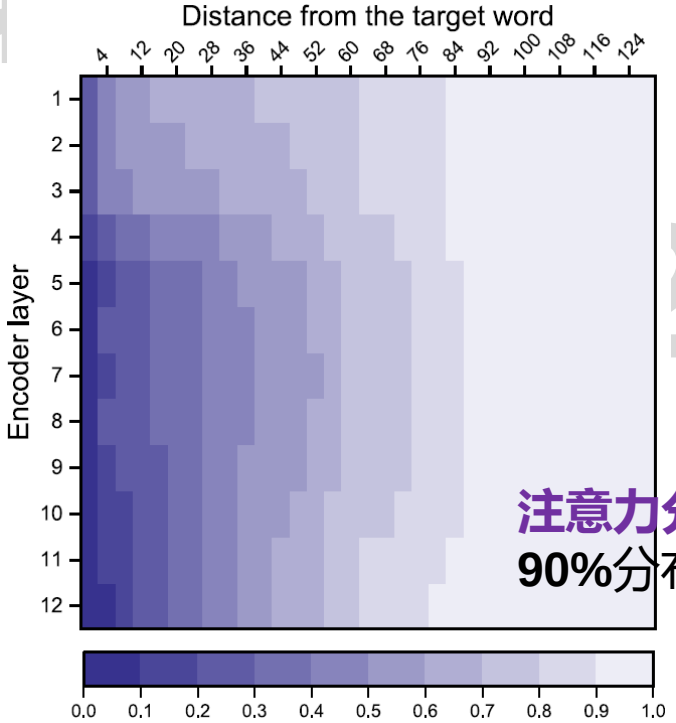
循环内容
- **初始化**：
  - 阈值为[CLS]
  - 所有token放入FIFO_I
  - 选定FIFO_I
- **重复**：
  - 阈值比较并放入FIFO
- **循环判断**：
  - 满足要求则输出
  - 不满则选定FIFO_S/L
  - 阈值为其中随机一个token

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**

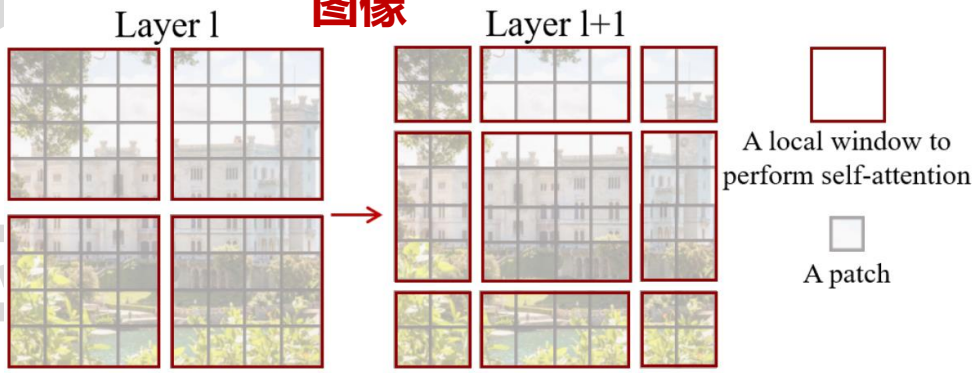
窗口注意力



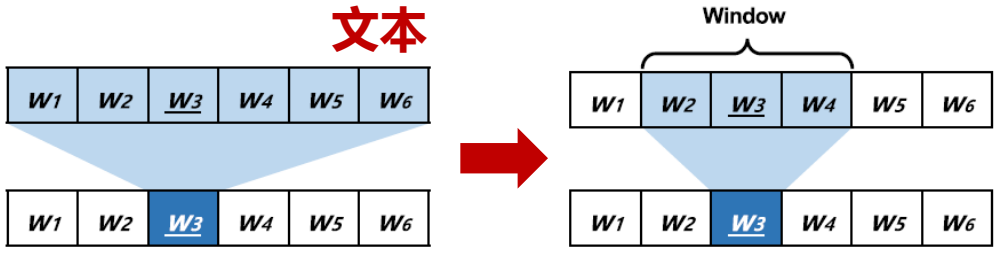
- 按注意力模块更关注目标词**附近**
- 目标词可以只和**附近token**进行注意力计算

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

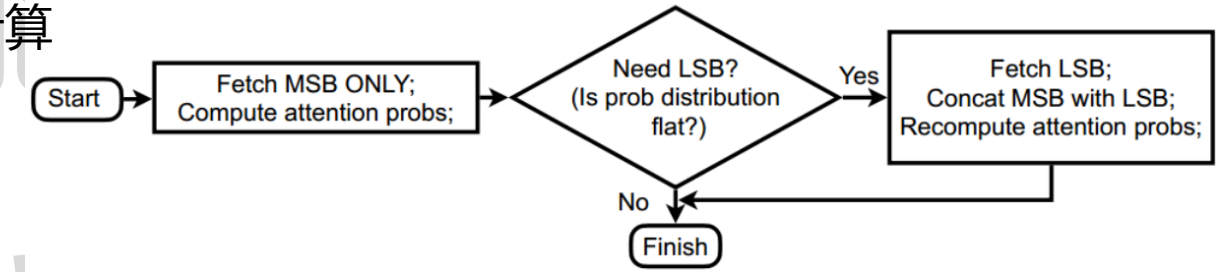
Kang, Myeonggu, Hyein Shin, and Lee-Sup Kim, *TCAD* 2021

模型级加速：稀疏性处理

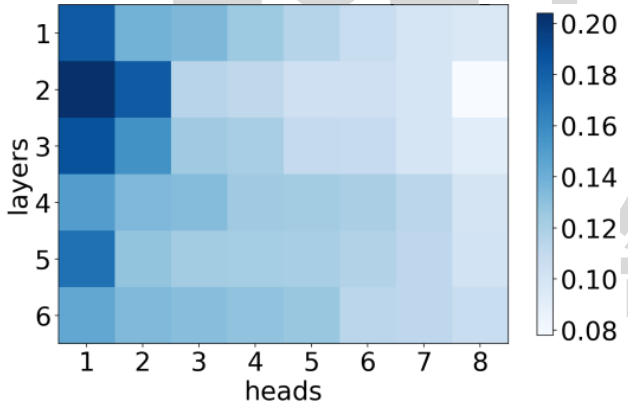
- **注意力模块内稀疏性**
  - **利用注意力机制更关注于关键词的特征**

渐进量化

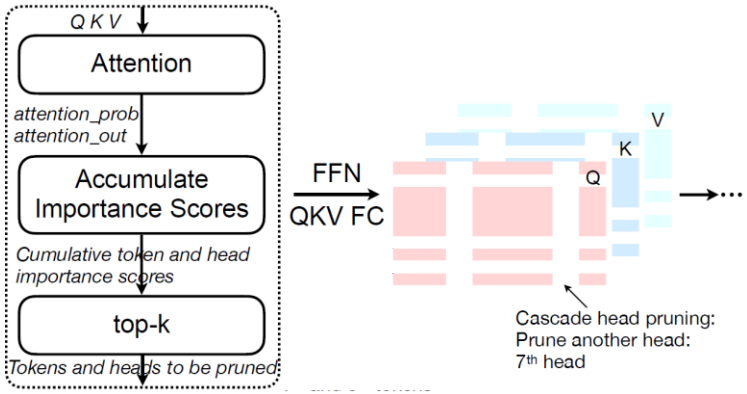
- 按**位**进行计算，而不是按**token**计算
- 每次计算**输入/权重**的一个比特
- 判断输出是否**平整**
- 不平整——已经实现注意力功能



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



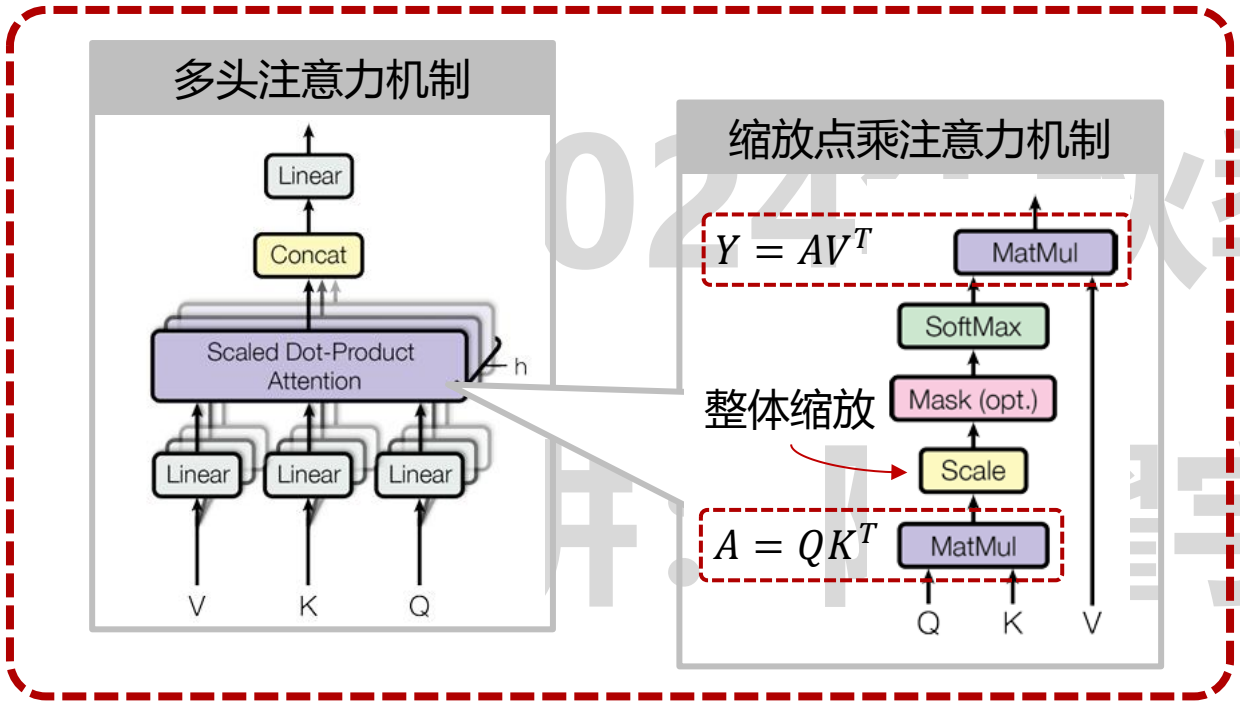
Voita, Elena, et al. *ACL* 2019

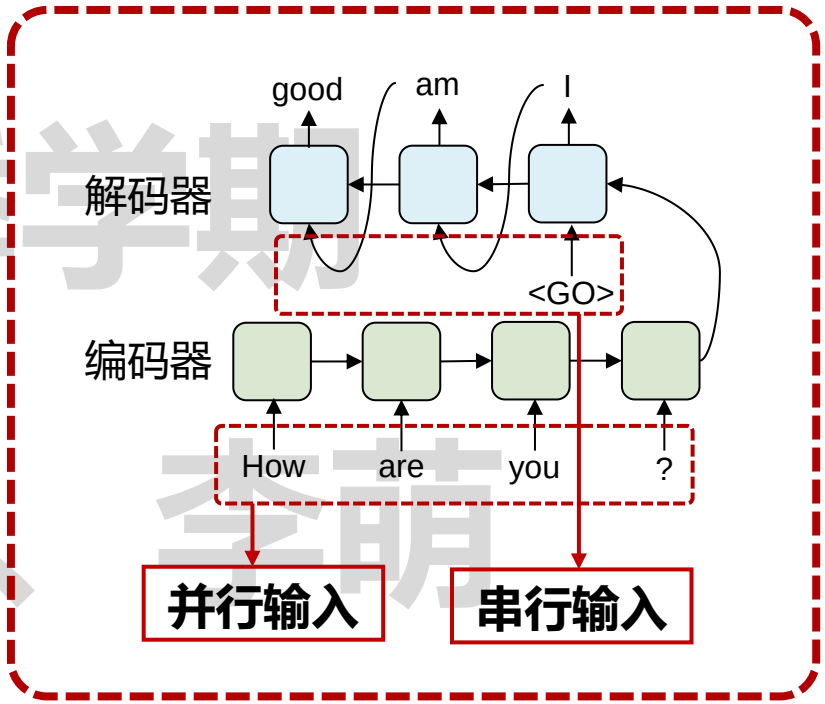
Wang, Hanrui, Zhekai Zhang, and Song Han. *HPCA* 2021

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

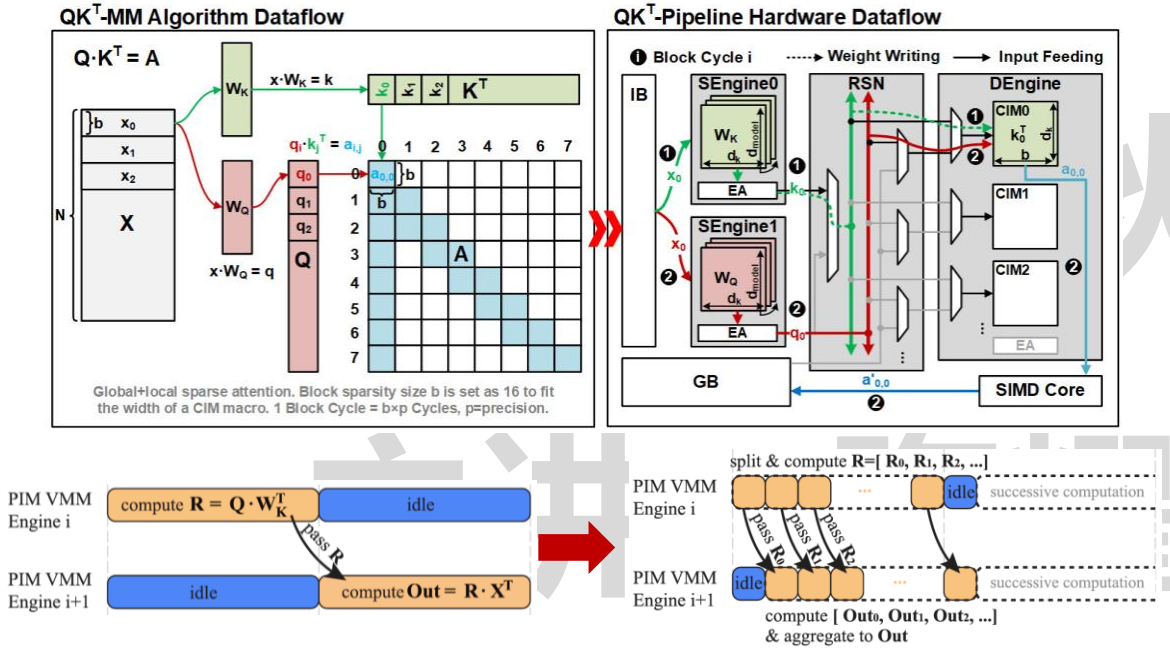




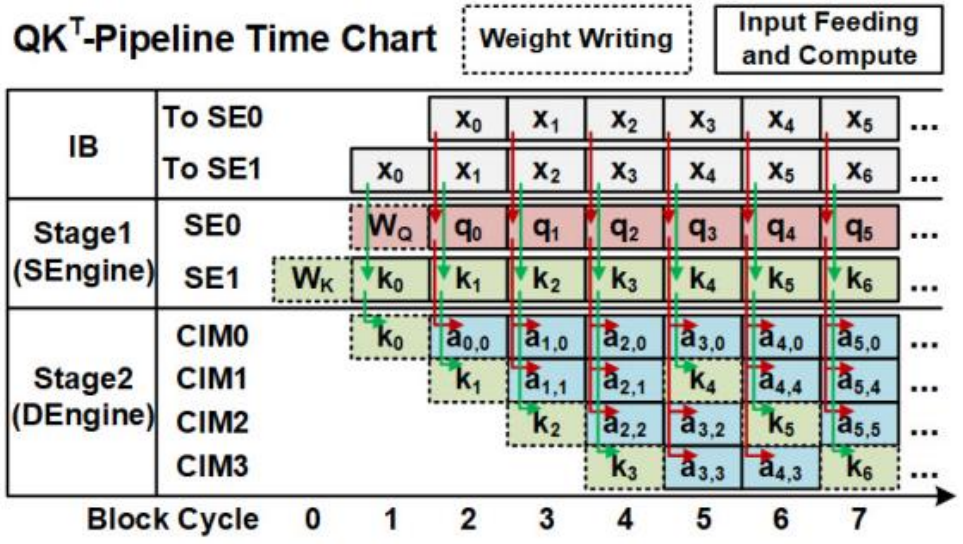
模块级加速：注意力模块加速

- **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



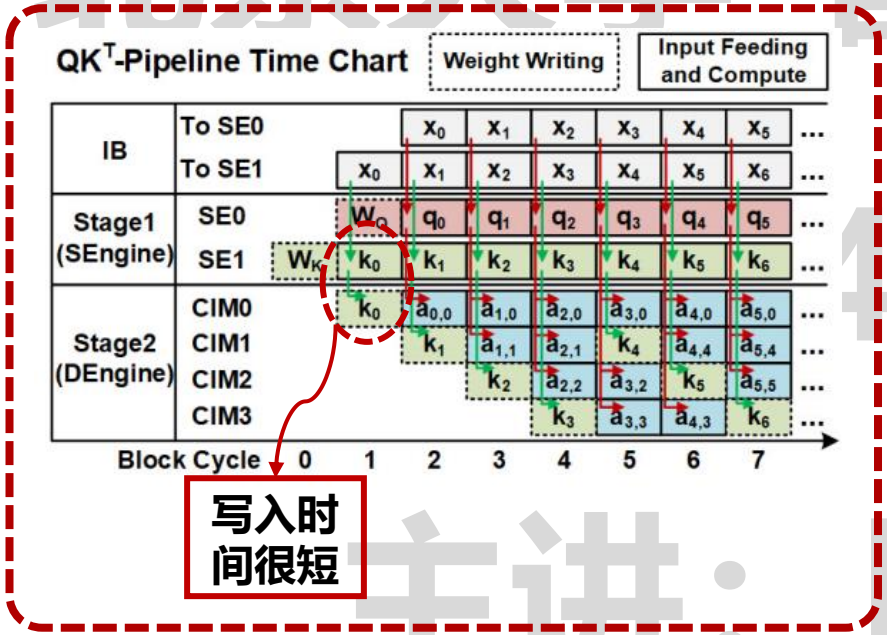
流水设计

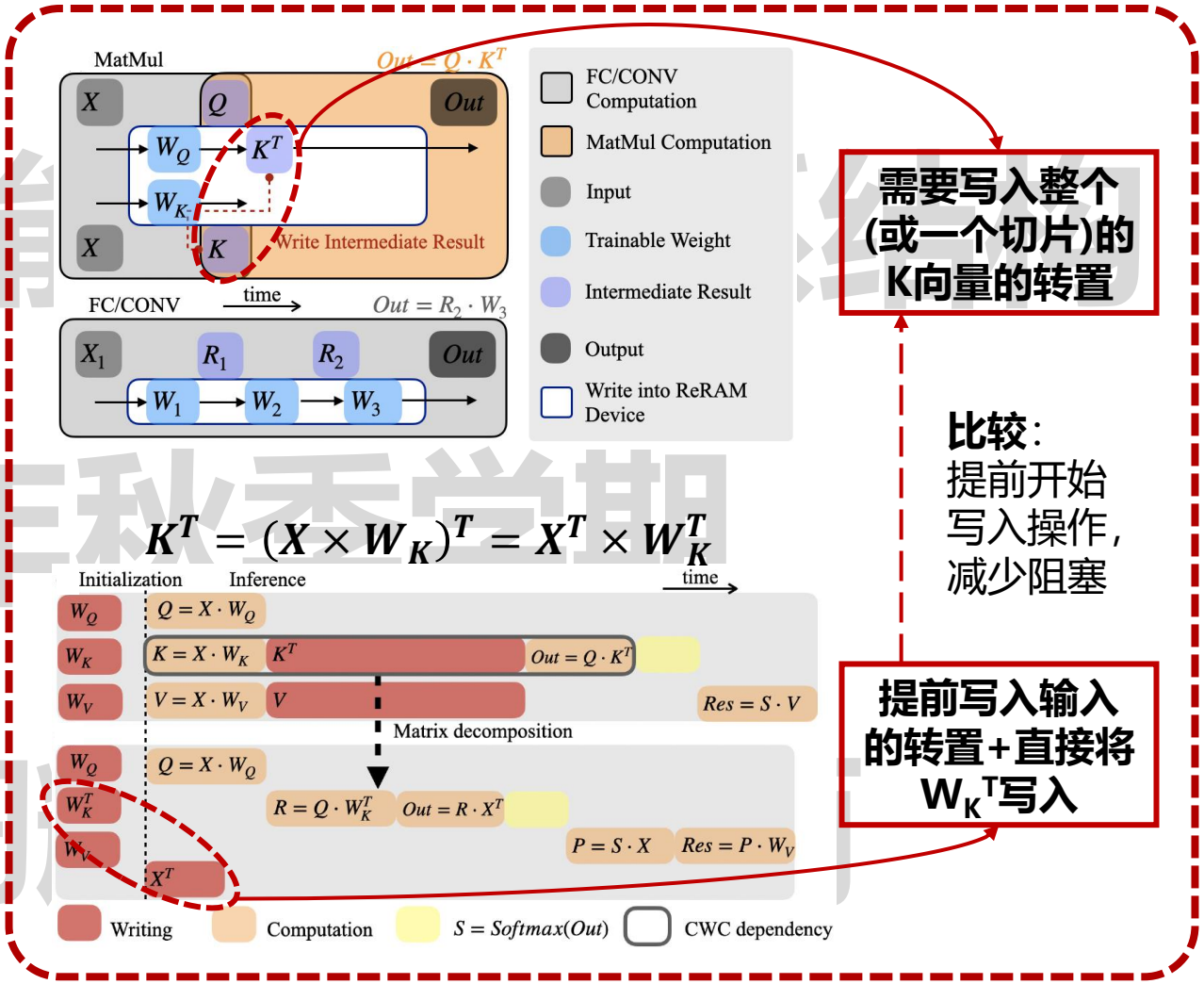


Tu, Fengbin, et al. *ISSCC* 2022　　Yang, Xiaoxuan, et al. *ICCAD* 2020

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

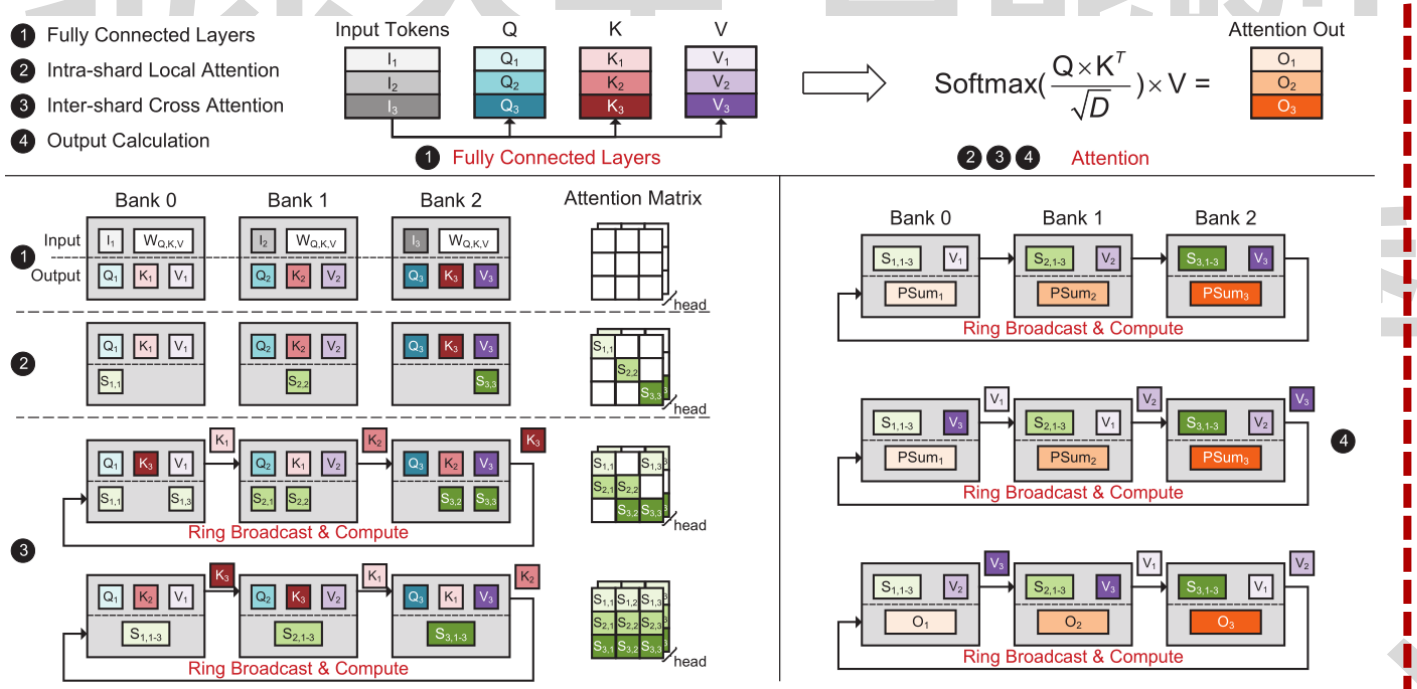




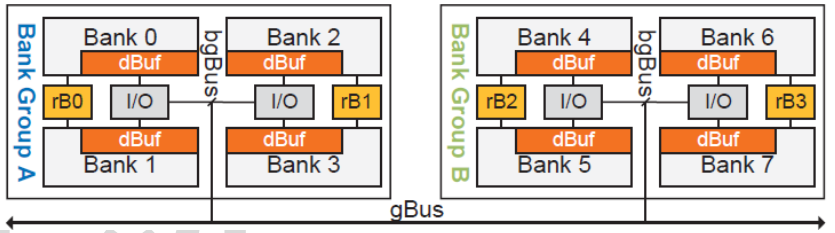
Yang, Xiaoxuan, et al. *ICCAD* 2020

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*

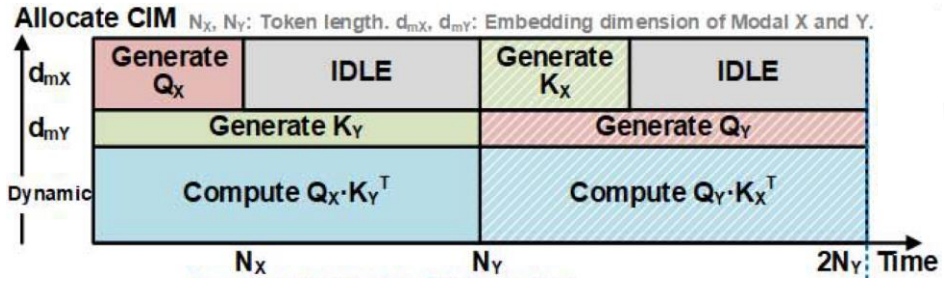
Zhou, Minxuan, et al. *HPCA* 2022

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

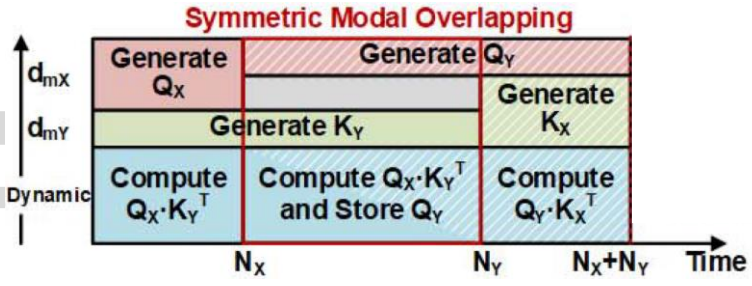
多模态输入的问题

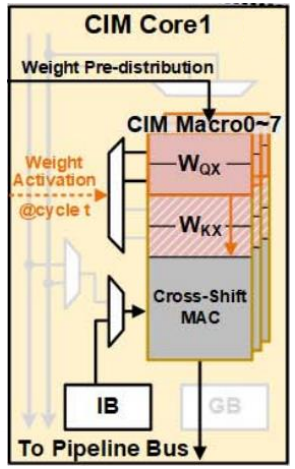
- **传统权重分布**：不同步骤在不同计算核心实现
- **多模态输入**：不同阶段运算延时不同
- 导致多模态输入不匹配时产生**额外延时**



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





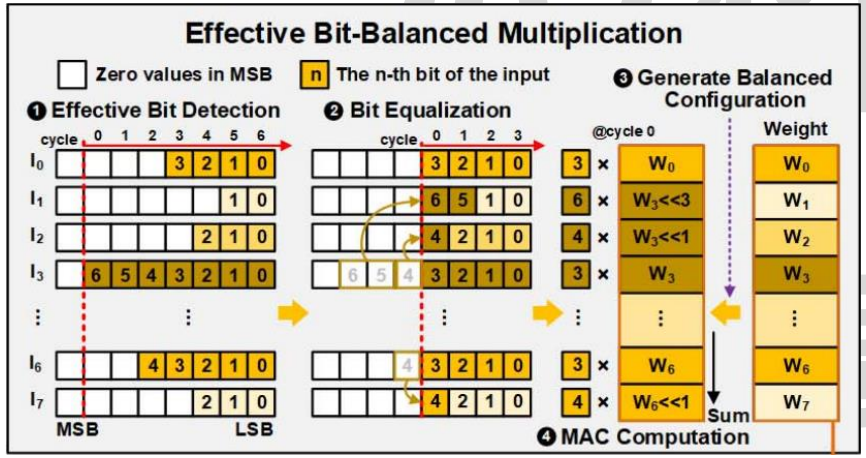
Tu, Fengbin, et al. *ISSCC* 2023

目录

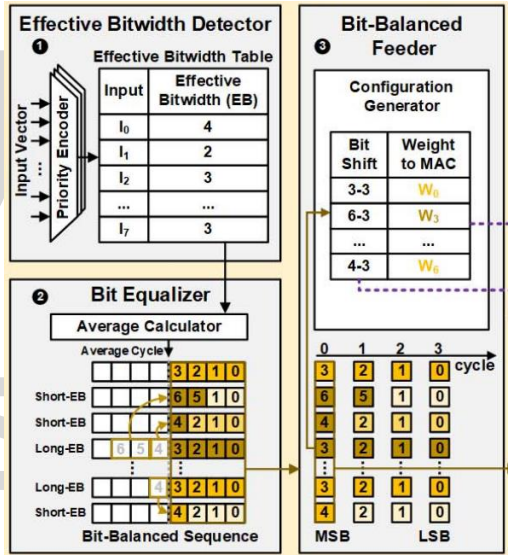
模块级加速：前馈神经网络加速

- **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

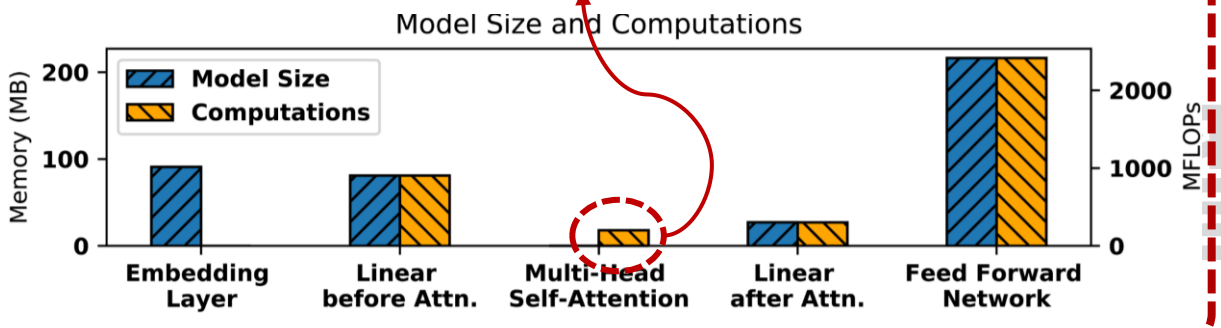
目录

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

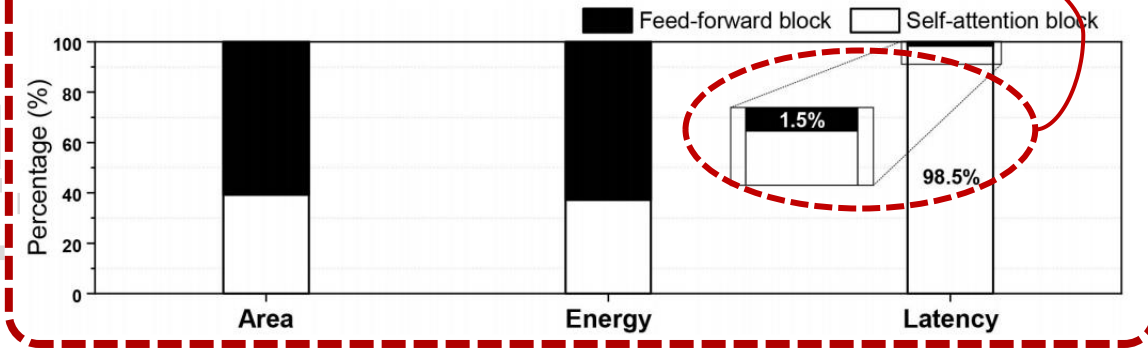
MatMul算子仅占**0.5%**左右



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



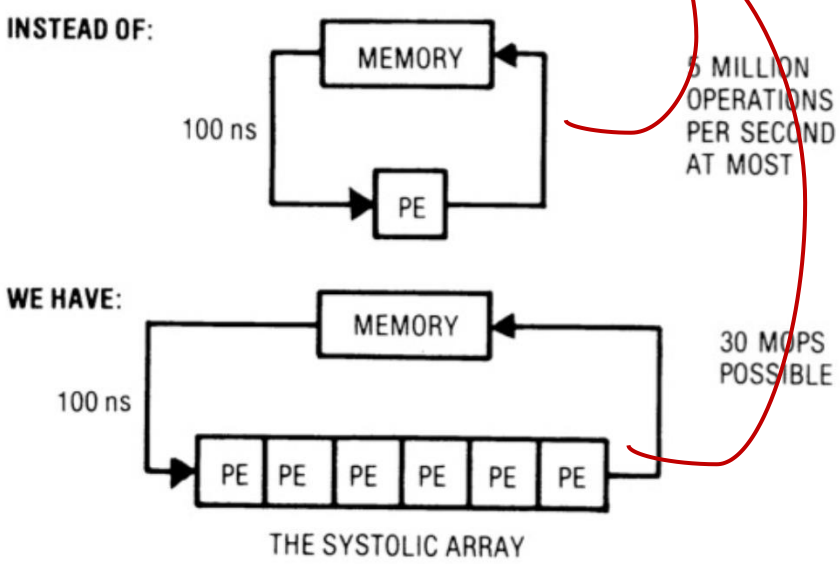
Kang, Myeonggu, Hyein Shin, and Lee-Sup Kim. *TCAD* 2021

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

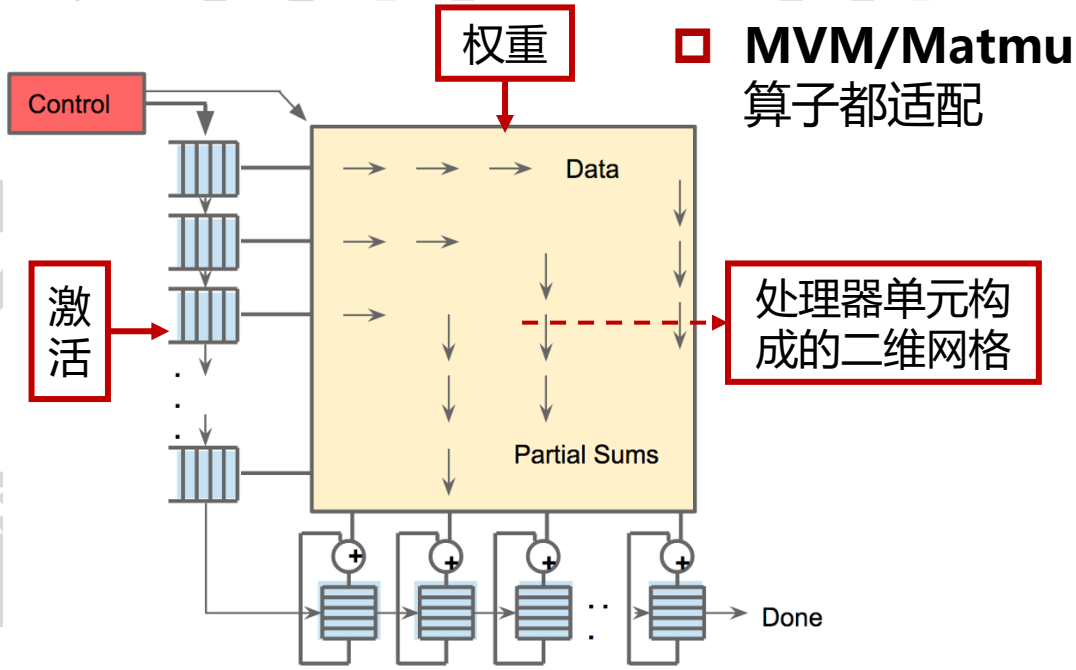
脉动阵列概念

减少存储器**调度**时间



Kung, Hsiang-Tsung. *Computer* 1982

脉动阵列应用

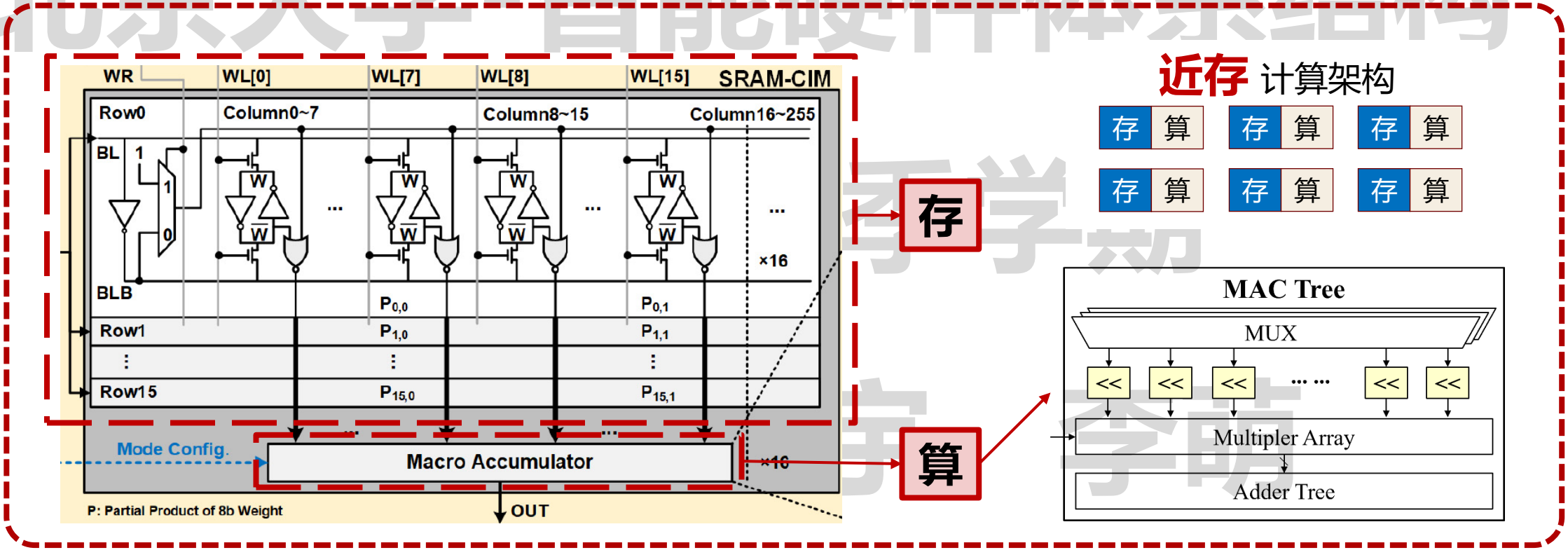


- **MVM/Matmul**
  算子都适配

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

算子级加速：线性算子加速

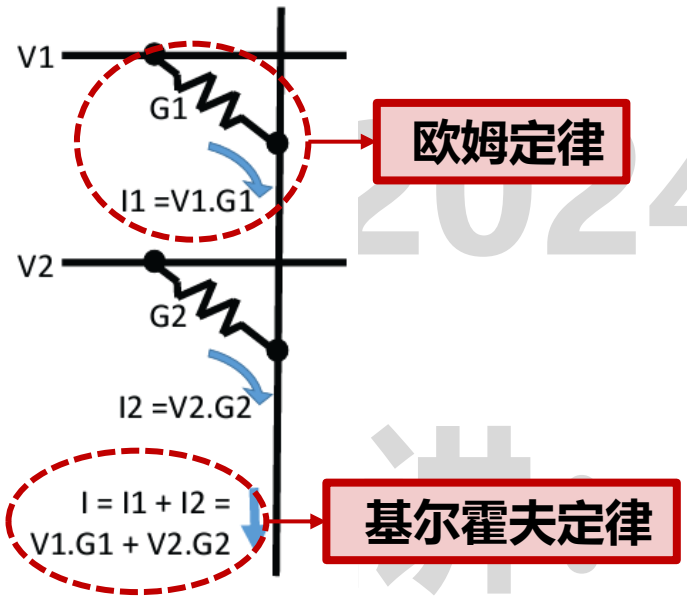
- **近存计算**
  - 存储阵列输出处增加逻辑单元，**减少传输**；DRAM、SRAM等存储介质
  - 乘法累加树——写入时间较长时影响MatMul算子实现



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

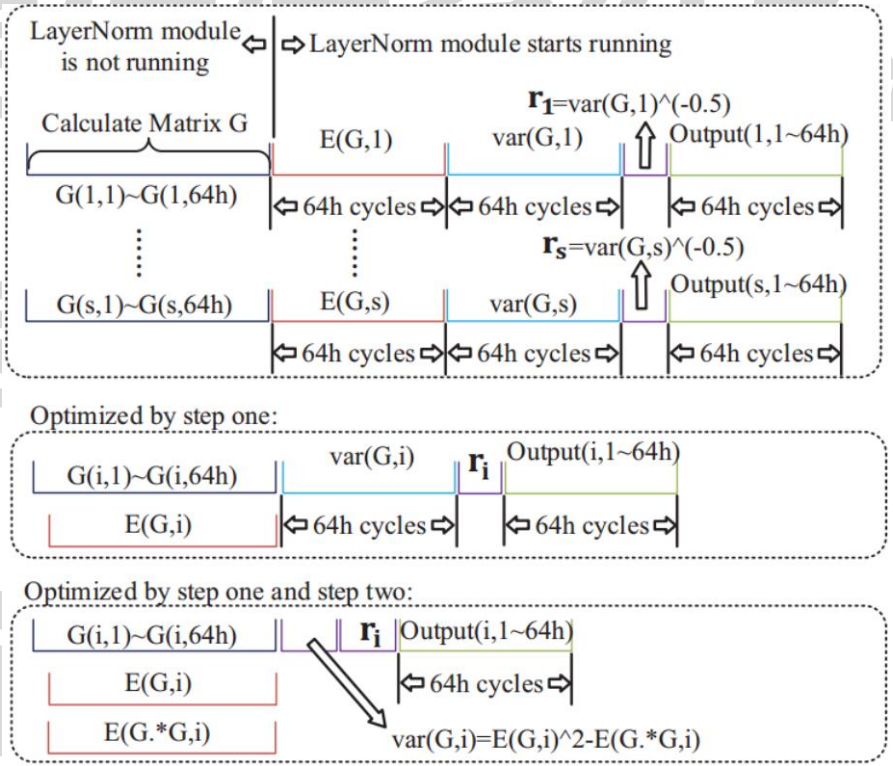
- **非线性算子加速：输入特征统计**
  - 均值统计/方差统计/指数统计

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

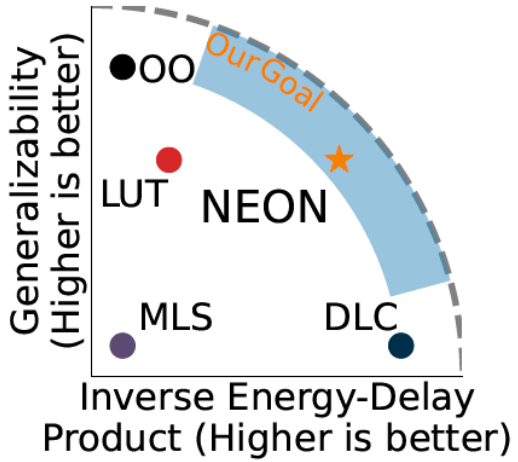
非线性函数加速

直接实现非线性函数

- **LUT**: 查找表
- **DLC**: 数字逻辑电路实现
- **MLS**: 忆阻器等新器件实现逻辑
- **OO**: 操作卸载到片外完成

将非线性函数变为其他函数

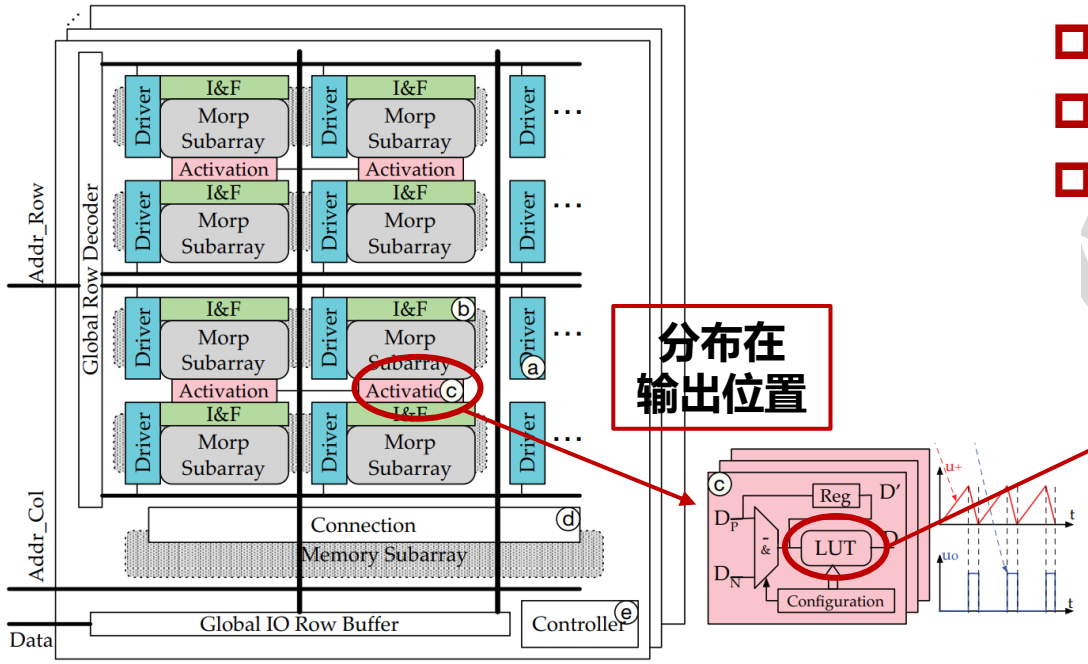
- **神经网络**实现: NEON-Net等
- **数学表达式**近似: I-BERT等



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

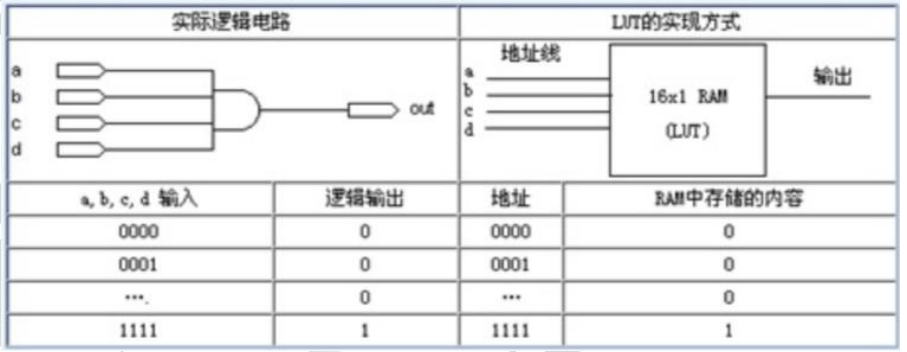
算子级加速：非线性算子加速

- **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**

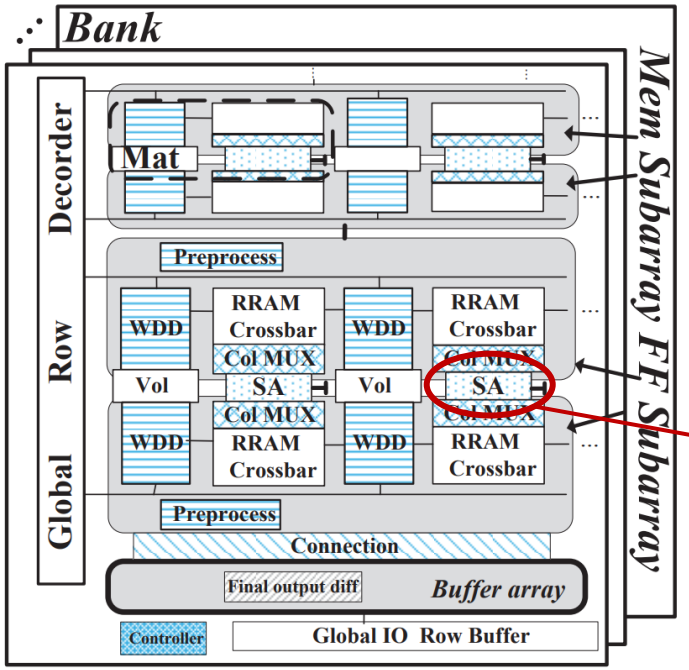


a, b, c, d 输入	逻辑输出	地址	RAM中存储的内容
0000	0	0000	0
0001	0	0001	0
...	0	...	0
1111	1	1111	1

Song, Linghao, et al. *HPCA* 2017

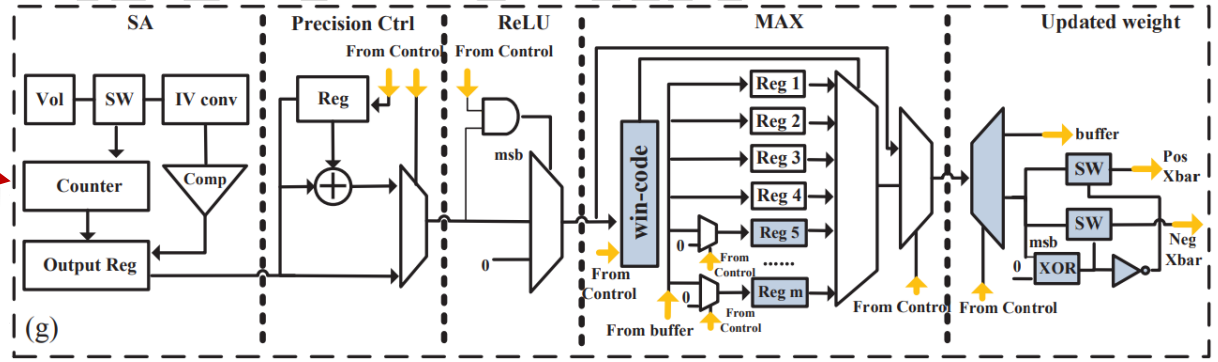
算子级加速：非线性算子加速

- **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



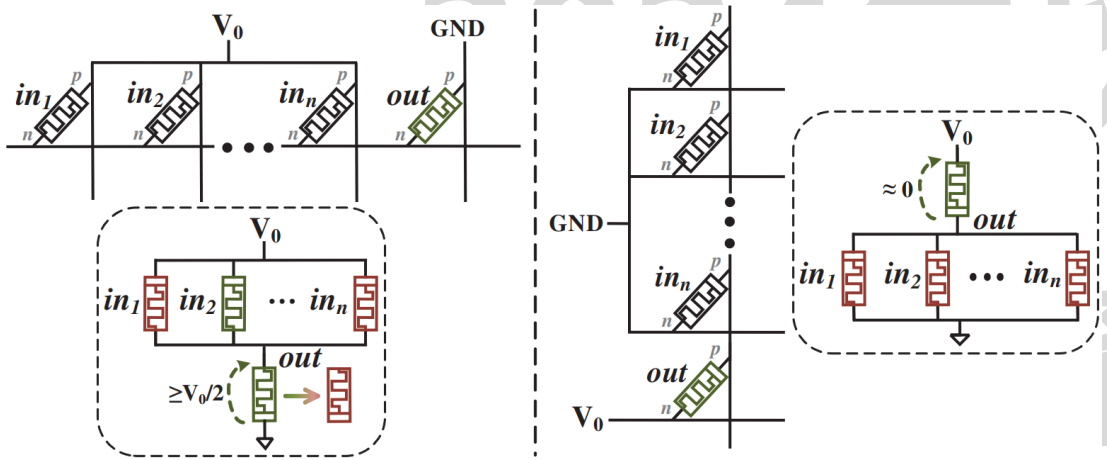
Cheng, Ming, et al. *DAC* 2017

算子级加速：非线性算子加速

- **非线性函数实现：MLS&OO**

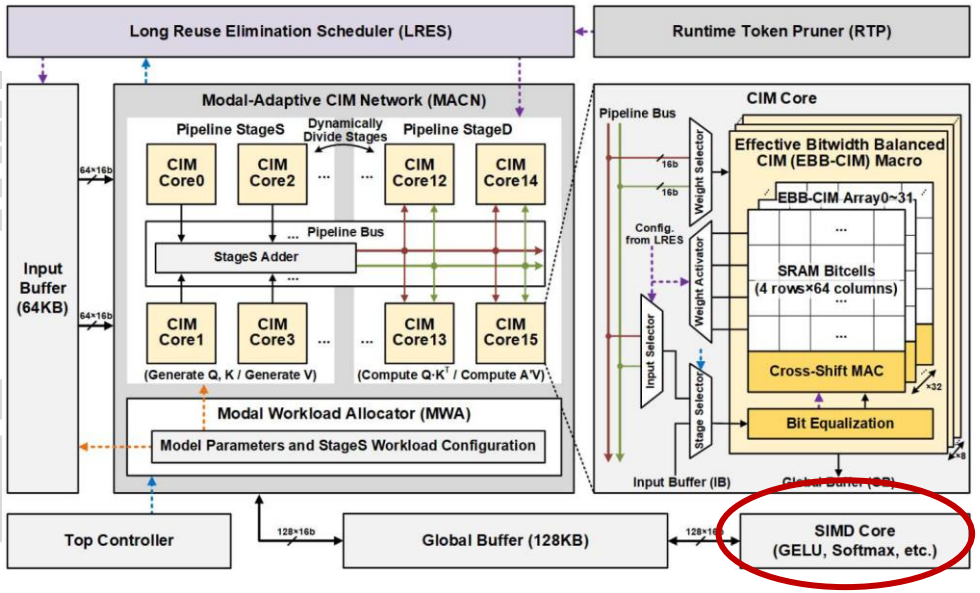
MLS(新器件实现逻辑)

- 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



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

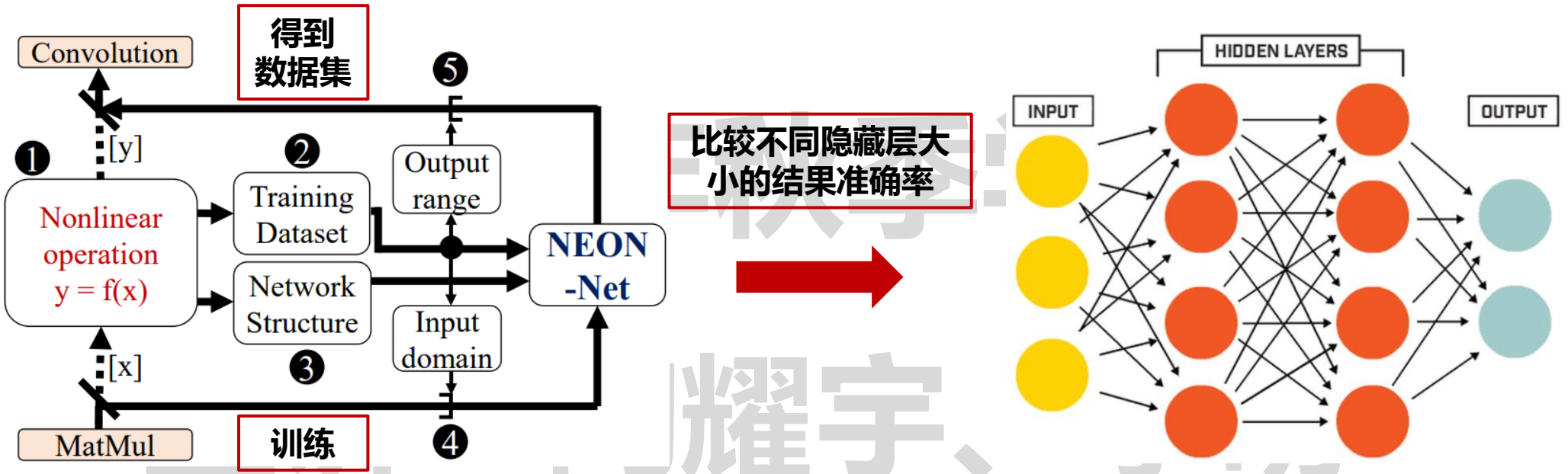


Gupta, Saransh, Mohsen Imani, and Tajana Rosing. *ICCAD* 2018

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

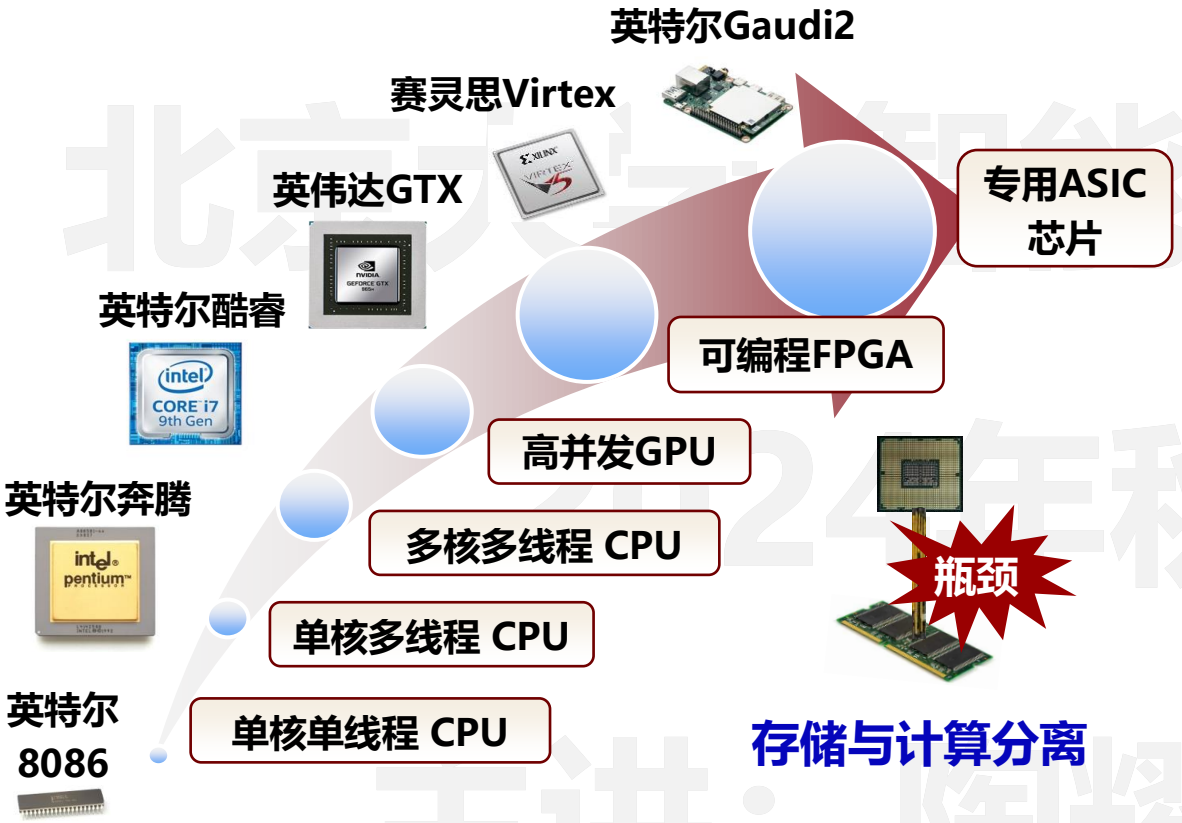
总结及展望

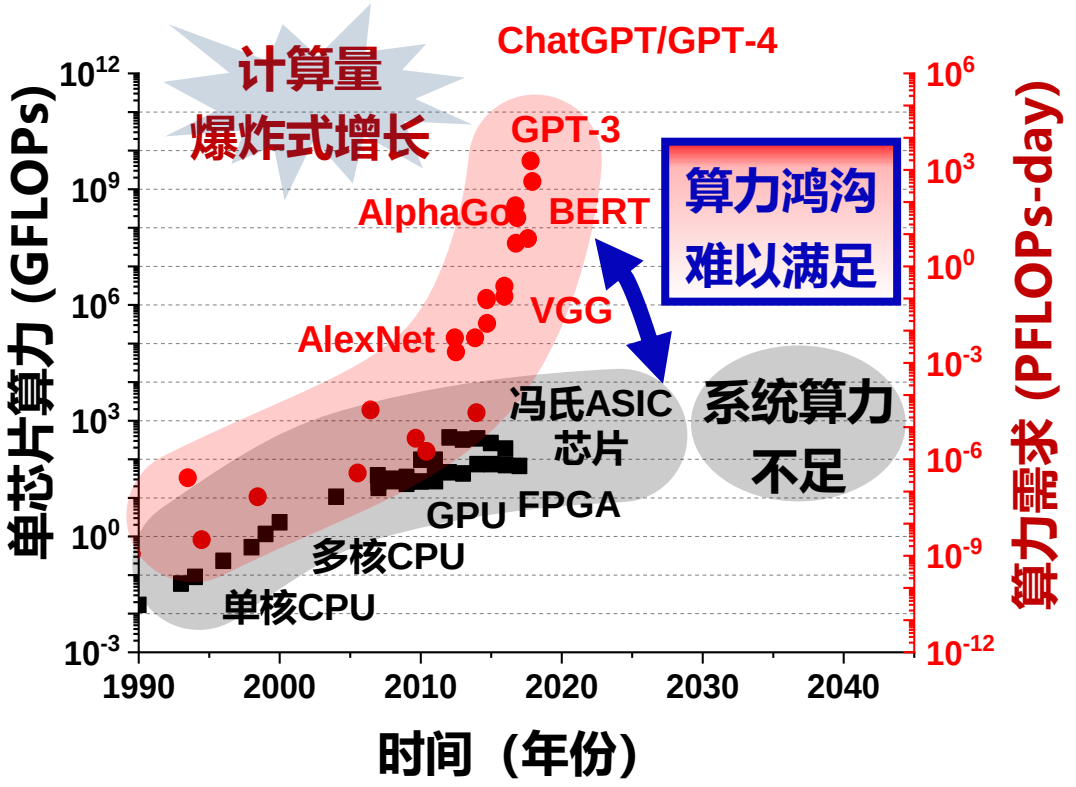
目 录

CONTENTS

当前半导体芯片产业发展的关键瓶颈

- 以冯诺依曼架构芯片为基石的传统算力系统难以满足爆炸式的算力增长需求



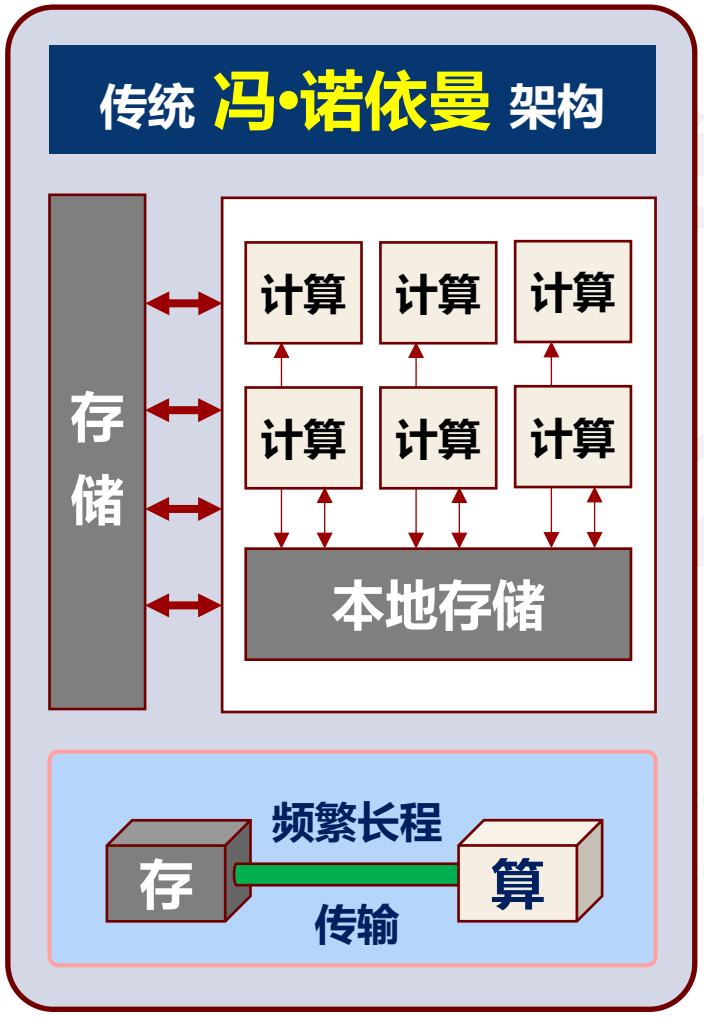


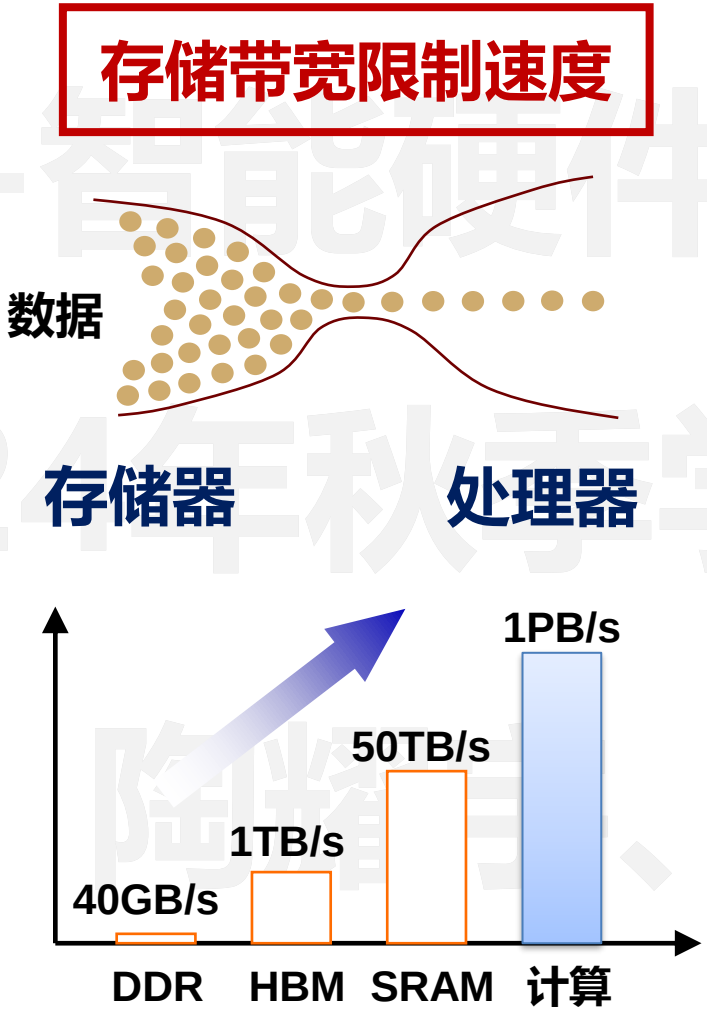
摩尔定律时代的冯诺依曼架构芯片演进图

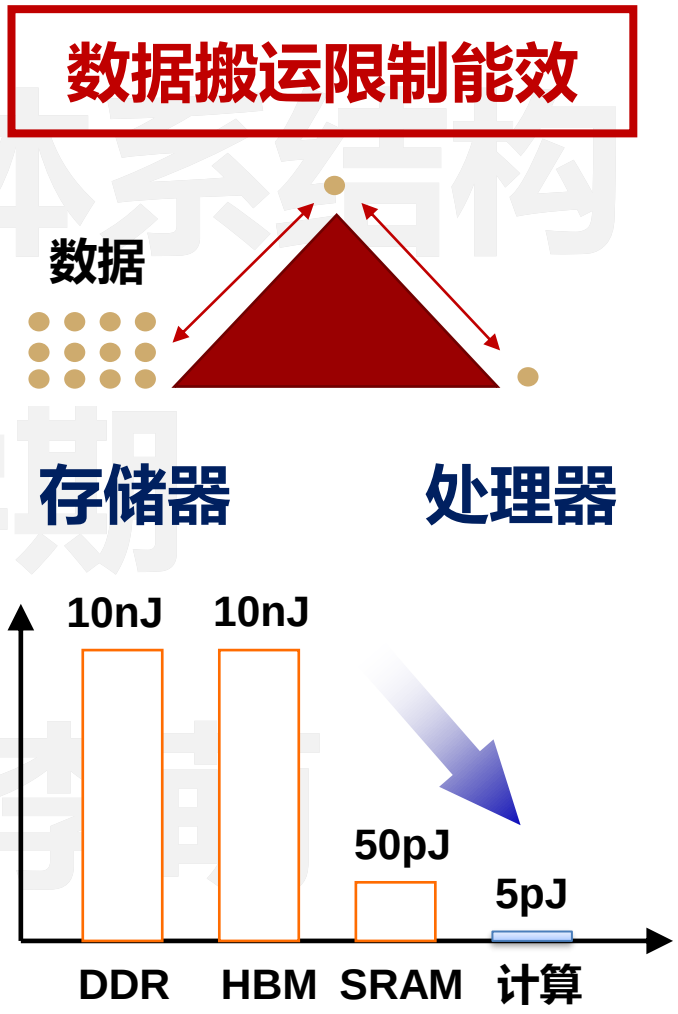
新兴计算任务所需的计算量呈爆炸式增长

当前半导体芯片产业发展的关键瓶颈

- 传统冯式架构的计算芯片难以突破大算力和高能效的性能瓶颈

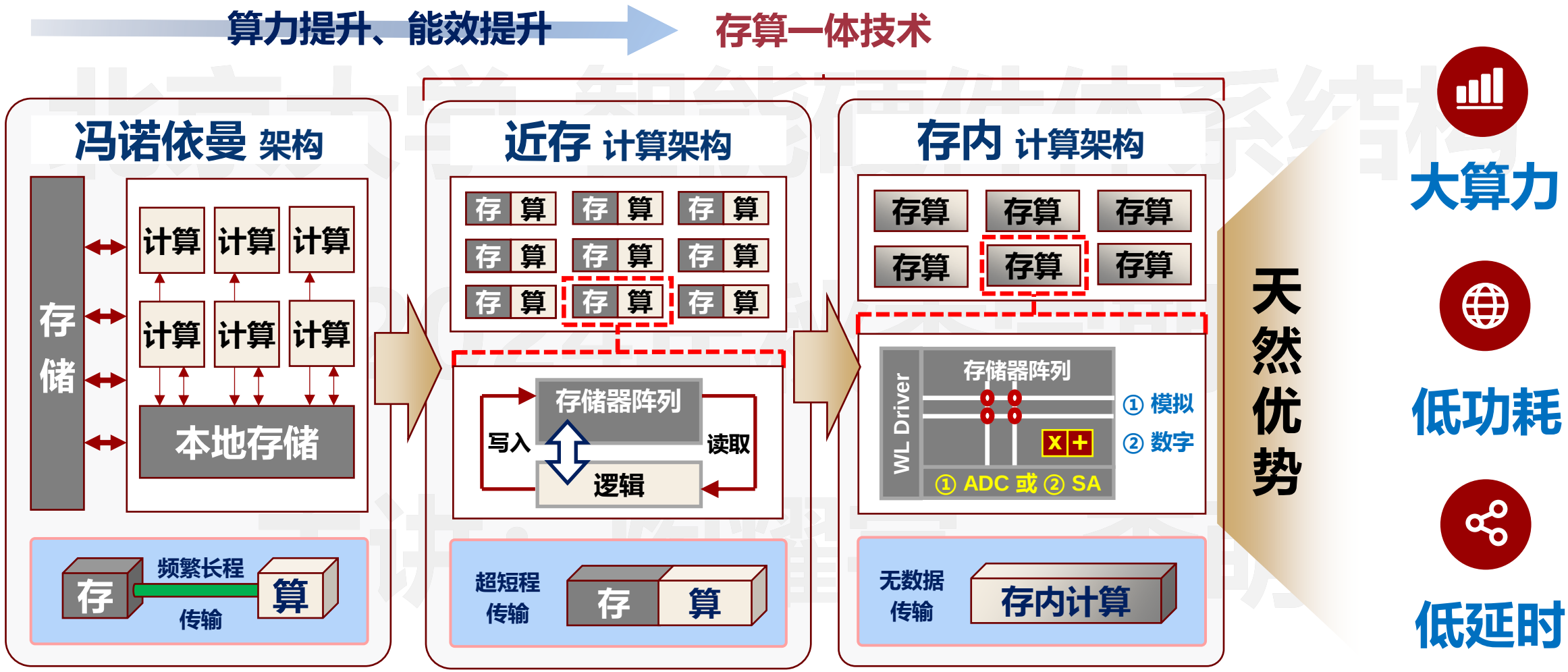






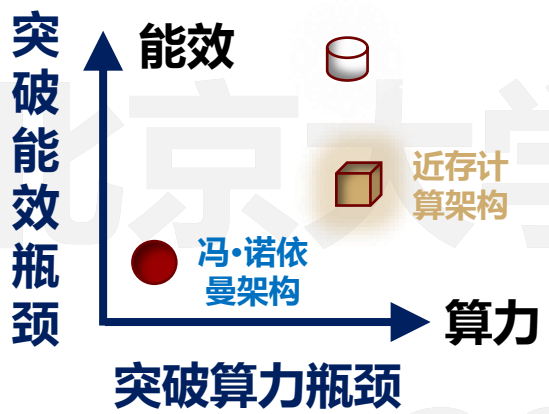
代表性半导体芯片前沿技术：存算一体

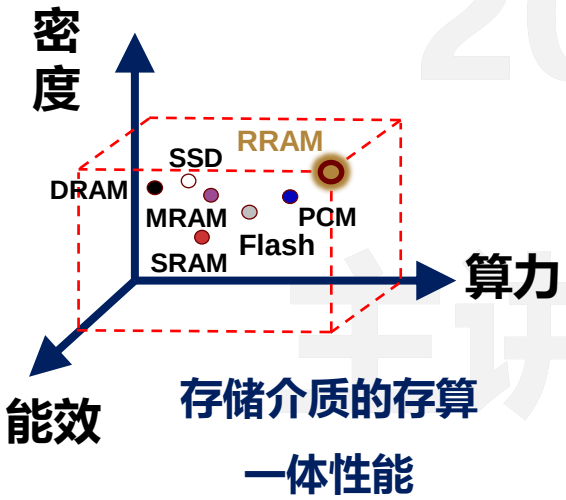
- 存算一体技术成为后摩尔时代打破算力瓶颈的重要路径



实现存算一体的多种存储介质

- ReRAM是最具发展潜力的新兴存储器之一，非常适合存算一体





存储器	主要优势	缺点与不足	非易失	适合场景
DRAM	工艺成熟 密度高	速度低/刷新/ 只能做近存	否	冯氏架构 过渡
SRAM	工艺成熟 IP化	能效低/密度低	否	端侧/边缘 中小算力
SSD	容量大 成本低	速度低/近存	是	云端大容量
MRAM	能效/速度/ 密度高	大规模集成难	是	端侧/边缘 中小算力
Flash	密度高/ 成本低	PVT变化敏感	是	端侧/边缘 低成本
ReRAM	**算力/能效/ 密度高**	**良率爬坡**	**是**	**端侧/边缘 大算力**

存算一体在国内外产业界和学术界的布局

- 存算一体技术受到国内外产业界和学术界的高度重视，已有诸多产业化布局

实现存算一体的多种技术路径

- 存算一体技术受到国内外产业界和学术界的高度重视，已有诸多产业化布局

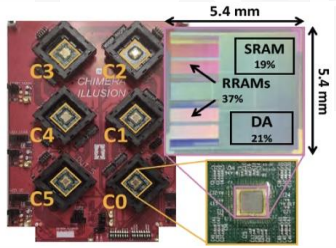
器件

计算架构

基于嵌入式存储的近存计算

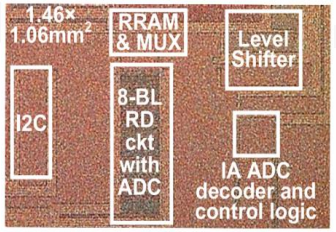
存内没有计算

多种嵌入式存储器（DRAM、SRAM等）



M. Giordano et al., VLSI 2021

基于新兴非易失存储器的存内计算



J.H. Yoon et al., JSSC 2022

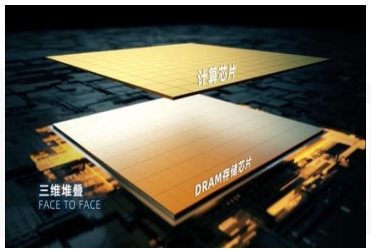
存内含计算

需新兴器件与电路架构的协同创新

基于先进封装的近存计算

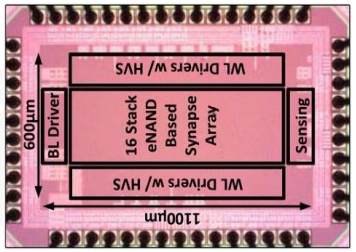
存内没有计算

多种先进封装（Chiplet、三维封装等）



D. Niu et al., ISSCC 2022

基于存储级存储器的存内计算



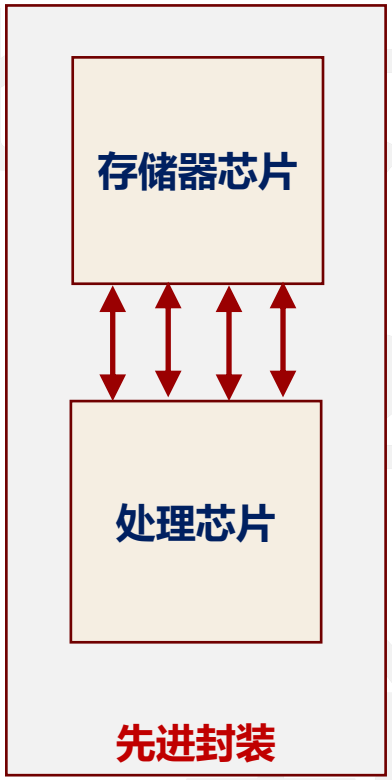
M. Kim et al., JSSC 2022

存内含计算

将存内计算应用于成熟的存储级存储器（如Flash）

路径一：先进封装实现近存计算

- 将存储器芯片与逻辑处理芯片通过多种先进封装方式拉进距离



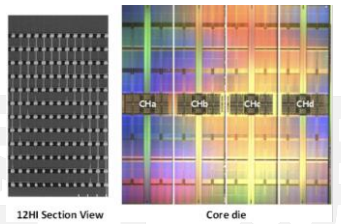
优势

与现有系统兼容性好
设计难度低

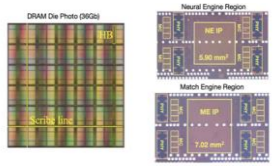
典型案例

AMD Zen CNM, SK-Hynix HBM-PIM, Alibaba CNM

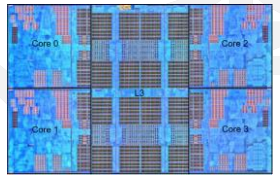
性能	先进封装近存计算
算力	高
能效	低
设计难度	低
可重构性	高
成本	低



SK-海力士
HBM-PIM



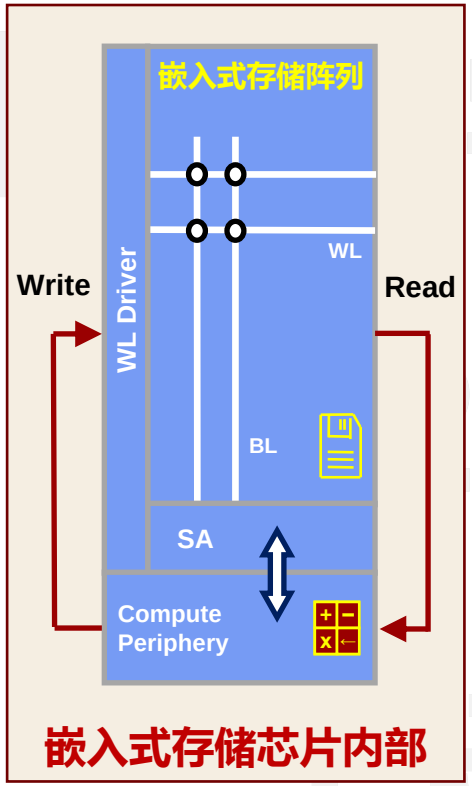
阿里巴巴达摩院3D
封装近存计算芯片



AMD Zen近存
计算样片

路径二：嵌入式存储实现近存计算

- 在嵌入式存储器芯片（DRAM/SRAM等）内部集成近存逻辑电路



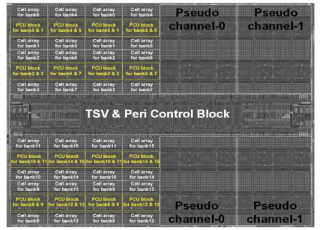
优势

数据搬运距离较先进封装更加缩短

典型案例

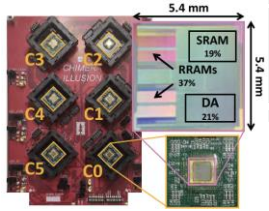
三星 HBM CNM,
GraphCore SRAM CNM,
Stanford RRAM CNM

性能	嵌入式存储近存
算力	中等
能效	中等
设计难度	中等
可重构性	中等
成本	中等



三星HBM CNM

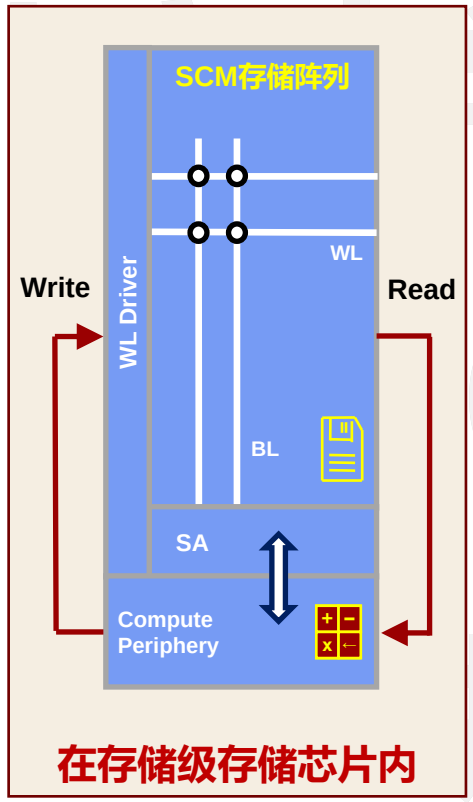
GraphCore SRAM CNM



Stanford RRAM CNM

路径二：存储级存储实现存内计算

- 利用成熟的存储级存储器（Flash等）进行存内计算

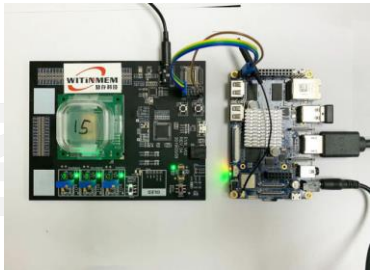


优势
工艺成熟、成本低

典型案例
Mythic Flash CNM, WitinMem Nor Flash CIM

性能	存储级存储器 存内计算
算力	中等
能效	低
设计难度	中等
可重构性	中等
成本	低

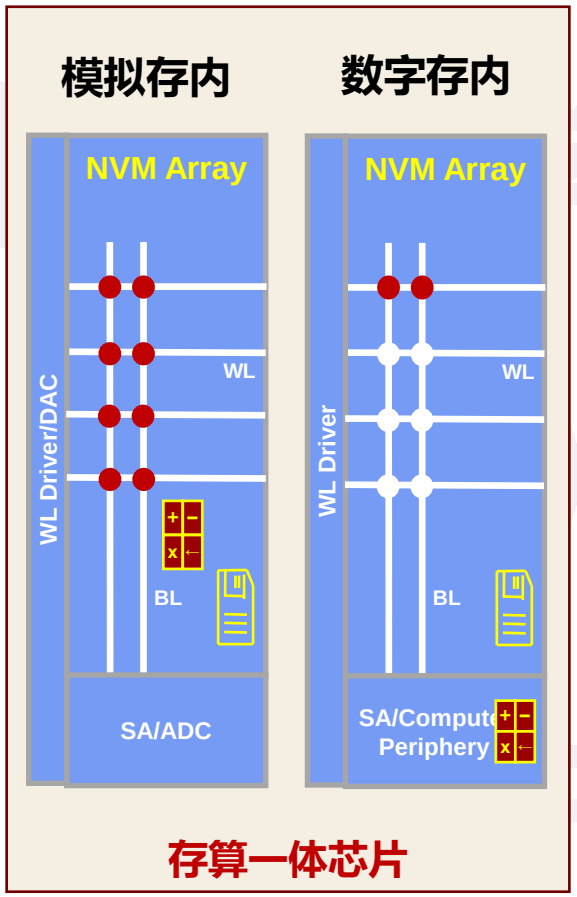
Mythic Flash CIM Chip



知存科技 Flash CIM

路径二：新兴非易失性器件实现存内计算

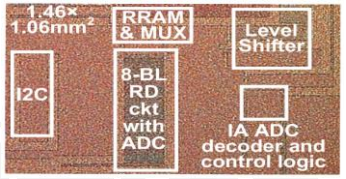
- 使用新兴的非易失性存储器（ReRAM等）进行存内计算



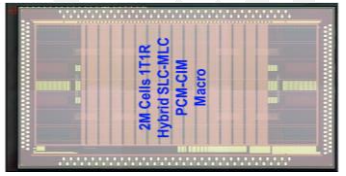
优势
大算力、高能效、高密度

典型案例
台积电ReRAM芯片、台积电PCM芯片、三星MRAM芯片

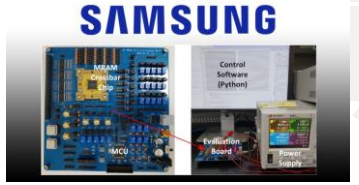
性能	新兴非易失性存储器存内计算
算力	高
能效	高
设计难度	高
可重构性	中等
成本	高



TSMC RRAM CIM



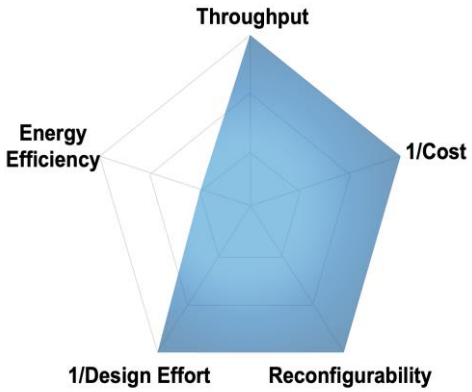
TSMC PCM CIM



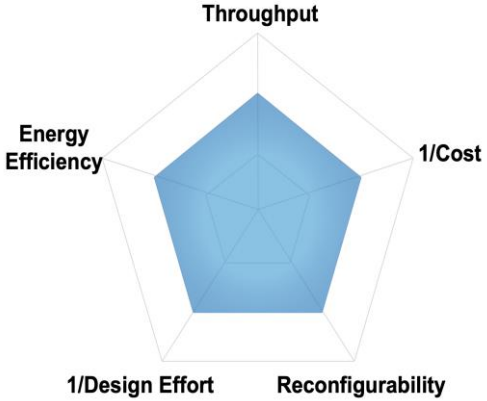
三星 MRAM CIM

存算一体多种技术路径的比较

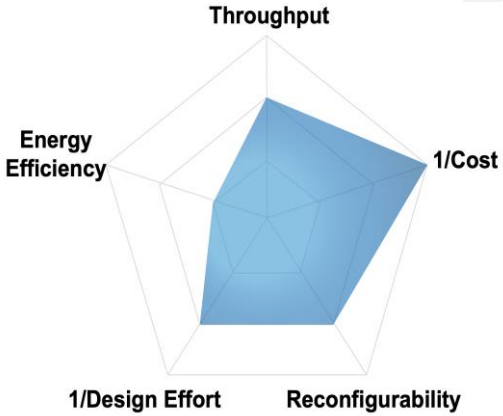
- 存算一体技术针对不同应用场景可以采用不同的技术路径



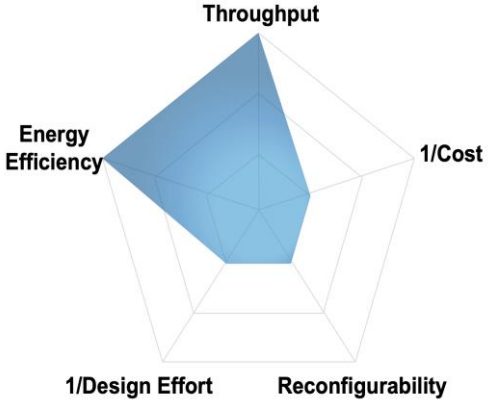
先进封装近存 CNM-3D



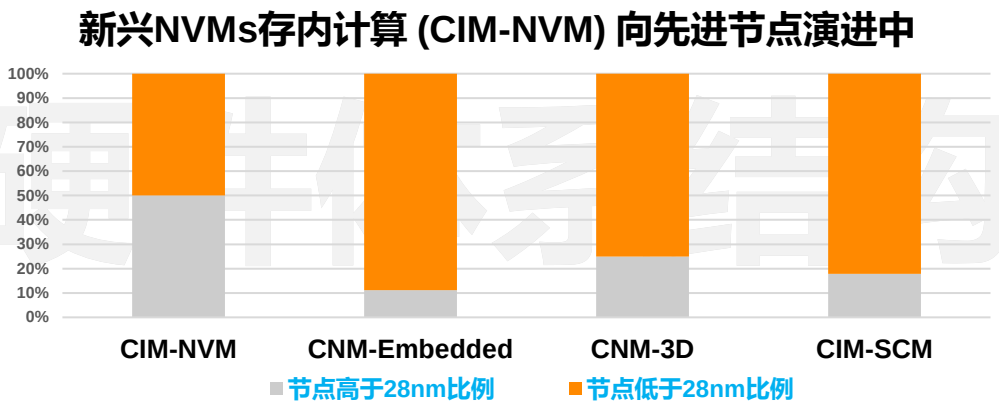
嵌入式近存 CNM-Emb.

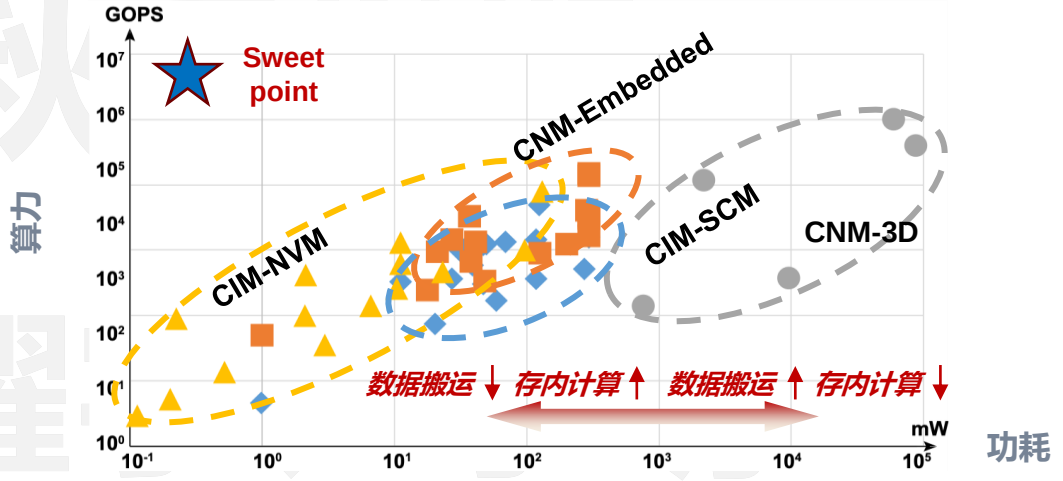


SCM近存/存内 CIM-SCM



新兴NVMs存内 CIM-NVM

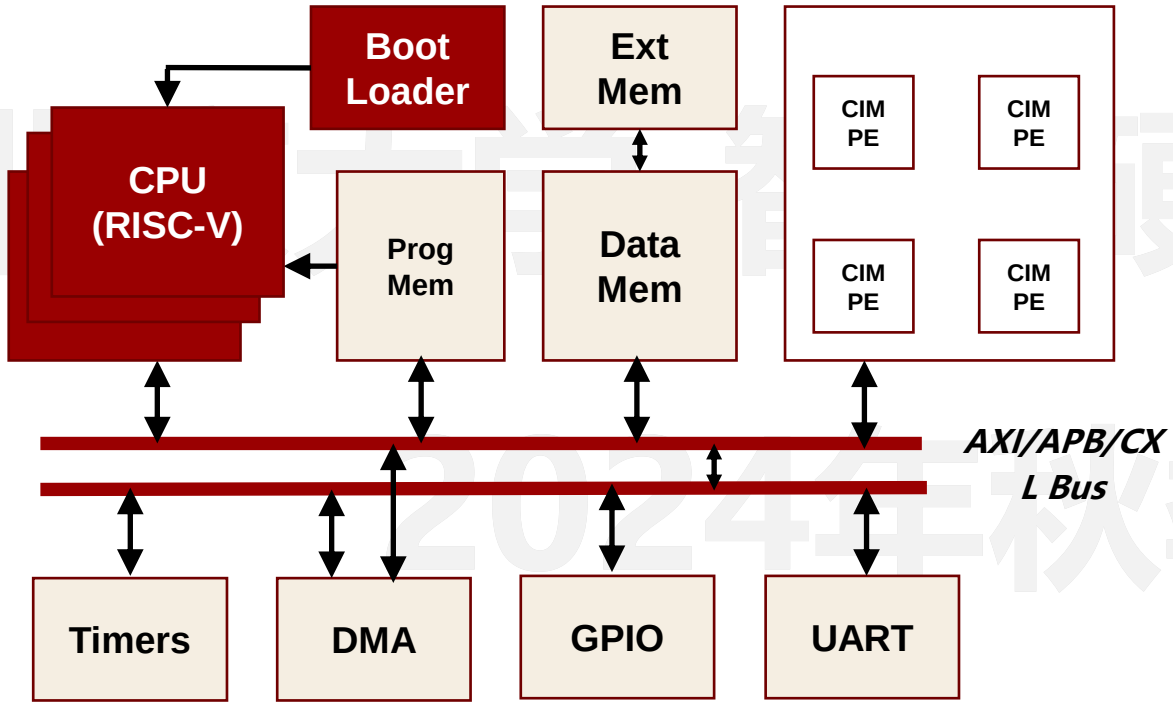


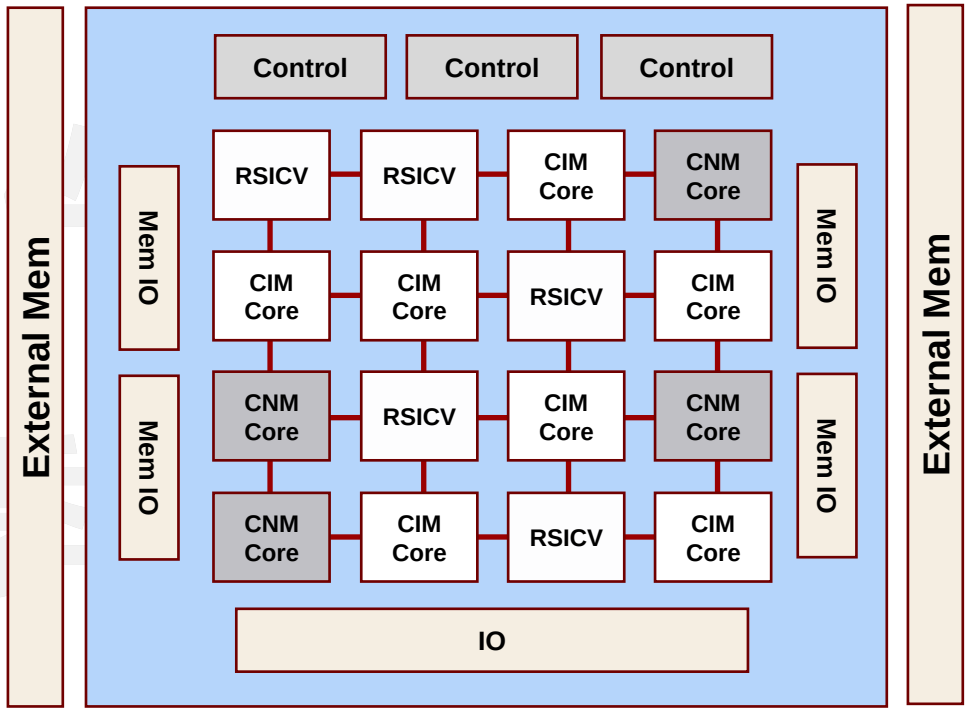


当前存算一体芯片算力、功耗图谱

存算一体的异构集成

- 融合存算一体技术的多阵列异构集成芯片





集中式异构集成架构

设计难度相对较低

但基于集中式总线的架构难以最大限度发挥存算一体的优势

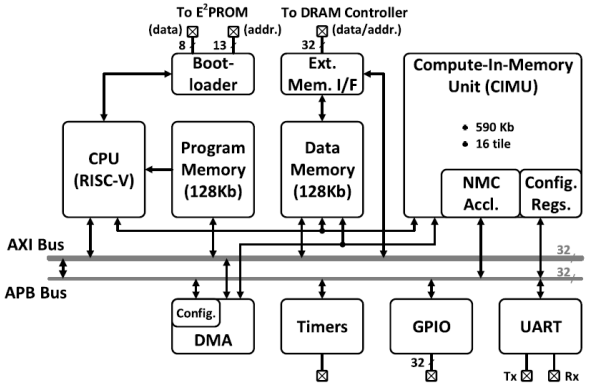
分布式异构集成架构

可扩展性、可重构性好

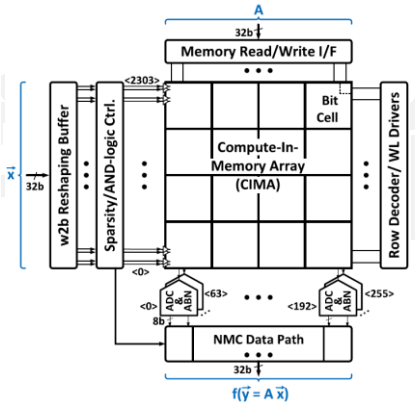
但片上网络设计及通信难度高

典型存算一体异构集成芯片 – 普林斯顿总线式集成芯片

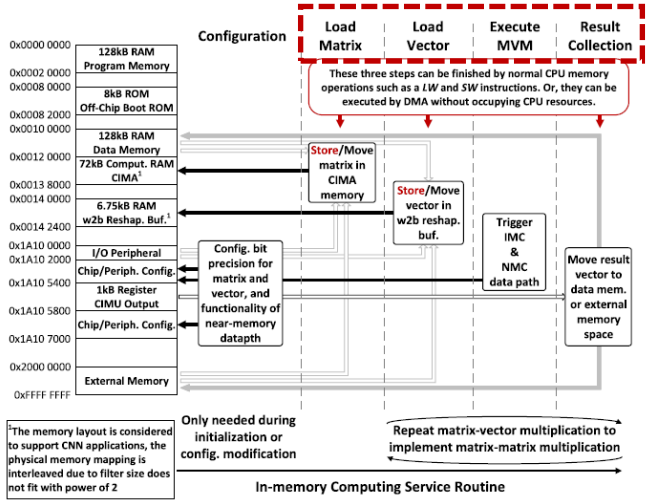
- 融合存算一体技术的多阵列异构集成芯片



顶层架构

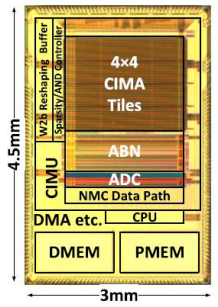


存内计算阵列架构

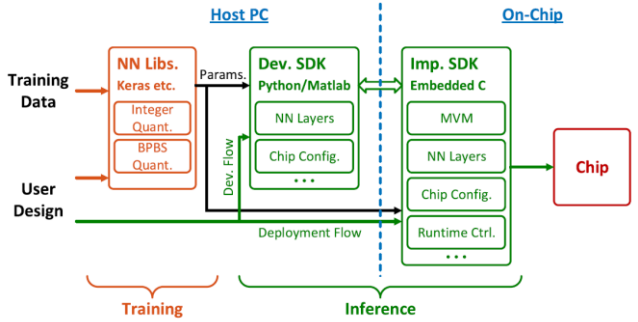


存内计算实现矩阵乘法时序图

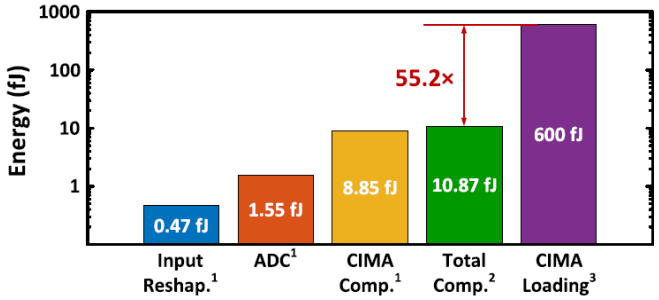
- 4x4 8T SRAM 存内计
- 阵列单阵列576x64，共 590Kb
- 集成RISCV CPU
- 128Kb ProgMem
- 128Kb DataMem
- 模拟存内 8b ADC 精度



芯片版图



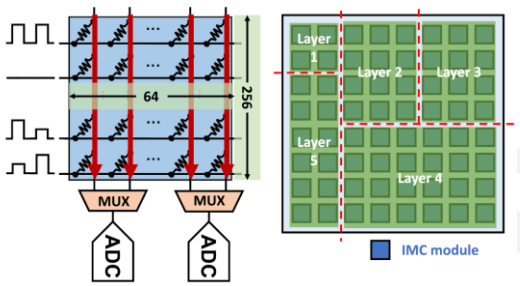
存内和非存内计算的映射方式



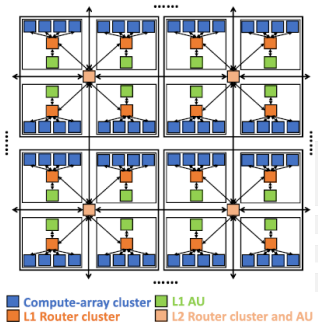
能耗分布分析

典型存算一体异构集成芯片 – 密歇根大学TAICHI分布式集成芯片

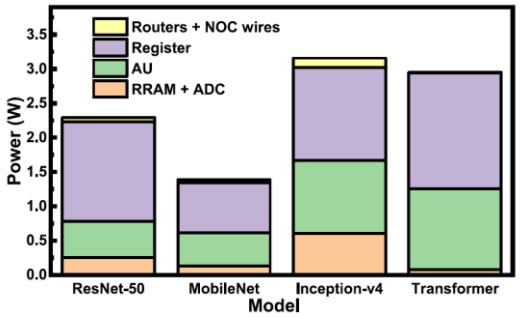
- 融合存算一体技术的多阵列异构集成芯片



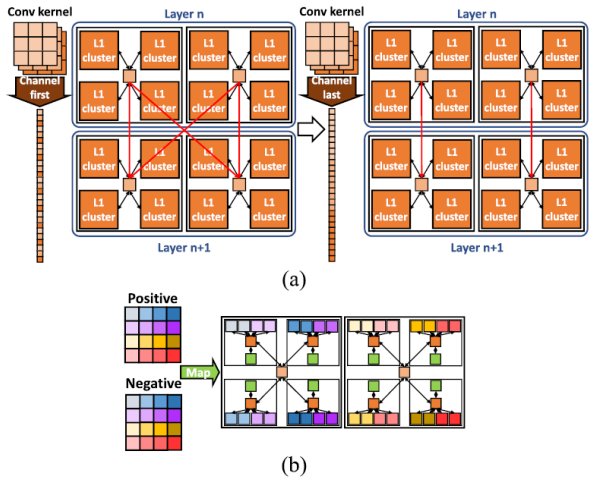
ReRAM模拟存内计算阵列



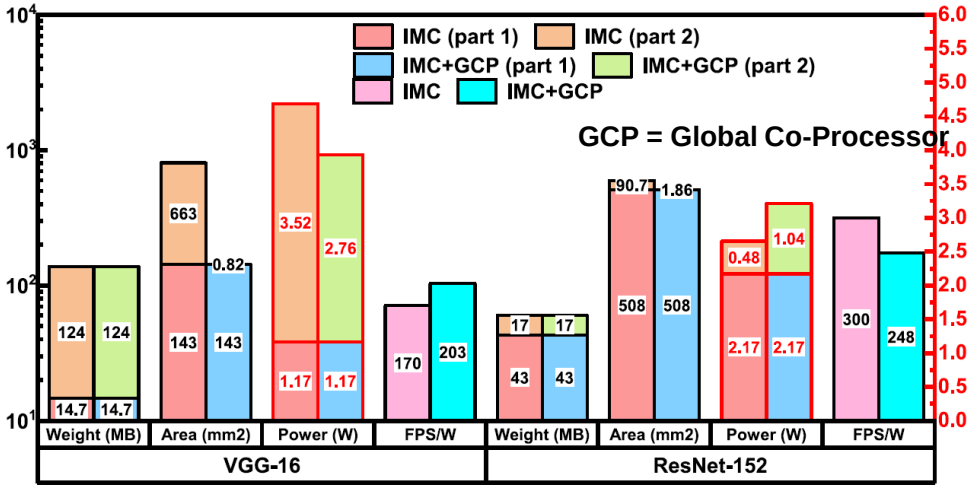
层次式片上网络



能耗分布分析



4x4个cluster、每个cluster
包含4x4个存内计算阵列

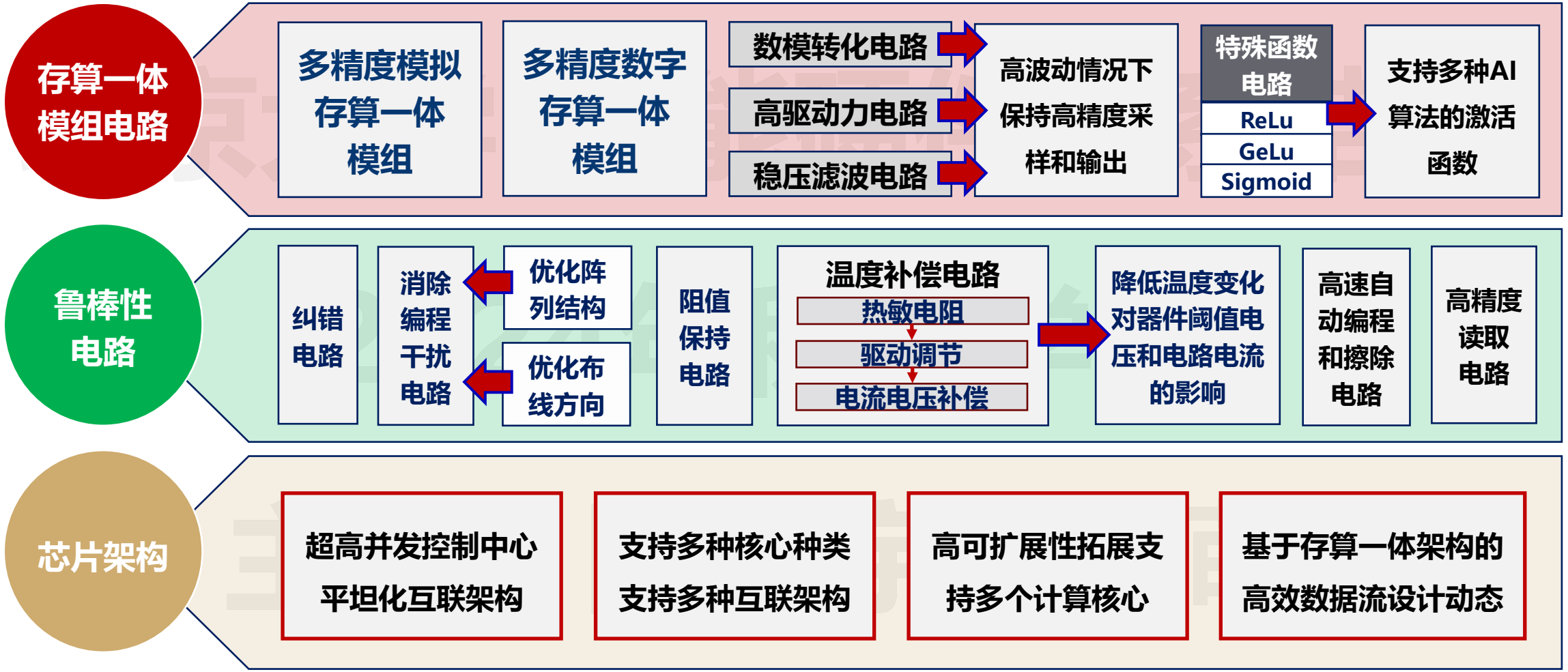


分布式异构架构运行VGG-16和ResNet-152的性能分析

- 16x16层级式网格片上网络
- 集成4种不同尺寸的ReRAM
  模拟存内计算阵列
- 模拟存内 8b ADC 精度

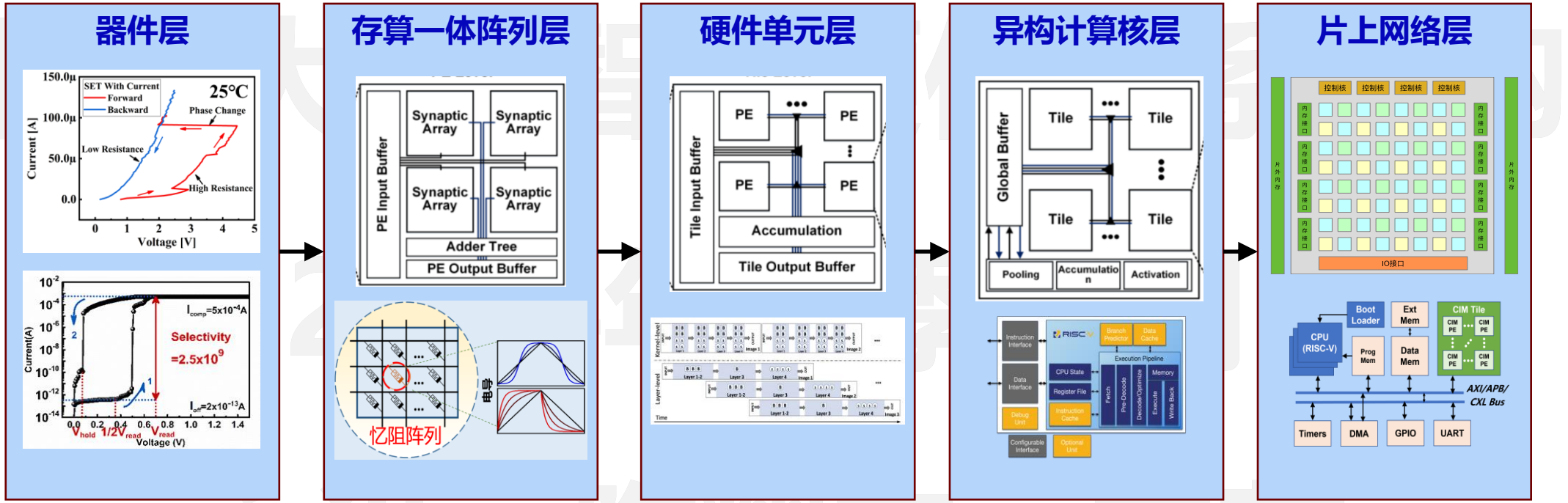
存算一体异构集成的电路与架构挑战

- 优化电路与芯片架构，保障能效优势和迭代演进能力



存算一体异构集成的系统级多层次挑战

- 需要考虑多层次的协同设计，设计空间非常大



Non-Ideality, etc.　BER, ADC, etc.　PE Pipeline, etc.　Core Configs, etc.　NoC/Bus Traffic, etc.

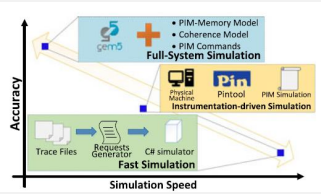
存算一体异构集成的架构仿真

- 架构仿真为设计空间非常大的存算一体异构集成提供支撑

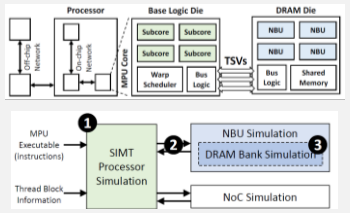
目标应用

通用计算

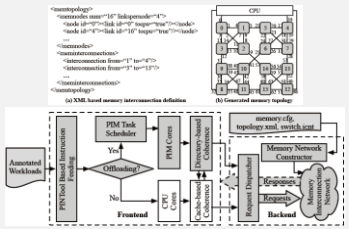
① PIMSIM USC (DRAM)



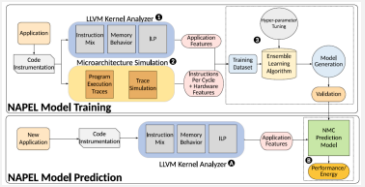
② MPUSIM UCSB (DRAM)



③ MultiPIM Virginia (DRAM)

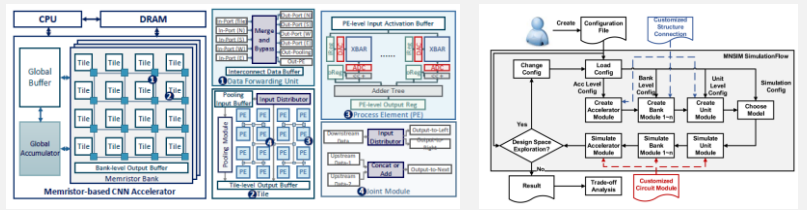


④ RamulatorPIM ETH (DRAM)

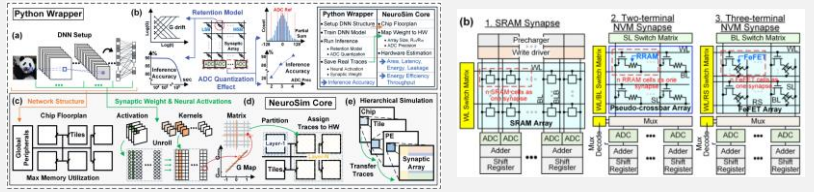


AI计算 (MAC/Conv)

⑤ MNSIM Tsinghua (DRAM/SRAM/eNVM)

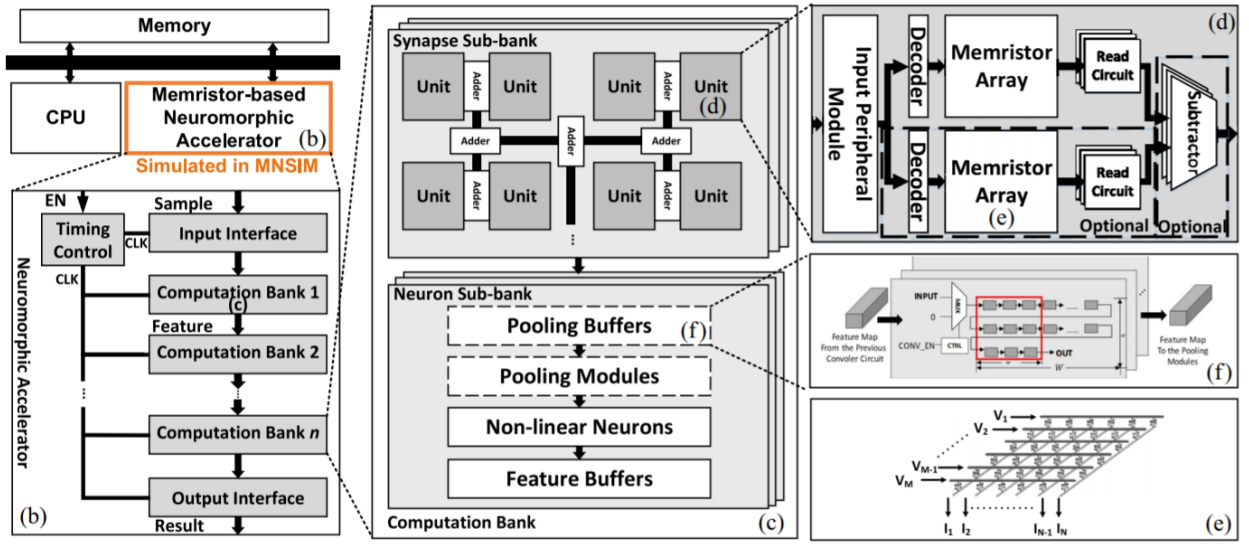


⑥ NeuroSIM GeorgiaTech (DRAM/SRAM/eNVM)

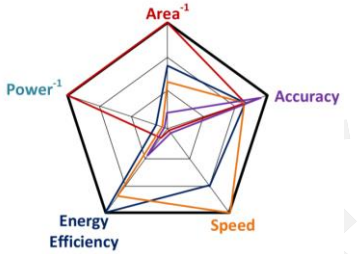


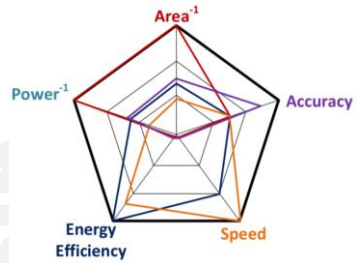
典型存算一体架构仿真器 – 清华MNSIM

- 面向AI计算的存算一体架构仿真器



Hierarchical structure: bank-level, tile level and PE-level





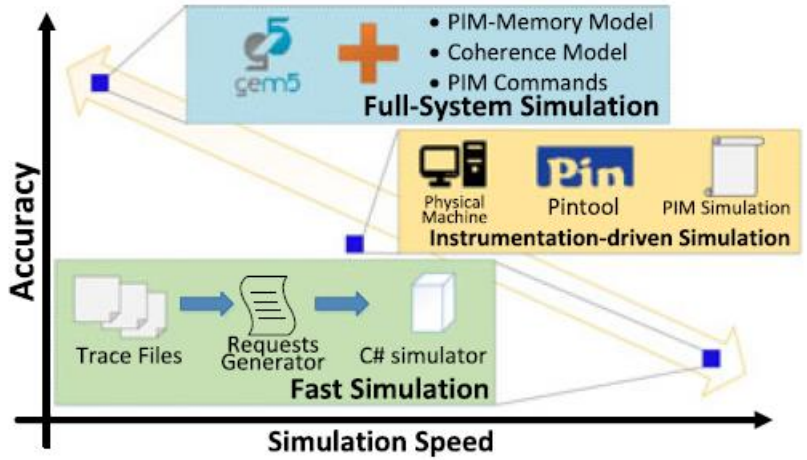
Deep CNN simulation demos

缺乏器件级的仿真

- 层级式的RRAM AI计算仿真器 – 覆盖算法、电路、架构、系统多个层级
- 灵活的接口配置 – 在各个层级接口均可以进行灵活配置，搜索空间很大
- 搜索速度快 – 仿真器均采用准确的行为级模型，可以并行仿真加快仿真速度

典型存算一体架构仿真器 – 南加州大学PIMSIM

- 面向通用计算的存算一体架构仿真器



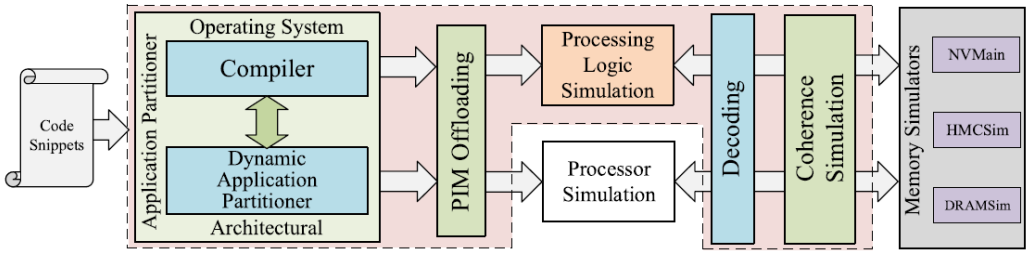

```
1:  #PIM_START
2:  #ID = 0          //define the execution logic ID
3:  #INPUT = a
4:  #INPUT = b
5:  #OUTPUT = c
6:     c = a + b;
7:  #PIM_END
```

```
1:  #PIM_START
2:  #ID = 0   //define the execution logic ID
3:     c = a + b;
4:  #PIM_END
```

```
1:  #PIM_CODE_ADD
2:  #ID = 0      //define the execution logic ID
3:  #DEFINE_INPUT Input1
4:  #DEFINE_INPUT Input2
5:  #DEFINE_OUTPUT Output1
6:        Output1 = Input1 + Input2;
7:  #PIM_END
8:     PIM_ADD(c, a, b);
```

```
1:  PIMKernel(c, a, b, 0); //the execution logic ID is 0
2:  //The function "PIMKernel" is preset in PIMSim
      to perform predefined simple kernerls
```

Need to manually label PIM tasks



PIMSIM Architecture Simulation Framework

Three Simulation Modes

① **Full-System**
② **Instrumentation-driven**
③ **Fast**

- **提供三种仿真模式 – Tradeoff between simulation speed and accuracy**
- **PIM任务标识 – 可执行的PIM任务在汇编编译的阶段进行代码标识**
- **兼容各类DRAM后端仿真器 – 例如DRAMSim等**

课程评估与总结

- 课程评估：扫右侧二维码或登录网上评估系统（kcpg.pku.edu.cn）
  - 希望同学们能给予好评，感谢！

- 希望大家后续科研学习一切顺利，也欢迎在课程结束后，同学们仍然能找我讨论有趣的问题

- 办公室：资源西楼2208B
- 邮箱：taoyaoyutyy@pku.edu.cn
- 电话：13095698730（微信同）