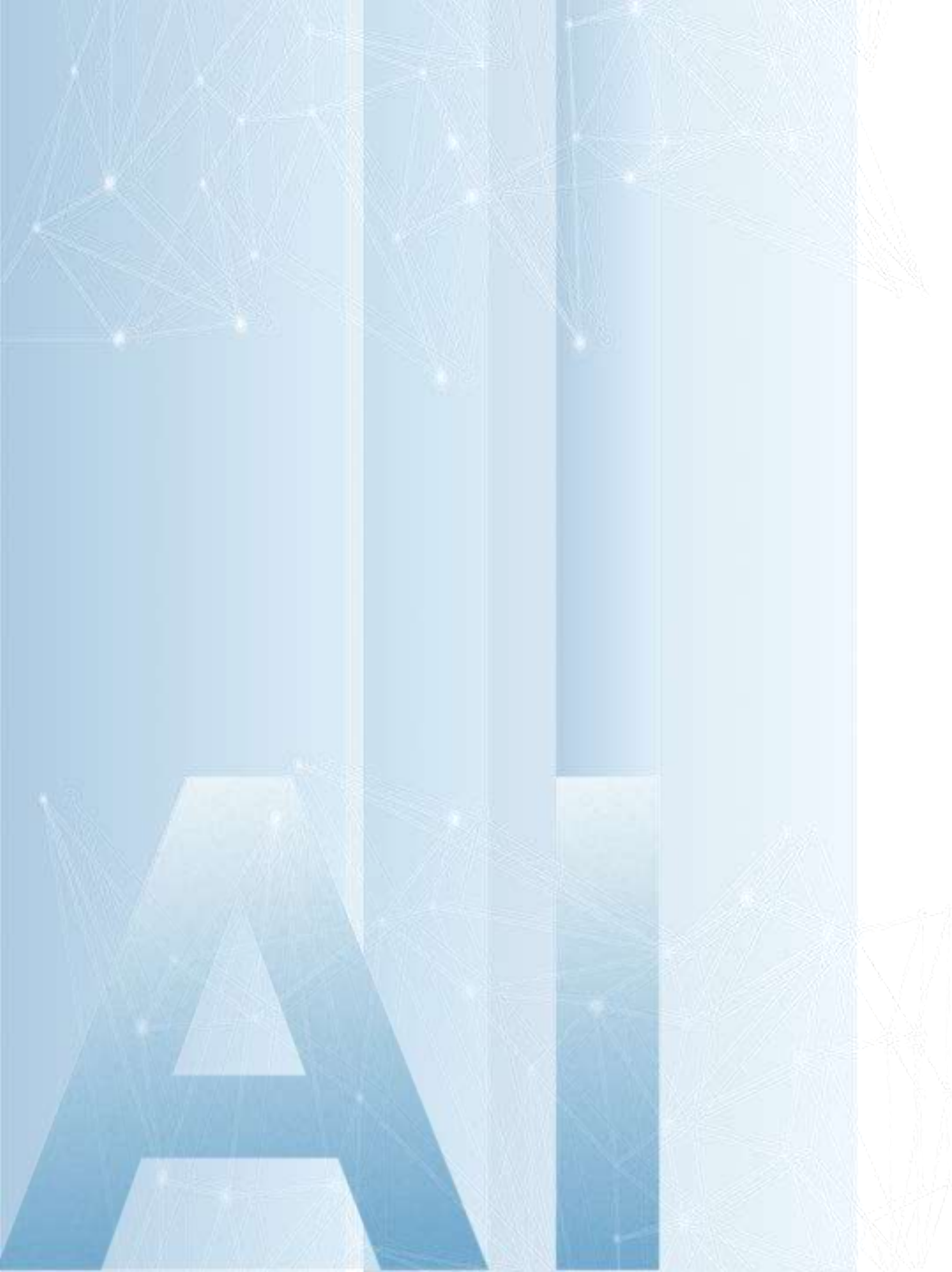
上海人工智能实验室
Shanghai Artificial Intelligence Laboratory

大模型并行

颜航



目录

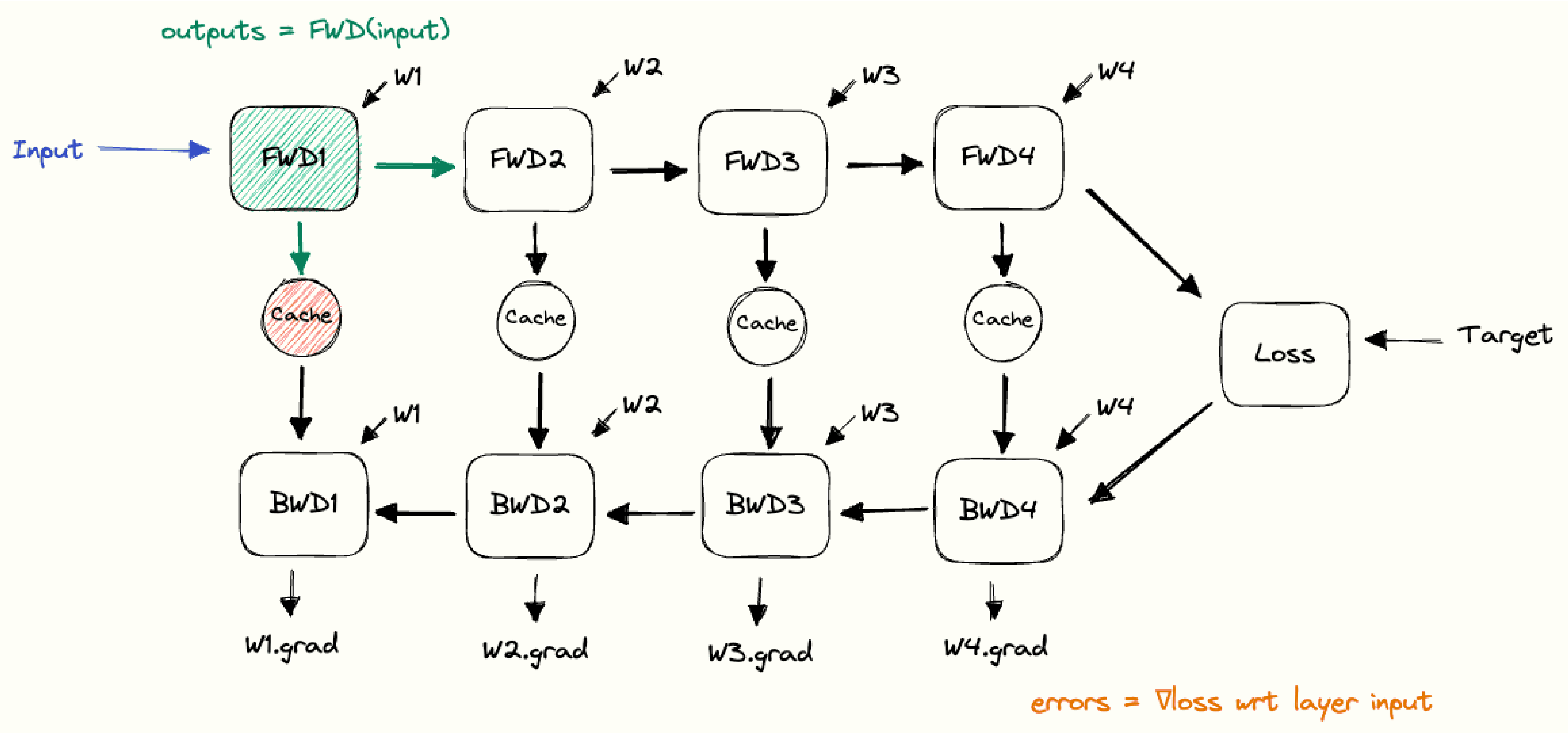
各类并行算法原理

各类并行算法

算法名称	主要思想	优点	缺点
数据并行 Data Parallel	将**模型并行**地**分散**到各个GPU上进行运算，通过梯度同步保证各个GPU上的模型仍然保持一致	使用非常简单，适用场景广泛，几乎已经成了多卡训练必用	无法单独应对模型巨大的情况
张量并行 Tensor Parallel	将**矩阵运算**进行**拆分**	可以将巨大模型拆小以适配单张GPU	1. 对通信的要求较高 2. 需要对模型算子进行处理
流水线并行 Pipeline Parallel	将大模型**按层**进行**拆分**	可以拆解巨大的模型，同时对模型算子影响相对较小	计算可能存在空泡，导致硬件利用率低
零冗余优化 Zero Redundancy Optimization	将模型在**数据并行维度**间进行**拆分**	简单易用	Zero2、Zero3对通信的要求较高
序列并行 Sequence Parallel	在**输入序列维度**进行**拆分**	可以应对超长上下文模型	长度不特别长的场景下效率不高

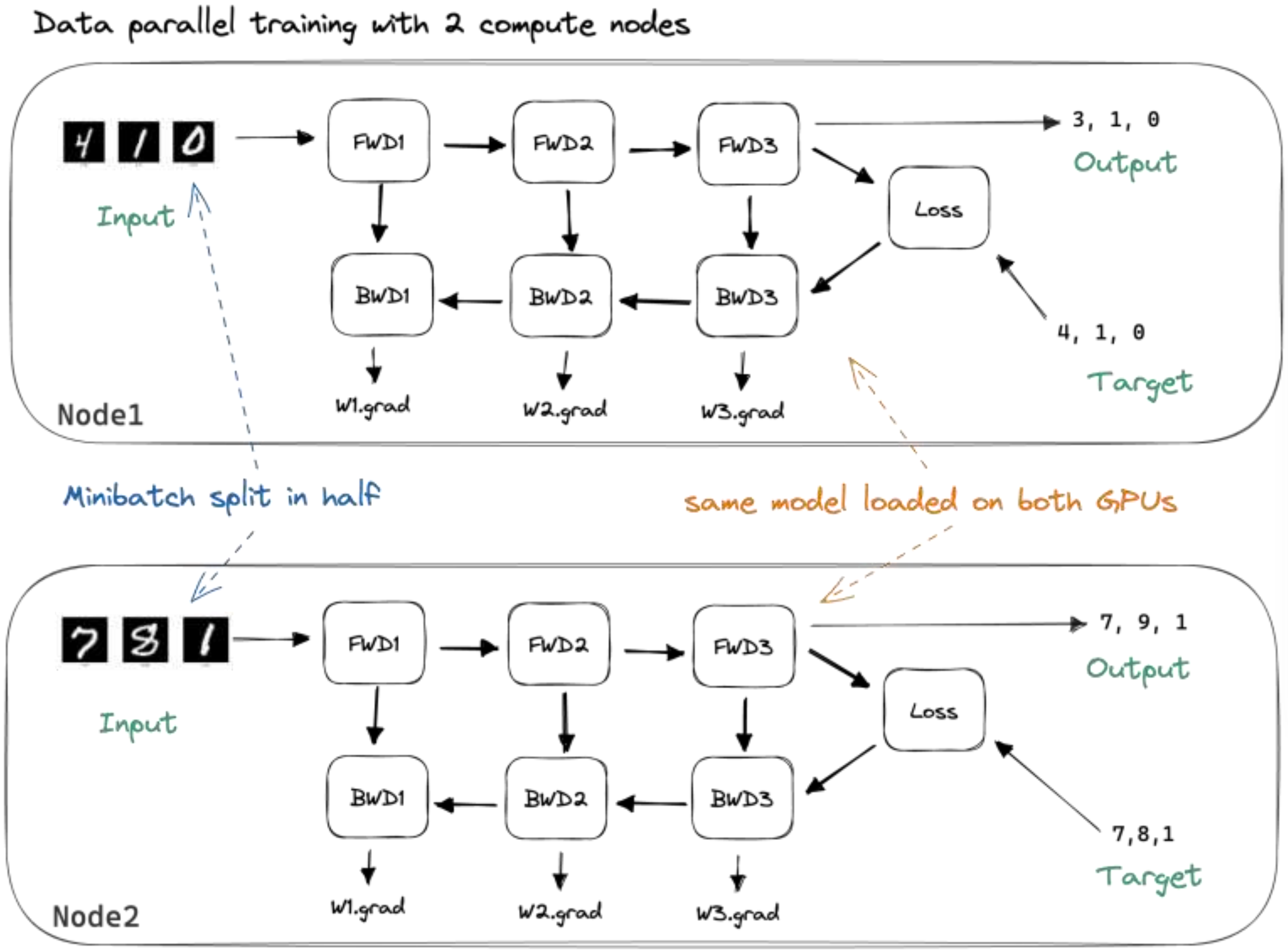
数据并行

单卡训练



https://siboehm.com/articles/22/data-parallel-training

数据并行



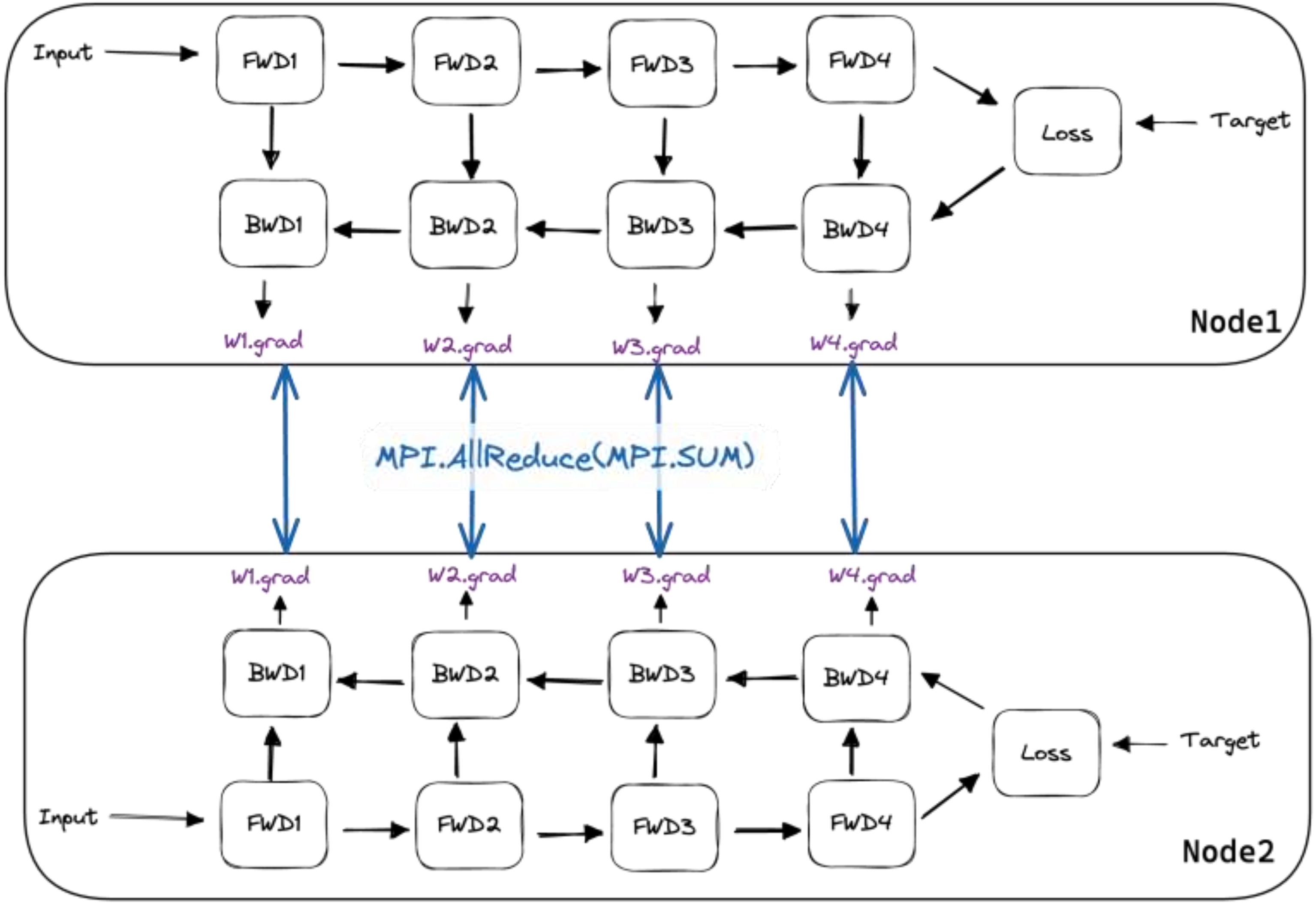
在每张卡上都执行各自前向、后向运算

注意

1. 每张卡上的数据是不同的
2. 每张卡上的模型需要初始的时候是完全一样的
3. 执行完梯度运算后，不能各自直接更新自己的模型，不然每张卡上的模型就不一致了

https://siboehm.com/articles/22/data-parallel-training

数据并行

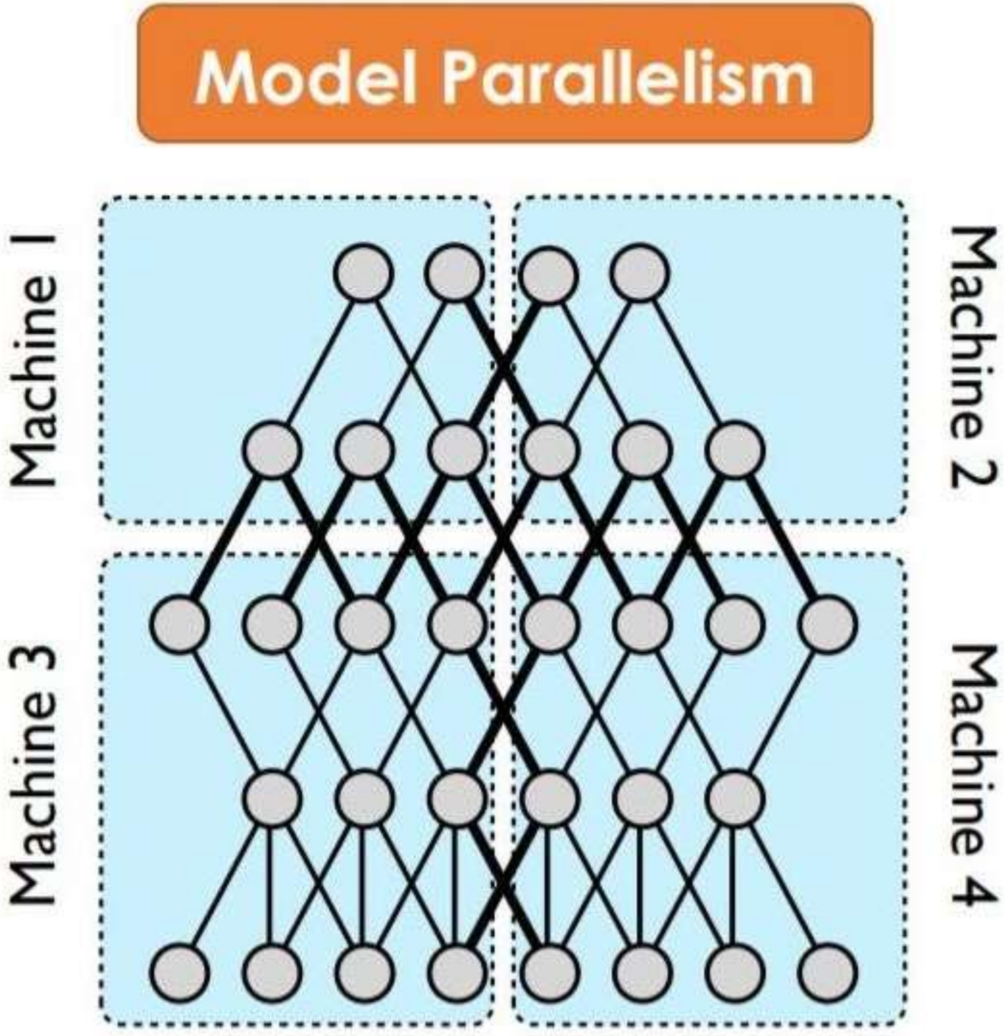


在完成梯度求解之后，需要首先进行梯度同步，经过这个步骤，不同GPU上模型的梯度是一致，由于它们本身起点也是一致的，所以更新之后不同GPU上模型仍然是一致的。

注意

1. 后向过程可以和梯度同步有时间上的重叠，提高效率
2. 不同GPU如果混用，可能导致更新结果不一致

模型并行



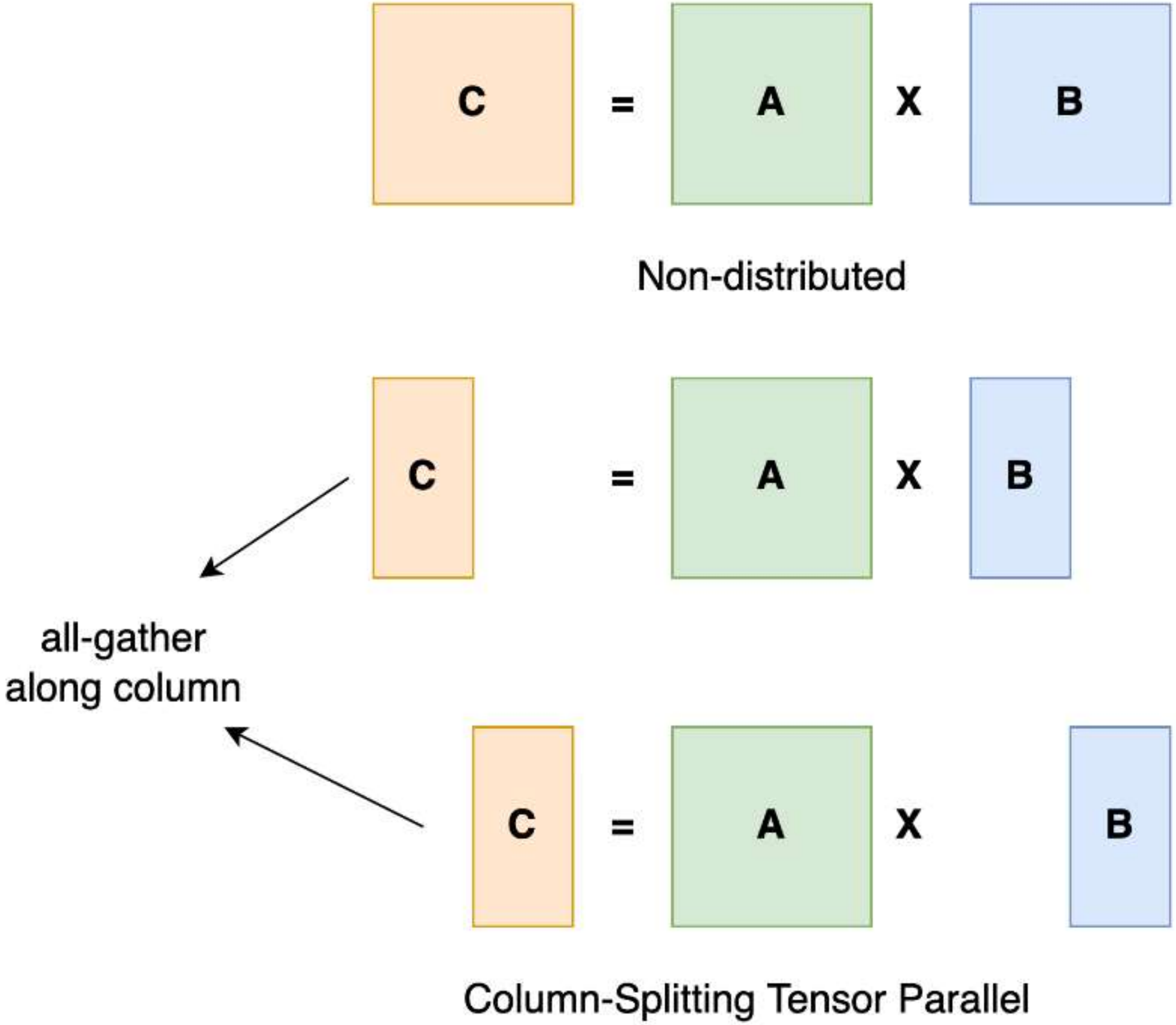
问题 👇

当模型特别大的时候，会出现一张卡甚至没办法放下一个模型的情况，这时就需要将模型拆分到多个GPU上。

拆分方案有 **2** 种：

1. 横向切分（张量并行）
2. 竖向切分（流水线并行）

张量并行

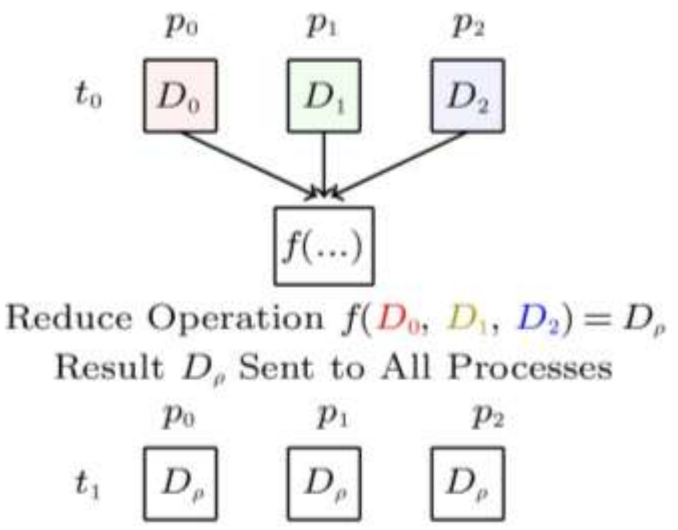


可以将矩阵乘法拆分成在两个设备上运行

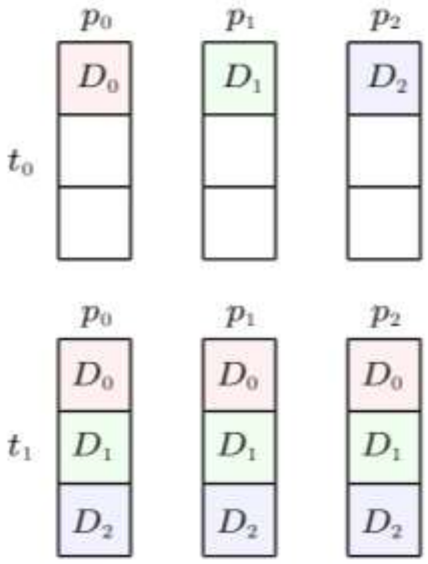
假设B就是那个模型的权重，实际上就是将模型的权重分散到两个设备上了

名词解释

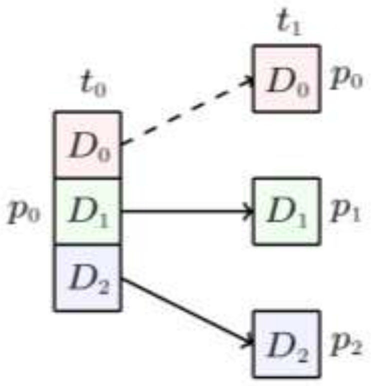
操作	定义	示例
All-reduce	对所有进程的数据执行一个归约操作（如求和、最大值等），并将结果返回给所有进程	每个进程持有一个值。执行All-reduce操作（如求和）后，每个进程将获得所有值的总和
All-gather	将所有进程的数据收集并分发给所有进程	每个进程拥有数据的一部分。执行All-gather后，每个进程将拥有包括所有进程部分的完整数据集
Scatter	将一个进程的数据分散到所有其他进程	一个进程将其数据集分割成多部分，将每部分发送到不同进程
All-to-All	每个进程向所有其他进程发送数据，并同时从所有其他进程接收数据	在All-to-all操作中，每个进程将其数据分割并发送到所有其他进程，并从其他所有进程接收数据



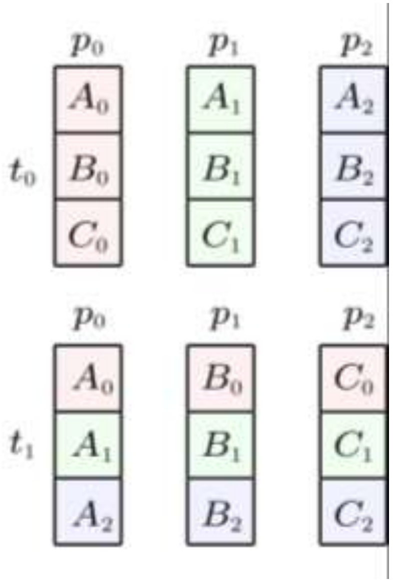
All-reduce



All-gather



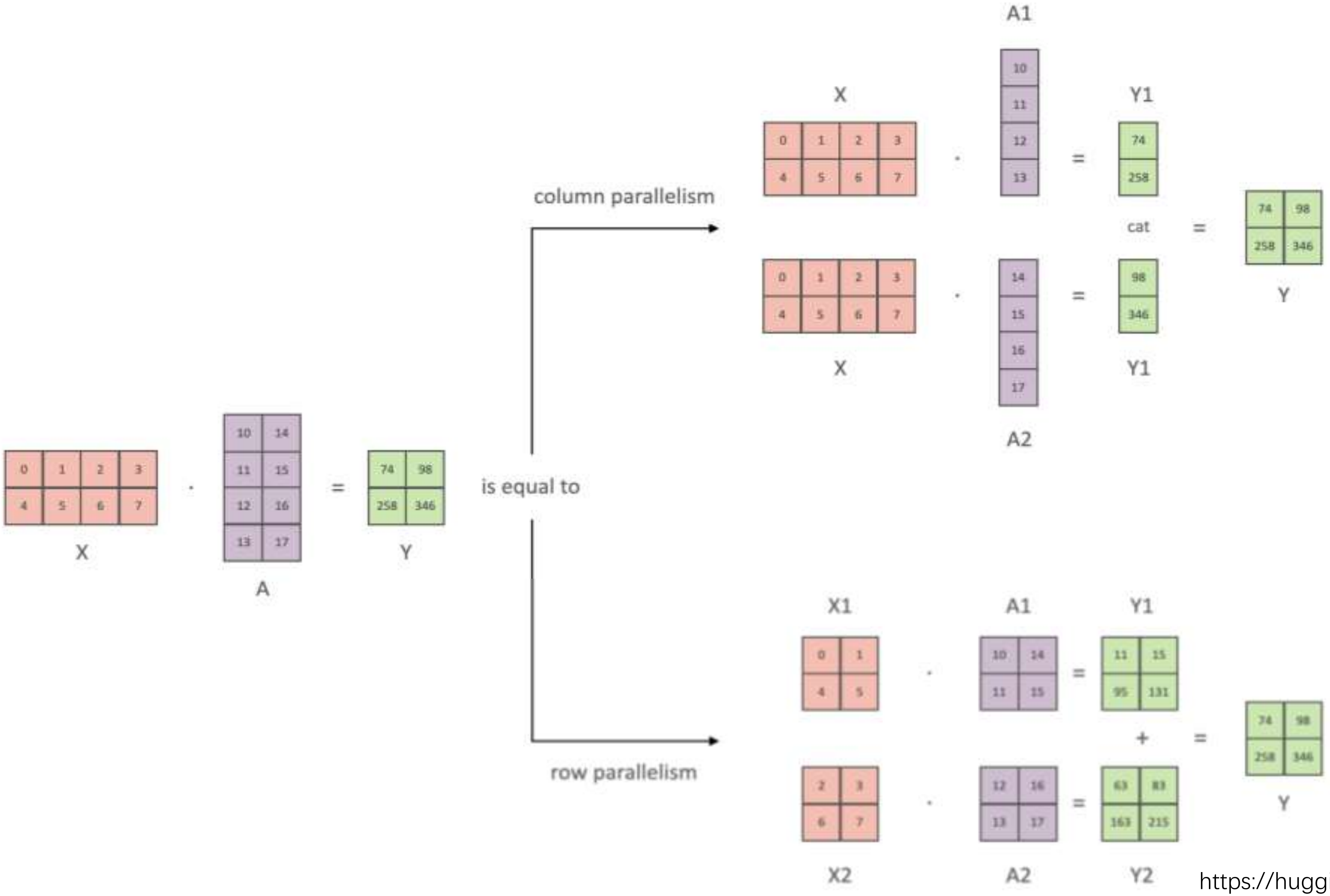
Scatter



All-to-All

xCCL: A Survey of Industry-Led Collective Communication Libraries for Deep Learning

张量并行——一个示例



切分X或者A都是可以的

https://huggingface.co/docs/transformers/v4.15.0/parallelism

张量并行—实际应用

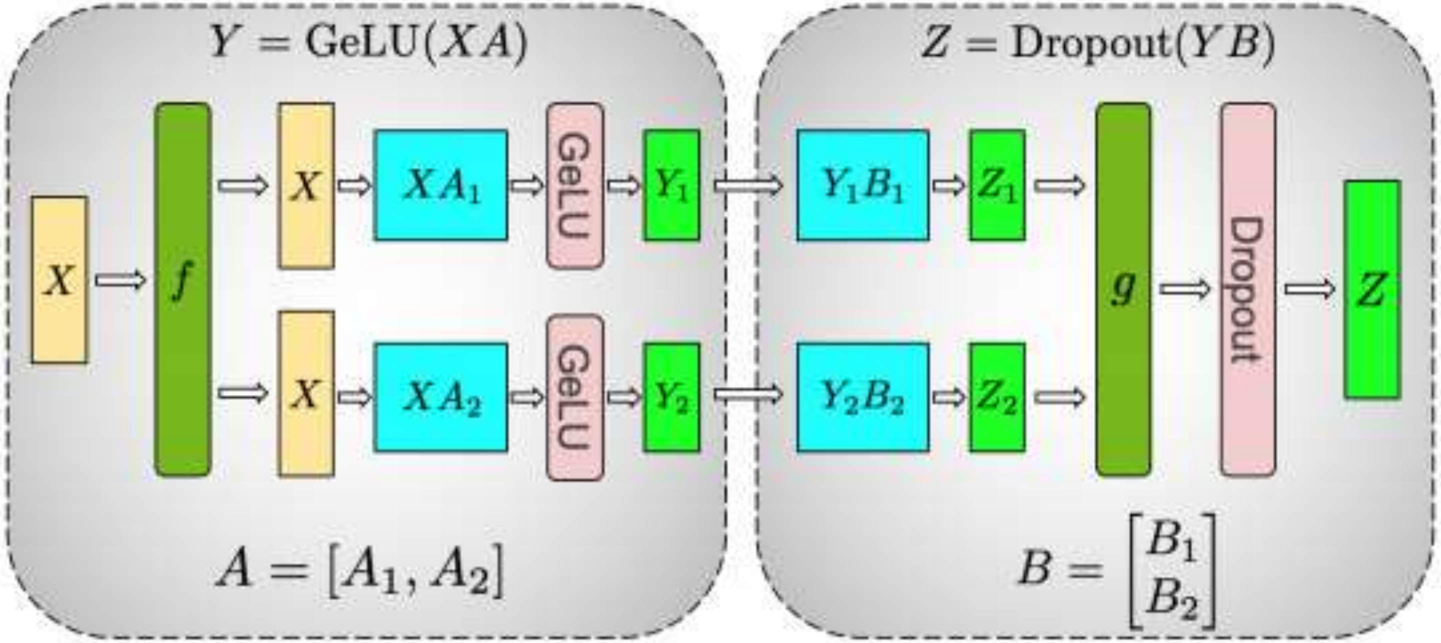
MLP运算

$$Y = XA \qquad (1)$$

$$Y = \mathrm{GeLU}(Y) \qquad (2)$$

$$Y = YB \qquad (3)$$

其中$X \in R^{L\times h}, A \in R^{h\times 4h}, B \in R^{4h\times h}$

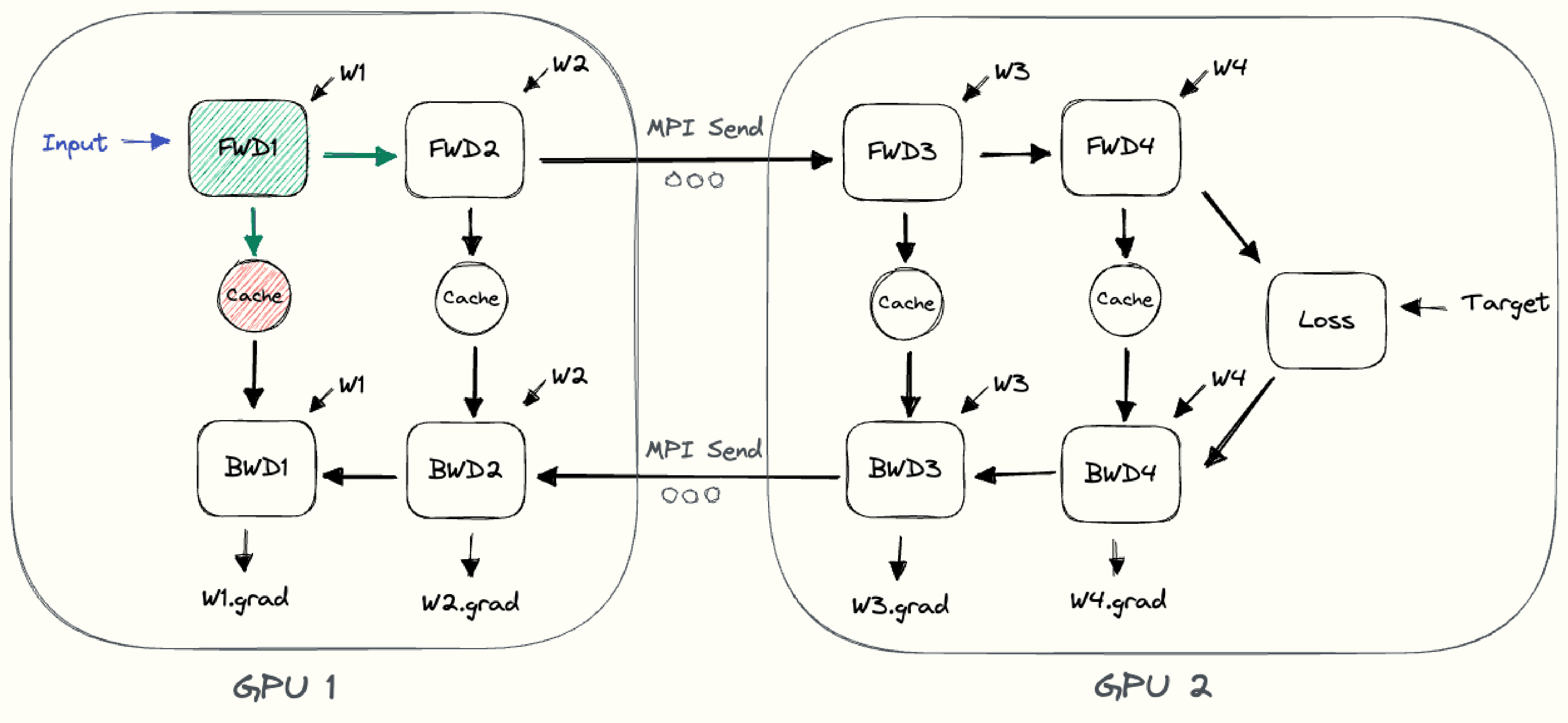


通过合理安排竖切和横切的位置，可以减少一次通信操作。

使用张量并行**需要修改模型算子**！！！同时张量并行需要比较**频繁**地进行**通信**，对带宽要求较高，一般仅在同一个节点内部使用

https://huggingface.co/docs/transformers/v4.15.0/parallelism

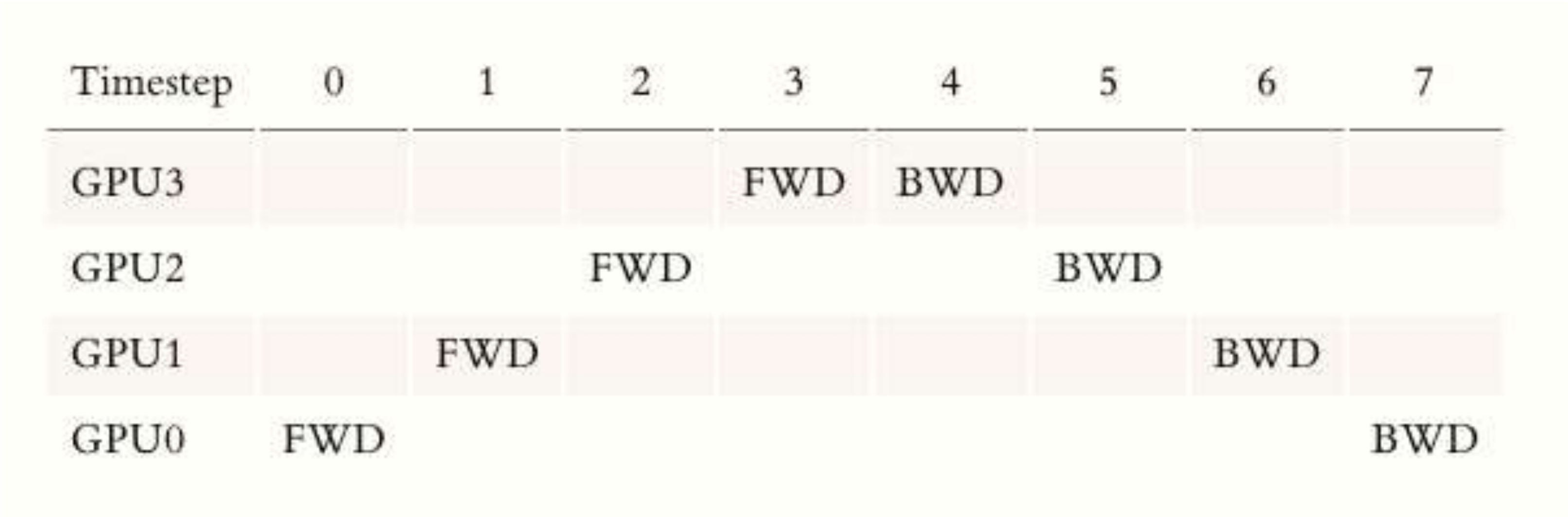
流水线并行



在层与层之间进行切分

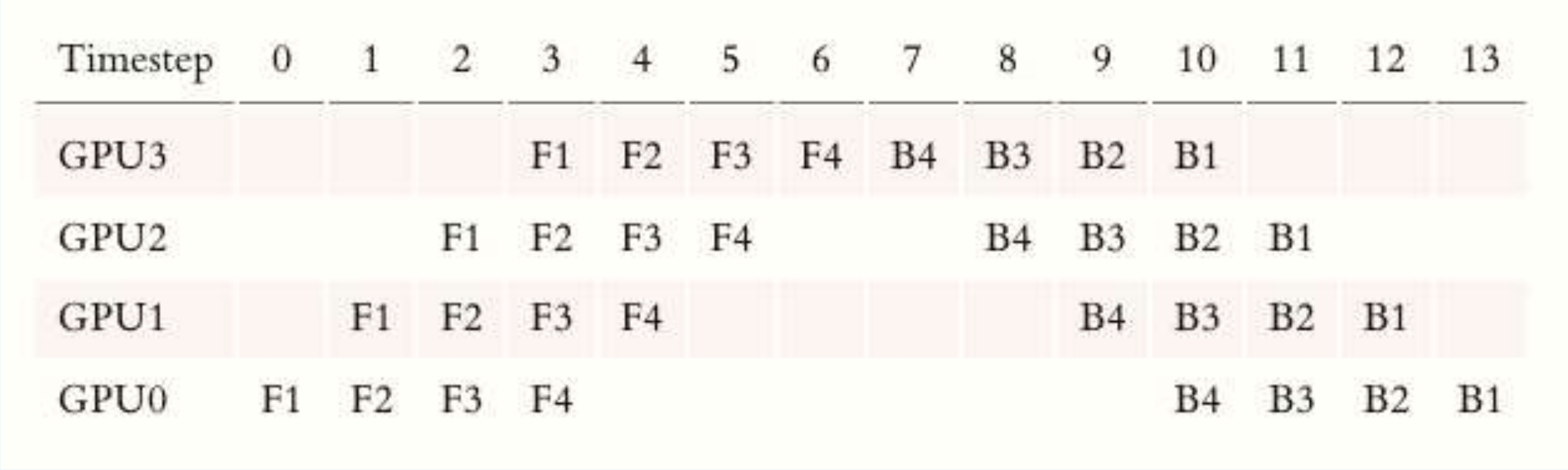
https://siboehm.com/articles/22/pipeline-parallel-training

流水线并行



Timestep	0	1	2	3	4	5	6	7
GPU3				FWD	BWD			
GPU2			FWD			BWD		
GPU1		FWD					BWD	
GPU0	FWD							BWD

这样实现会导致**大量浪费**

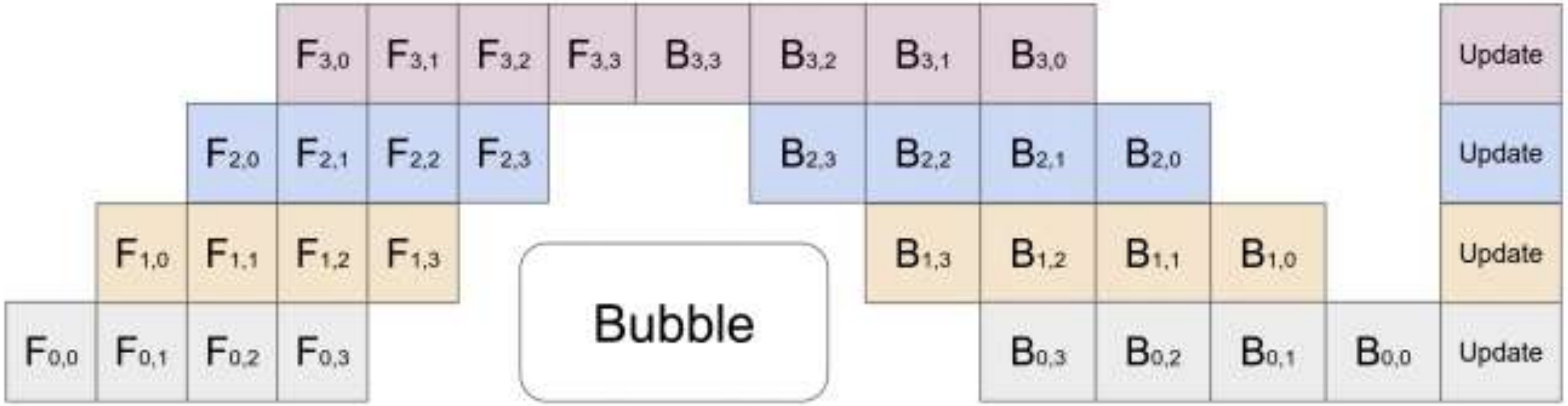


Timestep	0	1	2	3	4	5	6	7	8	9	10	11	12	13
GPU3				F1	F2	F3	F4	B4	B3	B2	B1			
GPU2			F1	F2	F3	F4			B4	B3	B2	B1		
GPU1		F1	F2	F3	F4					B4	B3	B2	B1	
GPU0	F1	F2	F3	F4							B4	B3	B2	B1

Gpipe提出将**每个batch再拆分为micro batch**

https://siboehm.com/articles/22/pipeline-parallel-training

流水线并行



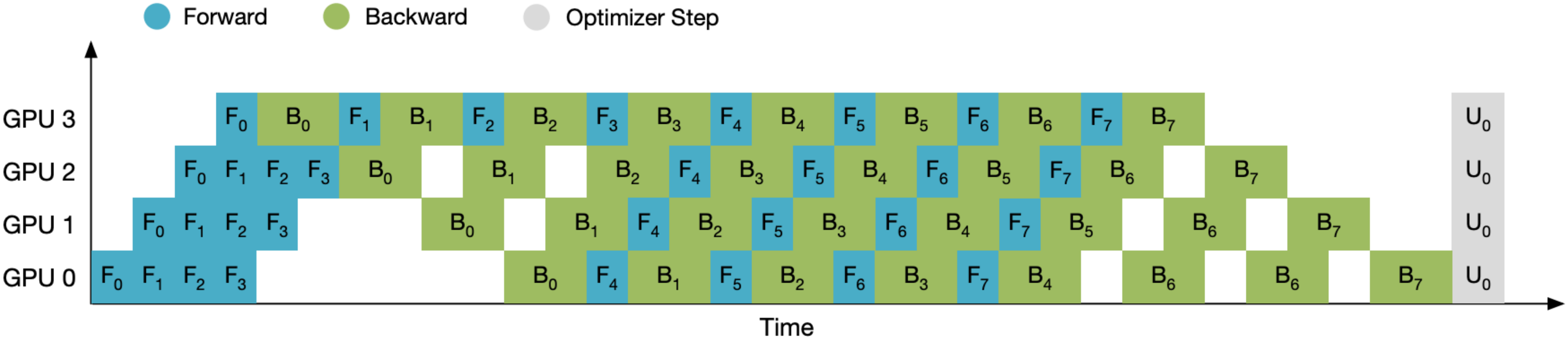
空泡率：

$$1-\frac{2nm}{2n(m+n-1)}=1-\frac{m}{m+n-1}$$

其中m是microbatch的数量，n是流水线阶段的数量。可以看出来，增大m和减少n都可以减少空泡

https://siboehm.com/articles/22/pipeline-parallel-training

流水线并行



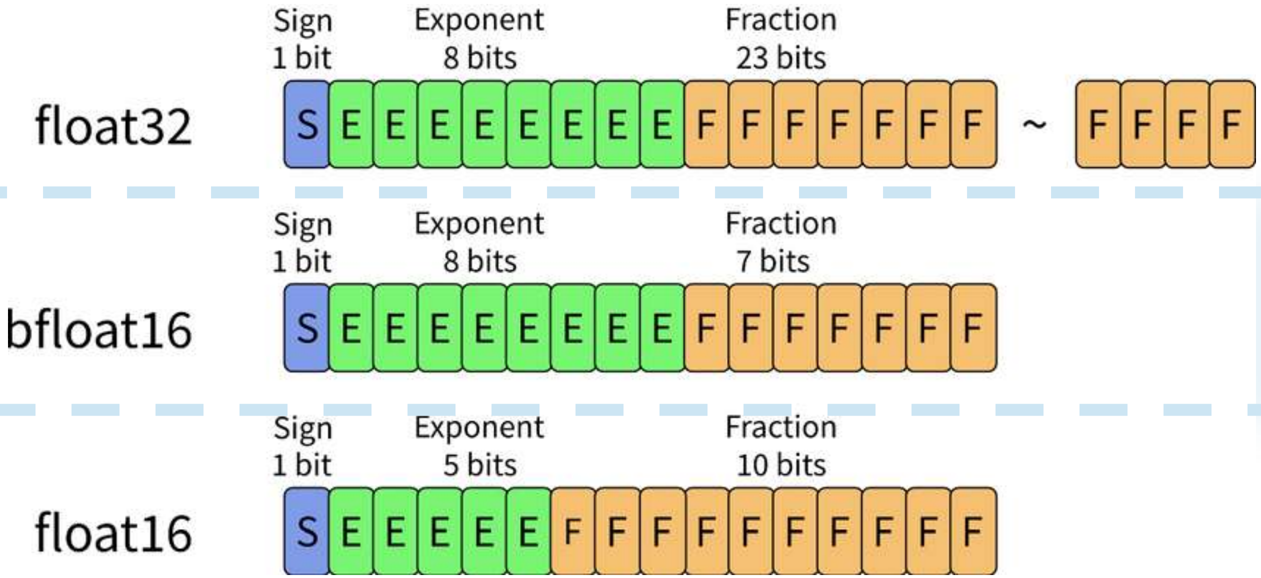
PipeDream可以稍微减少一些为反向传播cache的激活状态，是目前默认的流水线并行方案

流水线并行对通信的要求相较张量并行低一些，**但当流水线特别长的时候，空泡会导致其效率降低**

GLM-130B: AN OPEN BILINGUAL PRE-TRAINED MODEL
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

零冗余优化 – 数值精度

背景知识： 浮点数表达方式

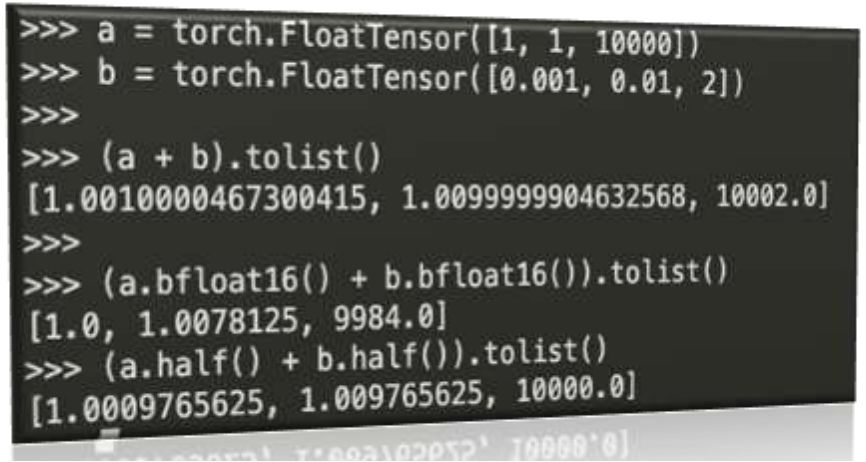


Float32表达精度高，范围大，但要占4个byte

Bfloat16表示精度低，但数字表达范围和float32一样大

目前大模型训练默认使用bfloat16

Float16的表示精度中，但数字表达范围较小



```
>>> a = torch.FloatTensor([1, 1, 10000])
>>> b = torch.FloatTensor([0.001, 0.01, 2])
>>>
>>> (a + b).tolist()
[1.0010000467300415, 1.0099999904632568, 10002.0]
>>>
>>> (a.bfloat16() + b.bfloat16()).tolist()
[1.0, 1.0078125, 9984.0]
>>> (a.half() + b.half()).tolist()
[1.0009765625, 1.009765625, 10000.0]
```

不同表达精度，计算结果不一致

零冗余优化 – 显存占用分析

Adam优化器算法

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

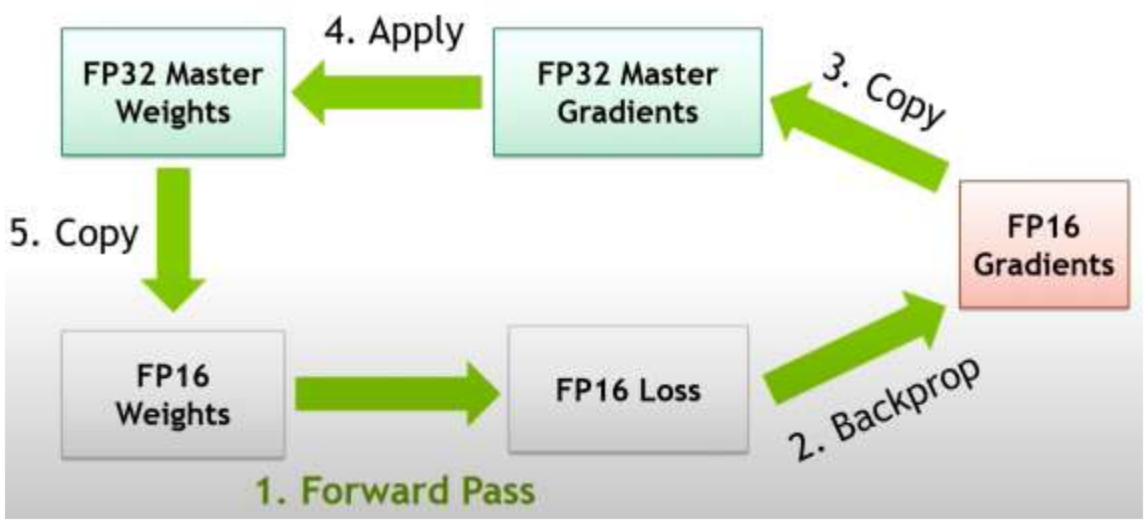
η : *Initial Learning rate*
g_t : *Gradient at time t along* ω^j
ν_t : *Exponential Average of gradients along* ω_j
s_t : *Exponential Average of squares of gradients along* ω_j
β_1, β_2 : *Hyperparameters*

正常训练显存占用：$4\Theta + 4\Theta + 4\Theta + 4\Theta = 16\Theta$

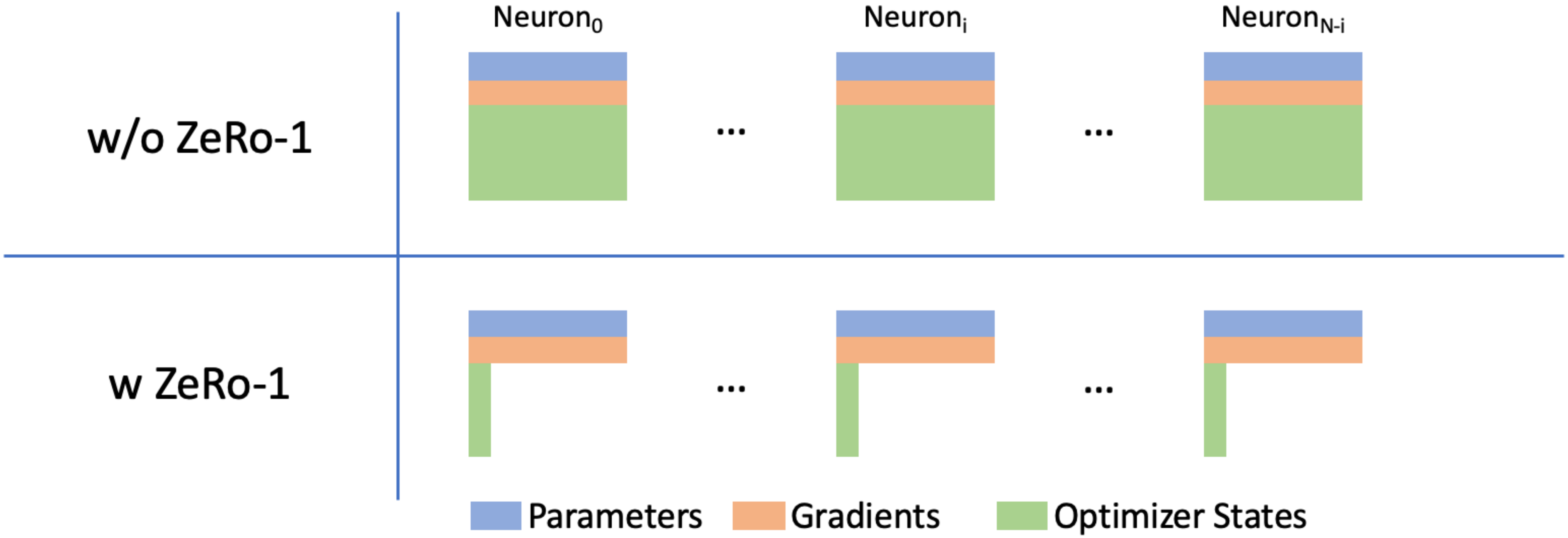
其中Θ是参数的个数；上述占用依次是一阶、二阶动量、模型参数、梯度

混合精度训练显存占用：$4\Theta + 4\Theta + 4\Theta + 2\Theta + 2\Theta = 16\Theta$

其中Θ是参数的个数；上述占用依次是一阶、二阶动量、复制的模型fp32参数、模型fp16参数、模型fp16梯度



零冗余优化—Zero1



Neuron_0 … Neuron_i … $\text{Neuron}_{N\text{-}i}$

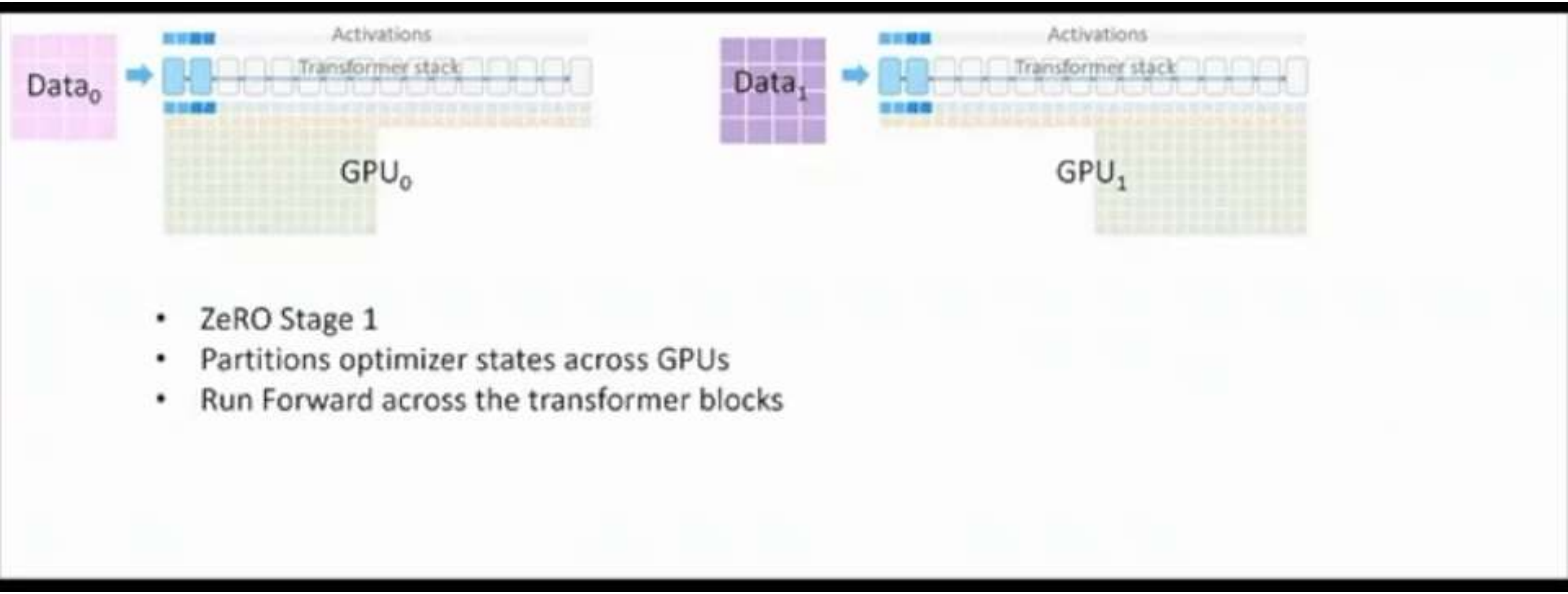
w/o ZeRo-1

w ZeRo-1

Parameters Gradients Optimizer States

Assume we use mixed precision training with Adam optimizer. Comparing the total memory usage with and with out ZeRO-1. ψ denotes model size (number of parameters), and N_d denotes DP degree. Then, without ZeRO-1 memory consumption is $(2+2+3*4)*\psi = 16\psi$, with ZeRO-1 is $2\psi + 2\psi + 12\psi/N_d$.

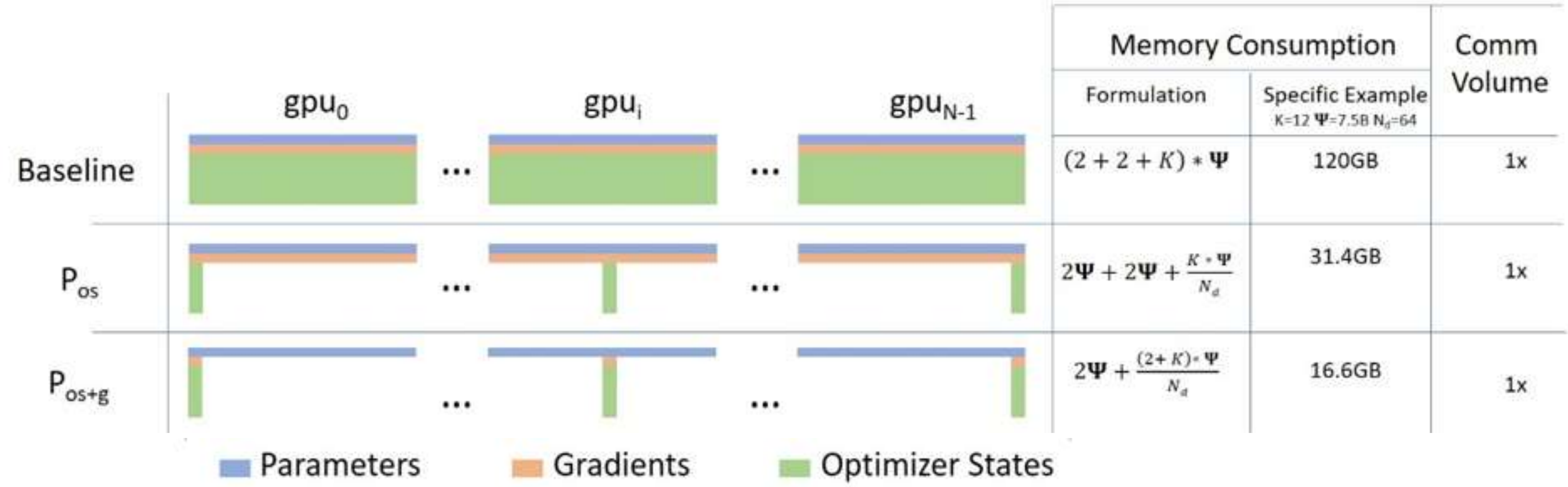
零冗余优化—Zero1



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

Zero1优化过程

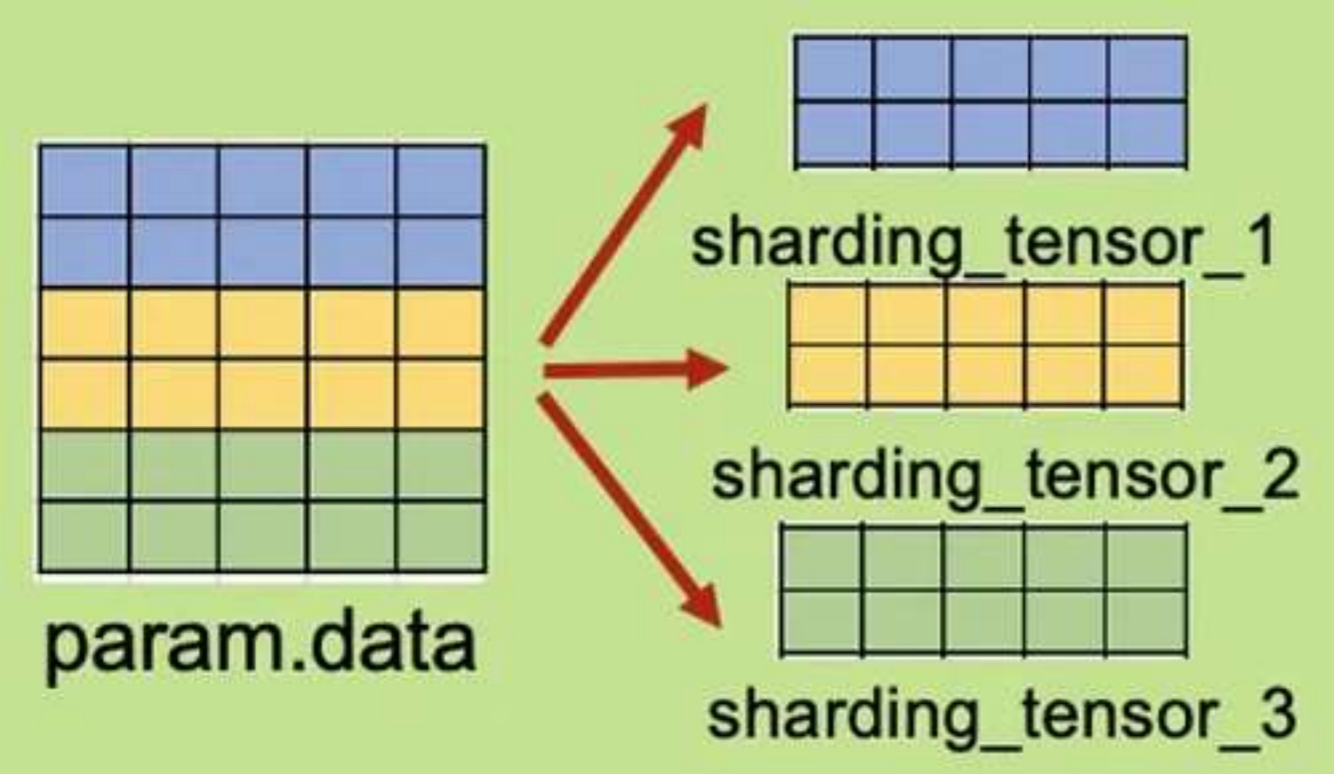
零冗余优化—Zero2



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x

零冗余优化—Zero3

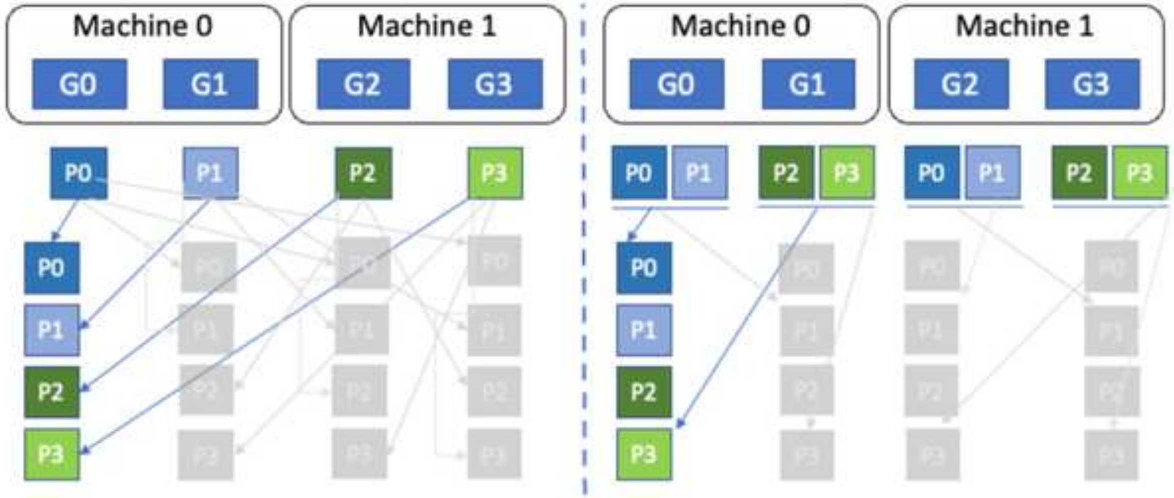
零冗余优化—Zero3



实际上的Zero3是按照**每个Parameter都拆分**的方式，而不是按照整Parameter拆分。

在计算当前层时，提前触发下一层的All-gather操作聚合参数，实现**计算和通信的overlap**。

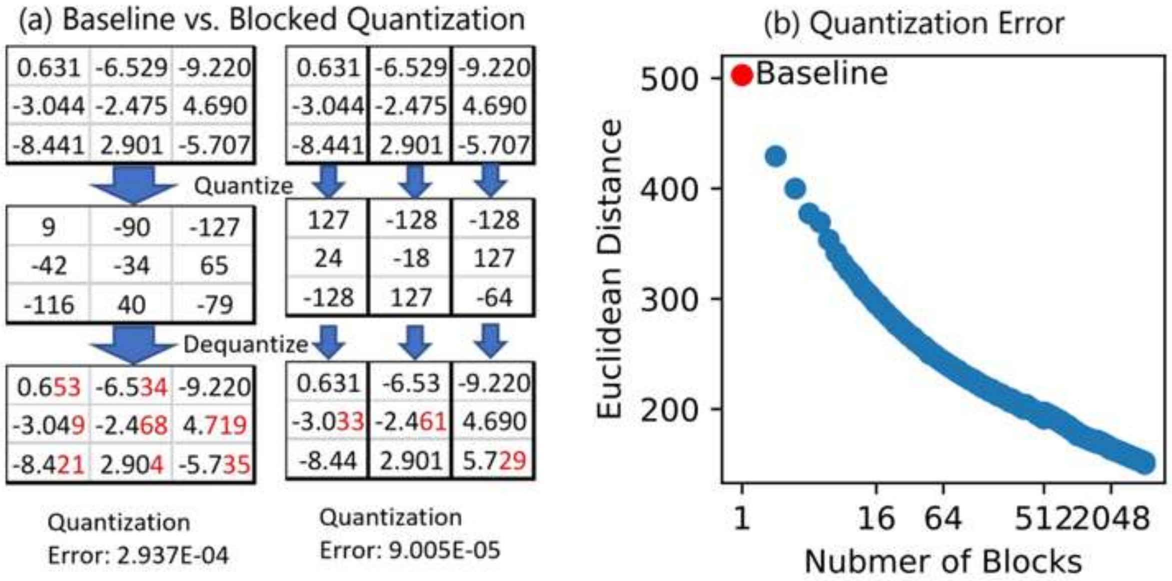
零冗余优化—Zero++



All-gather on primary weight partitions (ZeRO-3)

All-gather on secondary weight partitions (hpZ in ZeRO++)

在节点内进行切片，前向gather更快


(a) Baseline vs. Blocked Quantization

0.631	-6.529	-9.220
-3.044	-2.475	4.690
-8.441	2.901	-5.707

9	-90	-127
-42	-34	65
-116	40	-79

0.653	-6.534	-9.220
-3.049	-2.468	4.719
-8.421	2.904	-5.735

Quantization Error: 2.937E-04

0.631	-6.529	-9.220
-3.044	-2.475	4.690
-8.441	2.901	-5.707

127	-128	-128
24	-18	127
-128	127	-64

0.631	-6.53	-9.220
-3.033	-2.461	4.690
-8.44	2.901	5.729

Quantization Error: 9.005E-05

(b) Quantization Error

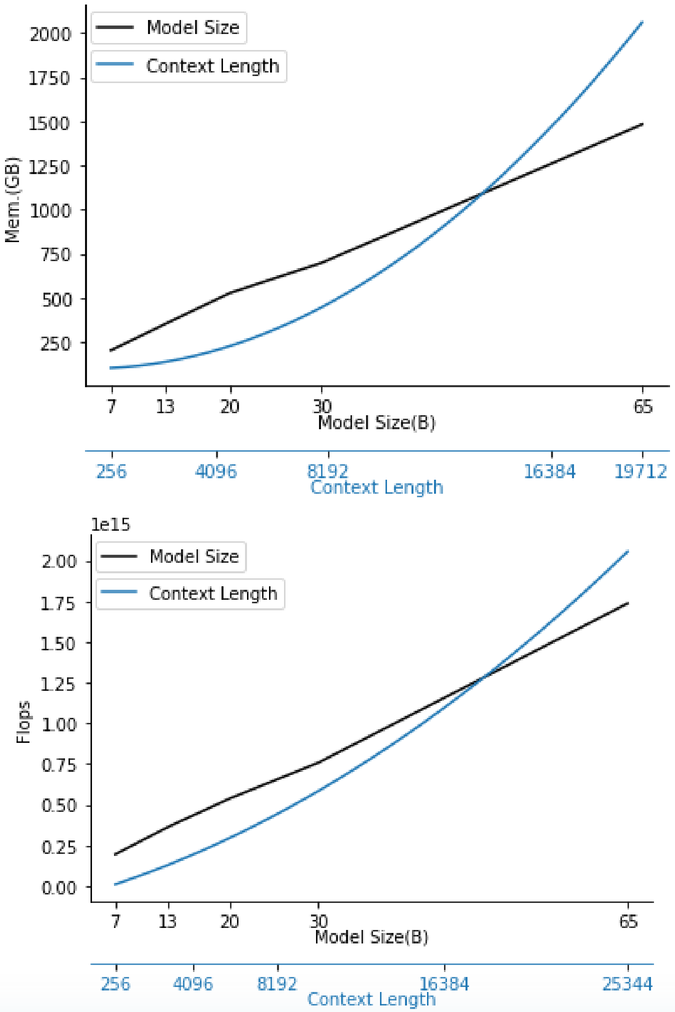
通信前将参数进行量化，减少通信量

ZeRO++: Extremely Efficient Collective Communication for Giant Model Training

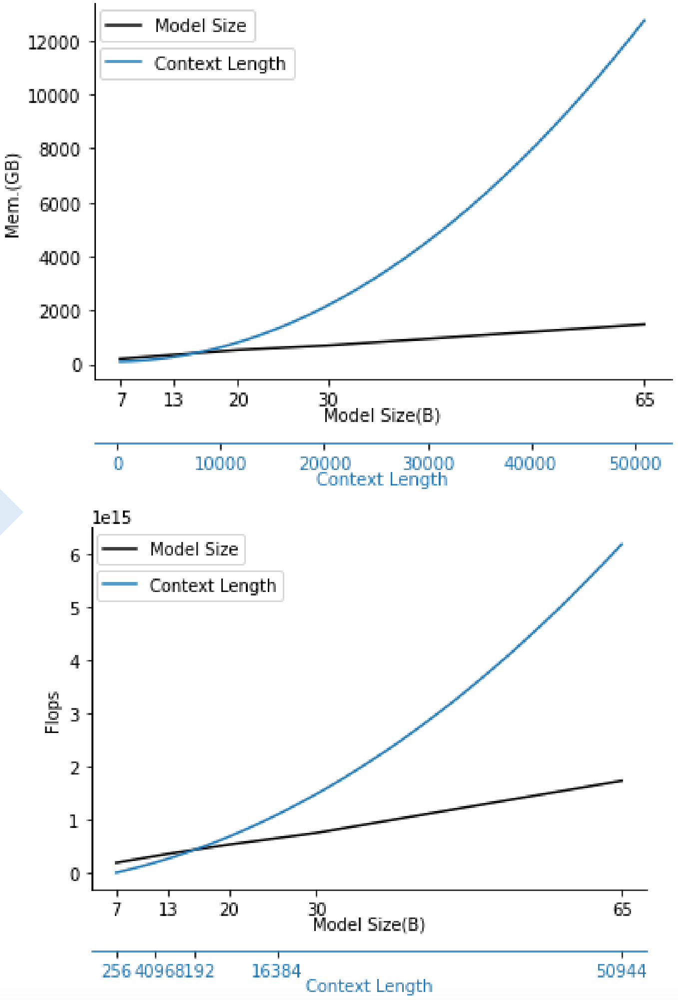
零冗余优化

类型	主要思想	备注
Zero-1	将优化器的状态进行切分	没有任何坏处，直接冲
Zero-2	切分优化器状态和梯度	对通信稍微有点要求
Zero-3	切分优化器、梯度和参数	对通信要求很高
Zero++	划分出多个Zero共享组，并使用量化减少通信量	对通信的要求比Zero-3低

序列并行--序之长，一卡放不下



当Context Length继续增加



黑色的线 是随着模型大小增加，训练需要的显存和Flops曲线

蓝色的线 是7B模型随着Context Length增加，训练需要的显存和Flops曲线

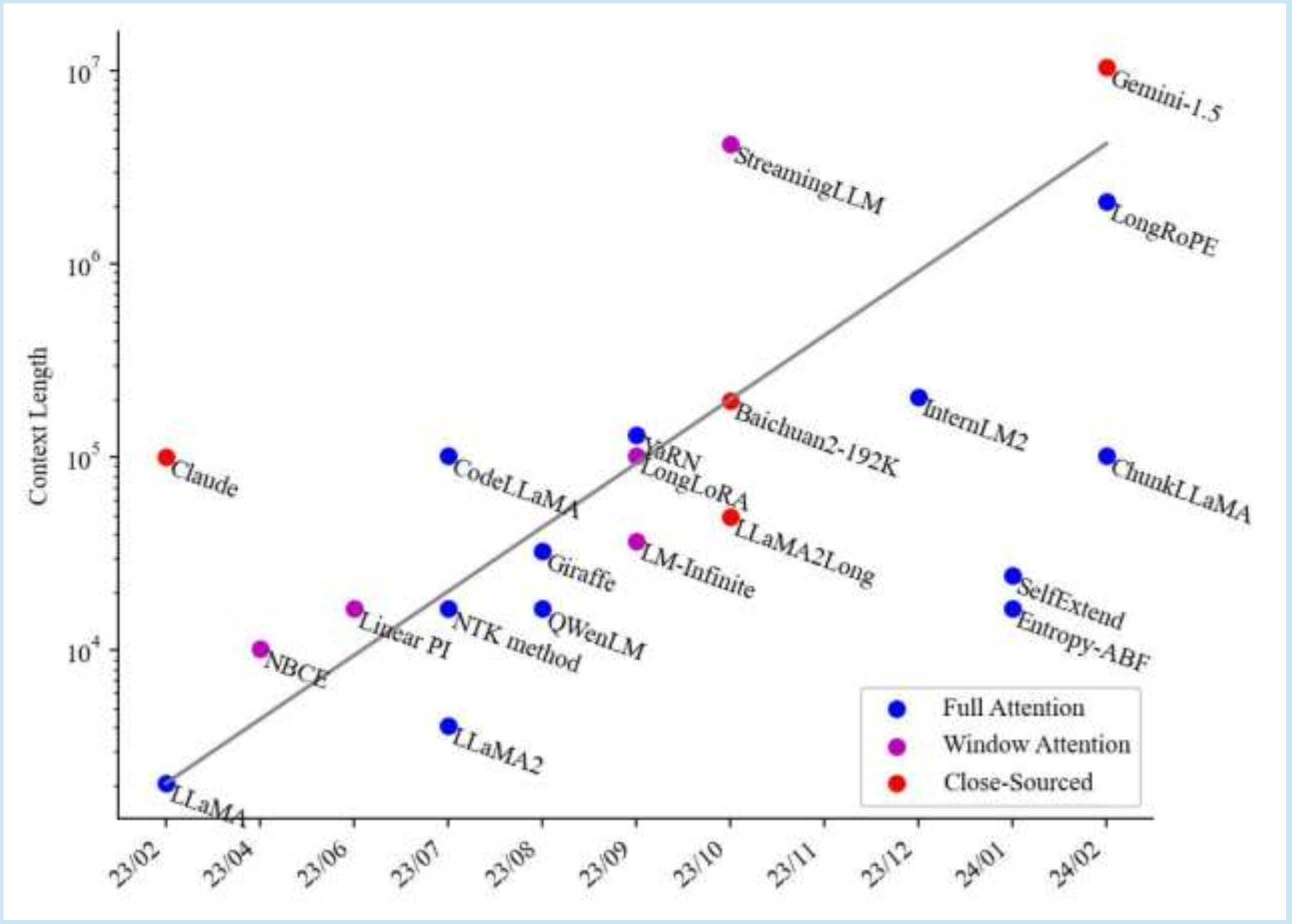
序列并行--长序列的“摩尔定律”

摩尔定律

- 每隔**18个月**芯片的性能提高一倍

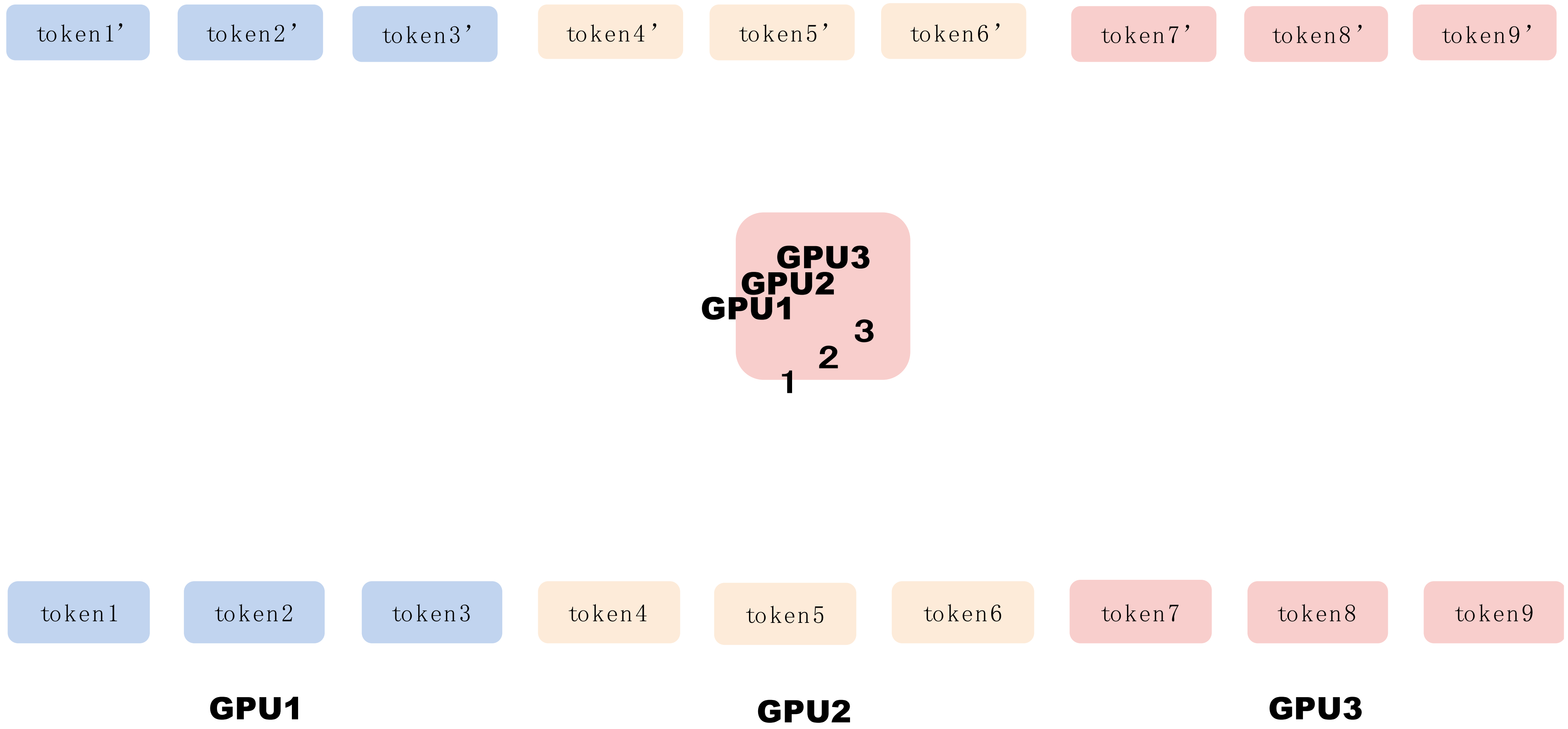
大模型语境长度?

- 每隔 **？ 个月**大模型的语境长度提高一倍



过去一年，大模型的语境长度约**每个月提高一倍**
截止目前，已经从2K飙升到**10M**，提高了约**4个数量级**

序列并行



token1’ token2’ token3’ token4’ token5’ token6’ token7’ token8’ token9’

GPU3
GPU2
GPU1
3
2
1

token1 token2 token3 token4 token5 token6 token7 token8 token9

GPU1 GPU2 GPU3

每张卡处理序列的一部分，在多头注意力的位置，head被均分到GPU上，但每个head都处理整个序列

缺点：并行的数量受限于head的数量

序列并行—另一个维度

假设以一个query的计算为例

$$o_i = \sum_{n=1}^{9} \frac{e^{q_i k_n} v_n}{\sum_{j=1}^{9} e^{q_i k_j}}$$

各部分m并行执行

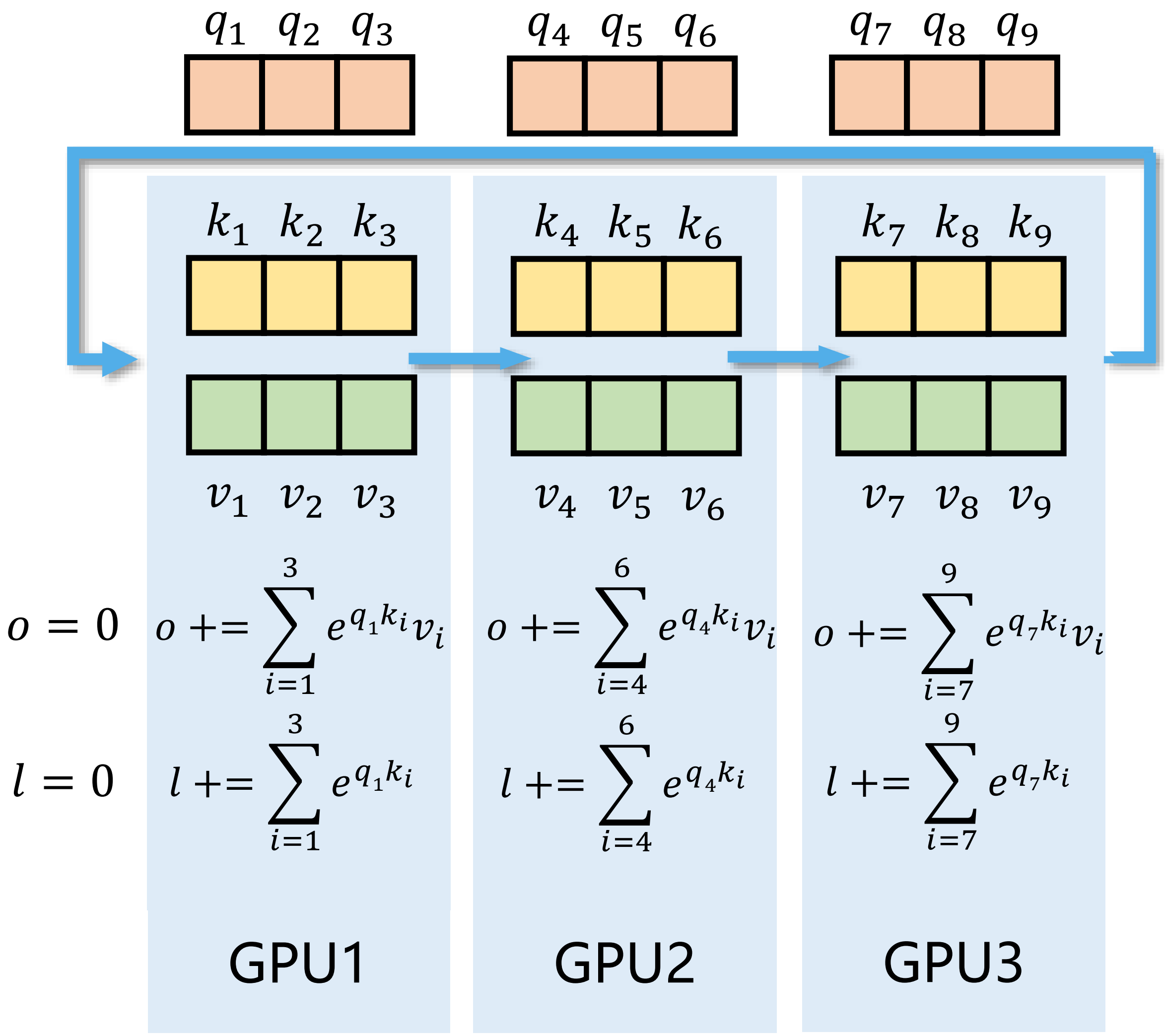
$$o_i^m = e^{q_i k_n} v_n + o_i^m$$

$$s_i^m = e^{q_i k_n} + s_i^m$$

合并执行

$$o_i = \frac{\sum_{m=1}^{M} o_i^m}{\sum_{m=1}^{M} s_i^m}$$

Ring Attention



$o = 0$

$l = 0$

GPU1:

$$o \mathrel{+}= \sum_{i=1}^{3} e^{q_1 k_i} v_i$$

$$l \mathrel{+}= \sum_{i=1}^{3} e^{q_1 k_i}$$

GPU2:

$$o \mathrel{+}= \sum_{i=4}^{6} e^{q_4 k_i} v_i$$

$$l \mathrel{+}= \sum_{i=4}^{6} e^{q_4 k_i}$$

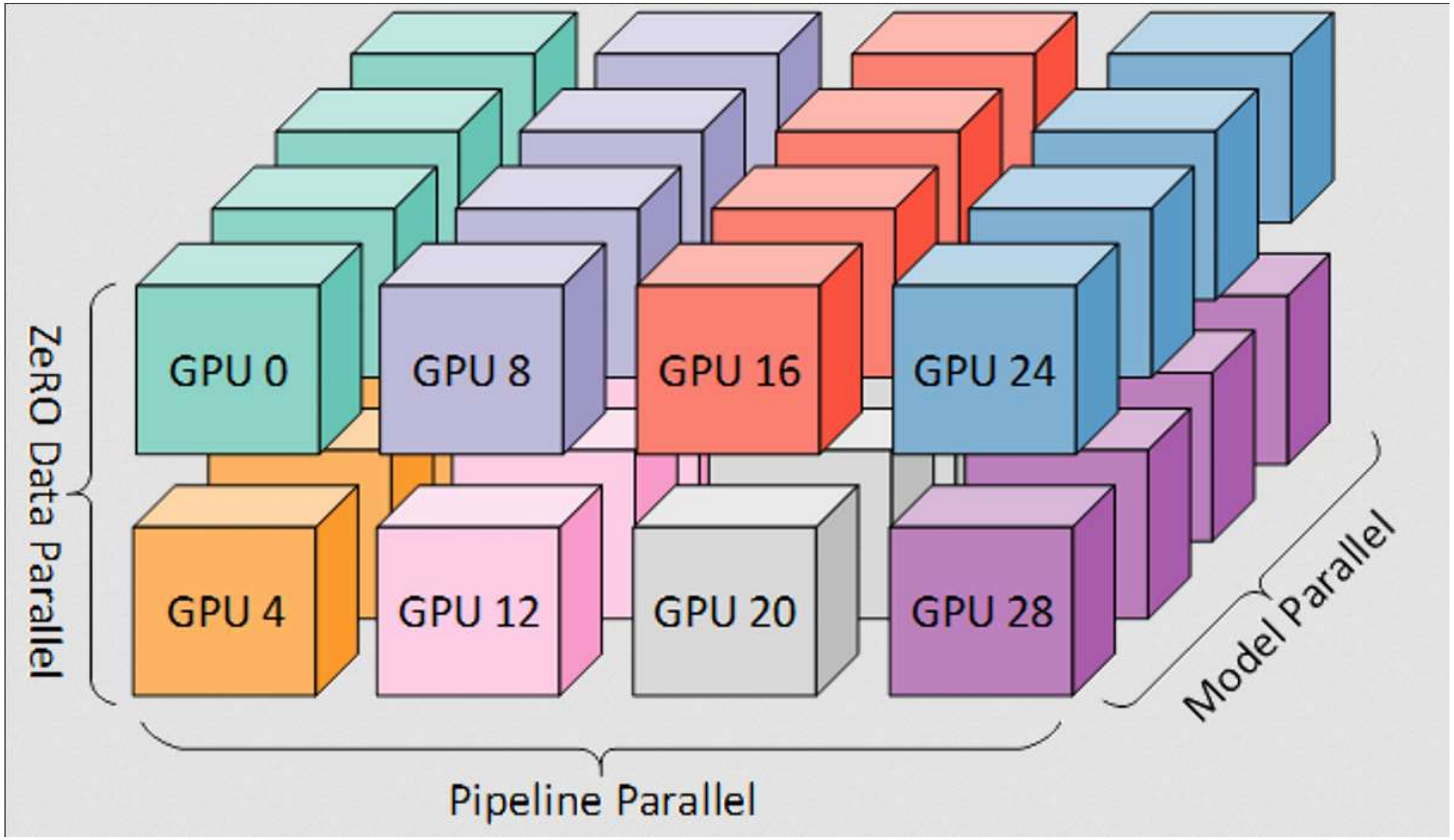
GPU3:

$$o \mathrel{+}= \sum_{i=7}^{9} e^{q_7 k_i} v_i$$

$$l \mathrel{+}= \sum_{i=7}^{9} e^{q_7 k_i}$$

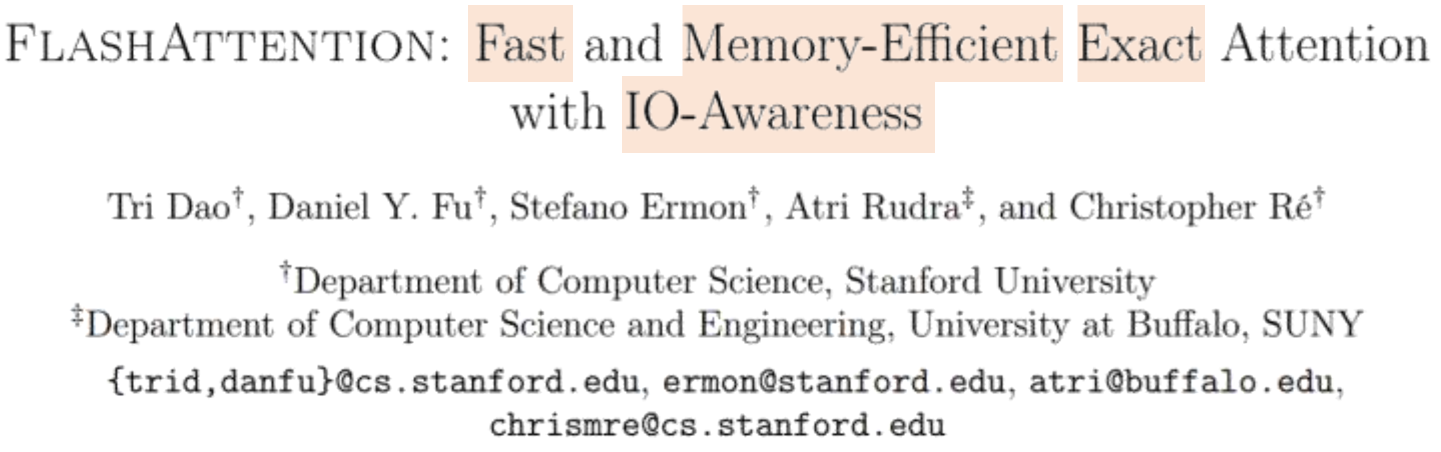
Ring Attention with Blockwise Transformers for Near-Infinite Context

实际训练会组合多种并行方式



| FlashAttention的原理

FlashAttention1的动机

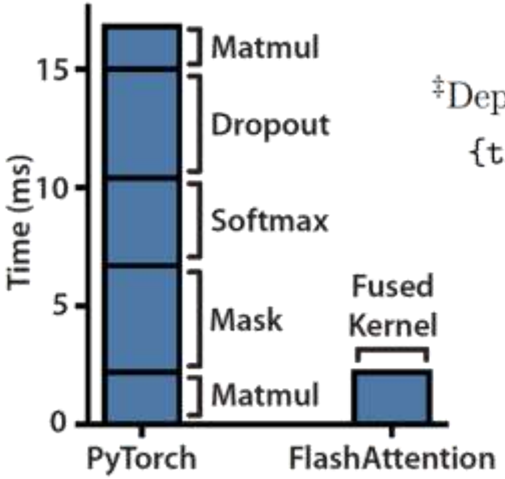

FLASHATTENTION: Fast and Memory-Efficient Exact Attention
with IO-Awareness

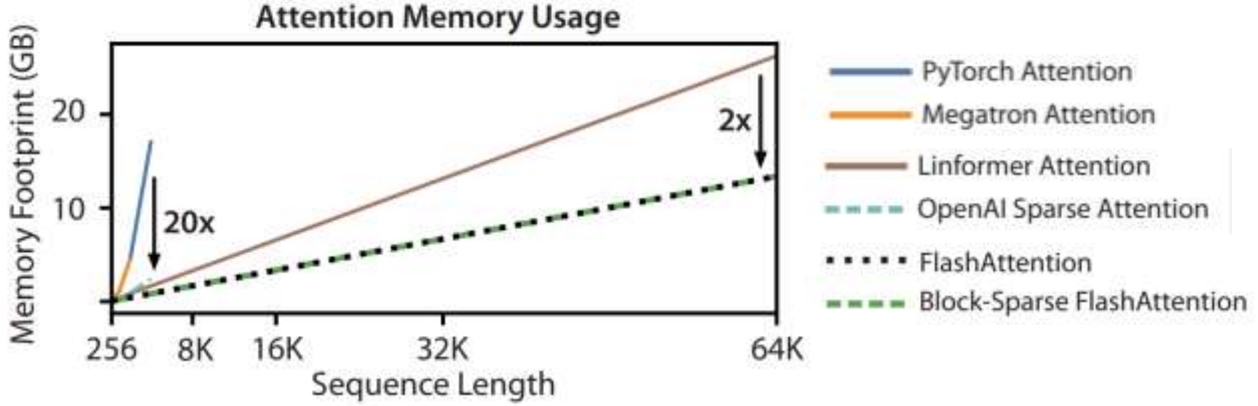
Tri Dao[†], Daniel Y. Fu[†], Stefano Ermon[†], Atri Rudra[‡], and Christopher Ré[†]

[†]Department of Computer Science, Stanford University
[‡]Department of Computer Science and Engineering, University at Buffalo, SUNY
{trid,danfu}@cs.stanford.edu, ermon@stanford.edu, atri@buffalo.edu,
chrismre@cs.stanford.edu

名称含义

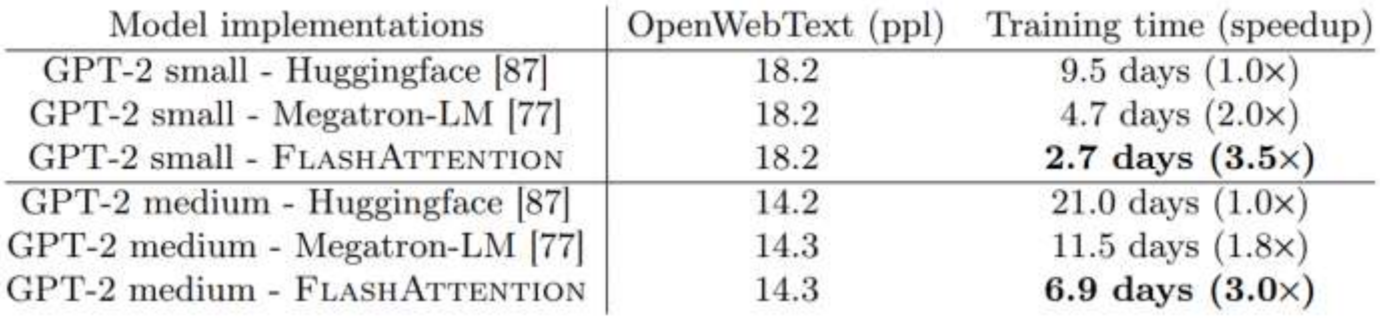
- Fast：耗时更短
  - 五个算子合成一个
- Memory-efficient
  - 极大降低存储开销
  - 实现超长序列输入
- Exact：不同于传统方法，没有任何近似
- IO aware：硬件角度加速（减少读写）





背景工作：传统注意力加速研究

- 稀疏方法：SparseTranformer ......
- 低秩方法：Linformer、Performer ...
- 方法局限：降低flops，并不是去提升“真正的速度”



Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FLASHATTENTION	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FLASHATTENTION	14.3	**6.9 days (3.0×)**

Training time reported on 8×A100s GPUs

FlashAttention1的动机

背景知识：GPU存储结构

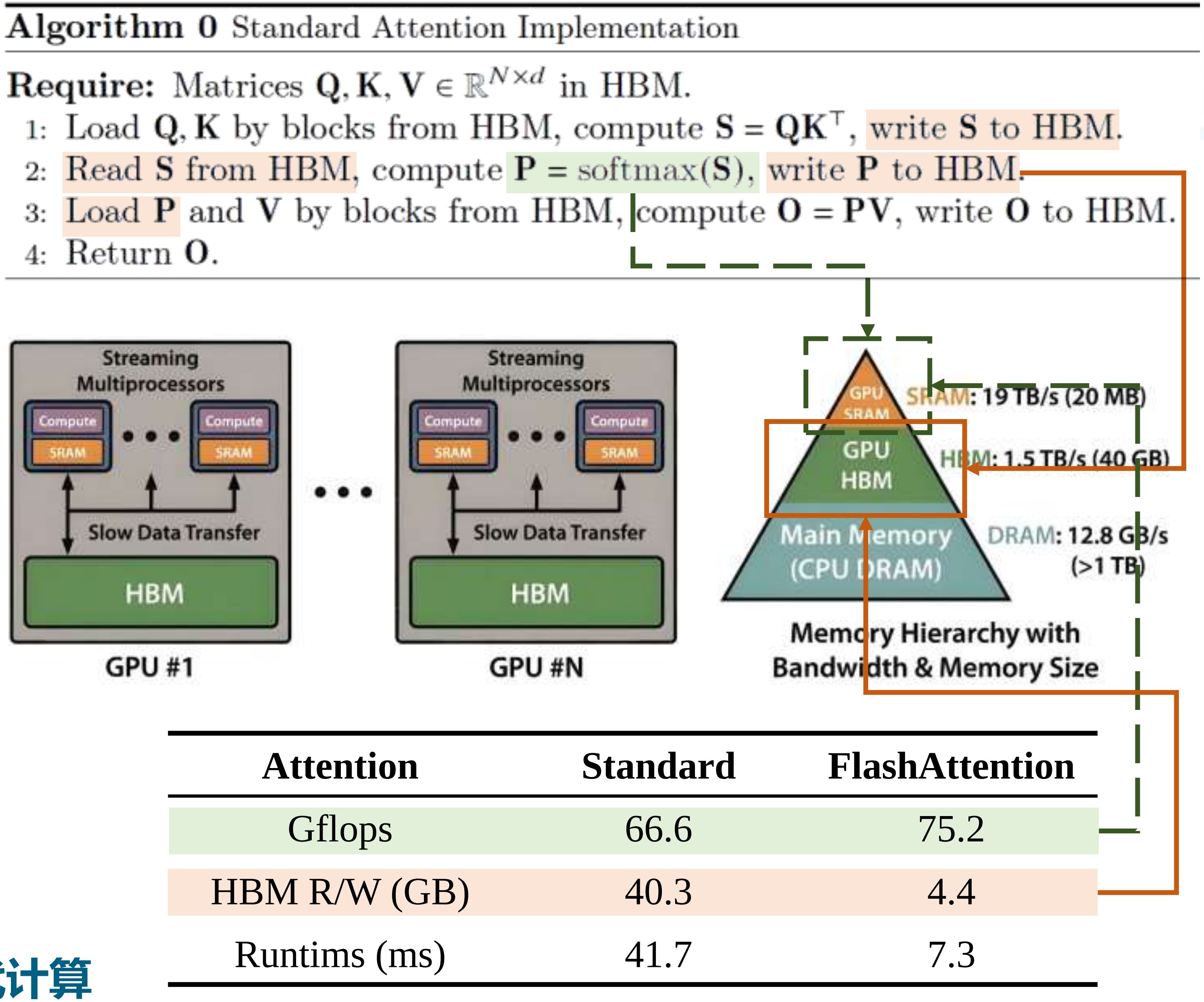
- SRAM：静态随机存储器
  - 存得少、算得快、类似高速缓存
- HBM：高带宽存储器
  - 存得多、算得慢、类似内存

核心思想：抓住主要矛盾

- 更多flops，充分利用SRAM效率
- 更少IO，减少不必要的读写开销

实现思路：经典方法的大胆组合

- 参考：on-line softmax
  - 将softmax算子从**分步计算**变成**迭代计算**



Algorithm 0 Standard Attention Implementation

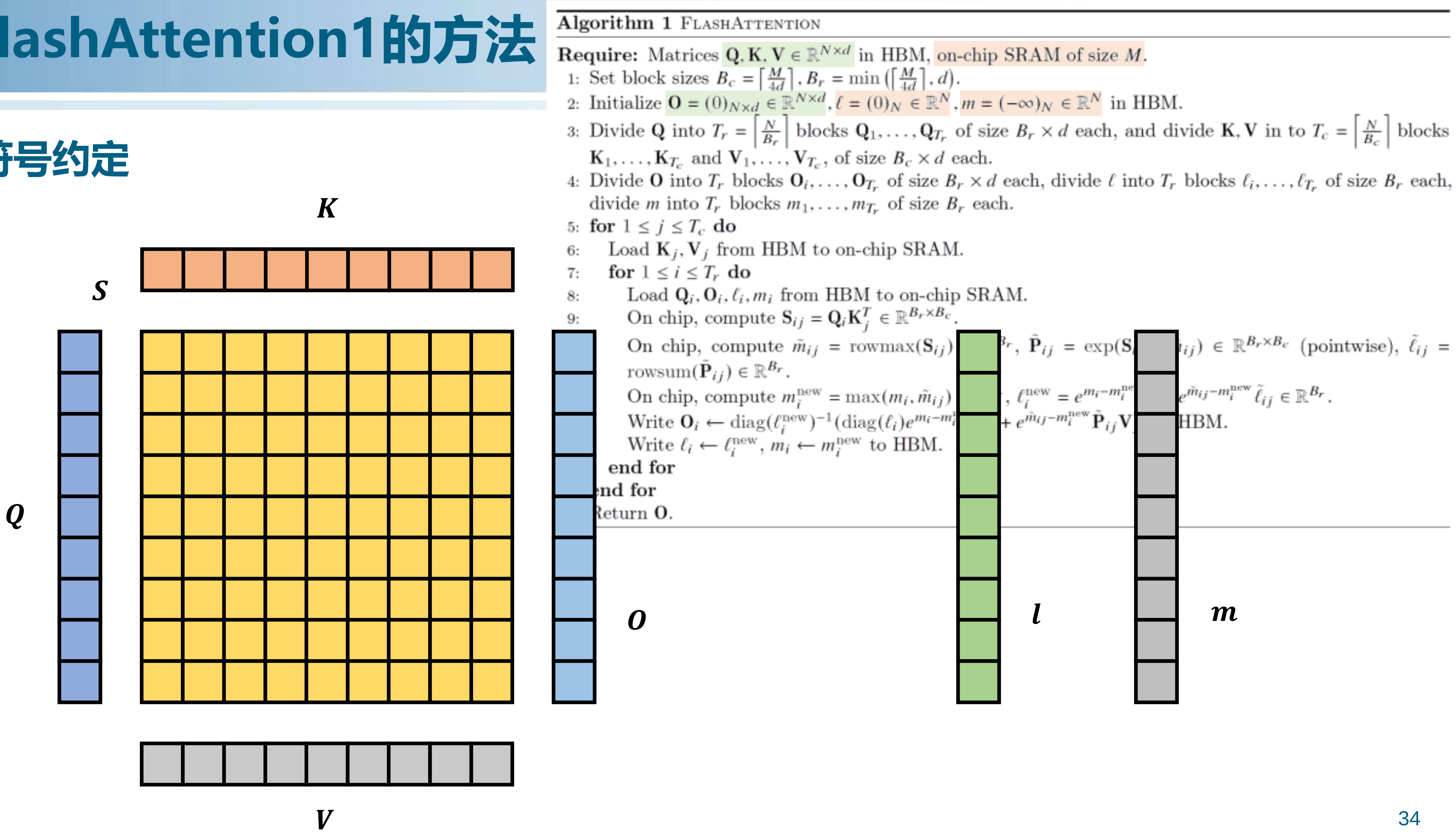
```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.
Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
Return $\mathbf{O}$.
```

Attention	Standard	FlashAttention
Gflops	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtims (ms)	41.7	7.3

- 方法：分块 tiling（前向+反向）、重计算 recomputation（仅反向，略）

FlashAttention1的方法

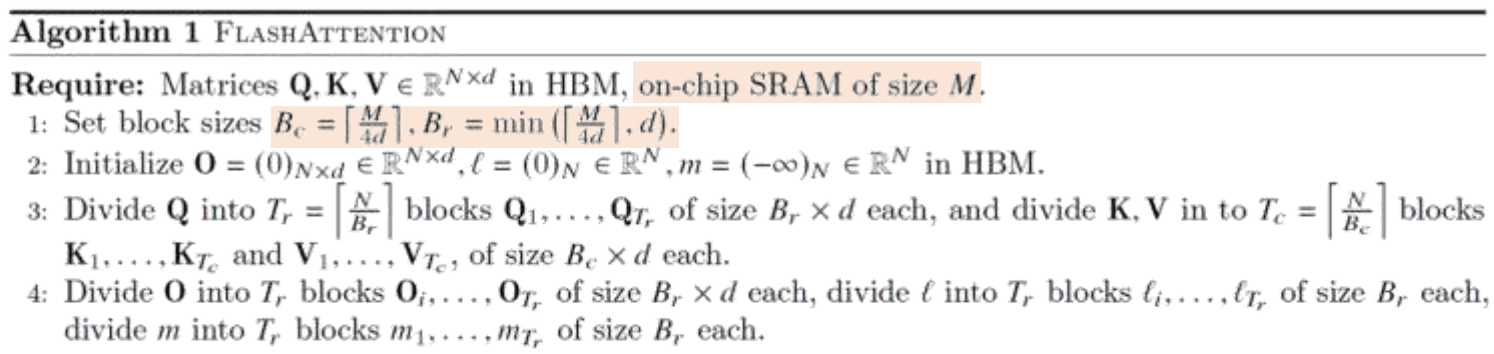
符号约定



Algorithm 1 FLASHATTENTION

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, on-chip SRAM of size $M$.
Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
Initialize $\mathbf{O} = (0)_{N\times d} \in \mathbb{R}^{N\times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
Divide $\mathbf{Q}$ into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \lceil \frac{N}{B_c} \rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide $\mathbf{O}$ into $T_r$ blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\ell$ into $T_r$ blocks $\ell_i, \dots, \ell_{T_r}$ of size $B_r$ each, divide $m$ into $T_r$ blocks $m_1, \dots, m_{T_r}$ of size $B_r$ each.
for $1 \le j \le T_c$ do
  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
  for $1 \le i \le T_r$ do
    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r\times B_c}$.
    On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij})$ $^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}$ $_{ij}) \in \mathbb{R}^{B_r\times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
    On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij})$ , $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{ne}}}$ $e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
    Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i)e^{m_i - m_i}$ $+ e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}$ HBM.
    Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
  end for
nd for
eturn $\mathbf{O}$.
```

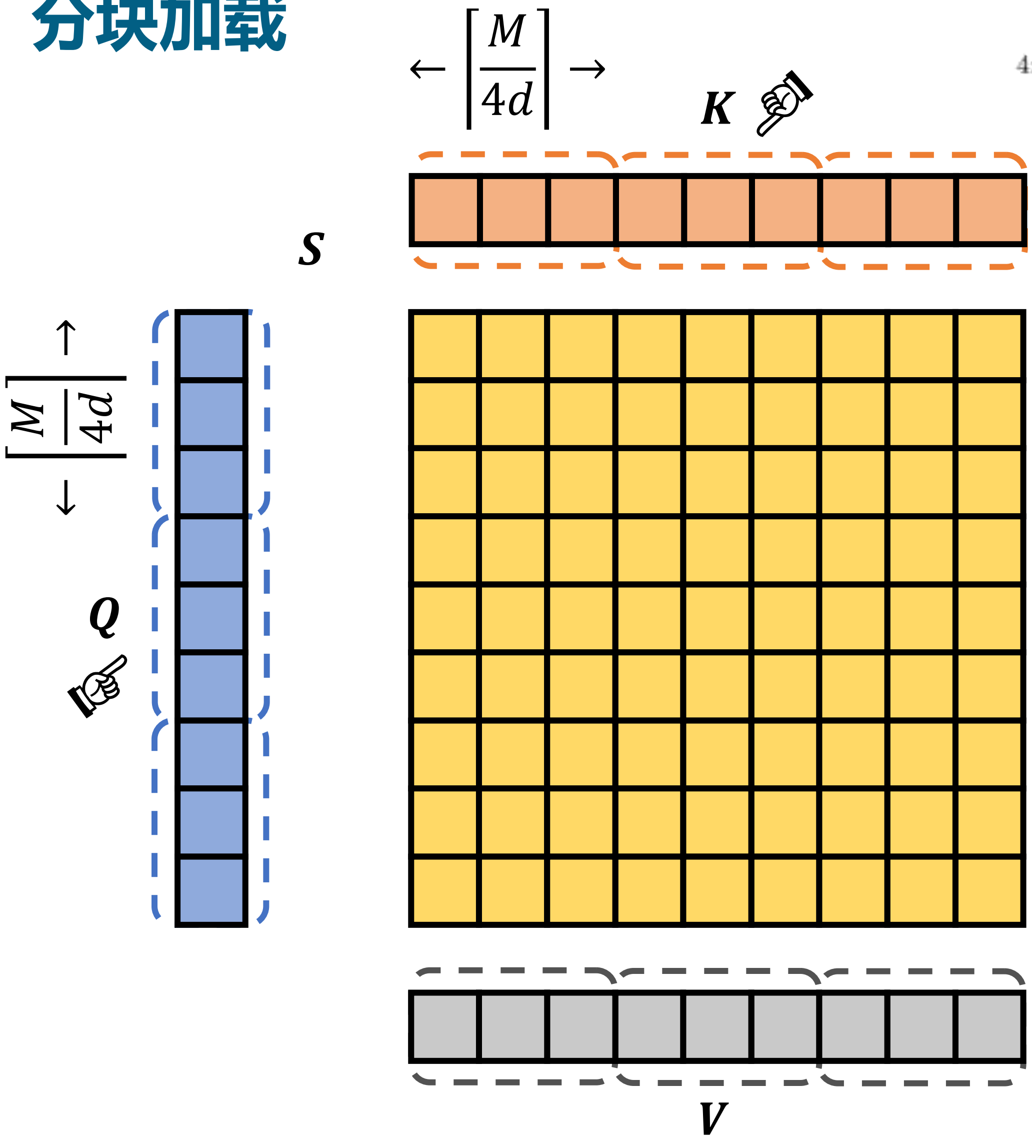
FlashAttention1的方法

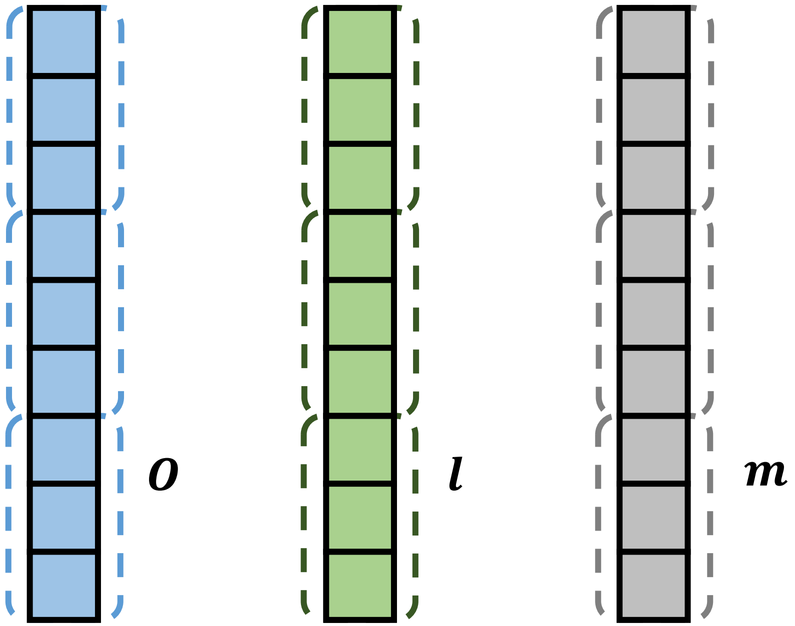


Algorithm 1 FLASHATTENTION

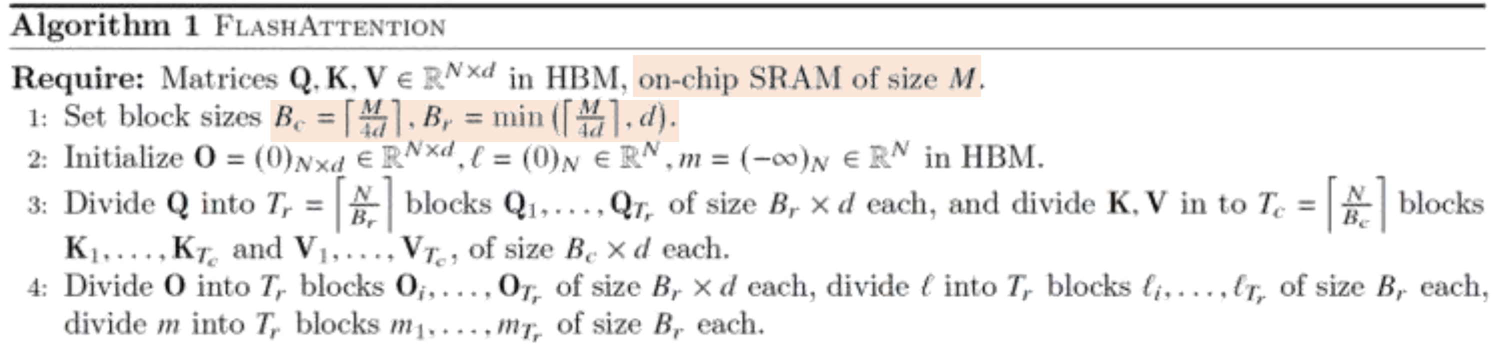
```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, on-chip SRAM of size $M$.
Set block sizes $B_c = \lceil \frac{M}{4d} \rceil, B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
Initialize $\mathbf{O} = (0)_{N\times d} \in \mathbb{R}^{N\times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
Divide $\mathbf{Q}$ into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \lceil \frac{N}{B_c} \rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide $\mathbf{O}$ into $T_r$ blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\ell$ into $T_r$ blocks $\ell_i, \dots, \ell_{T_r}$ of size $B_r$ each, divide $m$ into $T_r$ blocks $m_1, \dots, m_{T_r}$ of size $B_r$ each.
```

分块加载



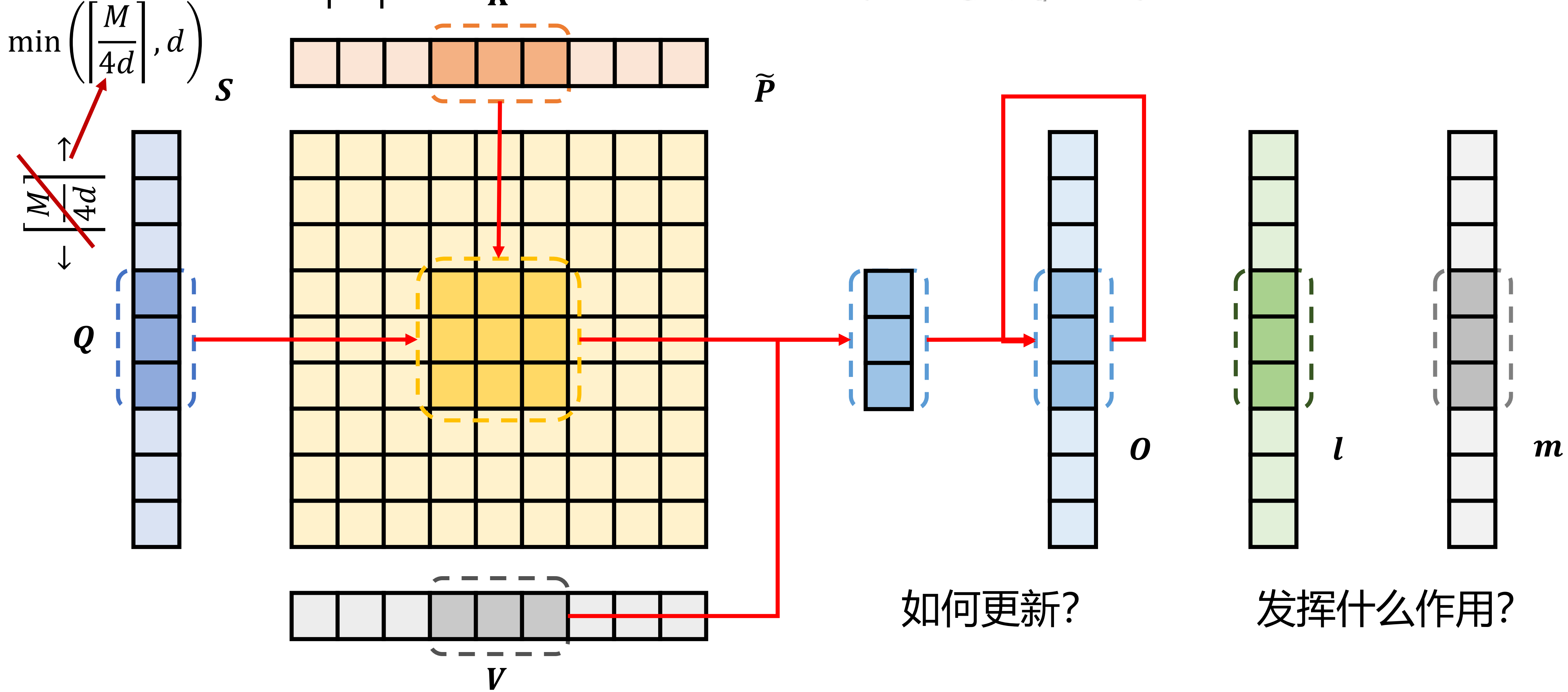


FlashAttention1的方法



```
Algorithm 1 FlashAttention
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size $M$.
  Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.
  Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
  Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
  Divide $\mathbf{O}$ into $T_r$ blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\ell$ into $T_r$ blocks $\ell_i, \dots, \ell_{T_r}$ of size $B_r$ each, divide $m$ into $T_r$ blocks $m_1, \dots, m_{T_r}$ of size $B_r$ each.
```

分块加载

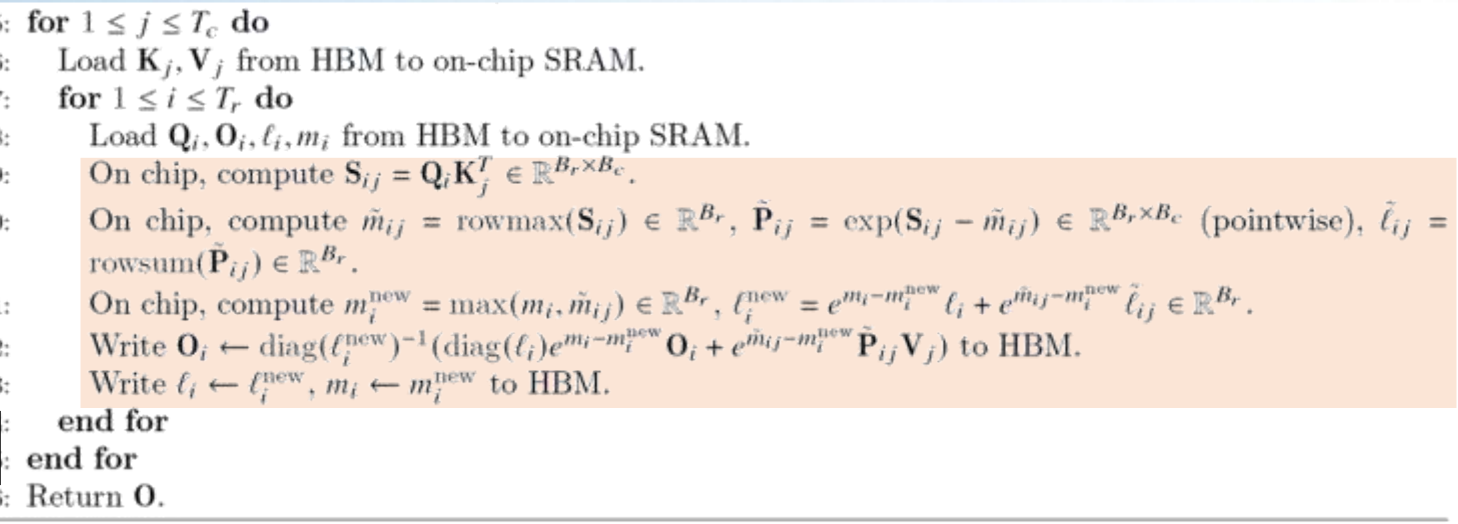


如何更新?

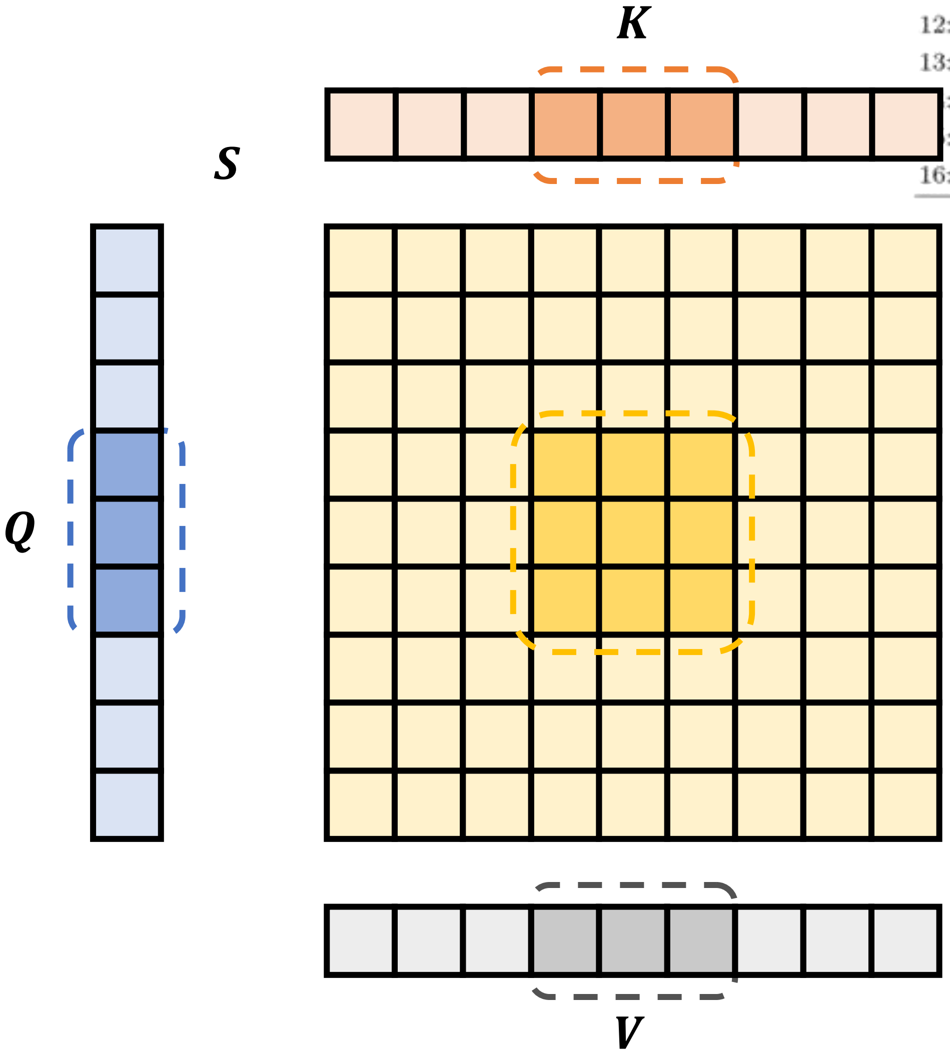
发挥什么作用?

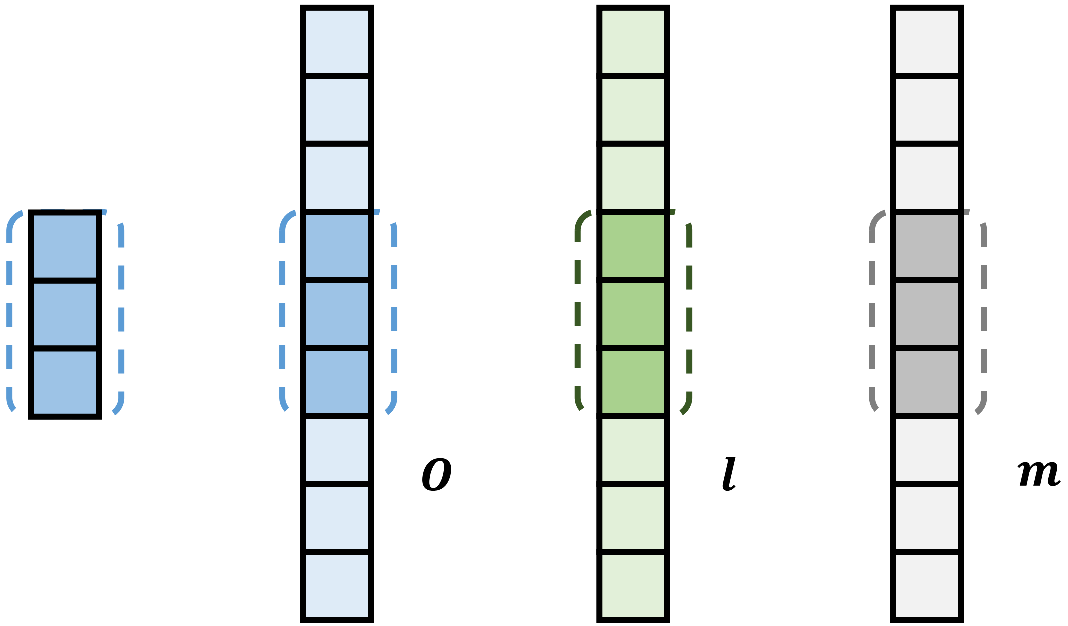
FlashAttention1的方法

softmax分解

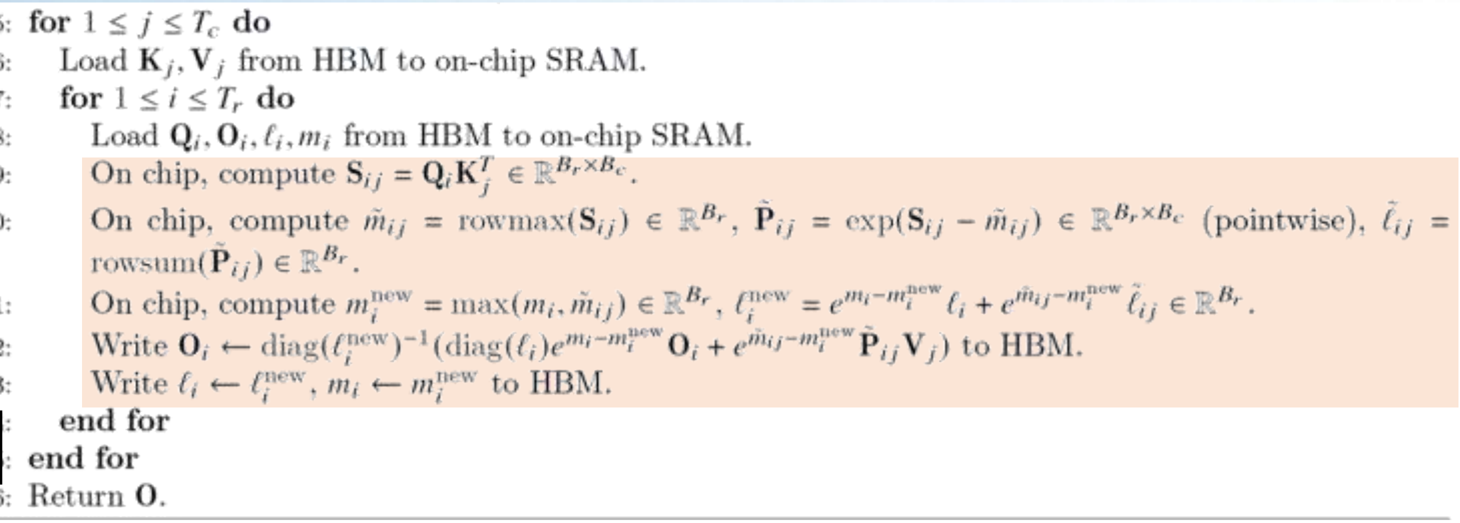


```
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i\mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
        On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}}\ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}}\tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i)e^{m_i - m_i^{\mathrm{new}}}\mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}}\tilde{\mathbf{P}}_{ij}\mathbf{V}_j)$ to HBM.
        Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
    end for
end for
Return $\mathbf{O}$.
```



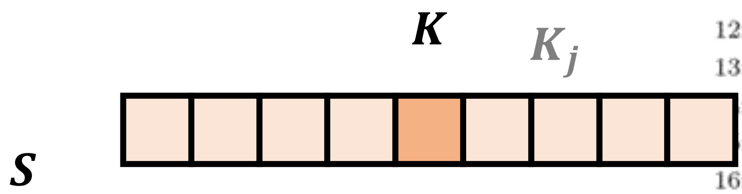


FlashAttention1的方法



```
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i\mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
        On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}}\ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}}\tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i)e^{m_i - m_i^{\mathrm{new}}}\mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}}\tilde{\mathbf{P}}_{ij}\mathbf{V}_j)$ to HBM.
        Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
    end for
end for
Return $\mathbf{O}$.
```

softmax分解（迭代分解）

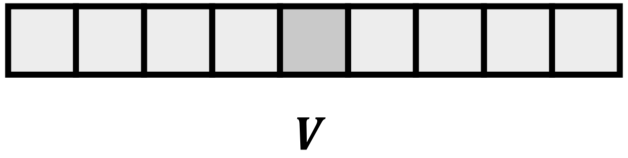


$$O_i = \frac{e^{Q_iK_1^T} \cdot V_1}{\sum_{s=1}^{N} e^{Q_iK_s^T}} + \cdots + \frac{e^{Q_iK_N^T} \cdot V_N}{\sum_{s=1}^{N} e^{Q_iK_s^T}} =$$

online softmax $$O_1 = \frac{e^{Q_iK_1^T} \cdot V_1}{\sum_{s=1}^{1} e^{Q_iK_s^T}} \quad O_j = O_{j-1} \cdot \frac{\sum_{s=1}^{j-1} e^{Q_iK_s^T}}{\sum_{s=1}^{j} e^{Q_iK_s^T}} + \frac{e^{Q_iK_j^T} \cdot V_j}{\sum_{s=1}^{j} e^{Q_iK_s^T}}$$

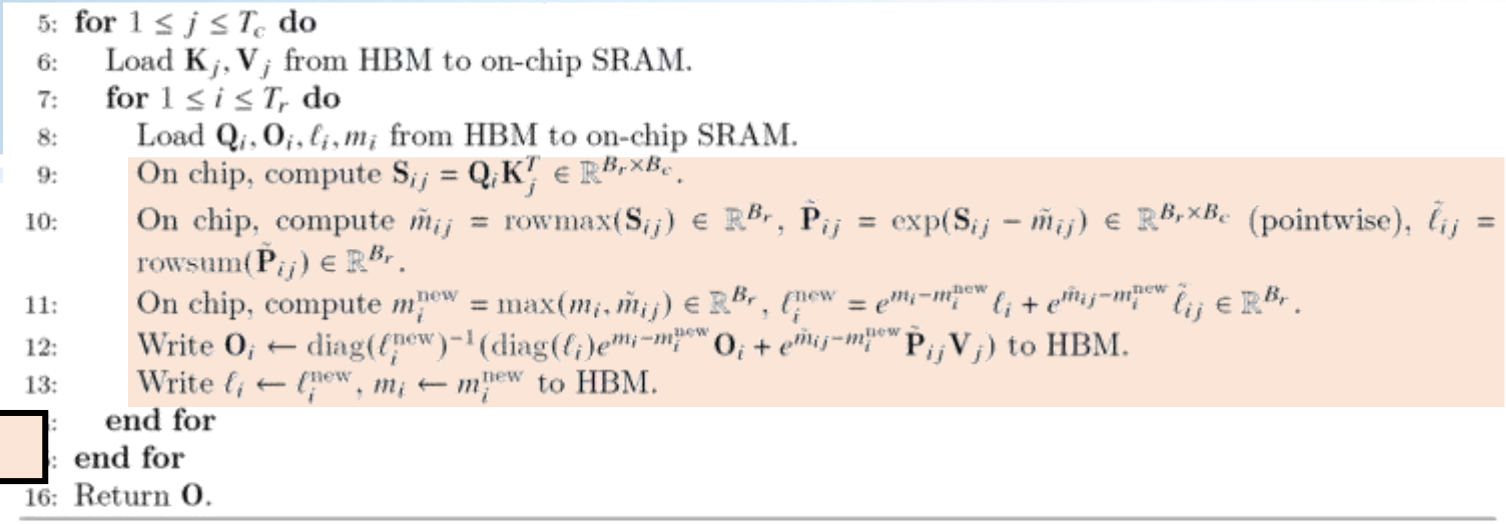
把softmax变成一个可以迭代完成的过程

把softmax+线性组合变成一个独立的算子

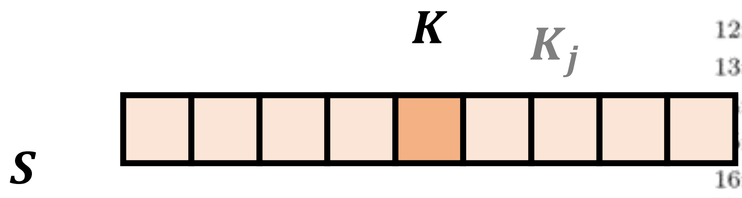


FlashAttention1的方法

softmax分解（公式简化）



```
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i\mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
        On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}}\ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}}\tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i)e^{m_i - m_i^{\mathrm{new}}}\mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}}\tilde{\mathbf{P}}_{ij}\mathbf{V}_j)$ to HBM.
        Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
    end for
end for
Return $\mathbf{O}$.
```



$$O_i = \frac{e^{Q_iK_1^T}\cdot V_1}{\sum_{s=1}^{N} e^{Q_iK_s^T}} + \cdots + \frac{e^{Q_iK_N^T}\cdot V_N}{\sum_{s=1}^{N} e^{Q_iK_s^T}} = \frac{e^{Q_iK_1^T}\cdot V_1}{\sum_{s=1}^{1} e^{Q_iK_s^T}}\cdot\frac{\sum_{s=1}^{1} e^{Q_iK_s^T}}{\sum_{s=1}^{N} e^{Q_iK_s^T}} + \cdots + \frac{e^{Q_iK_j^T}\cdot V_j}{\sum_{s=1}^{j} e^{Q_iK_s^T}}\cdot\frac{\sum_{s=1}^{j} e^{Q_iK_s^T}}{\sum_{s=1}^{N} e^{Q_iK_s^T}} + \cdots + \frac{e^{Q_iK_N^T}\cdot V_N}{\sum_{s=1}^{N} e^{Q_iK_s^T}}\cdot\frac{\sum_{s=1}^{N} e^{Q_iK_s^T}}{\sum_{s=1}^{N} e^{Q_iK_s^T}}$$

online softmax

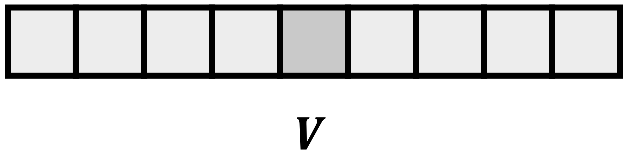
$$O_1 = 1\cdot V_1 \qquad O_j = O_{j-1}\cdot\frac{l_{ij-1}}{l_{ij}} + \frac{e^{Q_iK_j^T}\cdot V_j}{l_{ij}}$$

$$l_{i1} = e^{Q_iK_1^T}$$

$$l_{ij} = l_{ij-1} + e^{Q_iK_j^T}$$

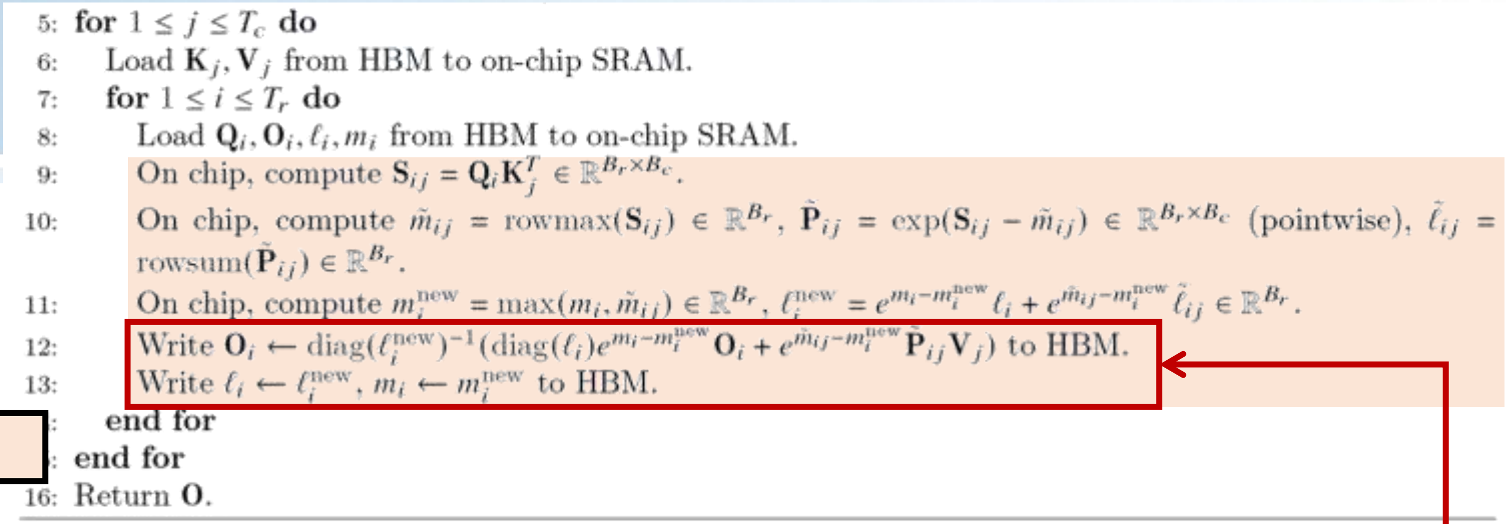
记录每行
到目前的
归一化项

$$l_{ij} = \sum_{s=1}^{j} e^{Q_iK_s^T}$$

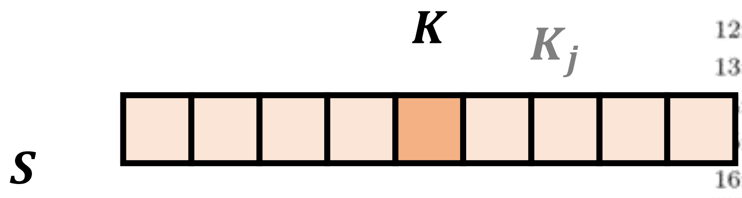


FlashAttention1的方法

softmax分解（公式简化）



```
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
        On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
        Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
    end for
end for
Return $\mathbf{O}$.
```



$$O_i = \frac{e^{Q_iK_1^T} \cdot V_1}{\sum_{s=1}^{N} e^{Q_iK_s^T}} + \cdots + \frac{e^{Q_iK_N^T} \cdot V_N}{\sum_{s=1}^{N} e^{Q_iK_s^T}} = \frac{e^{Q_iK_1^T} \cdot V_1}{\sum_{s=1}^{1} e^{Q_iK_s^T}} \cdot \frac{\sum_{s=1}^{1} e^{Q_iK_s^T}}{\sum_{s=1}^{N} e^{Q_iK_s^T}} + \cdots + \frac{e^{Q_iK_j^T} \cdot V_j}{\sum_{s=1}^{j} e^{Q_iK_s^T}} \cdot \frac{\sum_{s=1}^{j} e^{Q_iK_s^T}}{\sum_{s=1}^{N} e^{Q_iK_s^T}} + \cdots + \frac{e^{Q_iK_N^T} \cdot V_N}{\sum_{s=1}^{N} e^{Q_iK_s^T}} \cdot \frac{\sum_{s=1}^{N} e^{Q_iK_s^T}}{\sum_{s=1}^{N} e^{Q_iK_s^T}}$$

记录每行到目前的归一化项

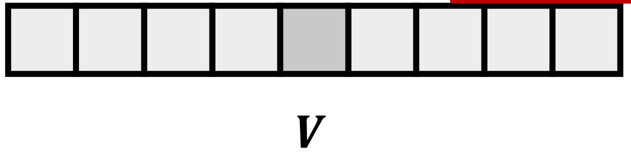
记录每行目前最大的attn score

online softmax

$$O_1 = V_1$$
$$m_{i1} = S_{i1}$$
$$l_{i1} = e^{S_{i1} - m_{i1}}$$

$$O_i = O_i^{\mathrm{old}} \cdot \frac{l_i^{\mathrm{old}}}{l_i} \cdot \frac{e^{m_i^{\mathrm{old}}}}{e^{m_i}} + \frac{e^{S_i - m_i} \cdot V_j}{l_i}$$
$$m_i = \max(m_i^{\mathrm{old}}, S_i)$$
$$l_i = l_i^{\mathrm{old}} \cdot e^{m_i^{\mathrm{old}} - m_i} + e^{S_i - m_i}$$

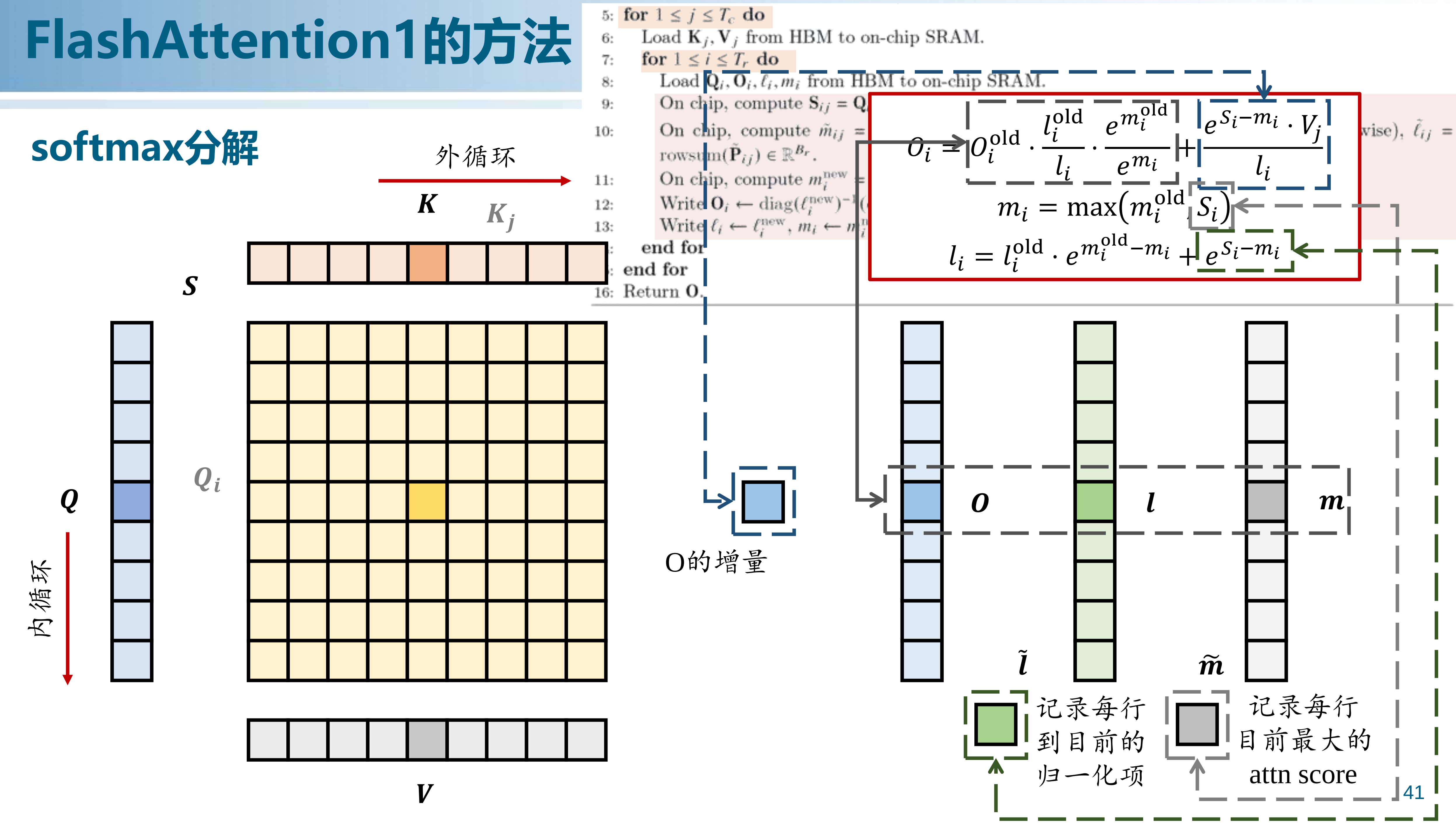
$$_{ij} = \sum_{s=1}^{j} e^{S_{is} - m_{ij}} \quad m_{ij} = \max_{1 \le s \le j} Q_iK_s^T$$



$$S_{ij} = Q_iK_j^T$$

FlashAttention1的方法

softmax分解



```
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}$
        On chip, compute $\tilde{m}_{ij} =$ wise), $\tilde{\ell}_{ij} =$
        rowsum($\tilde{\mathbf{P}}_{ij}$) $\in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\text{new}} =$
        Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1}($
        Write $\ell_i \leftarrow \ell_i^{\text{new}}, m_i \leftarrow m_i^{\text{n}}$
    end for
end for
Return $\mathbf{O}$.
```

$$O_i = O_i^{\text{old}} \cdot \frac{l_i^{\text{old}}}{l_i} \cdot \frac{e^{m_i^{\text{old}}}}{e^{m_i}} + \frac{e^{S_i - m_i} \cdot V_j}{l_i}$$

$$m_i = \max\left(m_i^{\text{old}}, S_i\right)$$

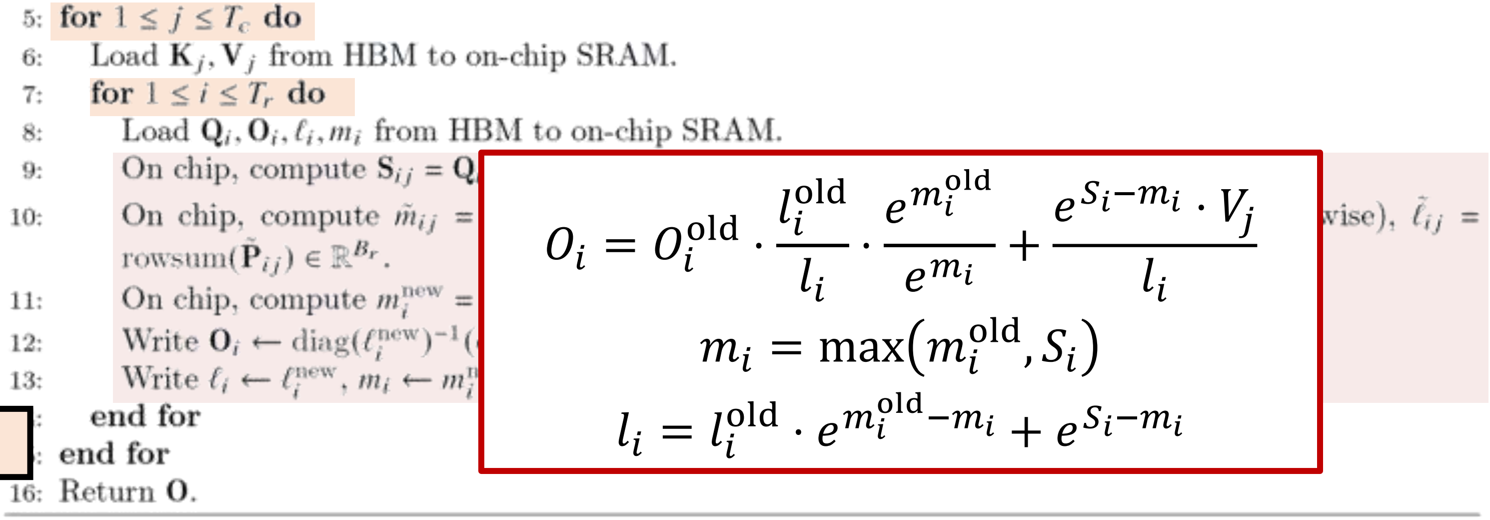
$$l_i = l_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + e^{S_i - m_i}$$

记录每行到目前的归一化项

记录每行目前最大的attn score

FlashAttention1的方法

分块更新

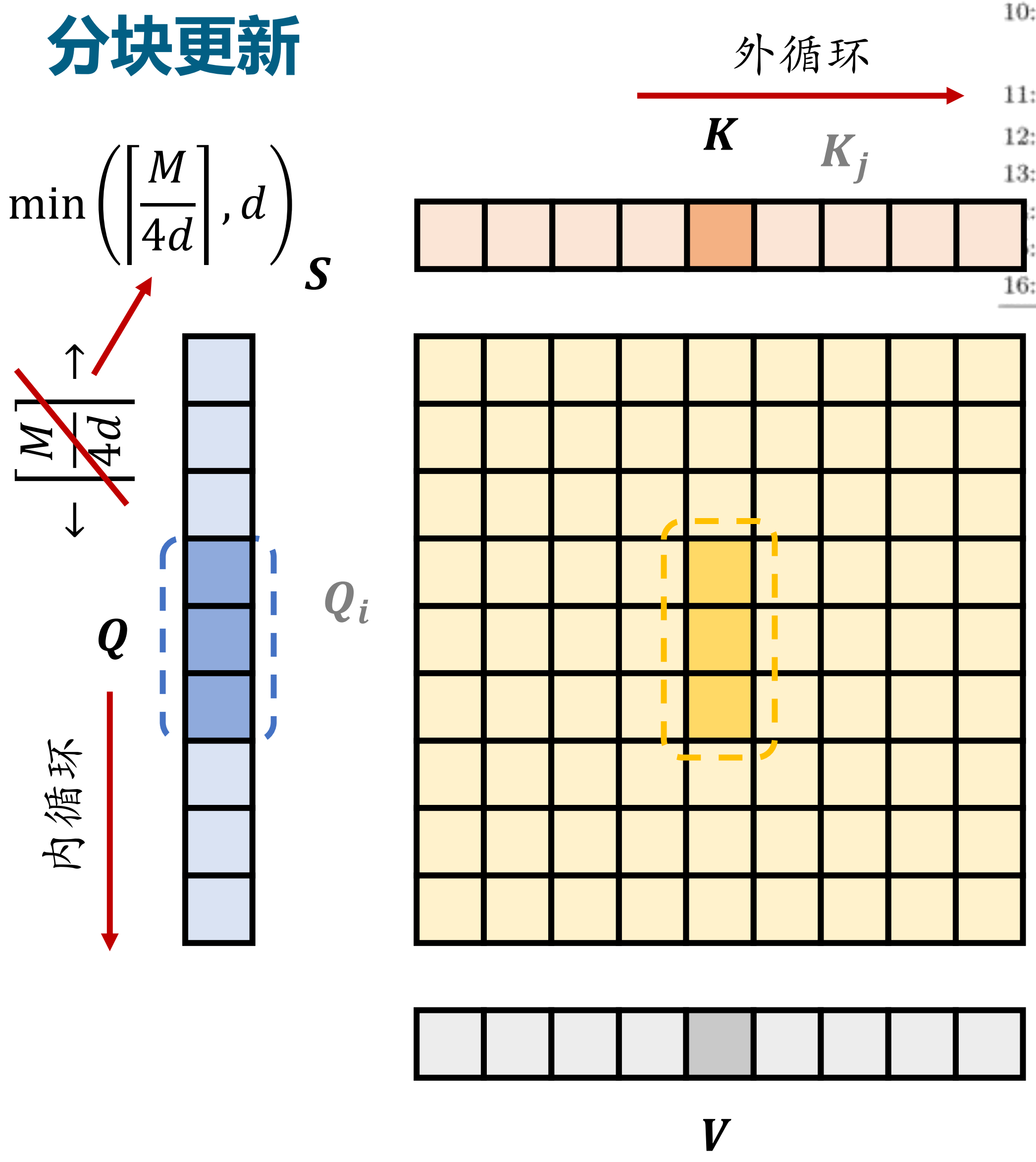


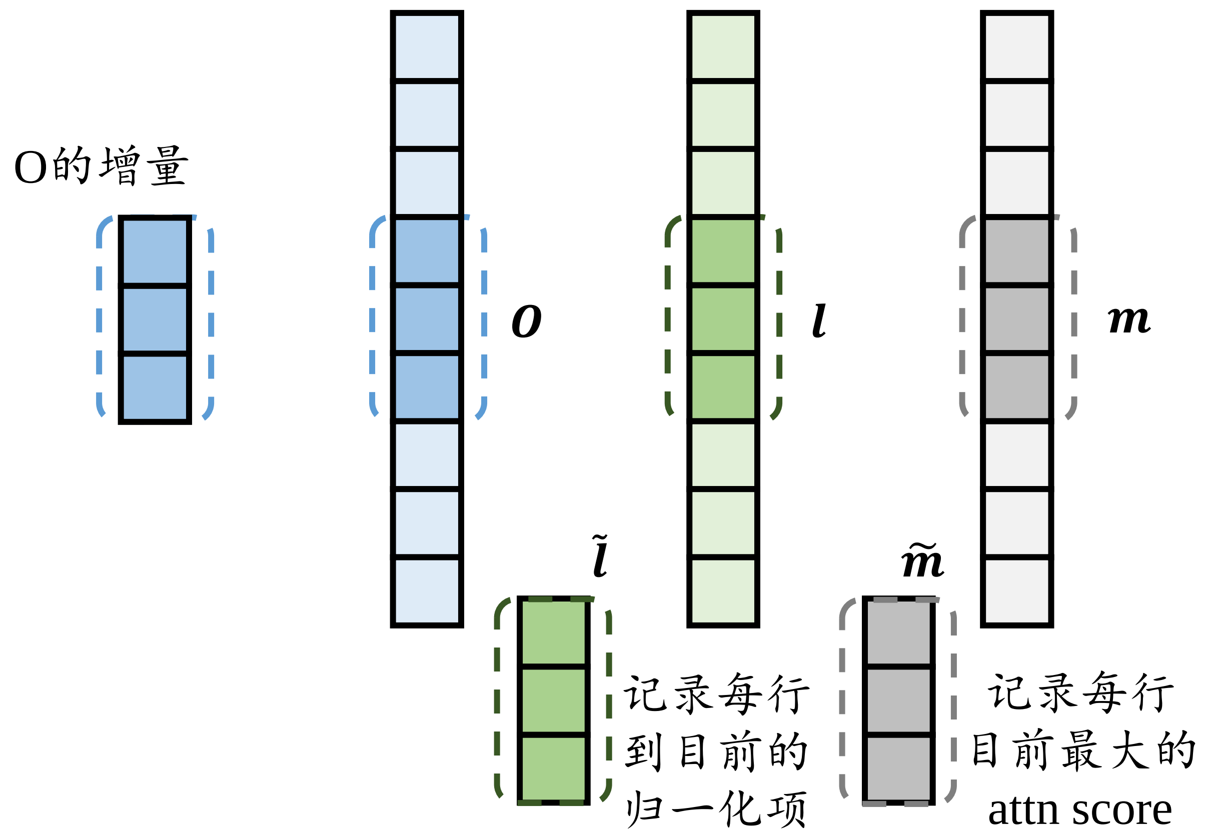
```
for $1 \le j \le T_c$ do
  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
  for $1 \le i \le T_r$ do
    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}$
    On chip, compute $\tilde{m}_{ij} =$ wise), $\tilde{\ell}_{ij} =$ rowsum($\tilde{\mathbf{P}}_{ij}$) $\in \mathbb{R}^{B_r}$.
    On chip, compute $m_i^{\text{new}} =$
    Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1}($
    Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{n}}$
  end for
end for
Return $\mathbf{O}$.
```

$$O_i = O_i^{\text{old}} \cdot \frac{l_i^{\text{old}}}{l_i} \cdot \frac{e^{m_i^{\text{old}}}}{e^{m_i}} + \frac{e^{S_i - m_i} \cdot V_j}{l_i}$$

$$m_i = \max\left(m_i^{\text{old}}, S_i\right)$$

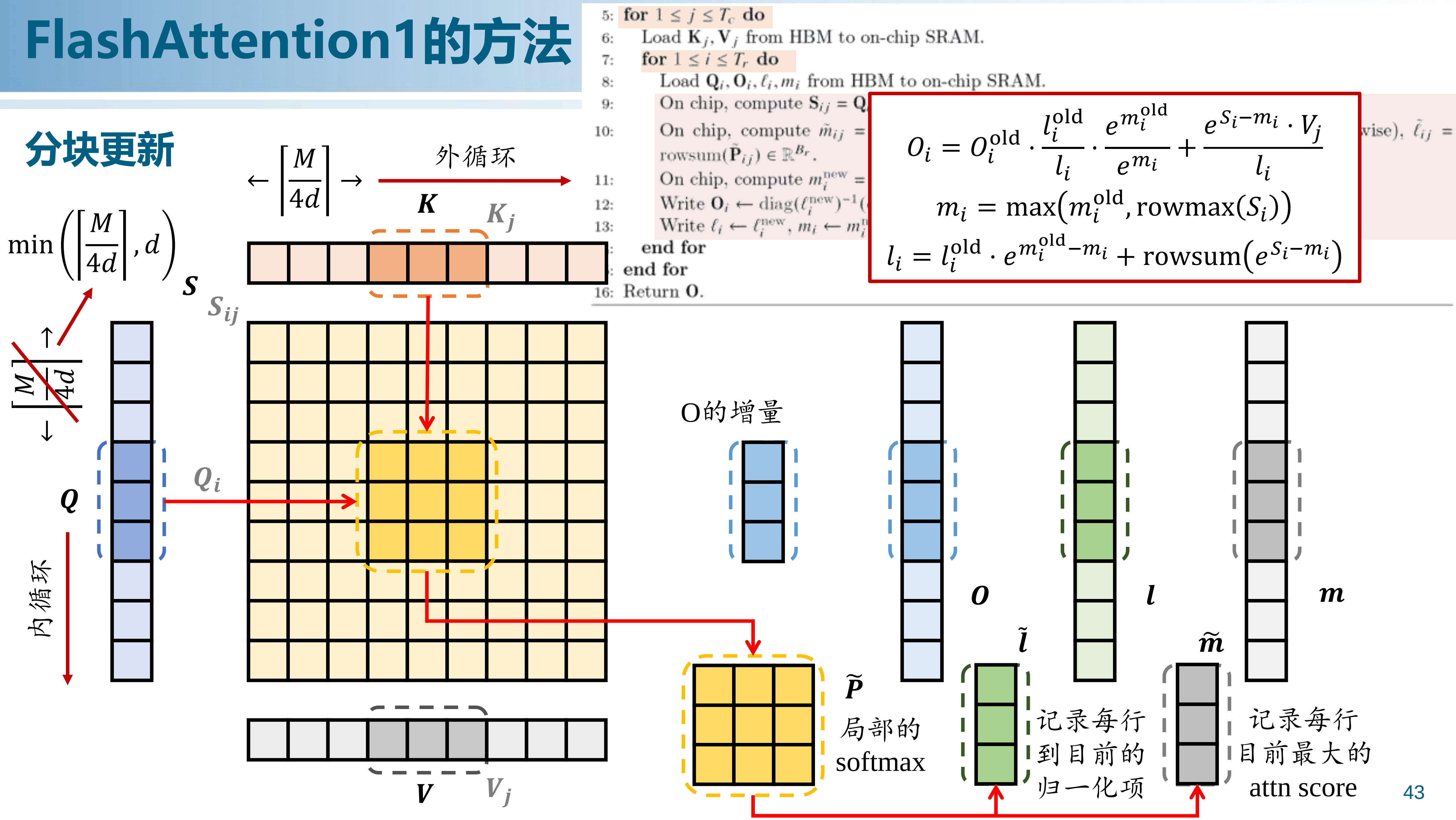
$$l_i = l_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + e^{S_i - m_i}$$





FlashAttention1的方法

分块更新



```
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}$
        On chip, compute $\tilde{m}_{ij} =$ wise), $\tilde{\ell}_{ij} =$ rowsum($\tilde{\mathbf{P}}_{ij}$) $\in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\mathrm{new}} =$
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}($
        Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{n}}$
    end for
end for
Return $\mathbf{O}$.
```

$$O_i = O_i^{\mathrm{old}} \cdot \frac{l_i^{\mathrm{old}}}{l_i} \cdot \frac{e^{m_i^{\mathrm{old}}}}{e^{m_i}} + \frac{e^{S_i - m_i} \cdot V_j}{l_i}$$

$$m_i = \max\left(m_i^{\mathrm{old}}, \mathrm{rowmax}(S_i)\right)$$

$$l_i = l_i^{\mathrm{old}} \cdot e^{m_i^{\mathrm{old}} - m_i} + \mathrm{rowsum}\left(e^{S_i - m_i}\right)$$

FlashAttention1的方法

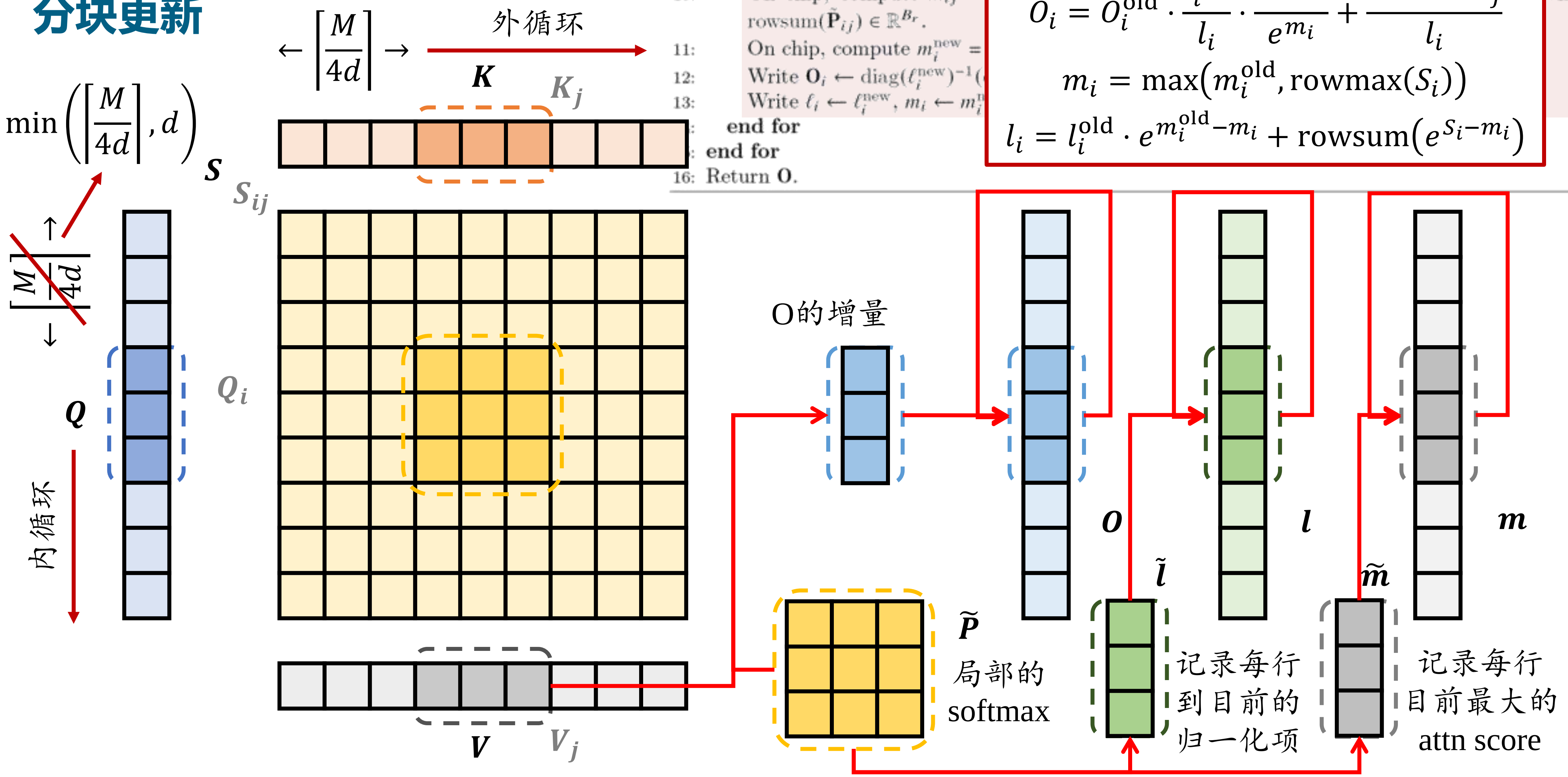
```
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}$
        On chip, compute $\tilde{m}_{ij} =$ wise), $\tilde{\ell}_{ij} =$ rowsum($\tilde{\mathbf{P}}_{ij}$) $\in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\text{new}} =$
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\text{new}})^{-1}($
        Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{n}}$
    end for
end for
Return $\mathbf{O}$.
```

$$O_i = O_i^{\text{old}} \cdot \frac{l_i^{\text{old}}}{l_i} \cdot \frac{e^{m_i^{\text{old}}}}{e^{m_i}} + \frac{e^{S_i - m_i} \cdot V_j}{l_i}$$

$$m_i = \max\left(m_i^{\text{old}}, \mathrm{rowmax}(S_i)\right)$$

$$l_i = l_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + \mathrm{rowsum}\left(e^{S_i - m_i}\right)$$

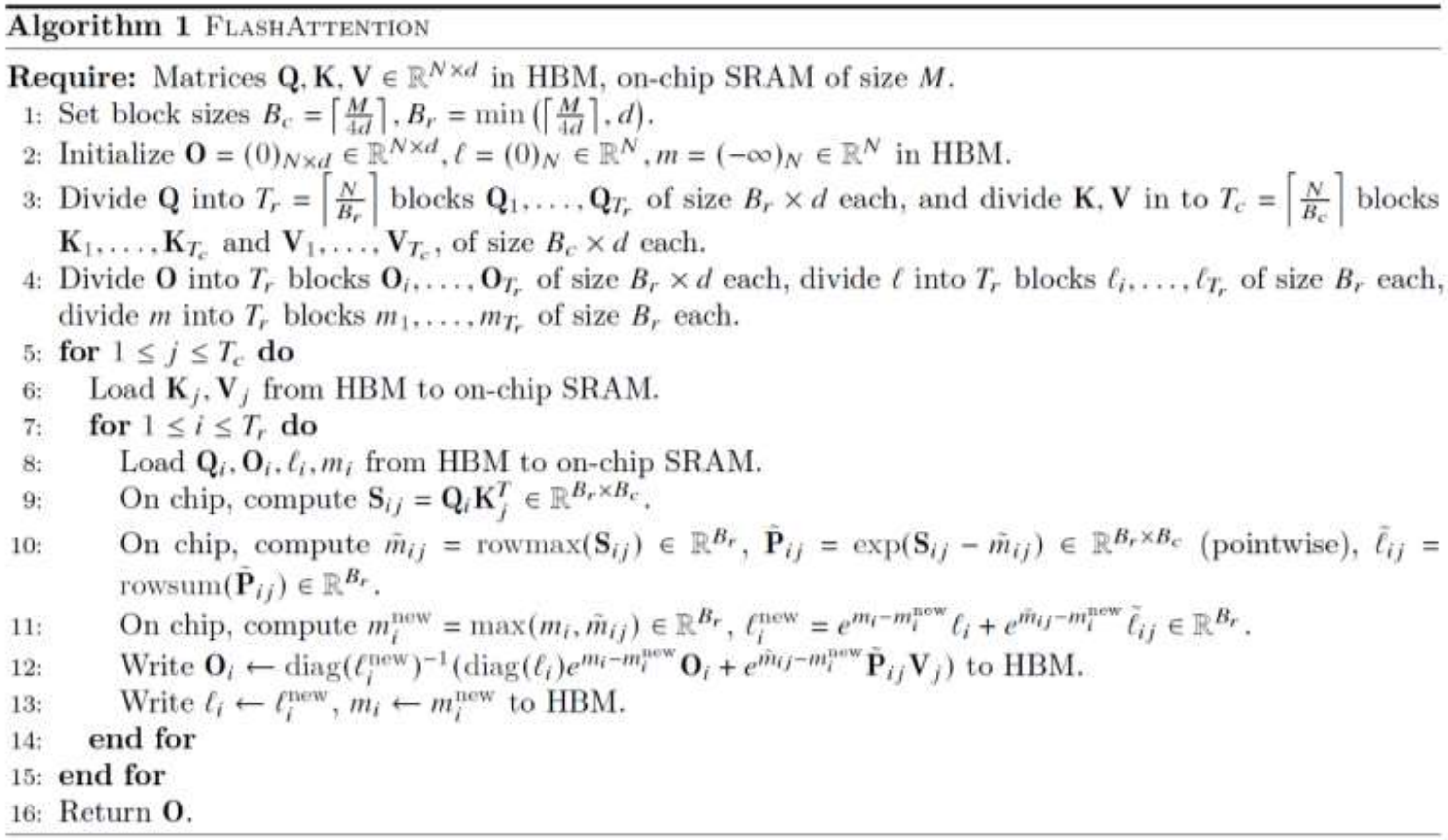
分块更新



FlashAttention1的方法

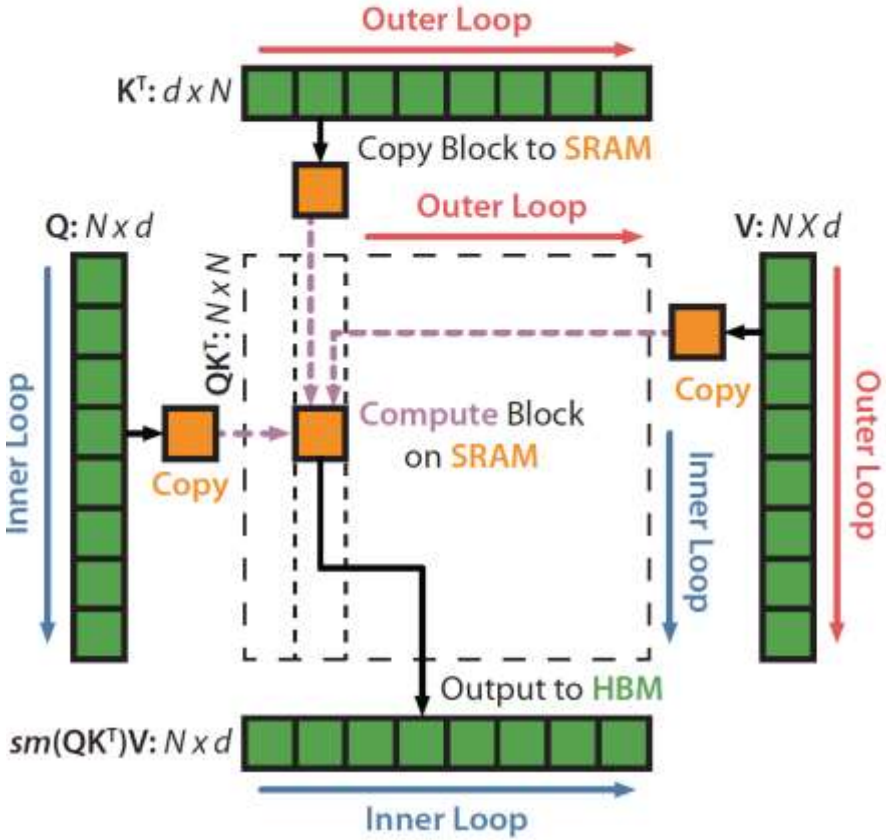
算法总结：分块加载、迭代softmax；复杂度：$O(Nd \cdot Nd/M) = O\left(N^2d^2/M\right)$

- 复杂度计算：内循环中Q和O的加载是主导因素，加载量 = Q和O的大小 * K和V的块数



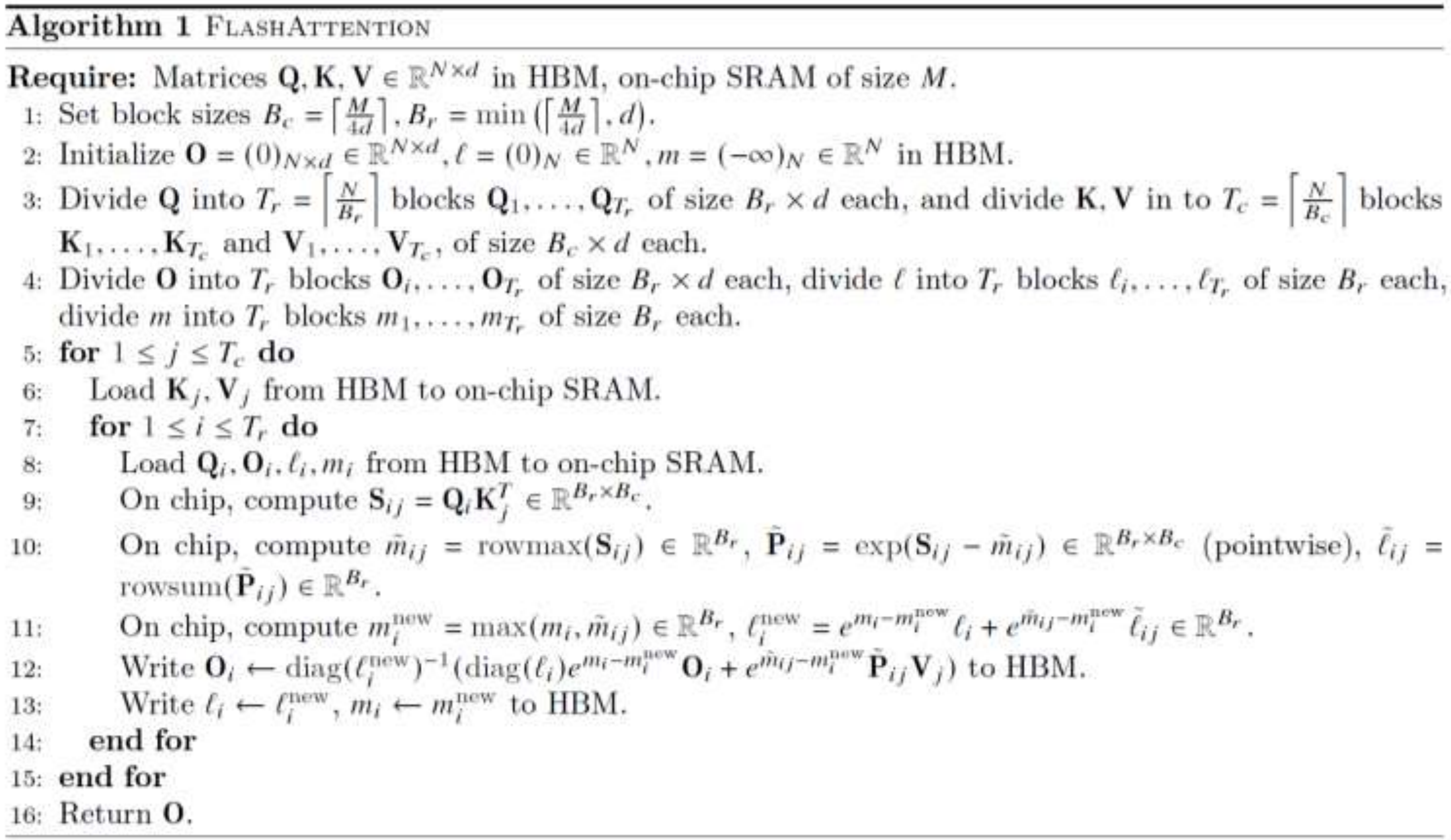
Algorithm 1 FlashAttention

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size $M$.
Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.
Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks
    $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide $\mathbf{O}$ into $T_r$ blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\ell$ into $T_r$ blocks $\ell_i, \dots, \ell_{T_r}$ of size $B_r$ each,
    divide $m$ into $T_r$ blocks $m_1, \dots, m_{T_r}$ of size $B_r$ each.
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
        On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} =$
            $\mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1} (\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
        Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
    end for
end for
Return $\mathbf{O}$.
```

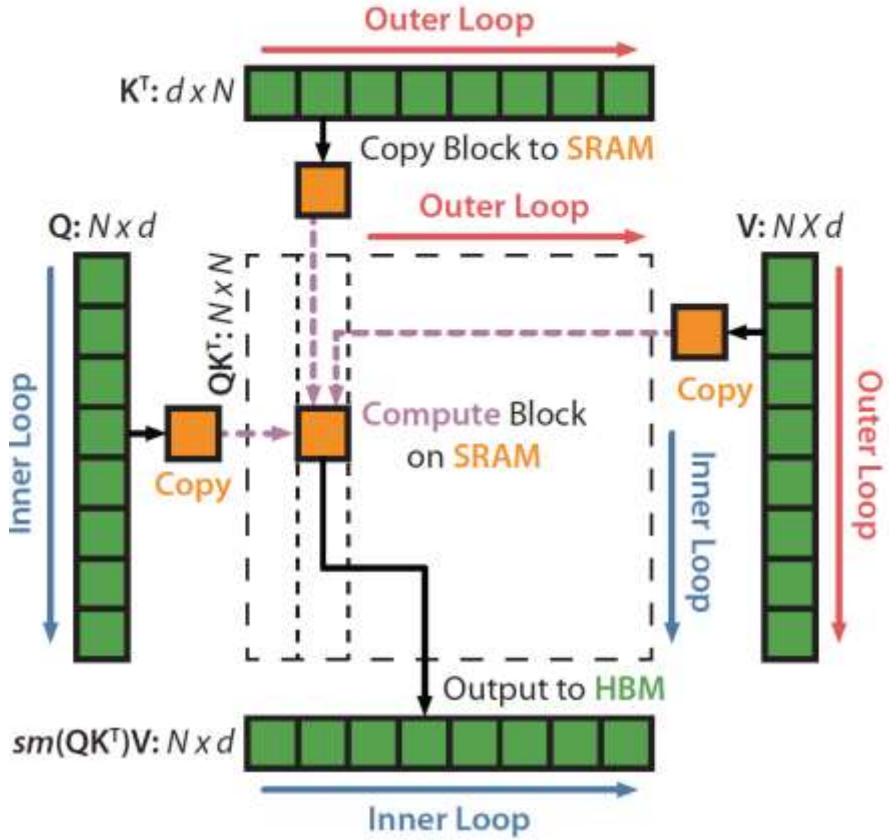


FlashAttention2的动机

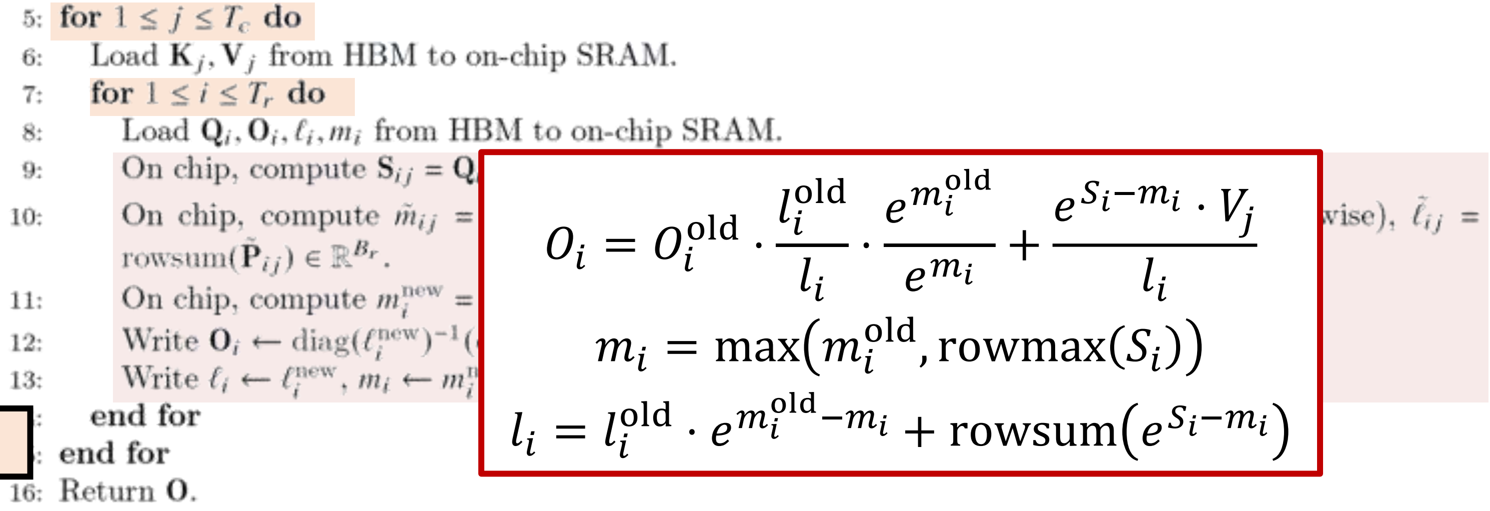
- **算法不足**： O实际上是不需要加载再写入的；softmax的归一化系数是可以一步除的
- 以Q为外循环，O只需要一次输出；GPU针对矩阵乘有加速，其他运算吞吐量显著更高



```
Algorithm 1 FLASHATTENTION
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size $M$.
Set block sizes $B_c = \lceil \frac{M}{4d} \rceil, B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
Divide $\mathbf{Q}$ into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \lceil \frac{N}{B_c} \rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide $\mathbf{O}$ into $T_r$ blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\ell$ into $T_r$ blocks $\ell_i, \dots, \ell_{T_r}$ of size $B_r$ each, divide $m$ into $T_r$ blocks $m_1, \dots, m_{T_r}$ of size $B_r$ each.
for $1 \le j \le T_c$ do
  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
  for $1 \le i \le T_r$ do
    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
    On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
    On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
    Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1} (\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
    Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}, m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
  end for
end for
Return $\mathbf{O}$.
```



FlashAttention1的方法



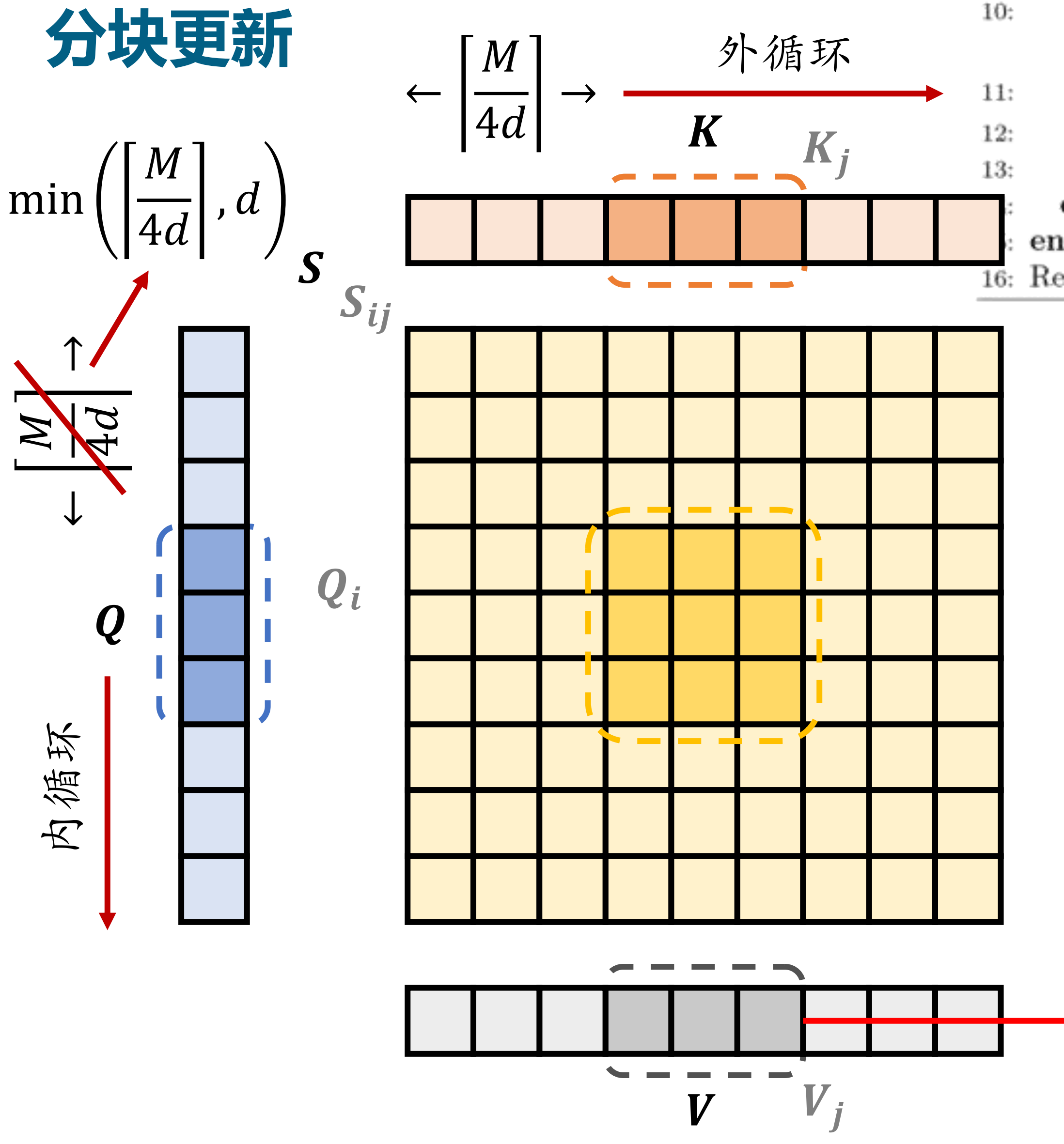
```
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}$
        On chip, compute $\tilde{m}_{ij} =$ wise), $\tilde{\ell}_{ij} =$ rowsum$(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\mathrm{new}} =$
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}($
        Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{n}}$
    end for
end for
Return $\mathbf{O}$.
```

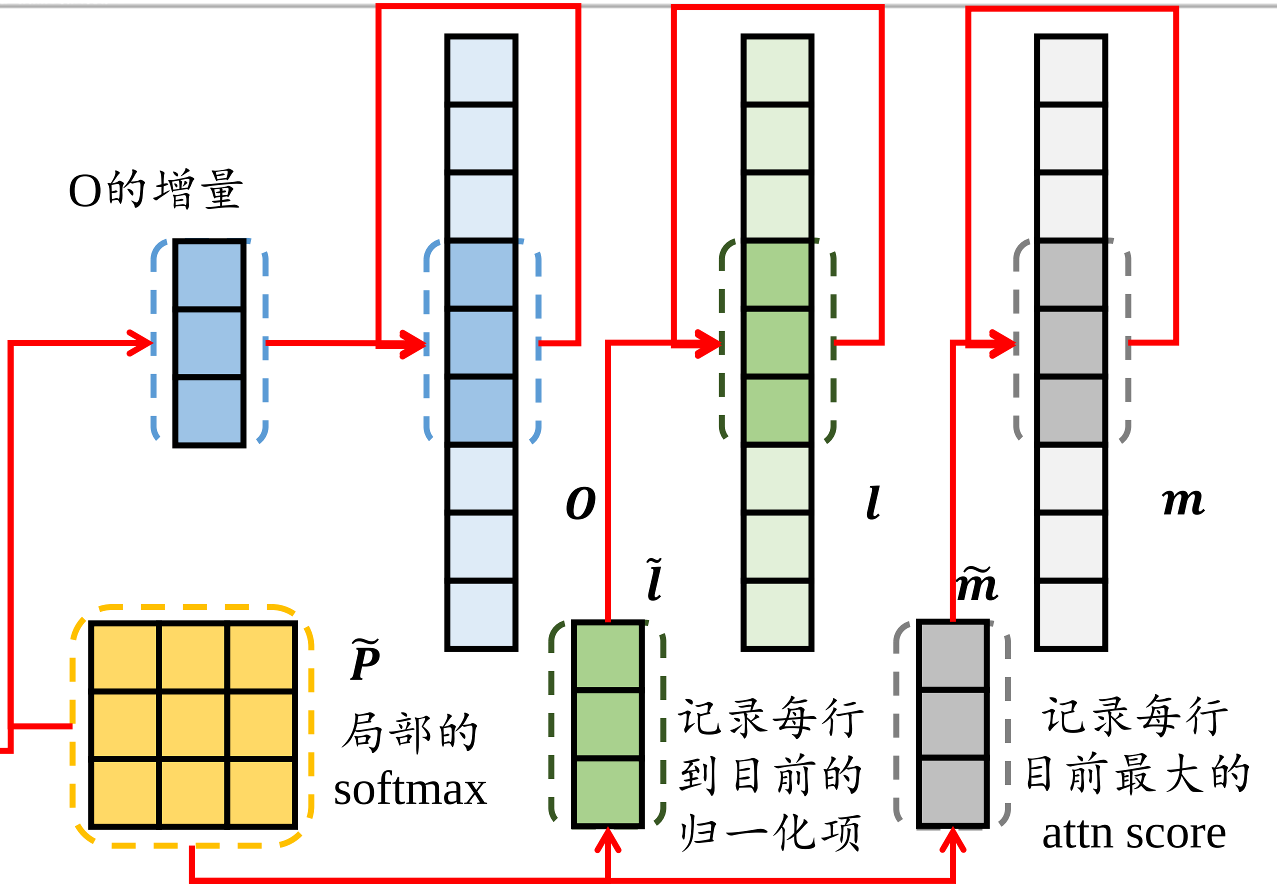
$$O_i = O_i^{\mathrm{old}} \cdot \frac{l_i^{\mathrm{old}}}{l_i} \cdot \frac{e^{m_i^{\mathrm{old}}}}{e^{m_i}} + \frac{e^{S_i - m_i} \cdot V_j}{l_i}$$

$$m_i = \max\left(m_i^{\mathrm{old}}, \mathrm{rowmax}(S_i)\right)$$

$$l_i = l_i^{\mathrm{old}} \cdot e^{m_i^{\mathrm{old}} - m_i} + \mathrm{rowsum}\left(e^{S_i - m_i}\right)$$

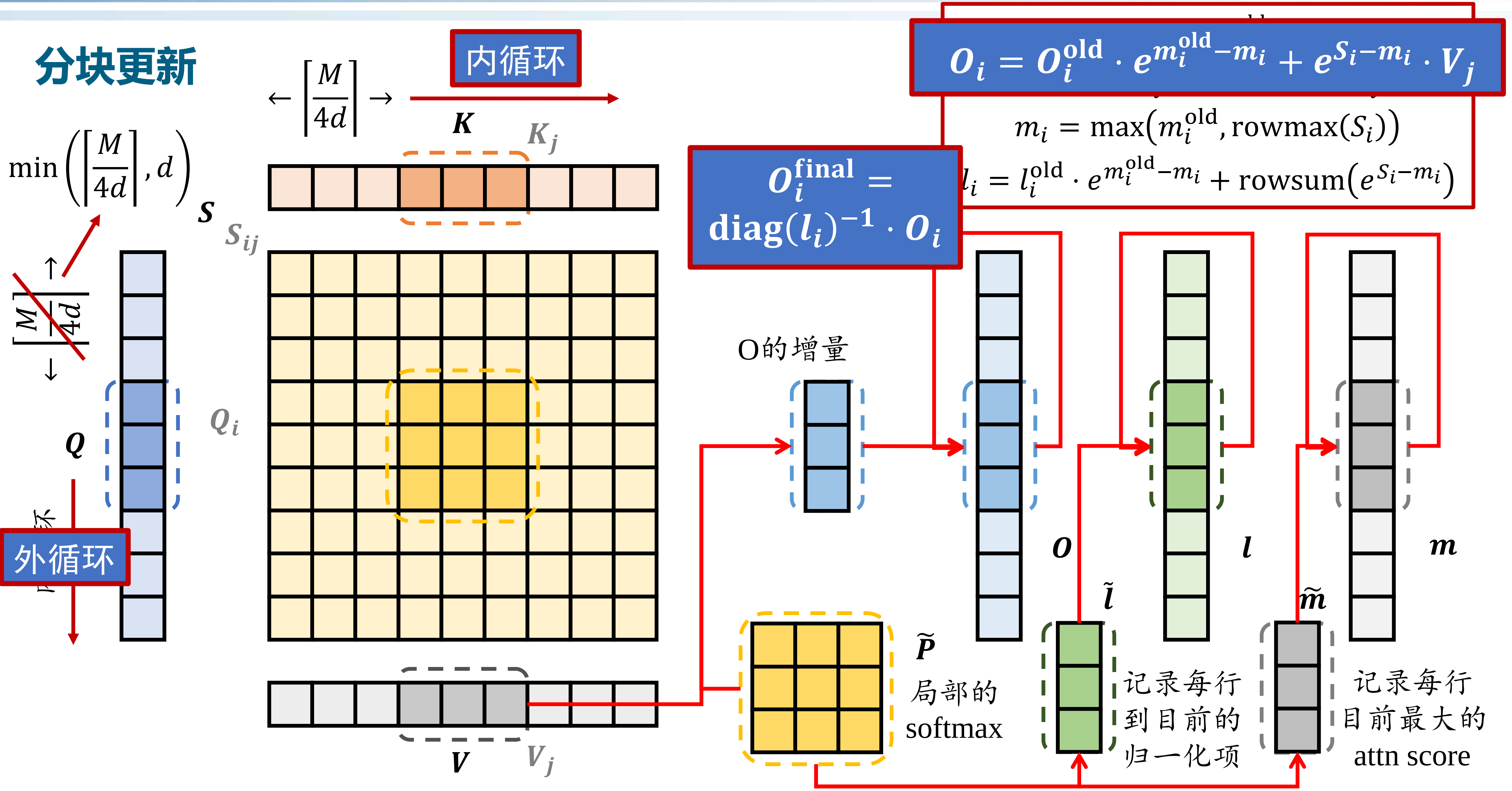
分块更新





FlashAttention2的方法

分块更新



$$O_i = O_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + e^{S_i - m_i} \cdot V_j$$

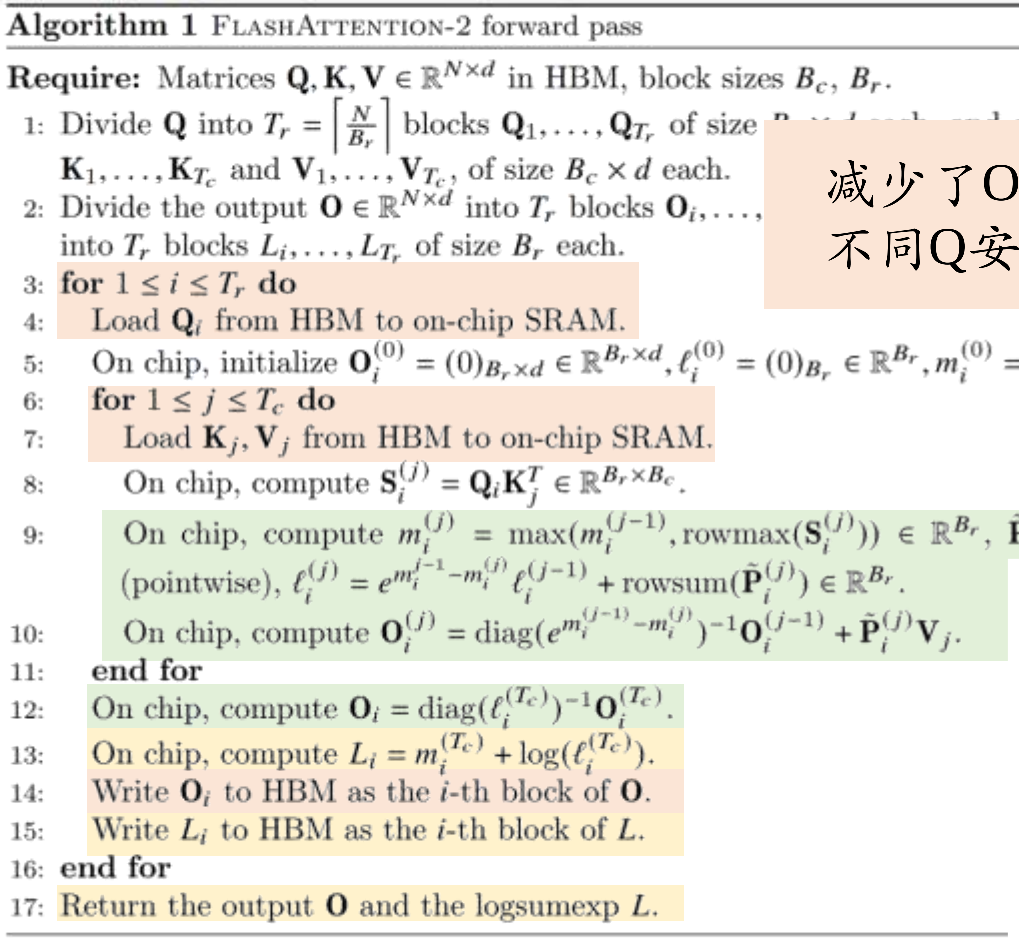
$$m_i = \max\left(m_i^{\text{old}}, \text{rowmax}(S_i)\right)$$

$$l_i = l_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + \text{rowsum}\left(e^{S_i - m_i}\right)$$

$$O_i^{\text{final}} = \text{diag}(l_i)^{-1} \cdot O_i$$

方法对比

- 改动一：内外循环的顺序调整
- 改动二：softmax归一化的滞后
- 改动三：logexpsum用于梯度回传



Algorithm 1 FlashAttention-2 forward pass

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, block sizes $B_c$, $B_r$.
Divide $\mathbf{Q}$ into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $\lceil \frac{N}{B_c} \rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide the output $\mathbf{O} \in \mathbb{R}^{N\times d}$ into $T_r$ blocks $\mathbf{O}_i, \dots,$ gsumexp $L$ into $T_r$ blocks $L_i, \dots, L_{T_r}$ of size $B_r$ each.
for $1 \le i \le T_r$ do
    Load $\mathbf{Q}_i$ from HBM to on-chip SRAM.
    On chip, initialize $\mathbf{O}_i^{(0)} = (0)_{B_r\times d} \in \mathbb{R}^{B_r\times d}, \ell_i^{(0)} = (0)_{B_r} \in \mathbb{R}^{B_r}, m_i^{(0)} = (-\infty)_{B_r} \in \mathbb{R}^{B_r}$.
    for $1 \le j \le T_c$ do
        Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_i^{(j)} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r\times B_c}$.
        On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \mathrm{rowmax}(\mathbf{S}_i^{(j)})) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r\times B_c}$ (pointwise), $\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}} \ell_i^{(j-1)} + \mathrm{rowsum}(\tilde{\mathbf{P}}_i^{(j)}) \in \mathbb{R}^{B_r}$.
        On chip, compute $\mathbf{O}_i^{(j)} = \mathrm{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1} \mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_i^{(j)} \mathbf{V}_j$.
    end for
    On chip, compute $\mathbf{O}_i = \mathrm{diag}(\ell_i^{(T_c)})^{-1} \mathbf{O}_i^{(T_c)}$.
    On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.
    Write $\mathbf{O}_i$ to HBM as the $i$-th block of $\mathbf{O}$.
    Write $L_i$ to HBM as the $i$-th block of $L$.
end for
Return the output $\mathbf{O}$ and the logsumexp $L$.
```

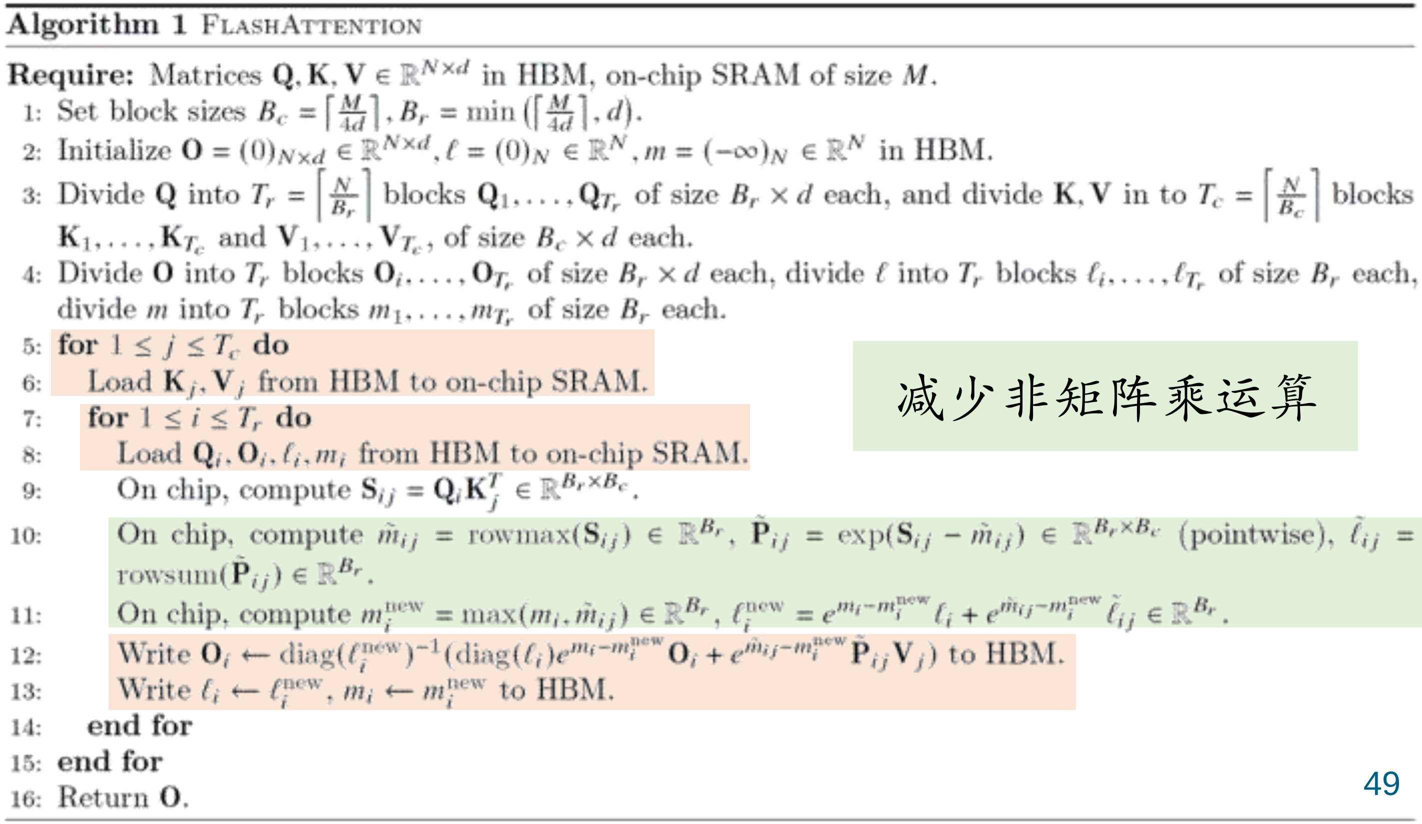
减少了O的相关读写
不同Q安排不同线程

$$O_i = O_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + e^{S_i - m_i} \cdot V_j$$

$$m_i = \max\left(m_i^{\text{old}}, \mathrm{rowmax}(S_i)\right)$$

$$l_i = l_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + \mathrm{rowsum}\left(e^{S_i - m_i}\right)$$

$$O_i^{\text{final}} = \mathrm{diag}(l_i)^{-1} \cdot O_i$$



Algorithm 1 FlashAttention

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, on-chip SRAM of size $M$.
Set block sizes $B_c = \lceil \frac{M}{4d} \rceil, B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
Initialize $\mathbf{O} = (0)_{N\times d} \in \mathbb{R}^{N\times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
Divide $\mathbf{Q}$ into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \lceil \frac{N}{B_c} \rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide $\mathbf{O}$ into $T_r$ blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\ell$ into $T_r$ blocks $\ell_i, \dots, \ell_{T_r}$ of size $B_r$ each, divide $m$ into $T_r$ blocks $m_1, \dots, m_{T_r}$ of size $B_r$ each.
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r\times B_c}$.
        On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r\times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\text{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
        Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.
    end for
end for
Return $\mathbf{O}$.
```

减少非矩阵乘运算

方法对比

- 改动一：内外循环的顺序调整
- 改动二：softmax归一化的滞后
- 改动三：logexpsum用于梯度回传

Algorithm 1 FlashAttention-2 forward pass

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, block sizes $B_c$, $B_r$.
Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $\left\lceil \frac{N}{B_c} \right\rceil$ blocks
    $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide the output $\mathbf{O} \in \mathbb{R}^{N\times d}$ into $T_r$ blocks $\mathbf{O}_i, \dots,$ gsumexp $L$
    into $T_r$ blocks $L_i, \dots, L_{T_r}$ of size $B_r$ each.
for $1 \le i \le T_r$ do
    Load $\mathbf{Q}_i$ from HBM to on-chip SRAM.
    On chip, initialize $\mathbf{O}_i^{(0)} = (0)_{B_r\times d} \in \mathbb{R}^{B_r\times d}, \ell_i^{(0)} = (0)_{B_r} \in \mathbb{R}^{B_r}, m_i^{(0)} = (-\infty)_{B_r} \in \mathbb{R}^{B_r}$.
    for $1 \le j \le T_c$ do
        Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_i^{(j)} = \mathbf{Q}_i\mathbf{K}_j^T \in \mathbb{R}^{B_r\times B_c}$.
        On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \mathrm{rowmax}(\mathbf{S}_i^{(j)})) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r\times B_c}$
            (pointwise), $\ell_i^{(j)} = e^{m_i^{j-1}-m_i^{(j)}}\ell_i^{(j-1)} + \mathrm{rowsum}(\tilde{\mathbf{P}}_i^{(j)}) \in \mathbb{R}^{B_r}$.
        On chip, compute $\mathbf{O}_i^{(j)} = \mathrm{diag}(e^{m_i^{(j-1)}-m_i^{(j)}})^{-1}\mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_i^{(j)}\mathbf{V}_j$.
    end for
    On chip, compute $\mathbf{O}_i = \mathrm{diag}(\ell_i^{(T_c)})^{-1}\mathbf{O}_i^{(T_c)}$.
    On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.
    Write $\mathbf{O}_i$ to HBM as the $i$-th block of $\mathbf{O}$.
    Write $L_i$ to HBM as the $i$-th block of $L$.
end for
Return the output $\mathbf{O}$ and the logsumexp $L$.
```

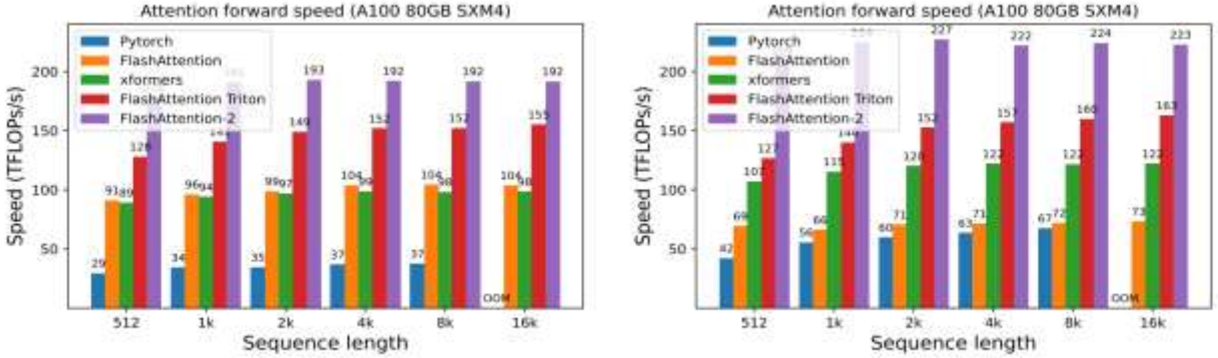
减少了O的相关读写
不同Q安排不同线程

$$O_i = O_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + e^{S_i - m_i} \cdot V_j$$

$$m_i = \max\left(m_i^{\text{old}}, \mathrm{rowmax}(S_i)\right)$$

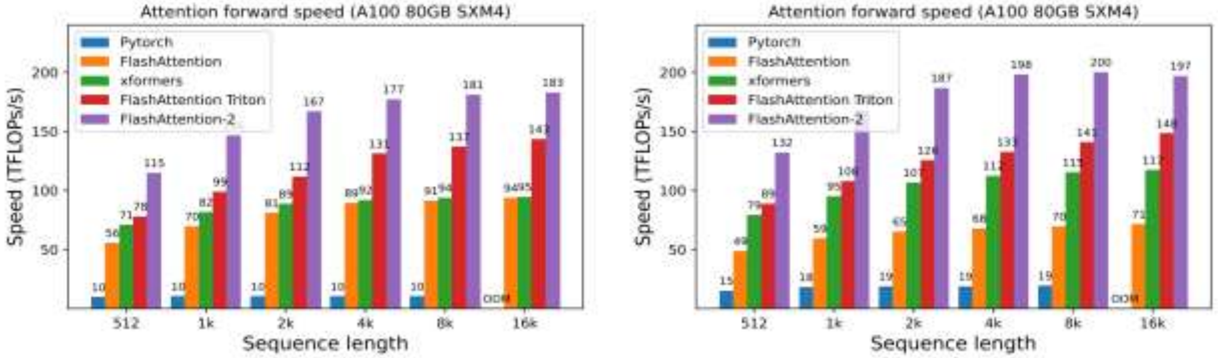
$$l_i = l_i^{\text{old}} \cdot e^{m_i^{\text{old}} - m_i} + \mathrm{rowsum}\left(e^{S_i - m_i}\right)$$

$$O_i^{\text{final}} = \mathrm{diag}(l_i)^{-1} \cdot O_i$$



(a) Without causal mask, head dimension 64

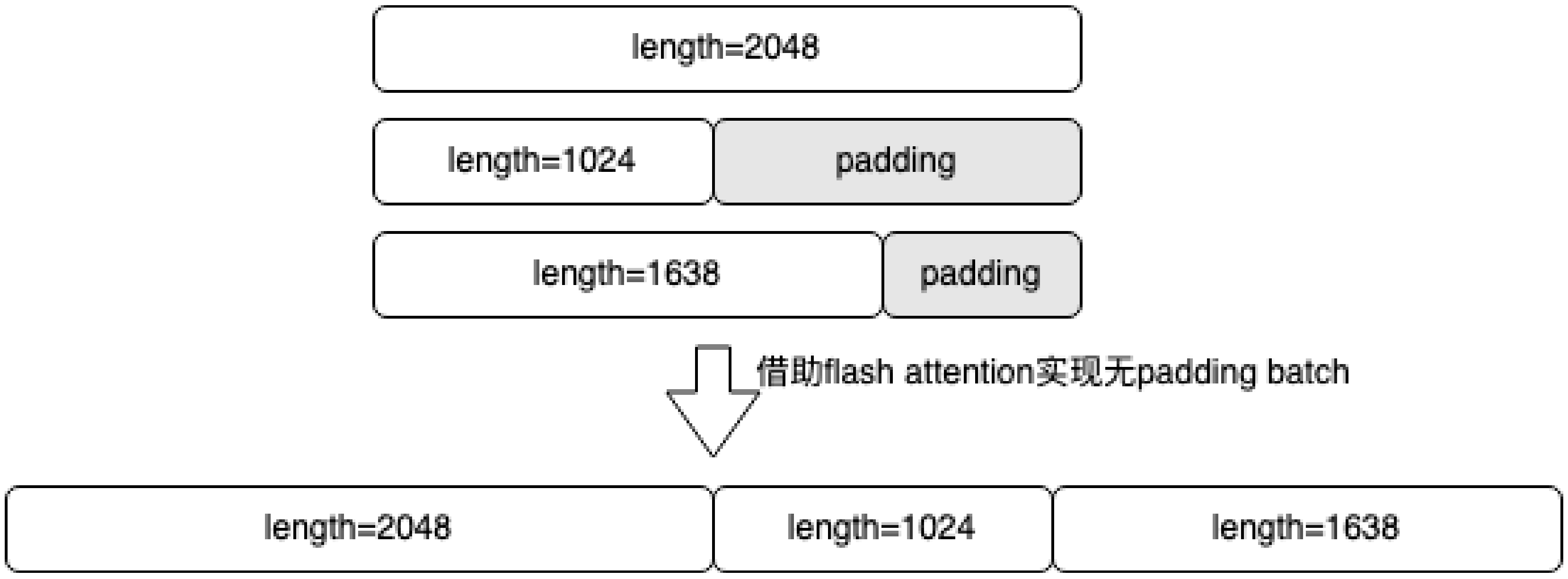
(b) Without causal mask, head dimension 128



(c) With causal mask, head dimension 64

(d) With causal mask, head dimension 128

避免padding



length=2048

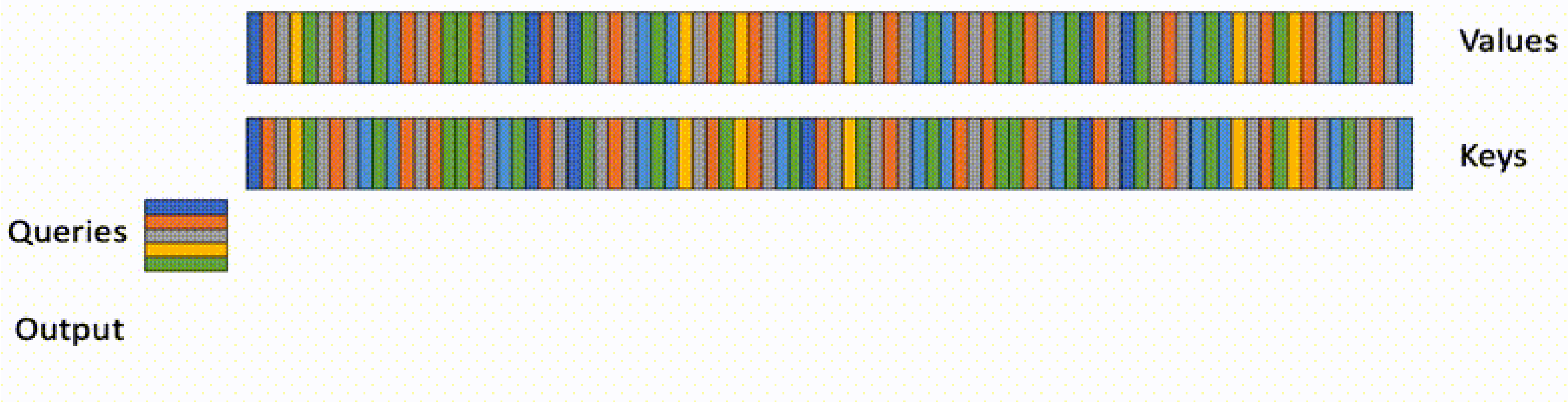
length=1024 padding

length=1638 padding

借助flash attention实现无padding batch

length=2048 length=1024 length=1638

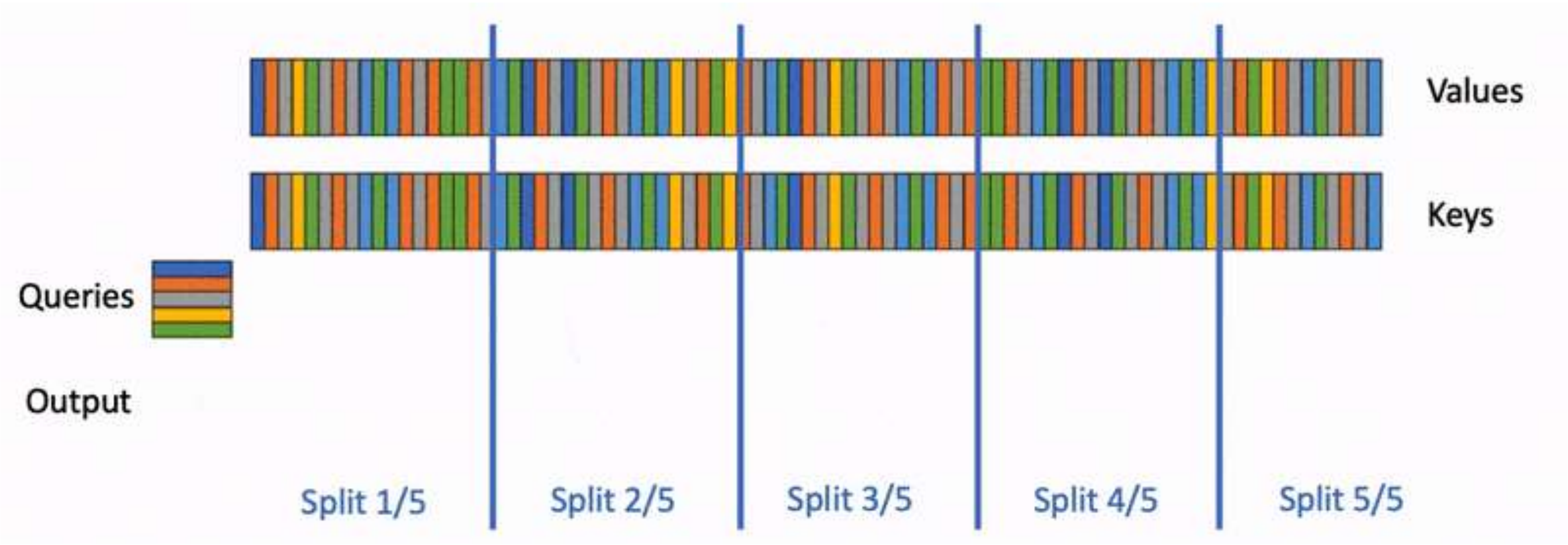
FlashDecoding



计算q和kv cache

https://pytorch.org/blog/flash-decoding/

FlashDecoding



这个过程可以并行计算

https://pytorch.org/blog/flash-decoding/

“LLM”宇宙常数

为什么我们需要LLM“宇宙常数”？

1. 一个Transformer模型的**参数量**是多少?
2. 一个Transformer模型的**计算量**是多少?
3. 训练需要多大的**显存**?
4. 训练需要多大的**硬盘空间**?
5. 什么是Nvlink、Infini-band?
6. 如何估算大模型的**训练速度**?
7. **训练**一个7B的模型需要多长**时间**?
8. 训练一个7B模型需要多少钱?

………

这些问题都是在大模型训练过程中会经常遇到的问题，我们需要速算方法来估计它们

为什么我们需要LLM“宇宙常数”？

训练需要多大的显存?

$$l * \left(12h^2 + 13h\right) + 2Vh$$

其中l是层数，h是隐藏层维度，V是词表大小

怎么算的?

*以MLP结构为FFN部分估算

一个Transformer模型的计算量是多少?

$$l * \left(24bsh^2 + 4bs^2h\right) + 2bshV \qquad \text{FLOPs}$$

其中l是层数，h是隐藏层维度，V是词表大小，s是序列长度，b是批大小。

以上是前向运算的估计，如果还需要考虑反向传播的话，还需要加上2倍。

想想为啥是2倍?

如果序列长度不是特别长（比如说32k），可以将前向和反向传播的算力约等为 $\mathbf{6}\boldsymbol{PD}$

特别长的上下文，算力会约为

$$\left(8 + \frac{4s}{3h}\right)PD$$

https://zhuanlan.zhihu.com/p/688178908

为什么我们需要LLM“宇宙常数”？

训练需要多大的显存?

$20P$ G

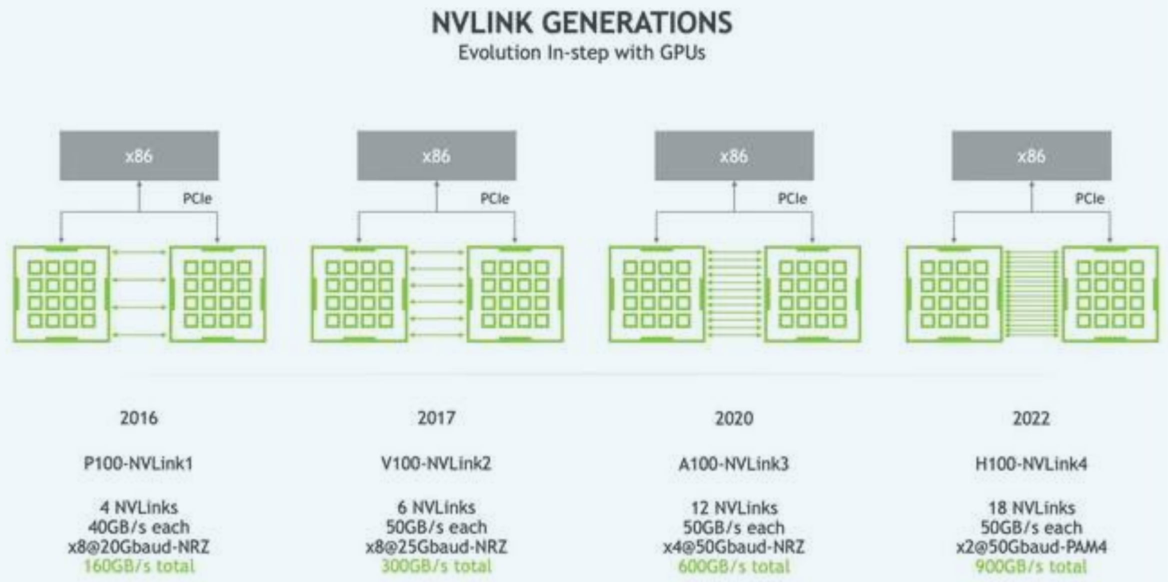
$$\underbrace{2+4}_{\text{weights}}+\underbrace{2+4}_{\text{gradients}}+\underbrace{4+4}_{\text{Adam states}}=20 \text{ bytes.}$$

此外还有中间的activation，不过这个可以尽量通过重计算避免掉很大一部分

训练需要多大的硬盘空间?

每个checkpoint需要$14P$ G或$12P$ G

什么是Nvlink、Infini-band?



2016	2017	2020	2022
P100-NVLink1	V100-NVLink2	A100-NVLink3	H100-NVLink4
4 NVLinks 40GB/s each x8@20Gbaud-NRZ 160GB/s total	6 NVLinks 50GB/s each x8@25Gbaud-NRZ 300GB/s total	12 NVLinks 50GB/s each x4@50Gbaud-NRZ 600GB/s total	18 NVLinks 50GB/s each x2@50Gbaud-PAM4 900GB/s total

为什么我们需要LLM“宇宙常数”？

如何估算大模型的训练速度、时间、成本?

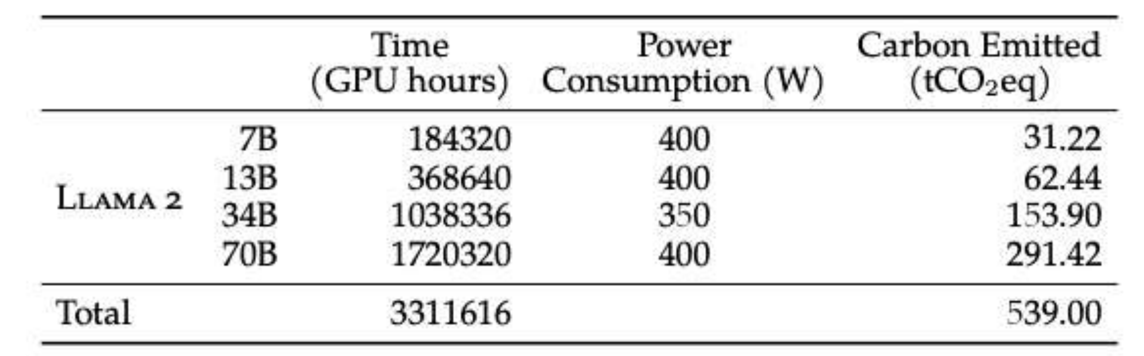
显卡型号	FLOPS
V100	125 T
A100	312 T
H100	990 T
B200	2250 T

根据我们之前对模型计算量的估计，计算量约等于 $6PD$

那么就可以通过这个计算量和每张卡可以输出的算力进行一个计算，一般来说，GPU算力利用率约50%左右。

估计一下LLaMA 7B需要的时间

$$\frac{7\times 10^{9}\times 2\times 10^{12}\times 6}{312\times 0.5\times 10^{12}\times 3600}=140562\ \text{GPU}*\text{Hour}$$



		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO_2eq)
LLAMA 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00

有了GPU*Hour，所需要的时间就根据投入卡的数量决定时间了。

那对应的成本呢？目前市场上A100每小时租金大约为10元左右，因此一个7B LLaMA的成本就是184万左右。

假设1.8T 的GPT4呢？ **> 2亿**

https://zhuanlan.zhihu.com/p/688178908
https://zhuanlan.zhihu.com/p/688248925

感谢观看

https://www.shlab.org.cn
Shanghai Artificial Intelligence Laboratory