4.3 大模型的分布式训练

林洲汉

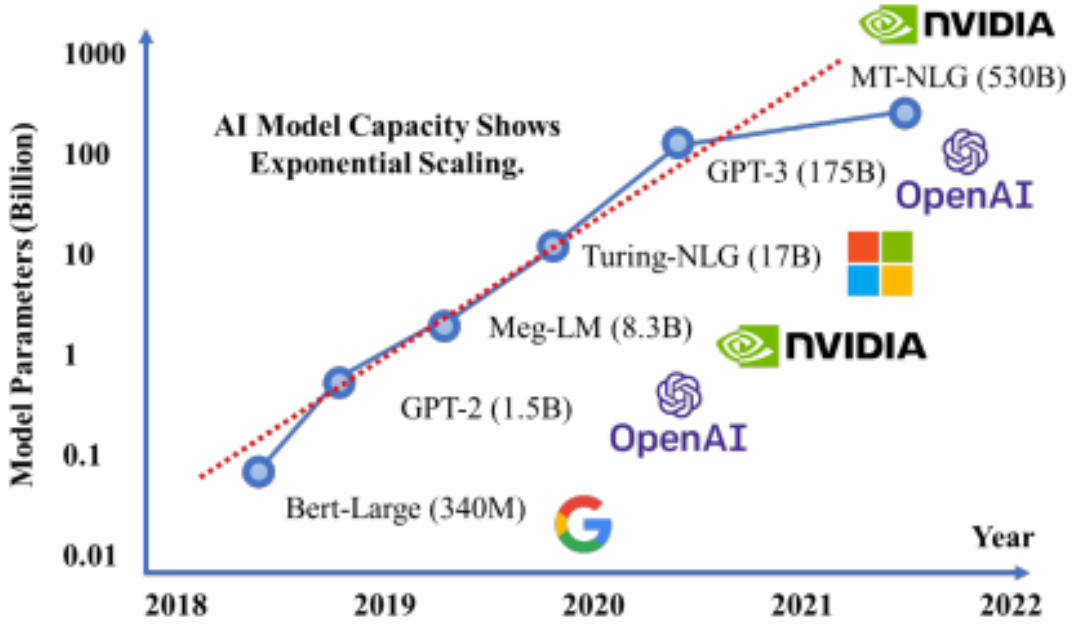
2024年秋季学期

目录

为什么需要并行计算

近年来，随着Transformer架构的提出，使得深度学习模型轻松突破上万亿规模参数，传统的单机单卡模式已经无法满足超大模型进行训练的要求。

因此，我们需要基于单机多卡、甚至是多机多卡进行分布式大模型的训练。



https://arxiv.org/abs/2111.14247

为什么需要并行计算

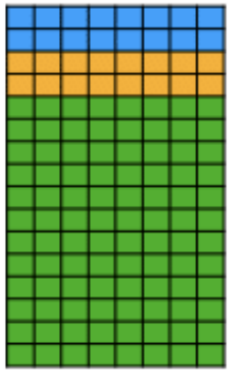
并行计算，能够大幅降低训练模型所需要的**时间开销**

并行计算使**超大参数**规模的模型训练成为可能

在WikiPedia上训练BERT模型时

- 单卡3090：58小时/epoch
- 8卡3090：9.5小时/epoch

加速6倍



模型参数：2 bytes

梯度：2 bytes

优化器状态：12 bytes (Adam)

即使是80G显存的H100，一张卡也只能装下

GPU显存分配

80GB/16Bytes=5.0B

如果不使用并行计算，只用一张GPU训练GPT-3需要355年！

目录

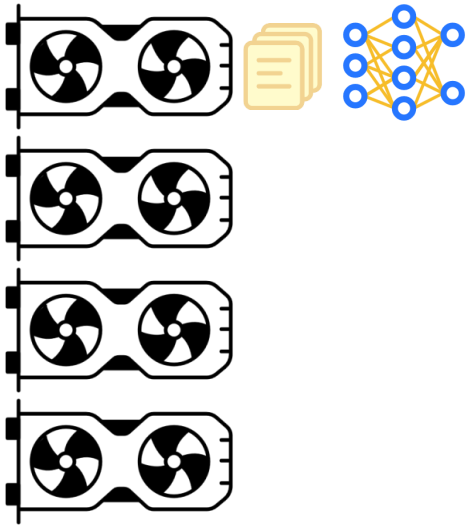
数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

首先由主GPU将数据分发到每个GPU上

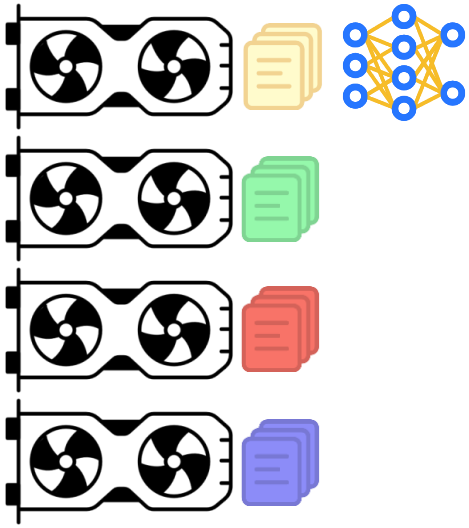


图标素材来自https://www.iconfont.cn/

数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

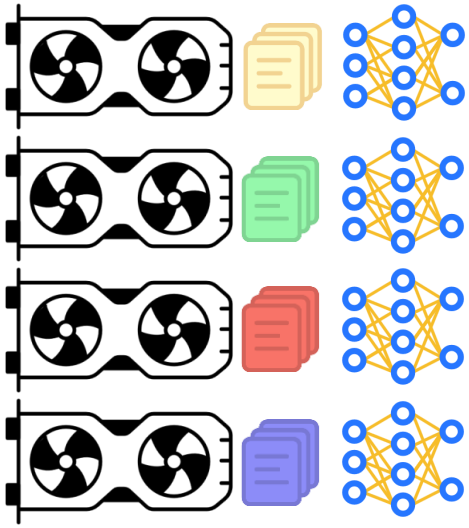
首先由主GPU将数据分发到每个GPU上
然后由主GPU将模型分发到每个GPU上



数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

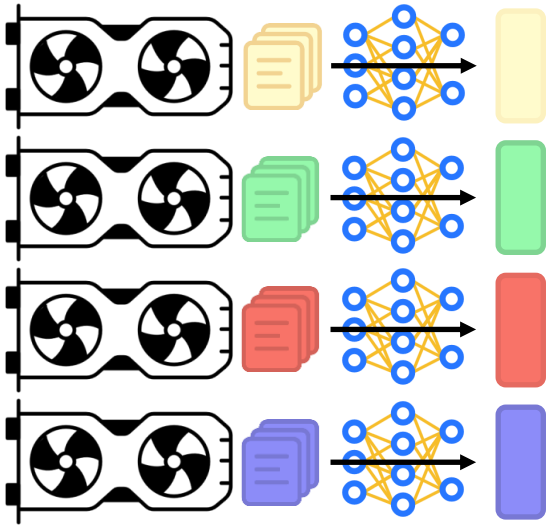
首先由主GPU将数据分发到每个GPU上
然后由主GPU将模型分发到每个GPU上



数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

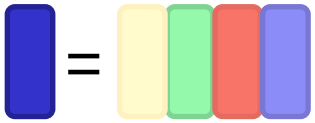
每个GPU独立进行前向计算，得到模型输出



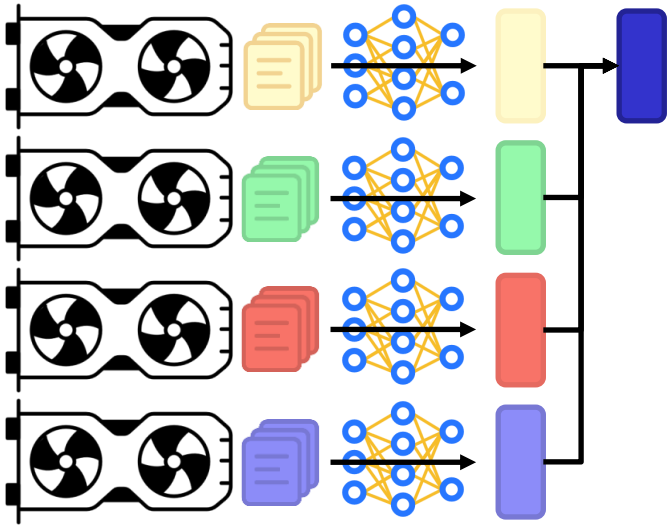
前向计算

数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。



将每个GPU的计算结果发回主GPU

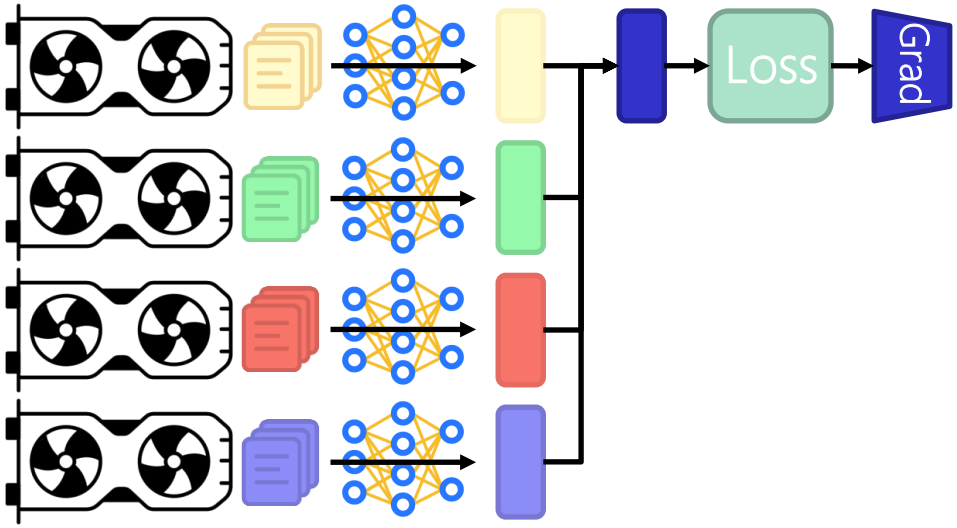


前向计算

数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

主GPU计算loss和梯度

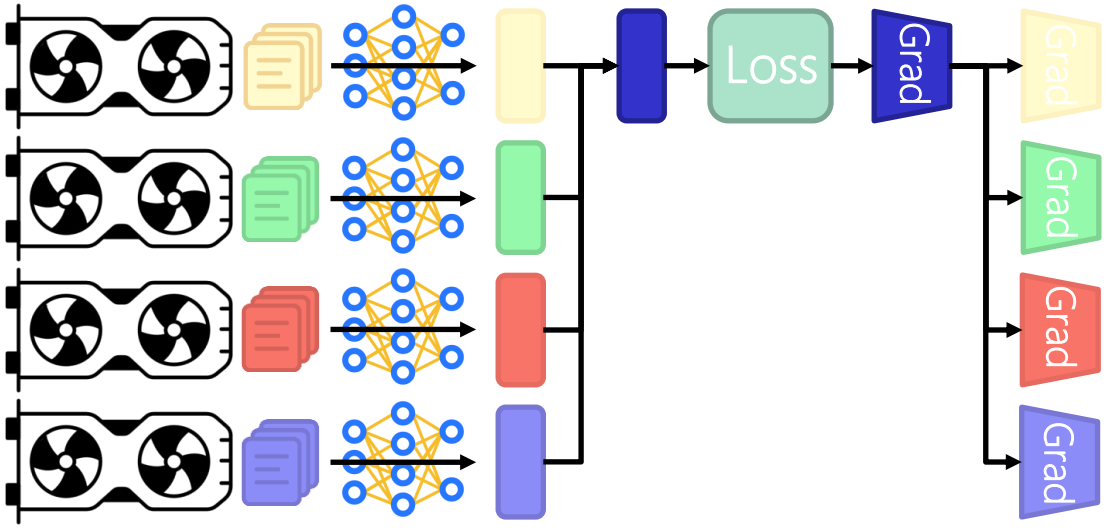


前向计算

数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

将梯度分发到每个GPU上

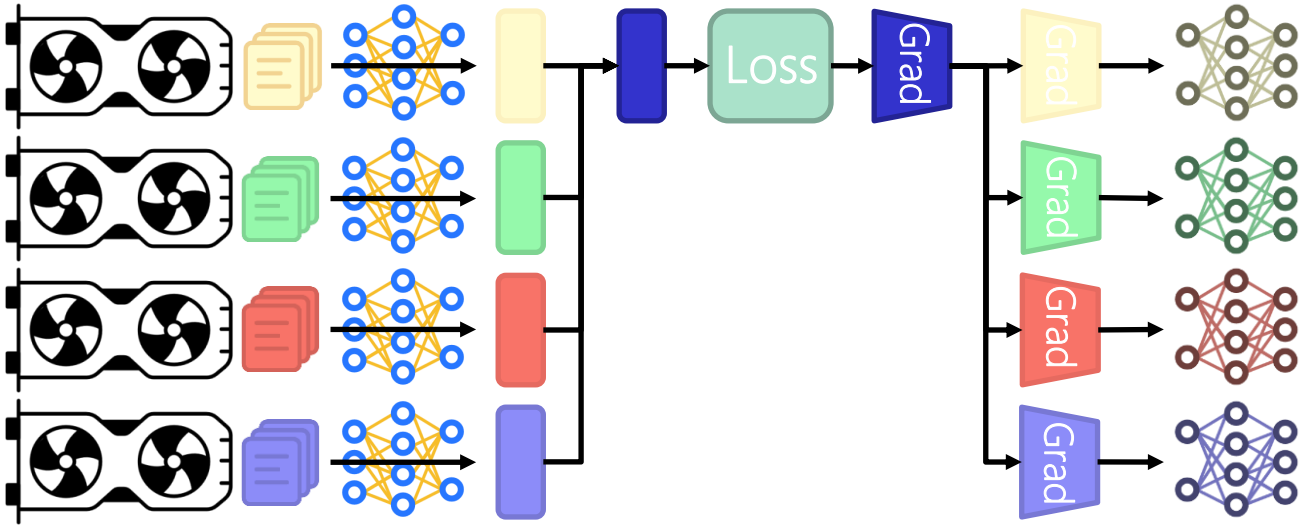


前向计算

数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

每个GPU独立计算参数更新

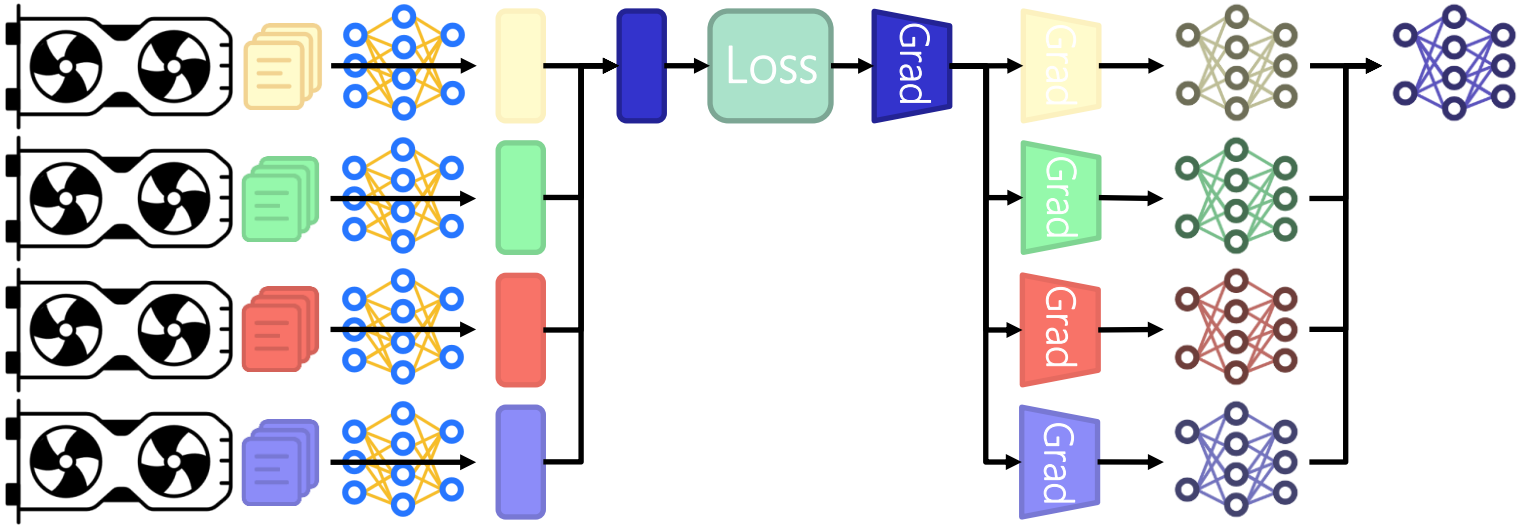


前向计算　　反向计算

数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

将每个GPU计算得到的参数更新发回主GPU

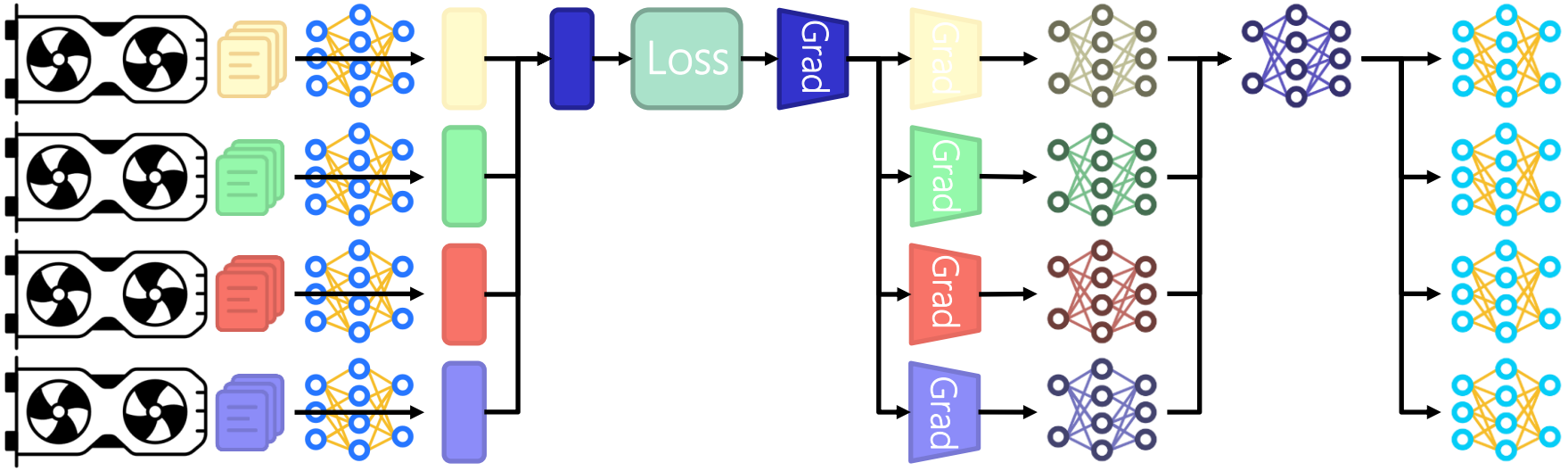


前向计算　　反向计算

数据并行

数据并行就是将把数据分摊到不同的计算设备上，同时进行计算。

主GPU更新模型参数，并同步到所有GPU



前向计算　　反向计算　　参数更新

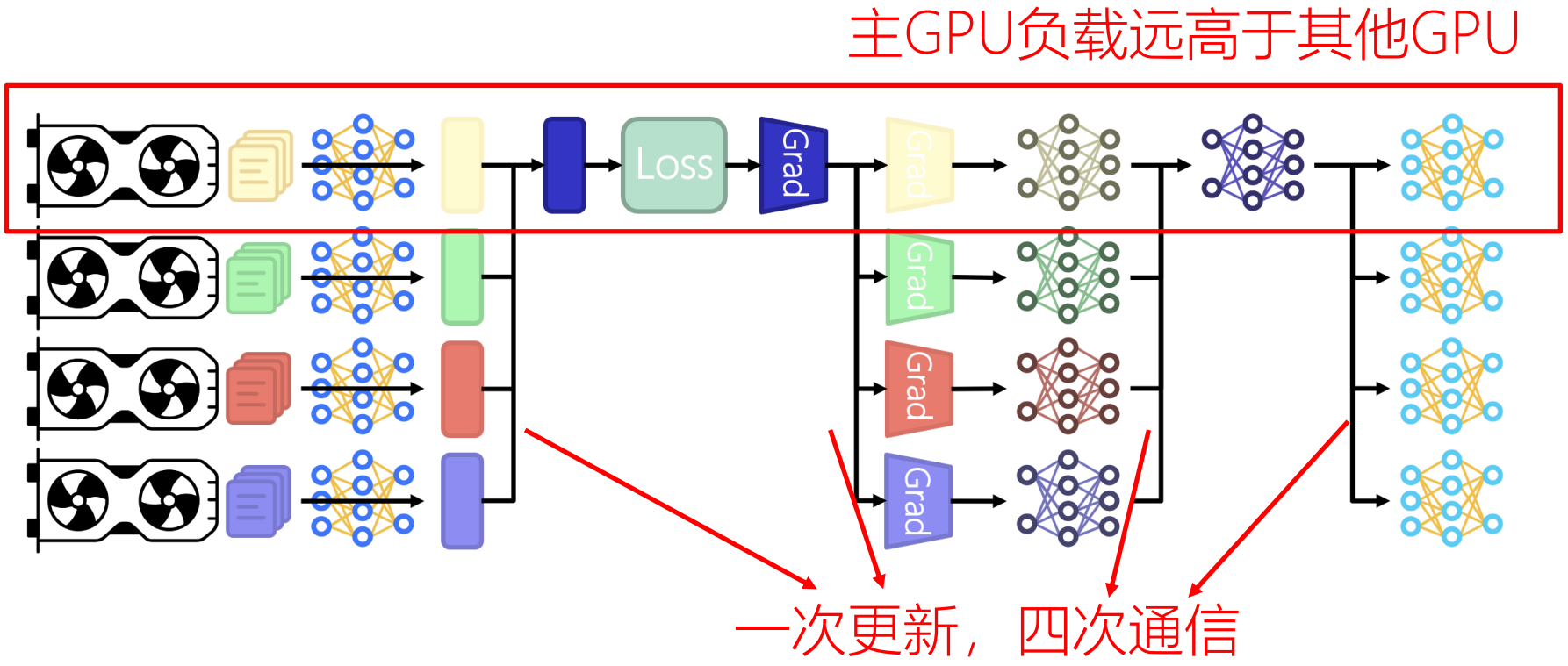
数据并行

PyTorch中的DP(DataParallel)就采用了上面的方法

优点：使用方便，基本无需修改已有的训练代码。

缺点：GPU负载严重不均衡，通信是很大的瓶颈

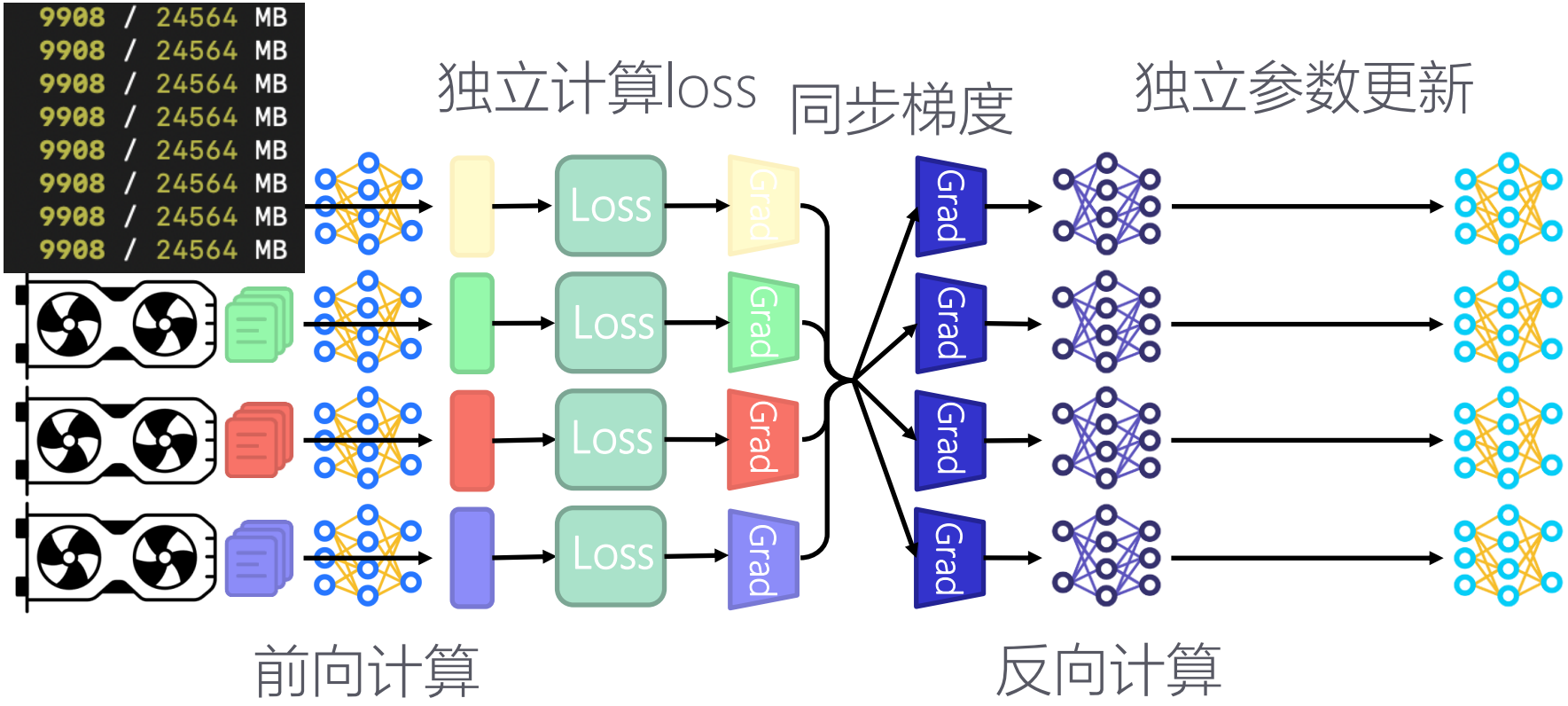
```
18186 / 24564 MB
 9824 / 24564 MB
 9824 / 24564 MB
 9824 / 24564 MB
 9824 / 24564 MB
 9824 / 24564 MB
 9824 / 24564 MB
 9824 / 24564 MB
```



数据并行

PyTorch中的DDP(DistributedDataParallel)对此进行了改进

Loss在各GPU中独立计算，各GPU之间只同步一次梯度，然后独立进行反向传播和参数更新



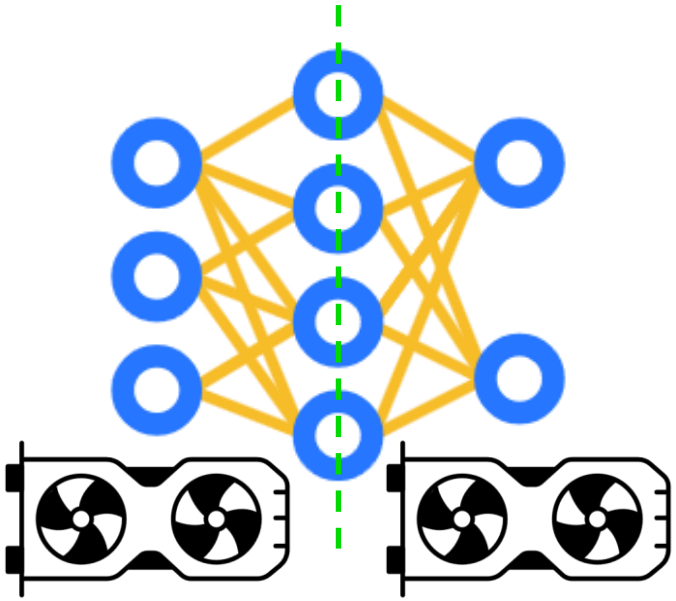
目录

模型并行

数据并行中，每个GPU上都加载完整的模型，能够训练的模型大小受到显存大小限制

如何训练更大参数量的模型？

我们需要进一步拆分模型



一张卡放不下，
那就放在两张卡上

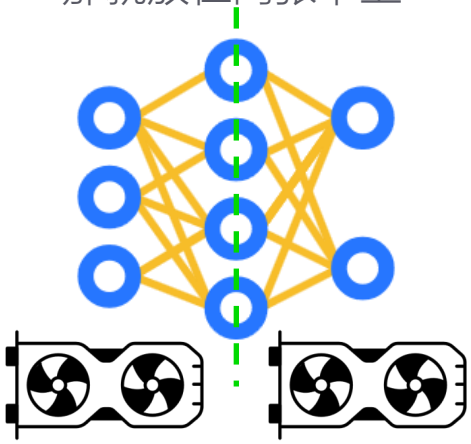
模型并行

数据并行中，每个GPU上都加载完整的模型，能够训练的模型大小受到显存大小限制

如何训练更大参数量的模型？

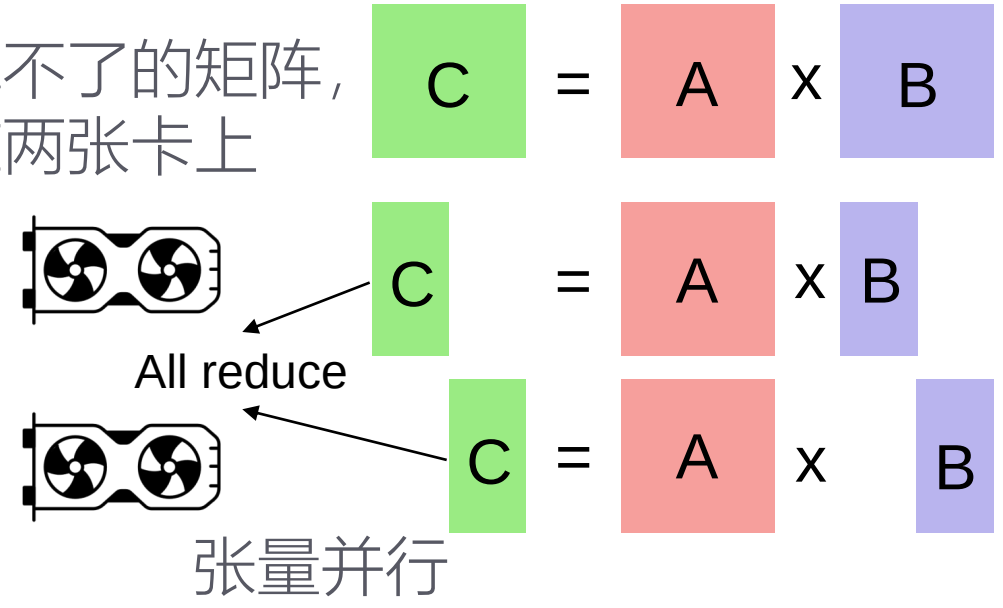
我们需要进一步拆分模型
- 流水线并行：将模型按层拆分
- 张量并行：将矩阵计算拆分

一张卡放不下的模型，那就放在两张卡上



流水线并行

一张卡算不了的矩阵，那就放在两张卡上

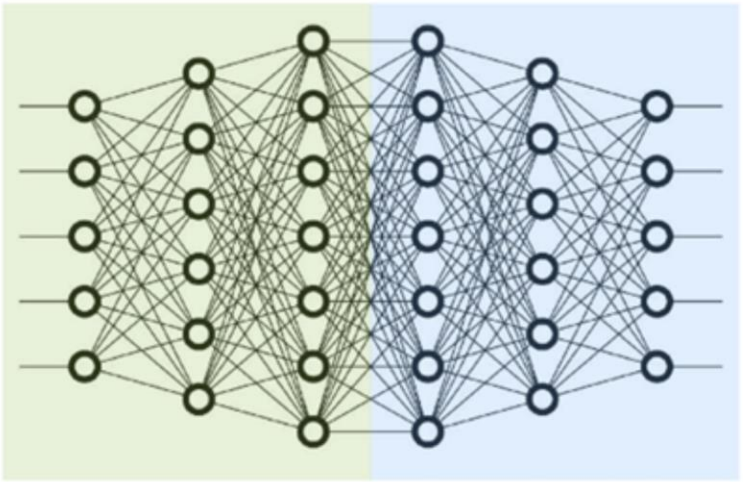


张量并行

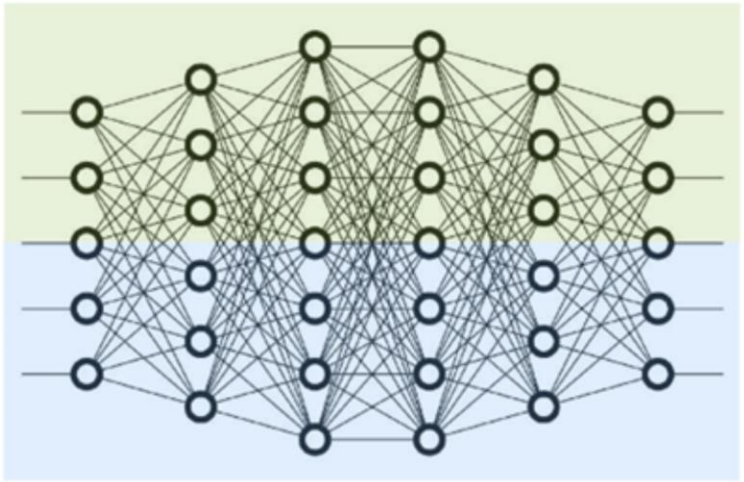
模型并行

模型并行从计算图的切分角度，可以分为以下几种：

- 按模型的layer层切分到不同设备，即层间并行，我们称之为流水线并行。
- 将计算图中的层内的参数切分到不同设备，即层内并行，我们称之为张量模型并行。



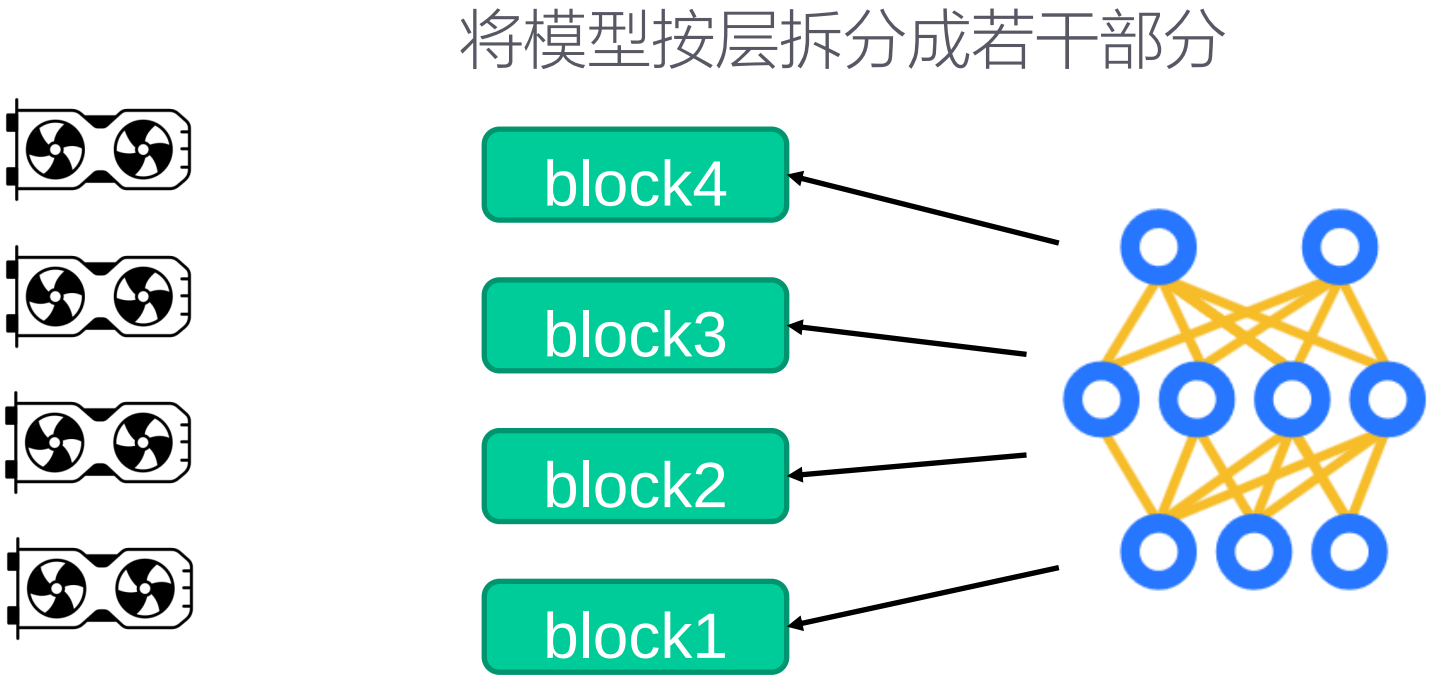
层间并行



层内并行

流水线并行

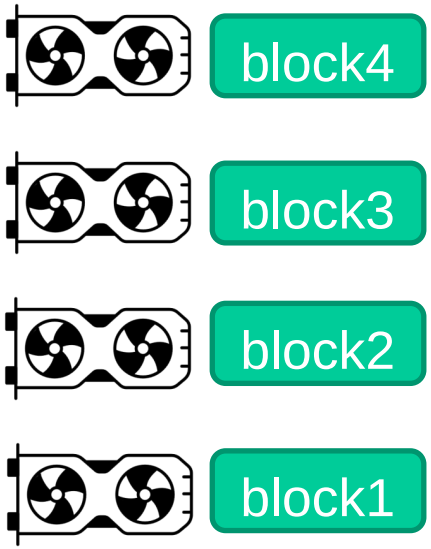
和计算机的流水线一样，将模型分为几个不同的部分，不同的显卡所加载的模型的不同的层，依次参与计算：



流水线并行

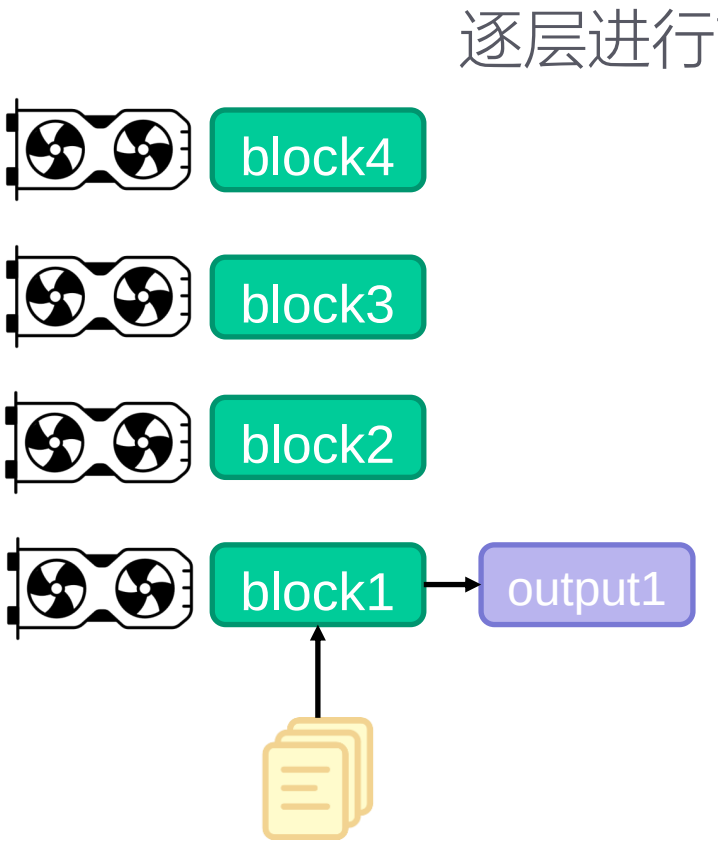
和计算机的流水线一样，将模型分为几个不同的部分，不同的显卡所加载的模型的不同的层，依次参与计算：

不同GPU上加载模型的不同层



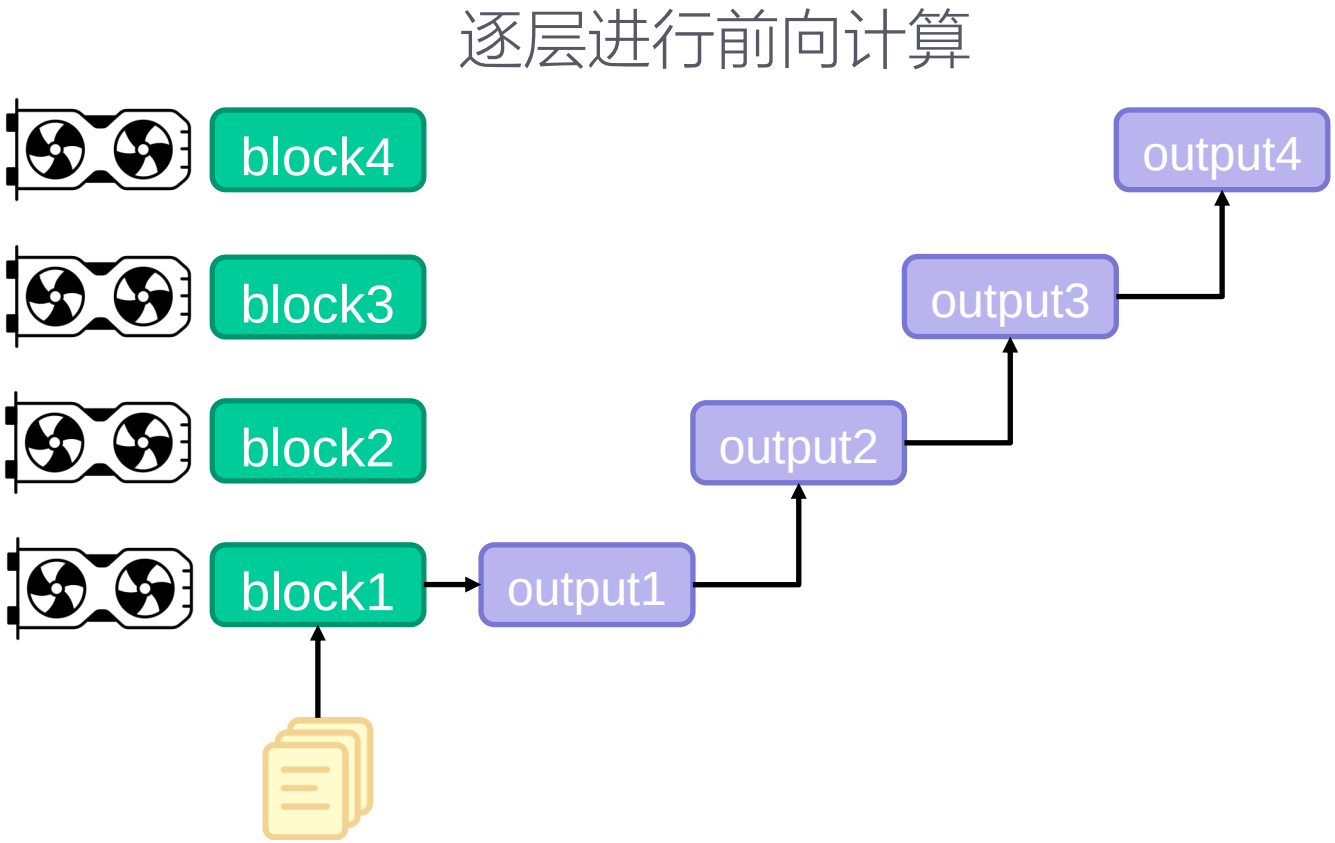
流水线并行

和计算机的流水线一样，将模型分为几个不同的部分，不同的显卡所加载的模型的不同的层，依次参与计算：



流水线并行

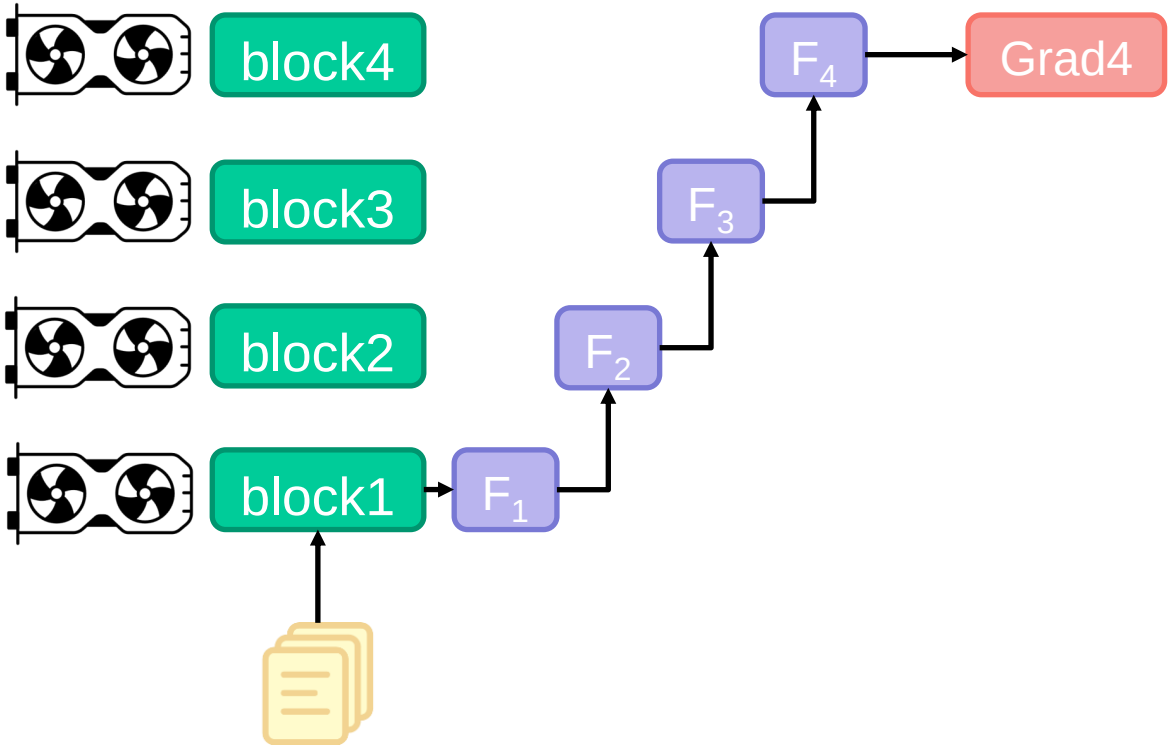
和计算机的流水线一样，将模型分为几个不同的部分，不同的显卡所加载的模型的不同的层，依次参与计算：



流水线并行

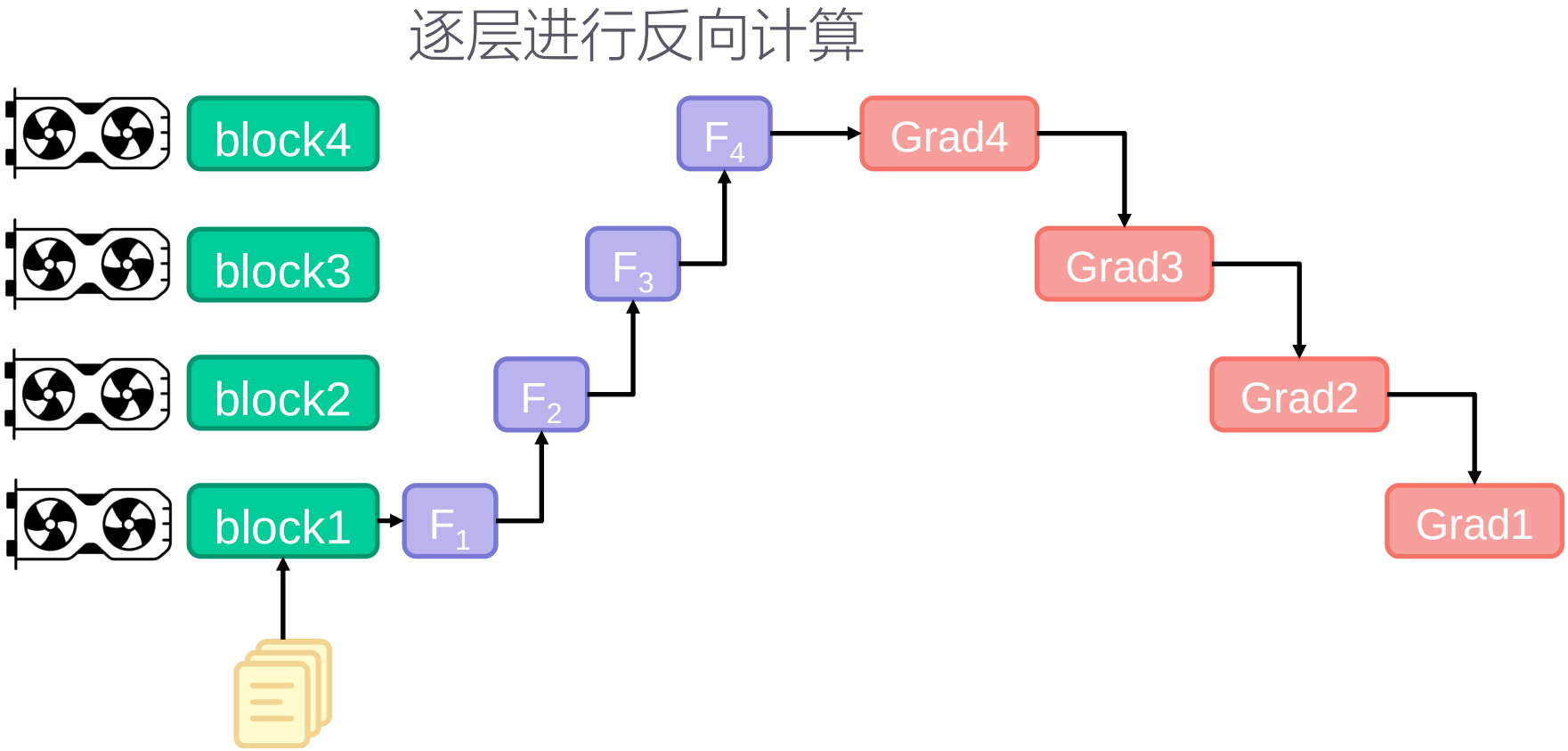
和计算机的流水线一样，将模型分为几个不同的部分，不同的显卡所加载的模型的不同的层，依次参与计算：

逐层进行反向计算



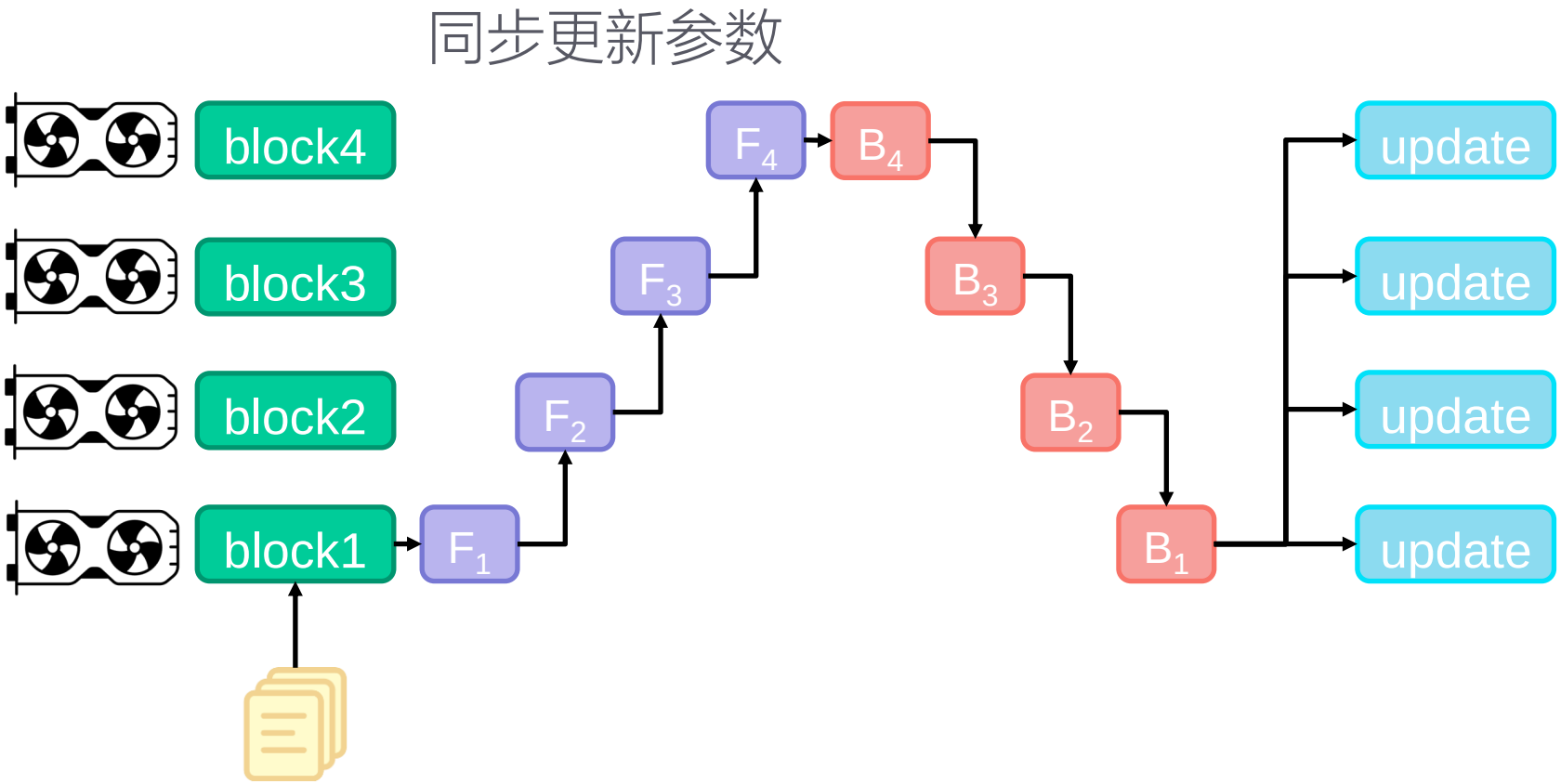
流水线并行

和计算机的流水线一样，将模型分为几个不同的部分，不同的显卡所加载的模型的不同的层，依次参与计算：



流水线并行

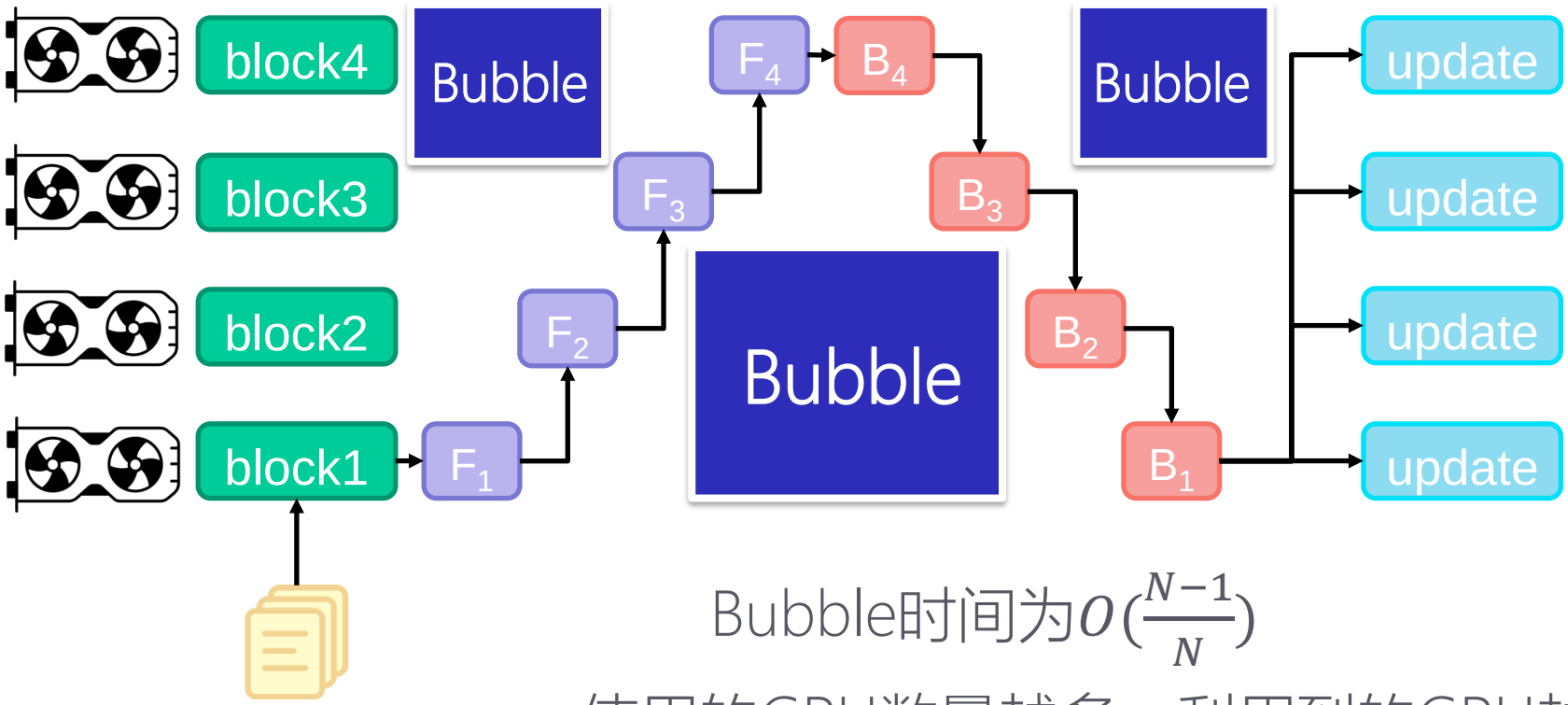
和计算机的流水线一样，将模型分为几个不同的部分，不同的显卡所加载的模型的不同的层，依次参与计算：



流水线并行

和计算机的流水线一样，将模型分为几个不同的部分，不同的显卡所加载的模型的不同的层，依次参与计算：

计算资源的利用率不高

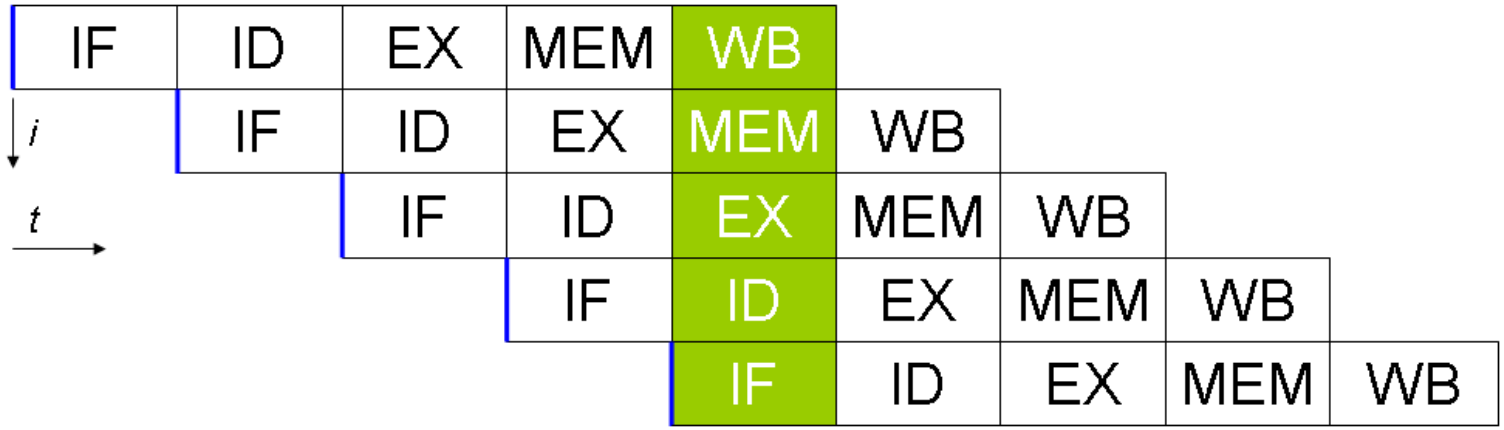


Bubble时间为$O(\frac{N-1}{N})$

使用的GPU数量越多，利用到的GPU越少

流水线并行

和计算机的流水线一样，将模型分为几个不同的部分，同步计算不同的批次数据，形成一个流水线



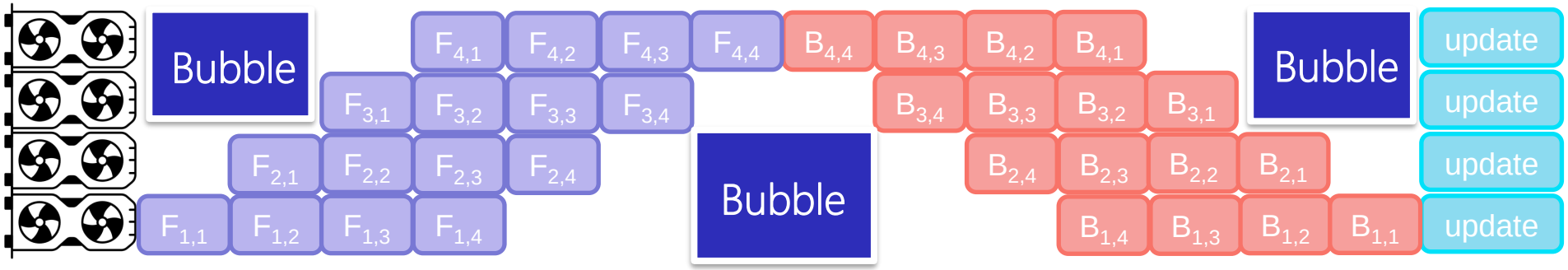
计算机流水线

https://commons.wikimedia.org/wiki/File:Fivestagespipeline.png

流水线并行

微批次(MicroBatch)流水线

将传入的小批次(MiniBatch)分块为微批次，人为创建流水线



Gpipe流水线

Bubble时间为$O(\frac{N-1}{N+M-1})$，M是划分MicroBatch的数量

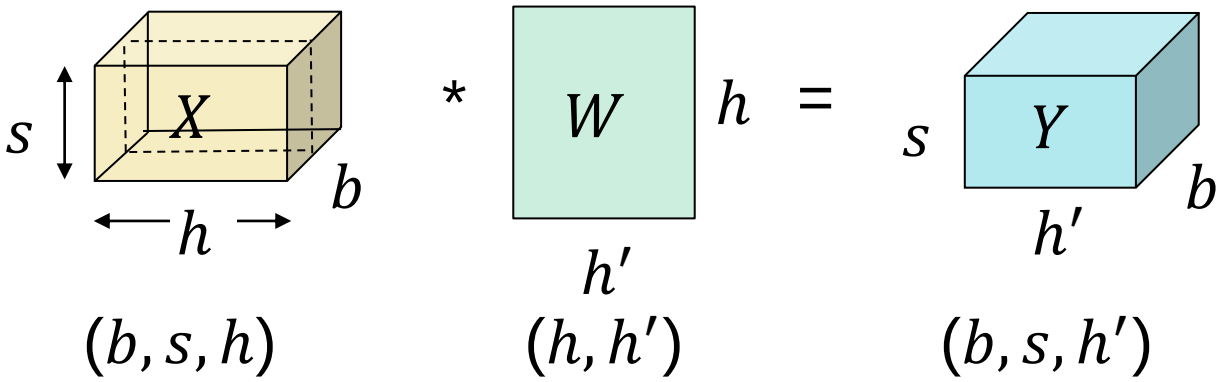
当$M \gg N$时，这个时间可以忽略不计

目录

张量并行

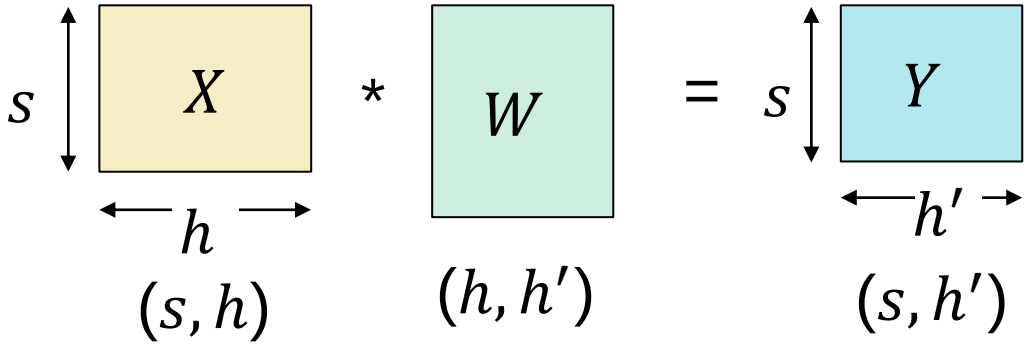
- 设输入数据为X，参数为W。X的维度 = (b, s, h)，W的维度 = (h, h')。其中：
  - b：批量（batch）大小；
  - s：输入序列的长度；
  - h：每个token向量的维度；
  - h'：参数W的hidden size。

- 那么每次forward的过程如下：



张量并行

- 在后续的讨论中如果不做特殊说明，我们不考虑batch大小，也就是假设$b = 1$：



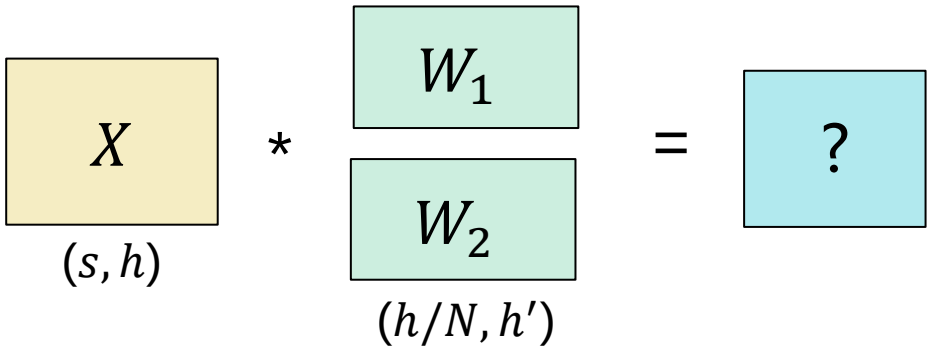
- 现在我们的问题是： W过大，导致单张显卡无法容纳。
- 我们需要将 W 切开并放到不同的GPU上。在这一过程中，我们面临三个主要问题：
  - 如何切分W；
  - 切分W后，如何进行forward；
  - 完成forward后，如何进行backward，以计算梯度并更新权重。

我们可以沿着W的行（h维度），或者列（h'维度）切分W。下面我们分别介绍这两种切割办法，并说明它们是如何做forward和backward的。

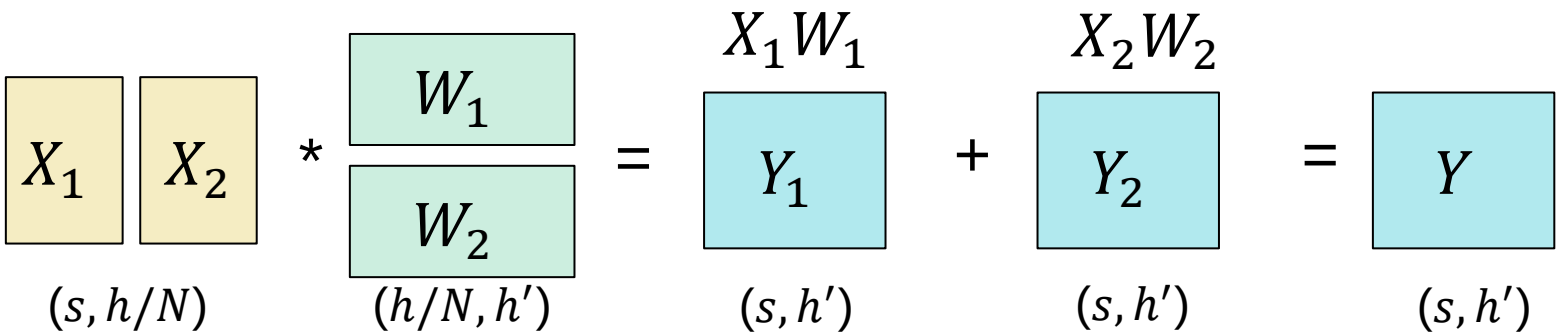
按行切分权重

(1) 矩阵切分

- N来表示GPU的数量。有几块GPU，就把W按行维度切成几份。下图展示了$N = 2$时的切割方式：



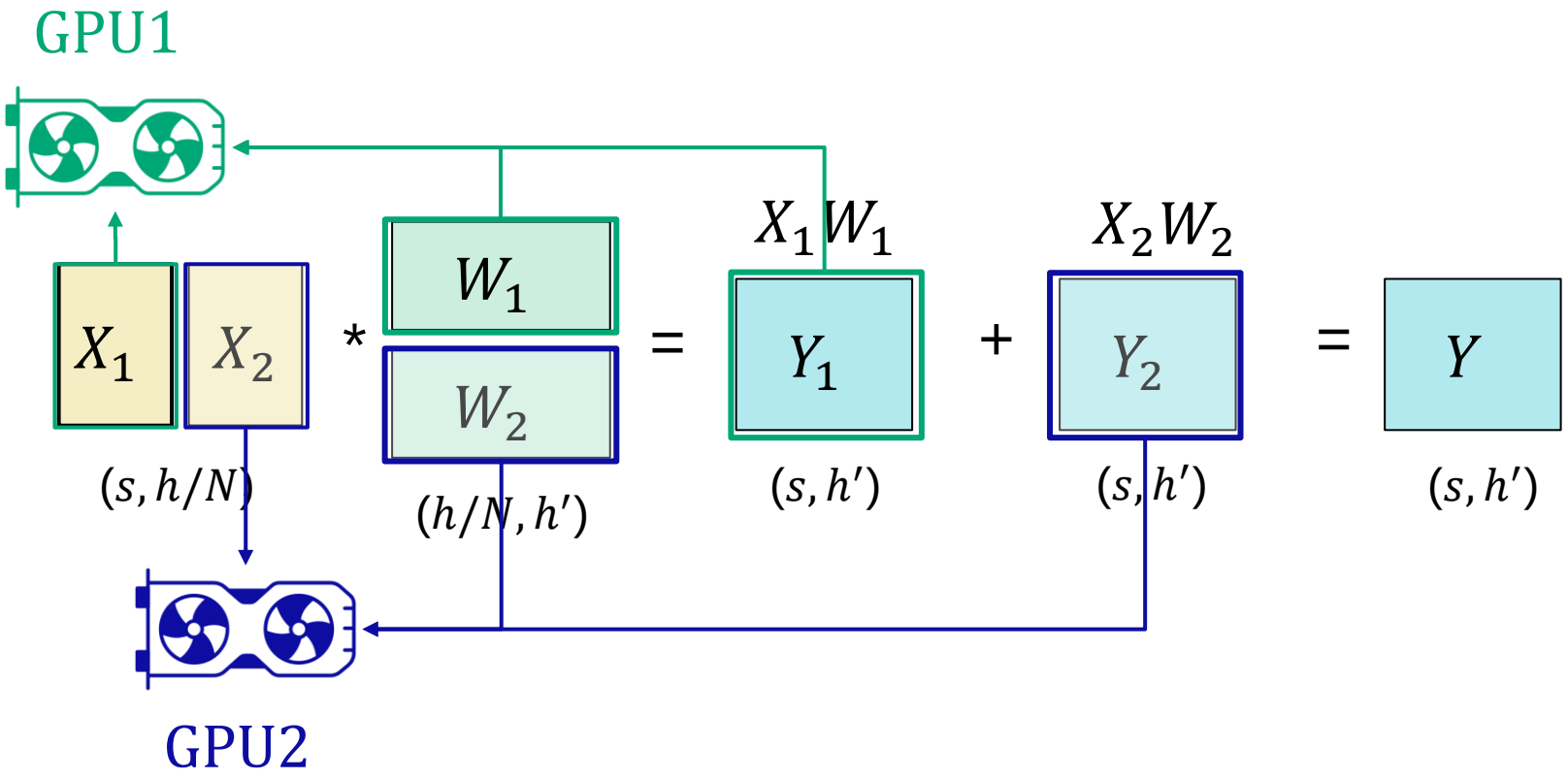
- W按照行维度切开后，X的维度和它不对齐了，如何再做矩阵乘法呢？
  解决办法：把X“按列切开”，如下图所示：



按行切分权重

(1) 矩阵切分

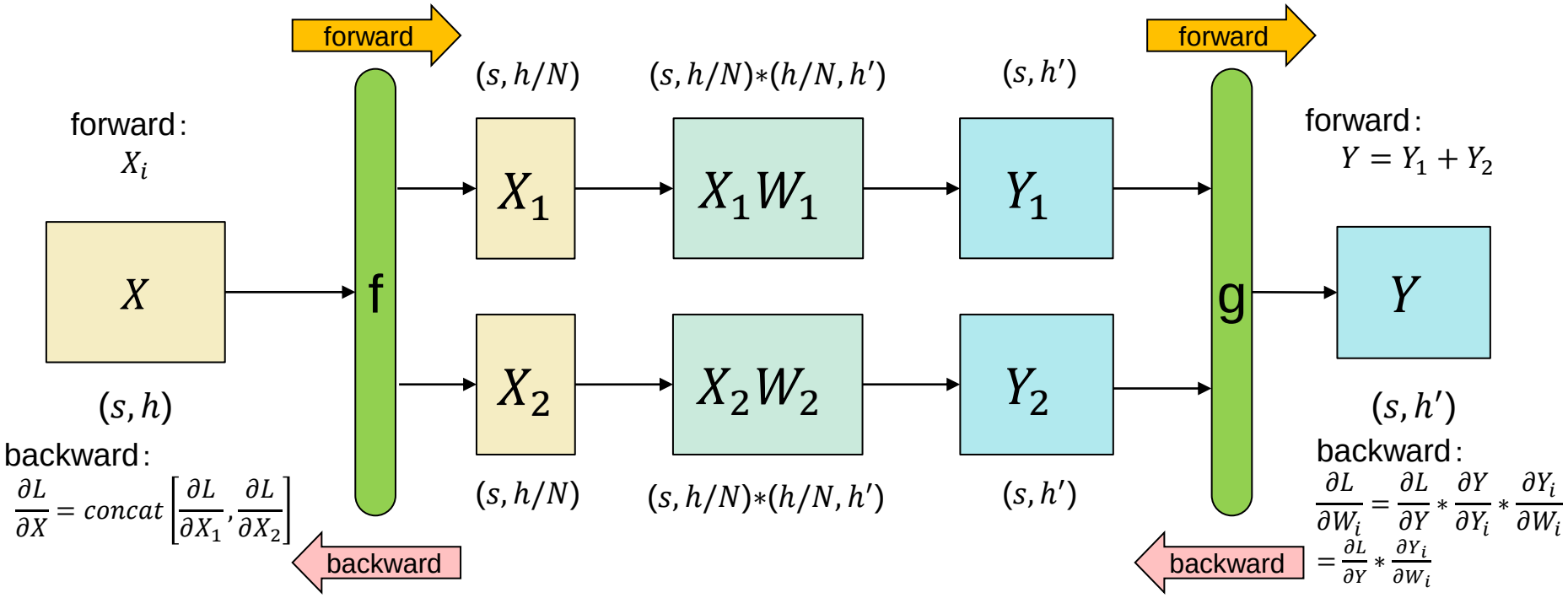
- 完成对输入数据和模型权重矩阵的切分后，我们就可以将X_1和W_1放在GPU1上，X_2和W_2放在GPU2上，分别进行计算。



张量并行

(2) Forward & Backward

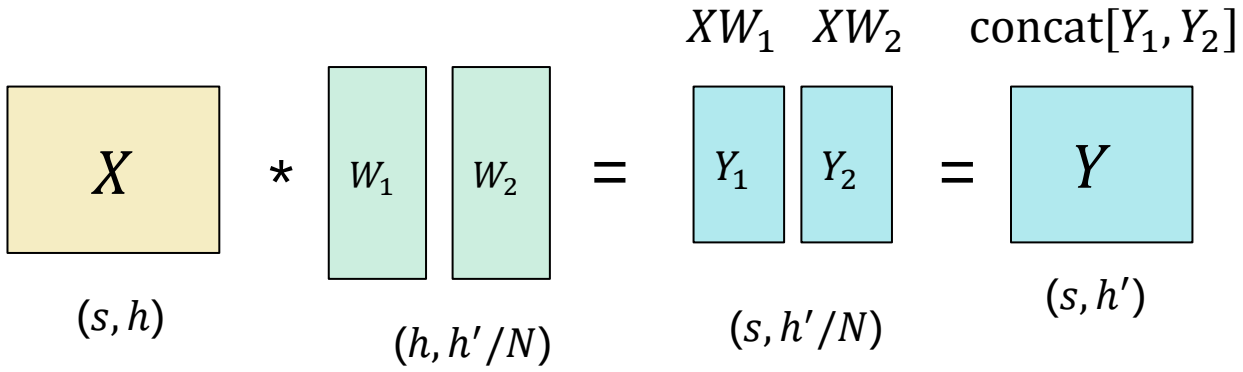
- 做完forward，取得预测值Y，进而可计算出损失L，接下来就能做backward。模型进行forward和backward的整体流程图如下：



按列切分权重

(1) 矩阵切分

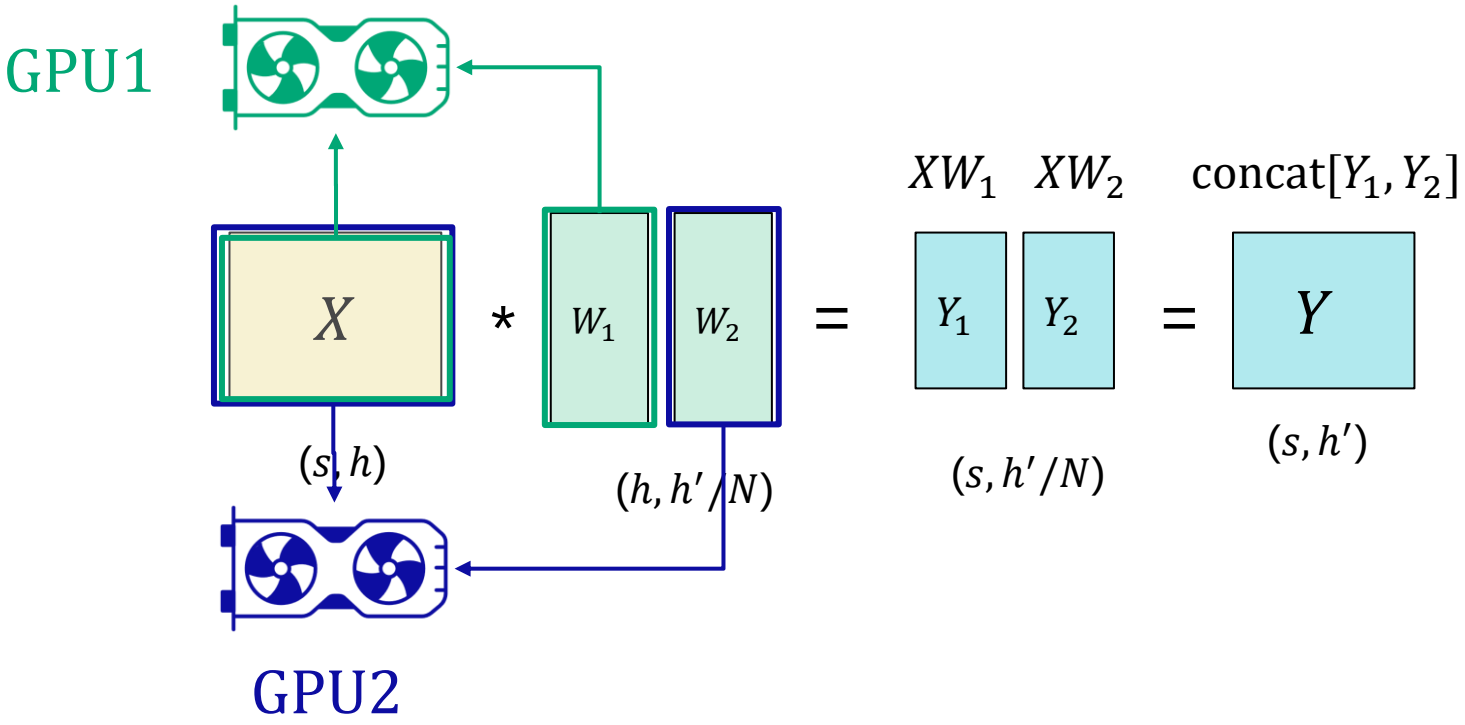
- 类似地，将权重按列切分后，相应的矩阵相乘计算图如下：



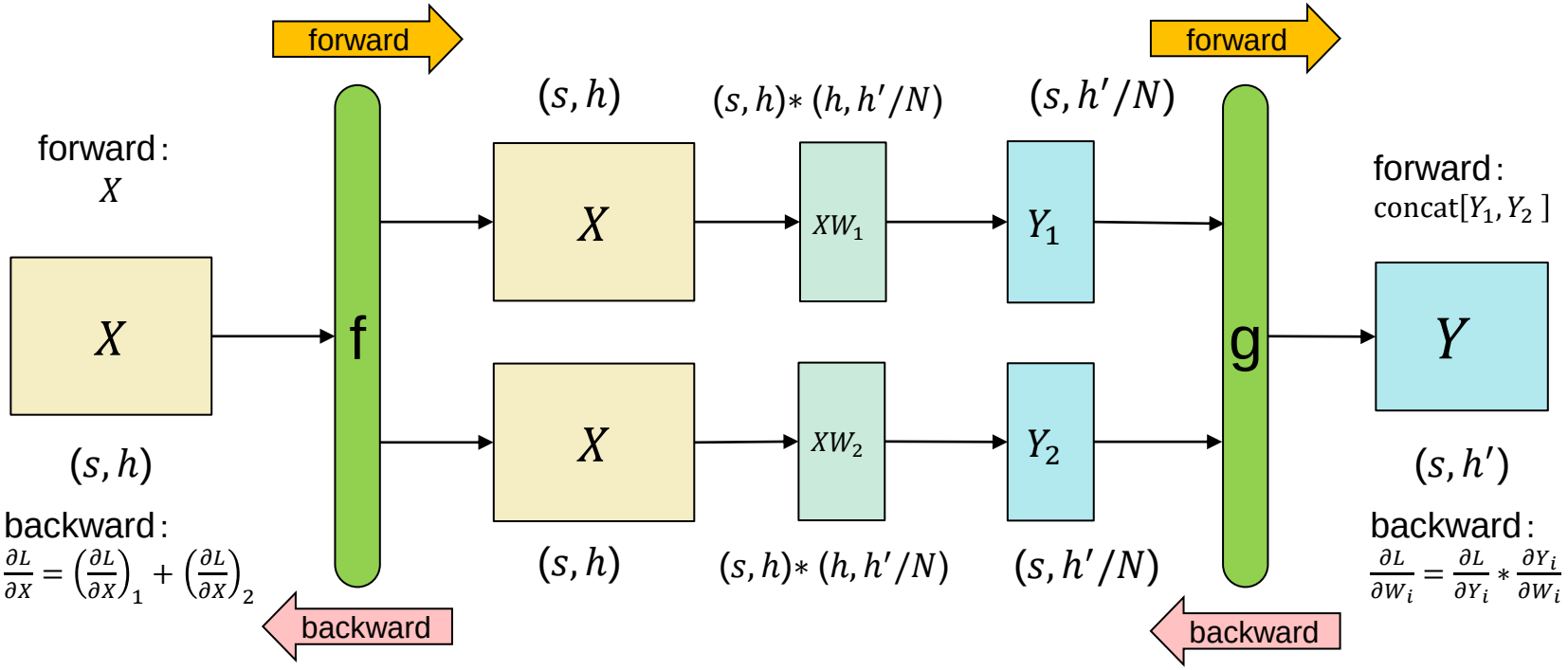
按列切分权重

(1) 矩阵切分

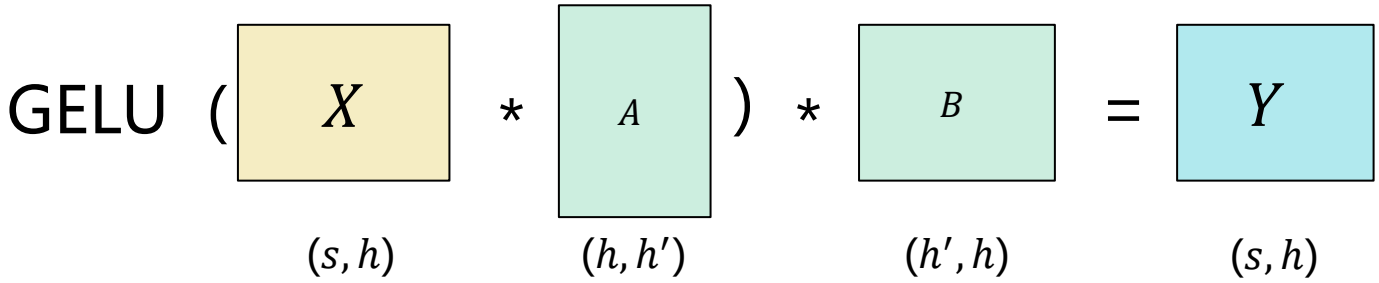
- 类似地，将权重按列切分后，相应的矩阵相乘计算图如下：



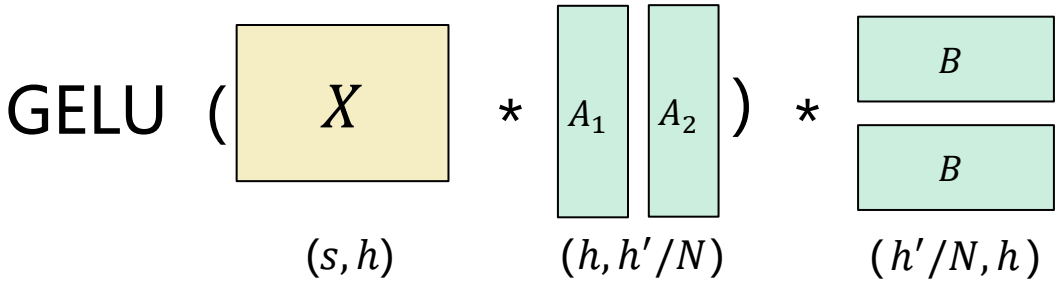
(2) Forward & Backward



- **MLP层**

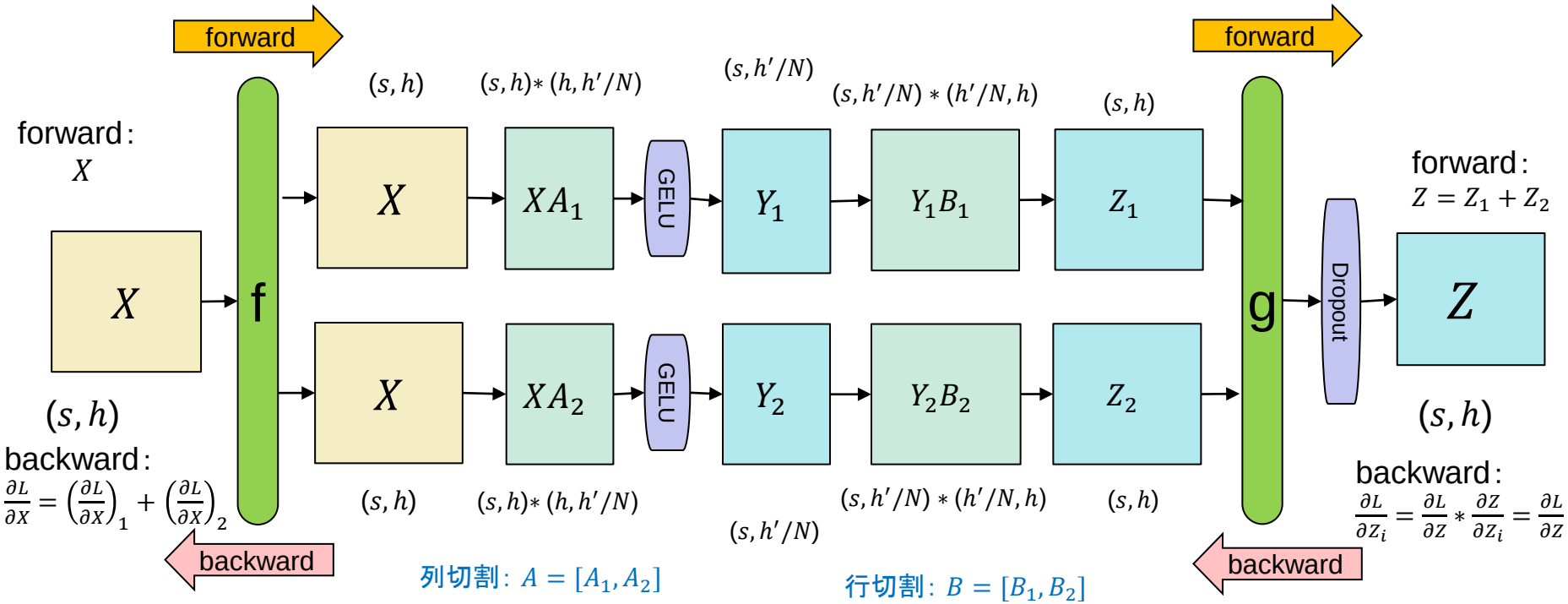


其中，GELU是激活函数，A和B分别为两个线性层。
权重切分：

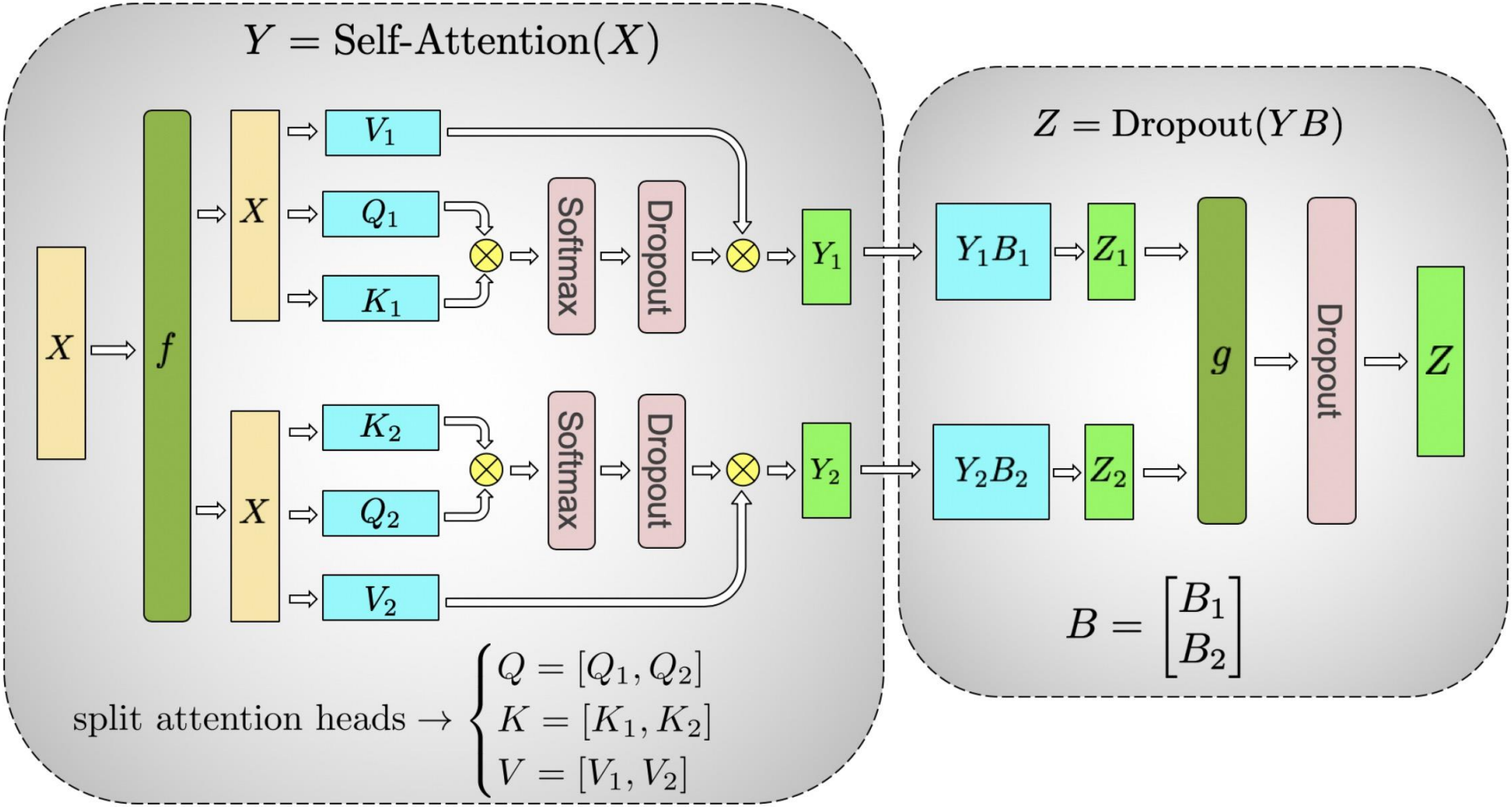


张量并行

- **MLP层张量并行**



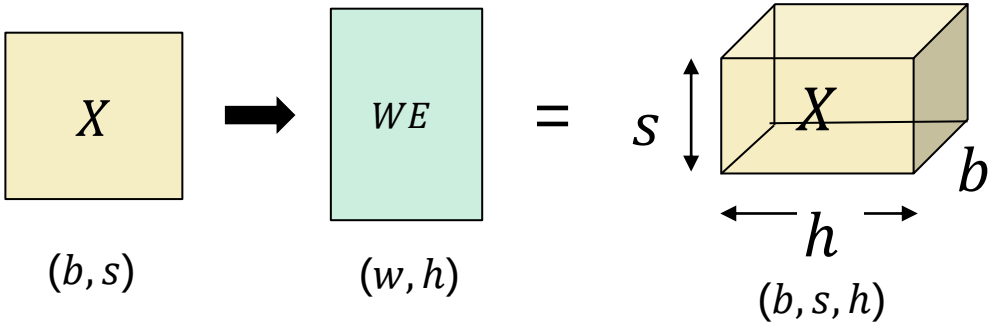
- **自注意力层张量并行**


$Y = \text{Self-Attention}(X)$
X
f
X
V_1
Q_1
K_1
Softmax
Dropout
Y_1
X
K_2
Q_2
V_2
Softmax
Dropout
Y_2
split attention heads $\rightarrow$ $Q = [Q_1, Q_2]$ $K = [K_1, K_2]$ $V = [V_1, V_2]$
$Z = \text{Dropout}(YB)$
Y_1B_1
Z_1
Y_2B_2
Z_2
g
Dropout
Z
$B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$

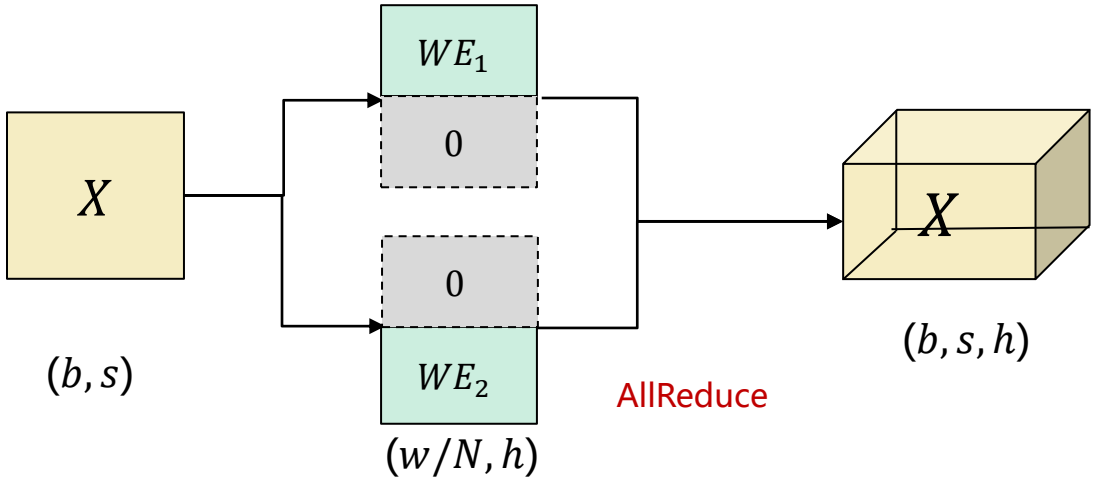
Embedding层

（1）输入层embedding

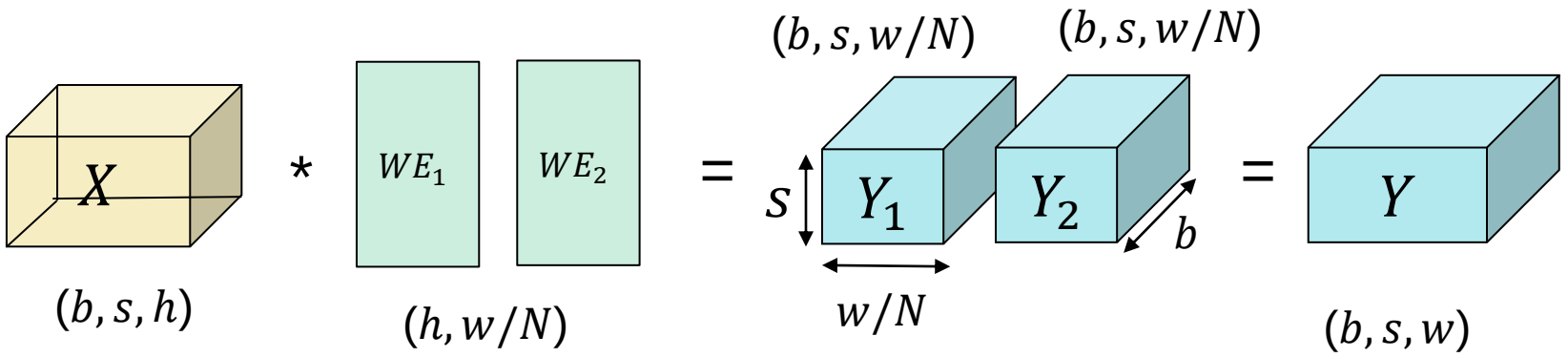
输入数据经过输入Embedding层的过程如下，其中WE代表词向量表，w代表总词表大小：



将Embedding层的参数按照词的维度进行切分，每张卡只存储部分词向量表。然后，通过AllReduce操作汇总各个设备上的部分词向量结果，从而得到完整的词向量结果。



(2)输出层embedding



- **CrossEntropy层**

经过输出层embedding后，CrossEntropy层负责计算模型的损失函数值。在实现张量模型并行时，我们可以将CrossEntropy层的参数按照类别的维度进行切分，每个设备只存储部分类别的参数。然后，在各个设备上并行计算损失函数值，并将各个设备上的损失函数值进行汇总，得到最终的损失函数值。

目录

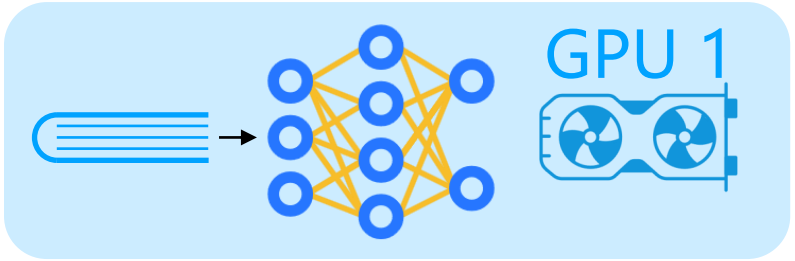
序列并行

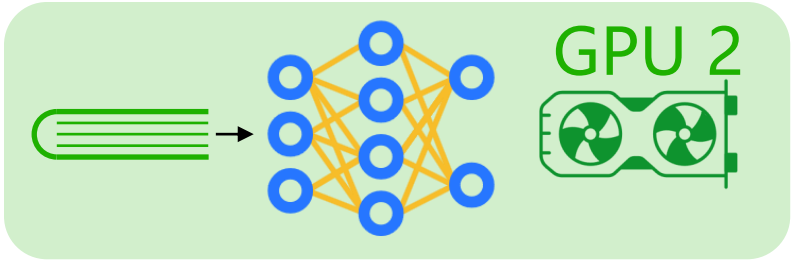
切分超长输入文本序列，同样可以实现并行。

用于训练的长文本

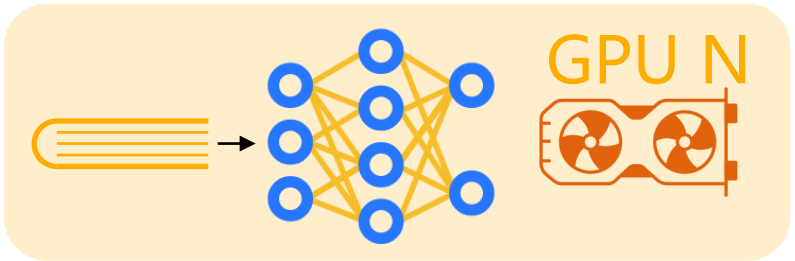
[**Batch Size**, Tokens, Hidden Dimension]

数据并行切分batch





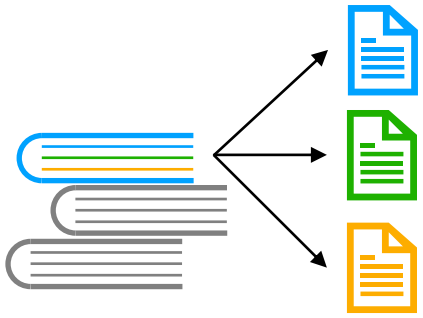
...



https://hanlab.mit.edu/courses/2024-fall-65940

序列并行

切分超长输入文本序列，同样可以实现并行。

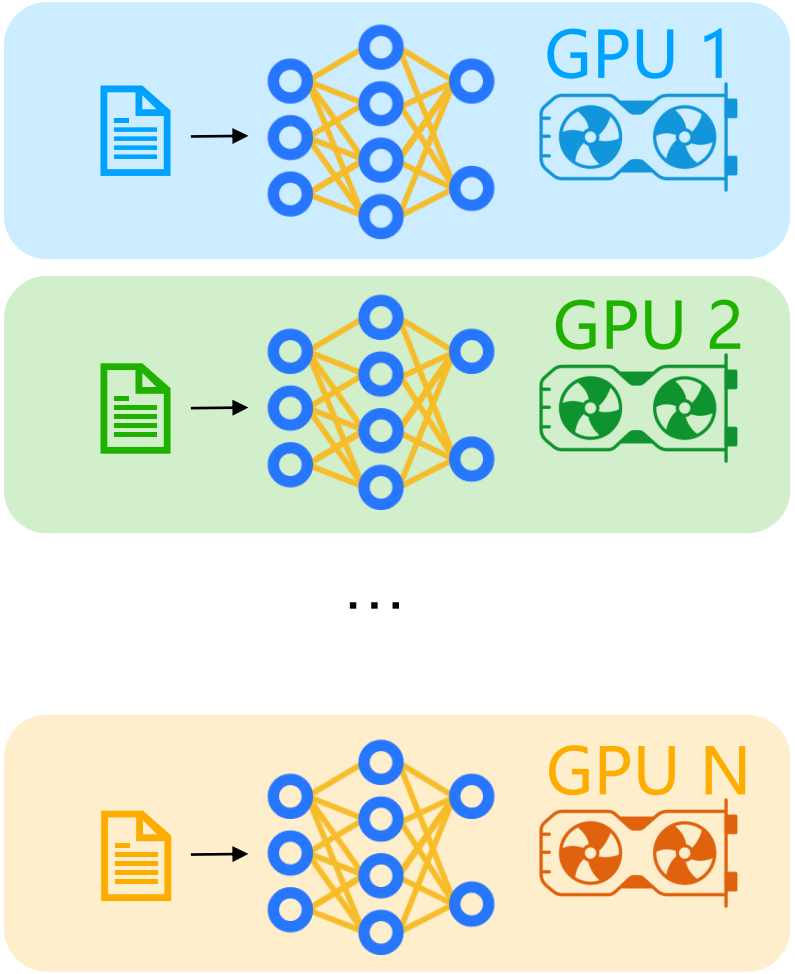


用于训练的长文本

[Batch Size, **Tokens**, Hidden Dimension]

可能会超过100k

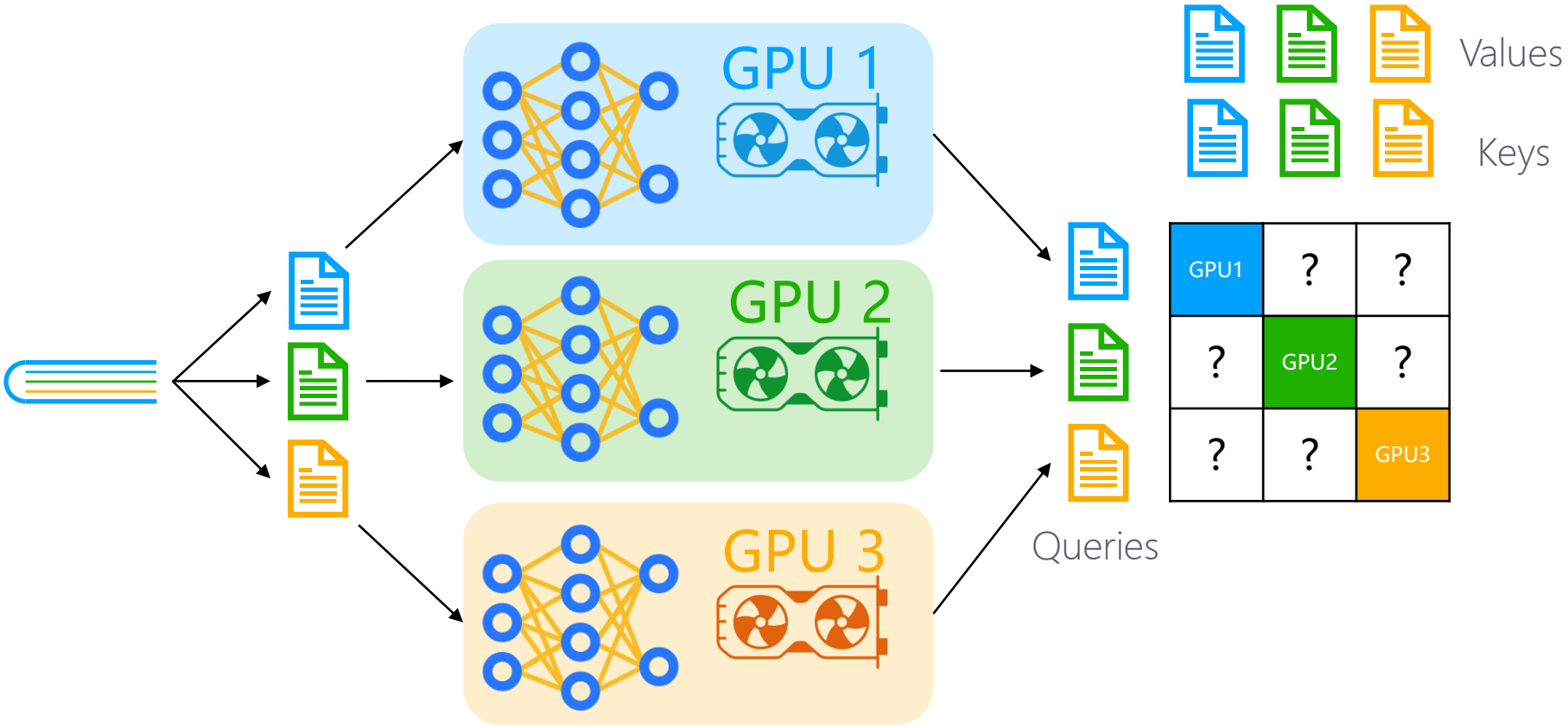
序列并行切分token



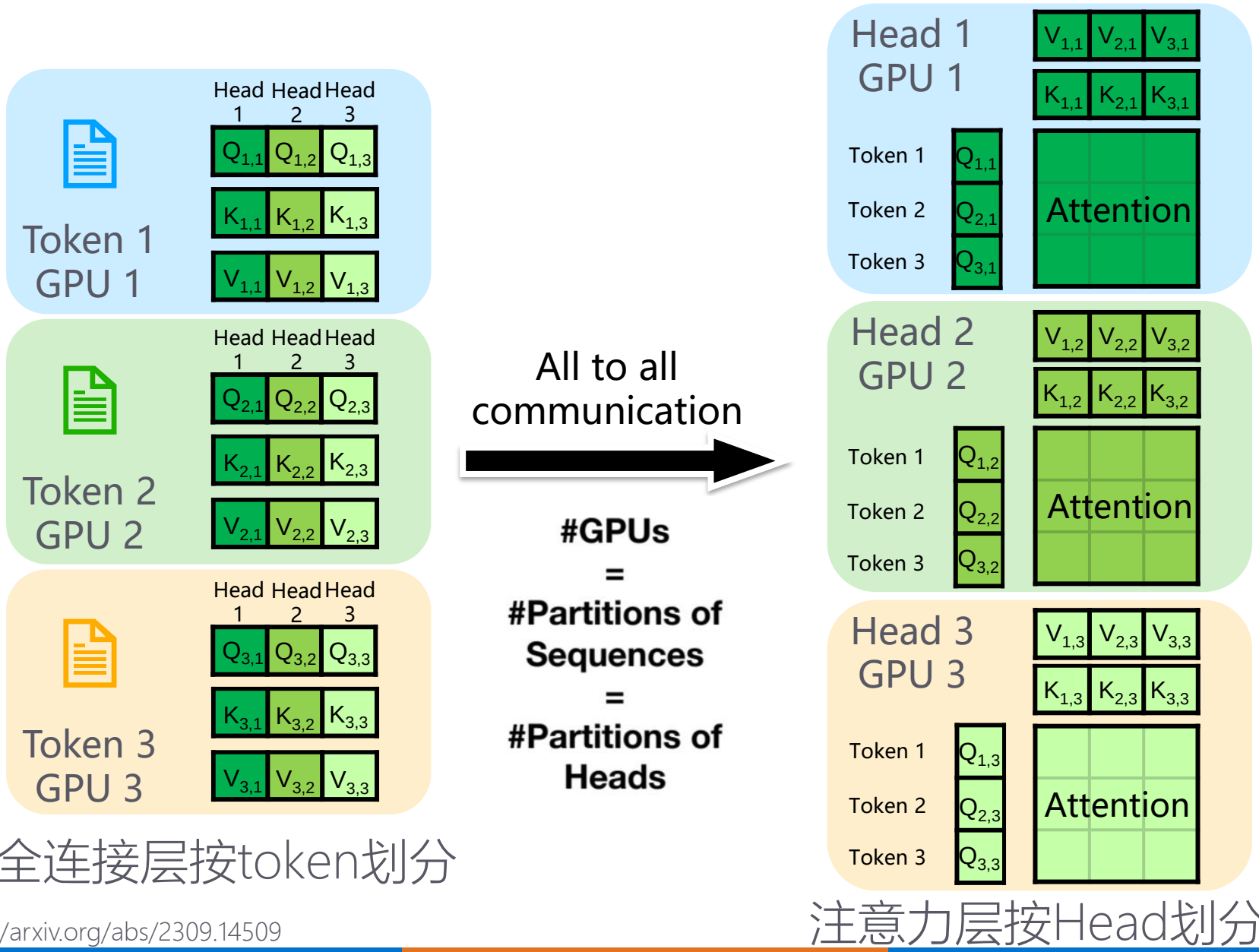
序列并行

全连接层的序列并行类似于数据并行，这里按下不表。

但是注意力层怎么办？超长序列上的attention如何计算？



序列并行：在attention层重组数据



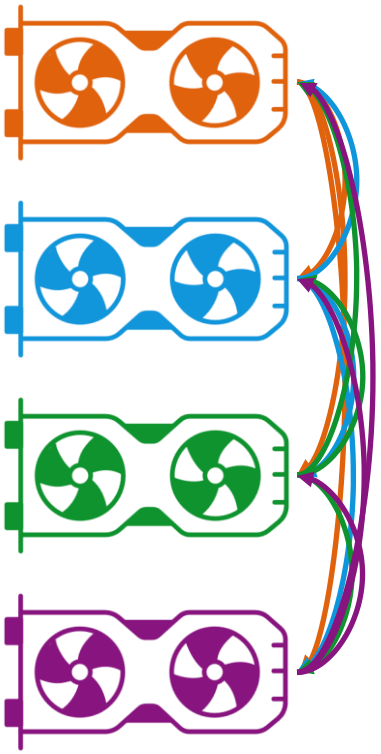
全连接层按token划分

注意力层按Head划分

https://arxiv.org/abs/2309.14509

序列并行：环注意力

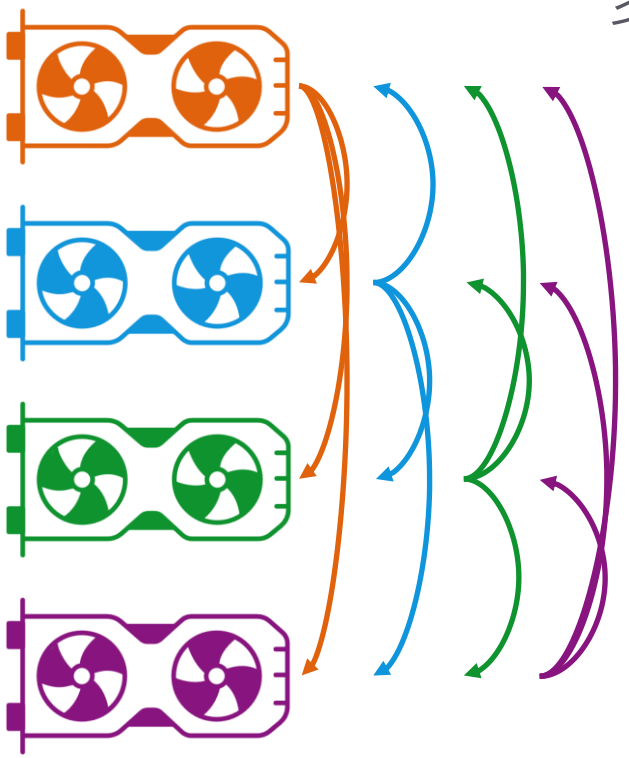
GPU间的通信瓶颈



序列并行：环注意力

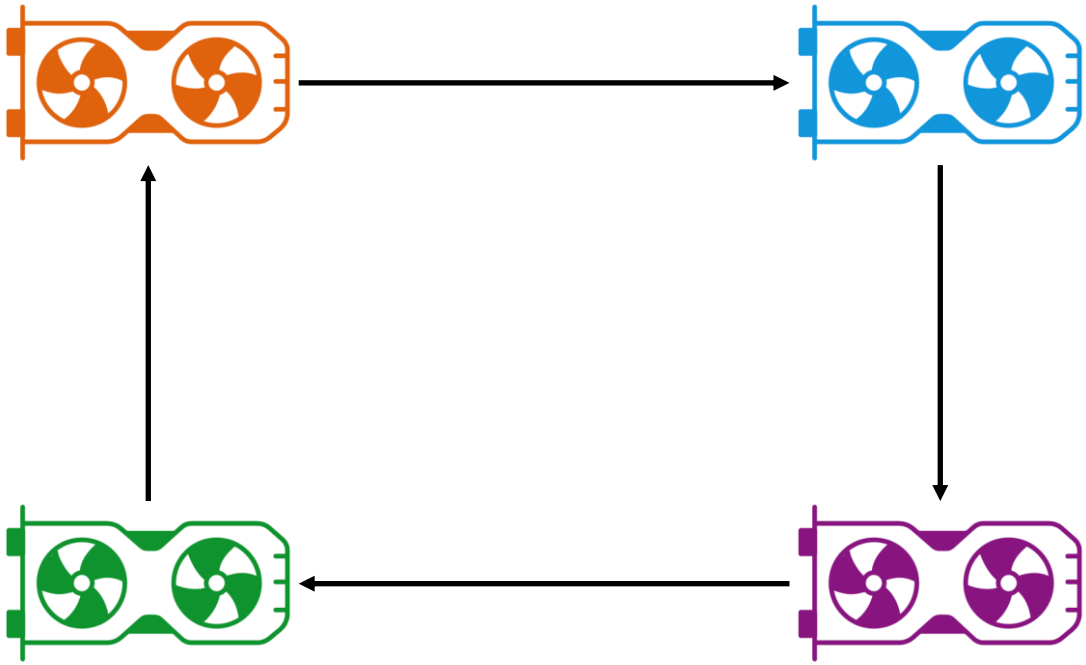
GPU间的通信瓶颈

多次通信的等待时间较长



序列并行：环注意力

GPU间的通信瓶颈

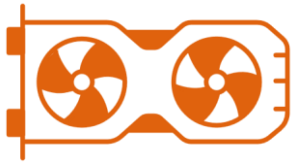


将GPU连成一个环！

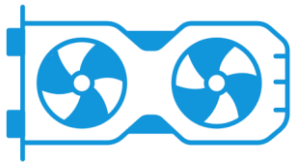
序列并行：环注意力

GPU间的通信瓶颈

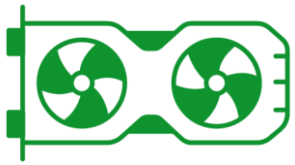
希望对各个部分分别求和



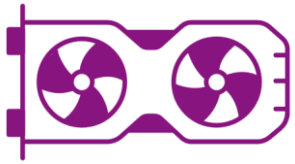
a_1	b_1	c_1	d_1



a_2	b_2	c_2	d_2



a_3	b_3	c_3	d_3

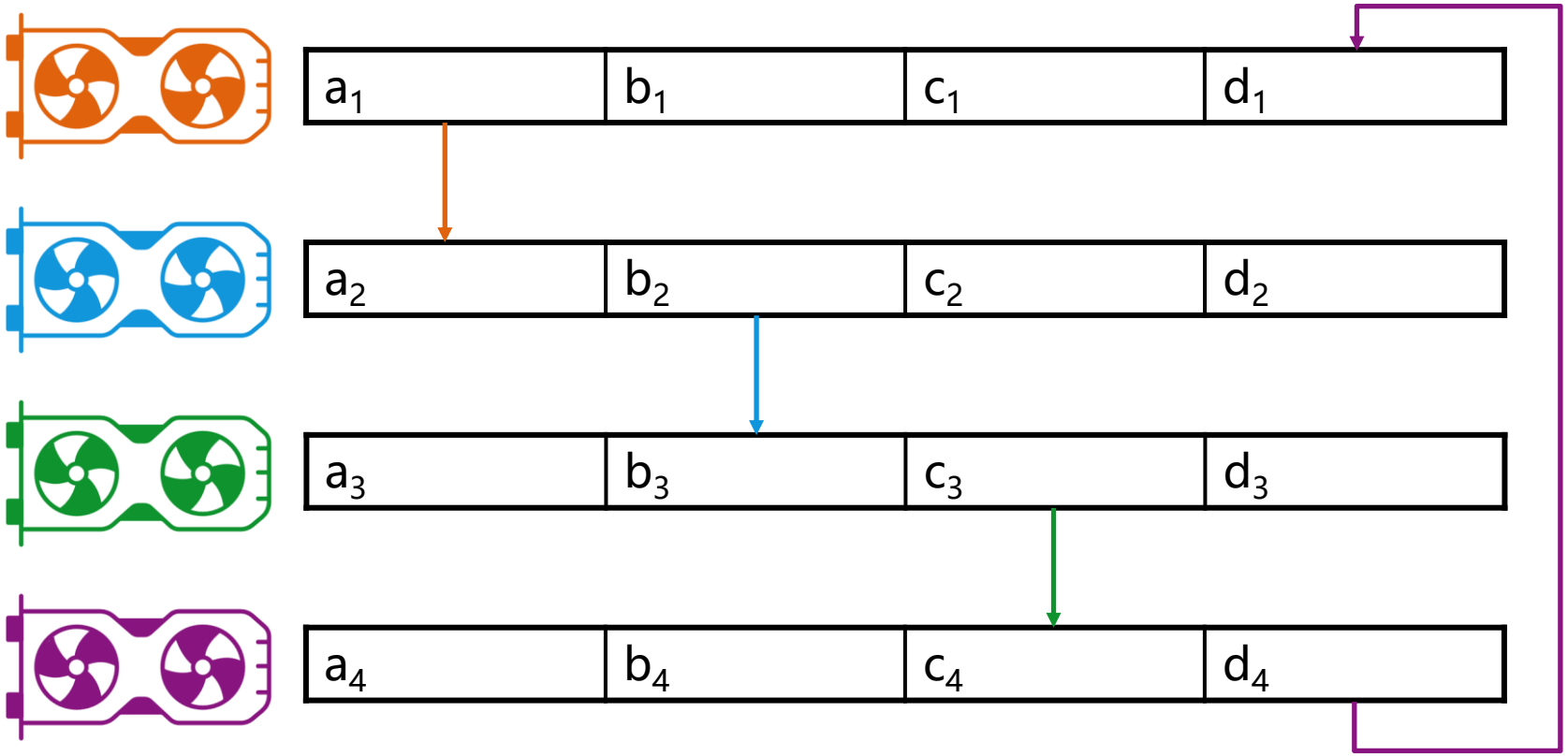


a_4	b_4	c_4	d_4

序列并行：环注意力

GPU间的通信瓶颈

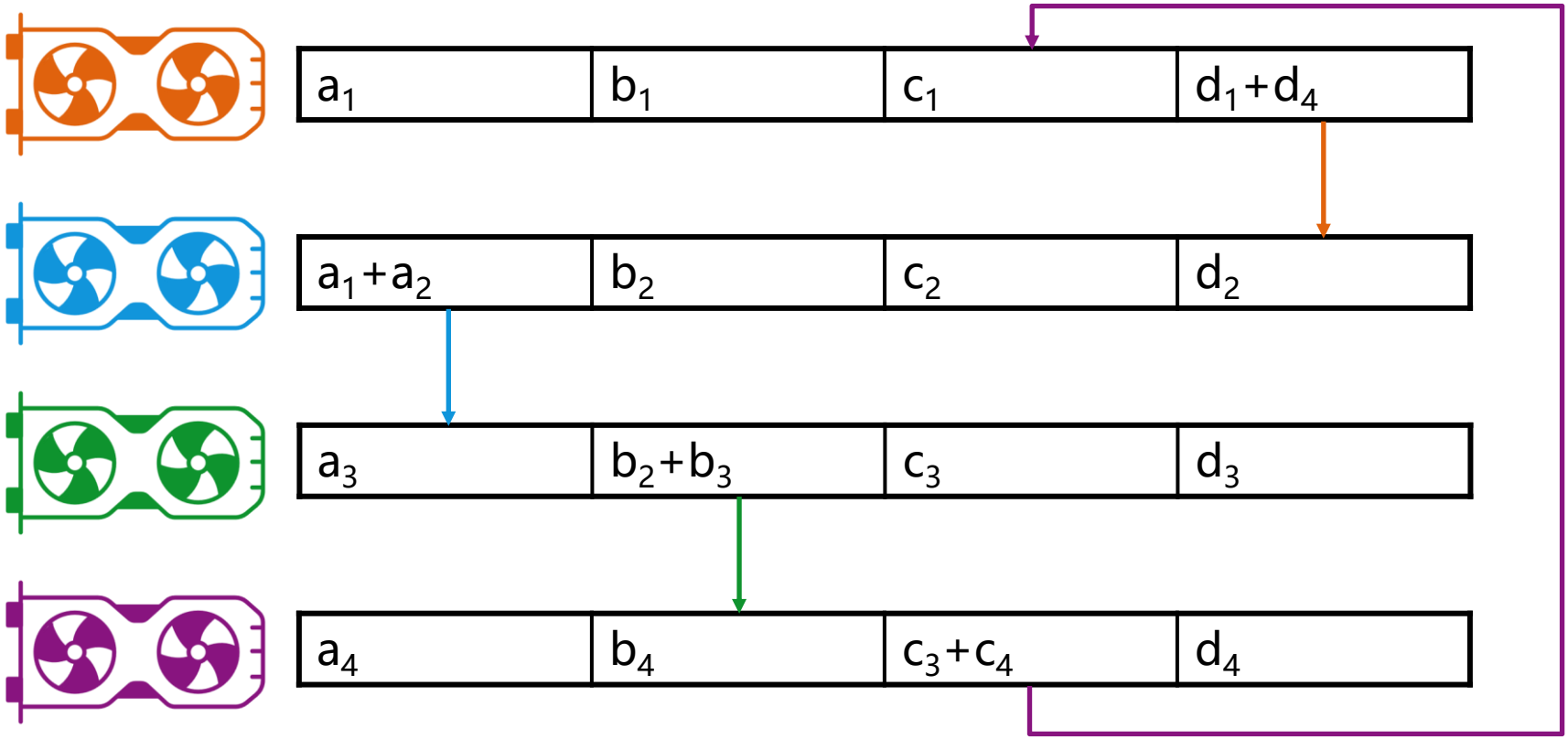
每个GPU只和“下一个”GPU通信



序列并行：环注意力

GPU间的通信瓶颈

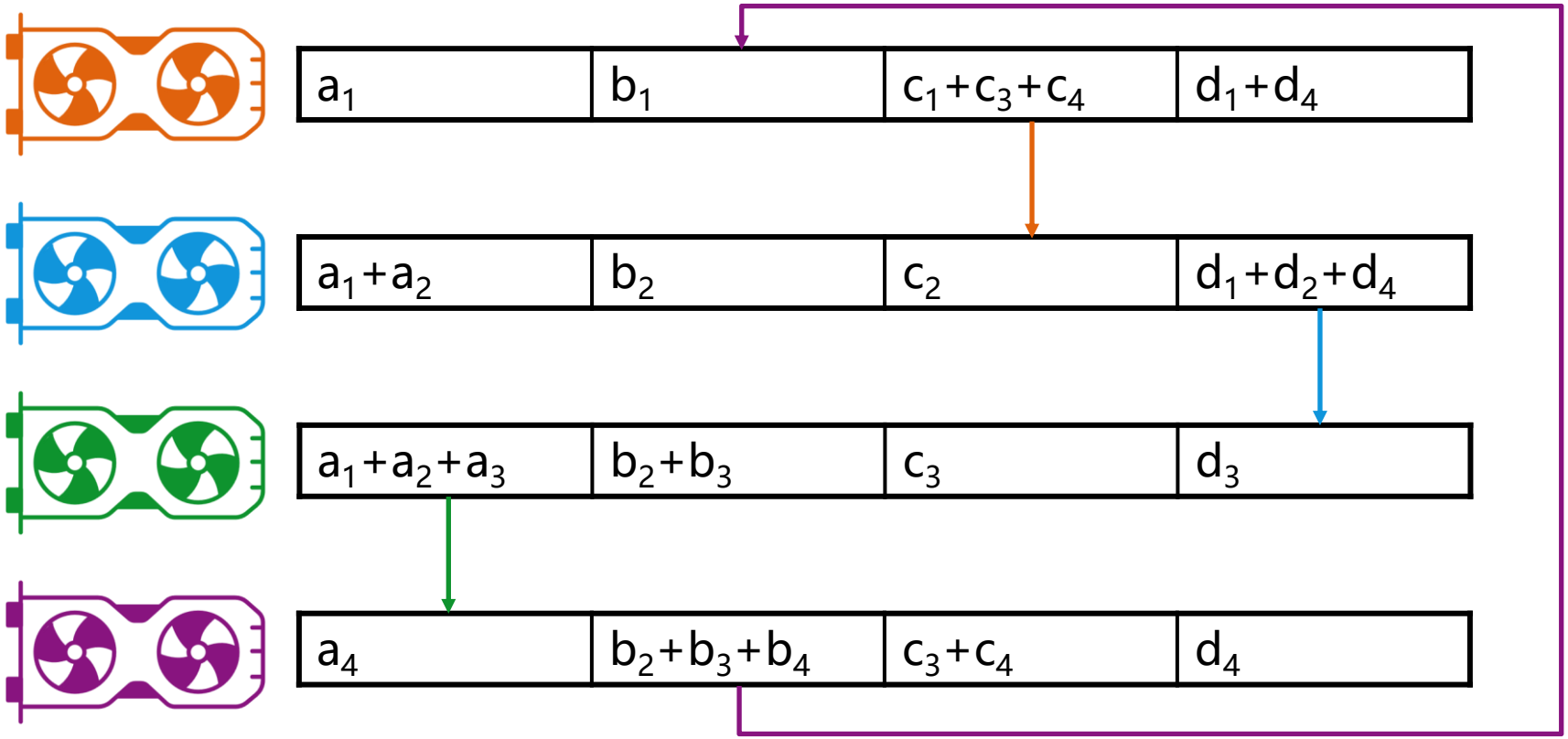
每个GPU只和“下一个”GPU通信



序列并行：环注意力

GPU间的通信瓶颈

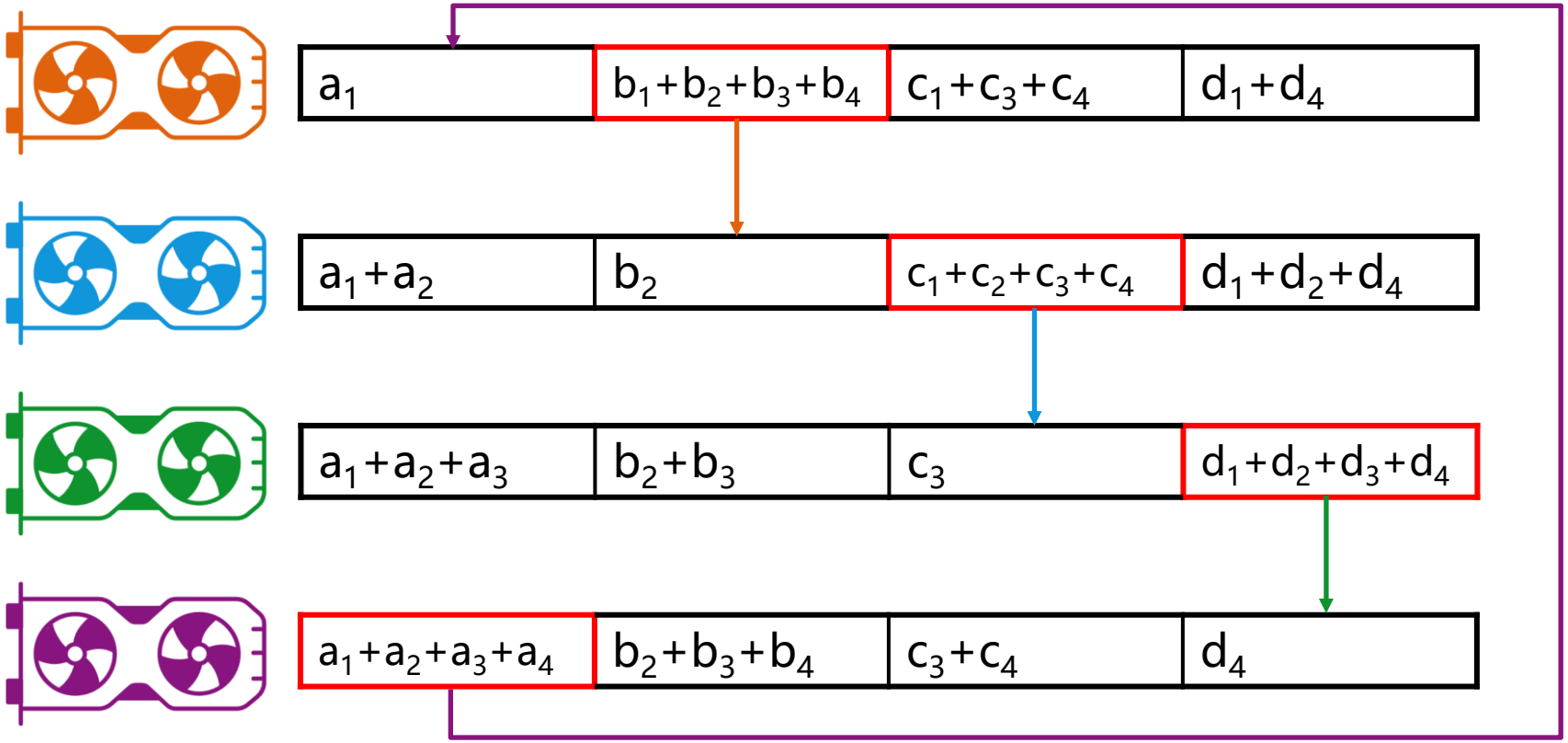
每个GPU只和“下一个”GPU通信



序列并行：环注意力

GPU间的通信瓶颈

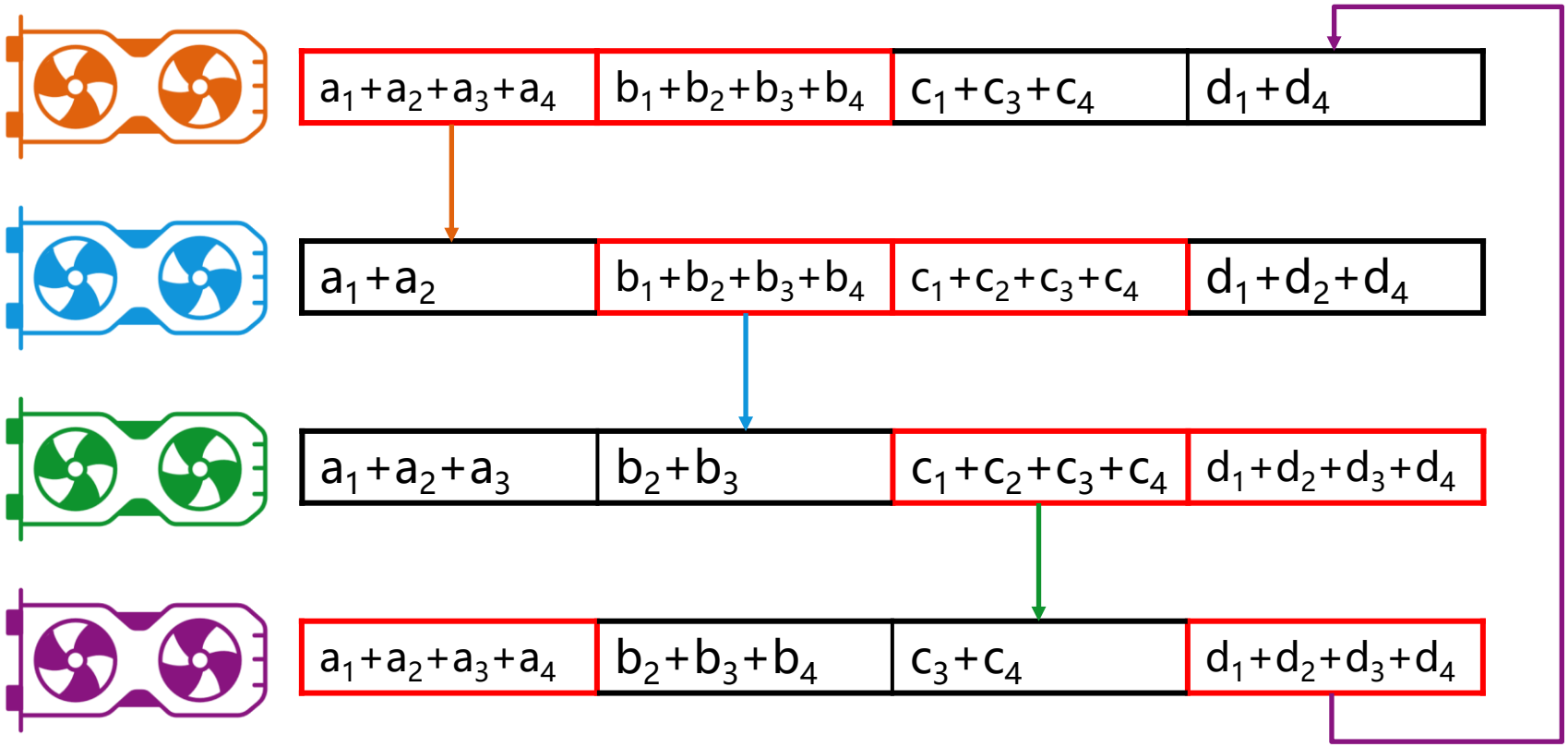
每个GPU只和“下一个”GPU通信



序列并行：环注意力

GPU间的通信瓶颈

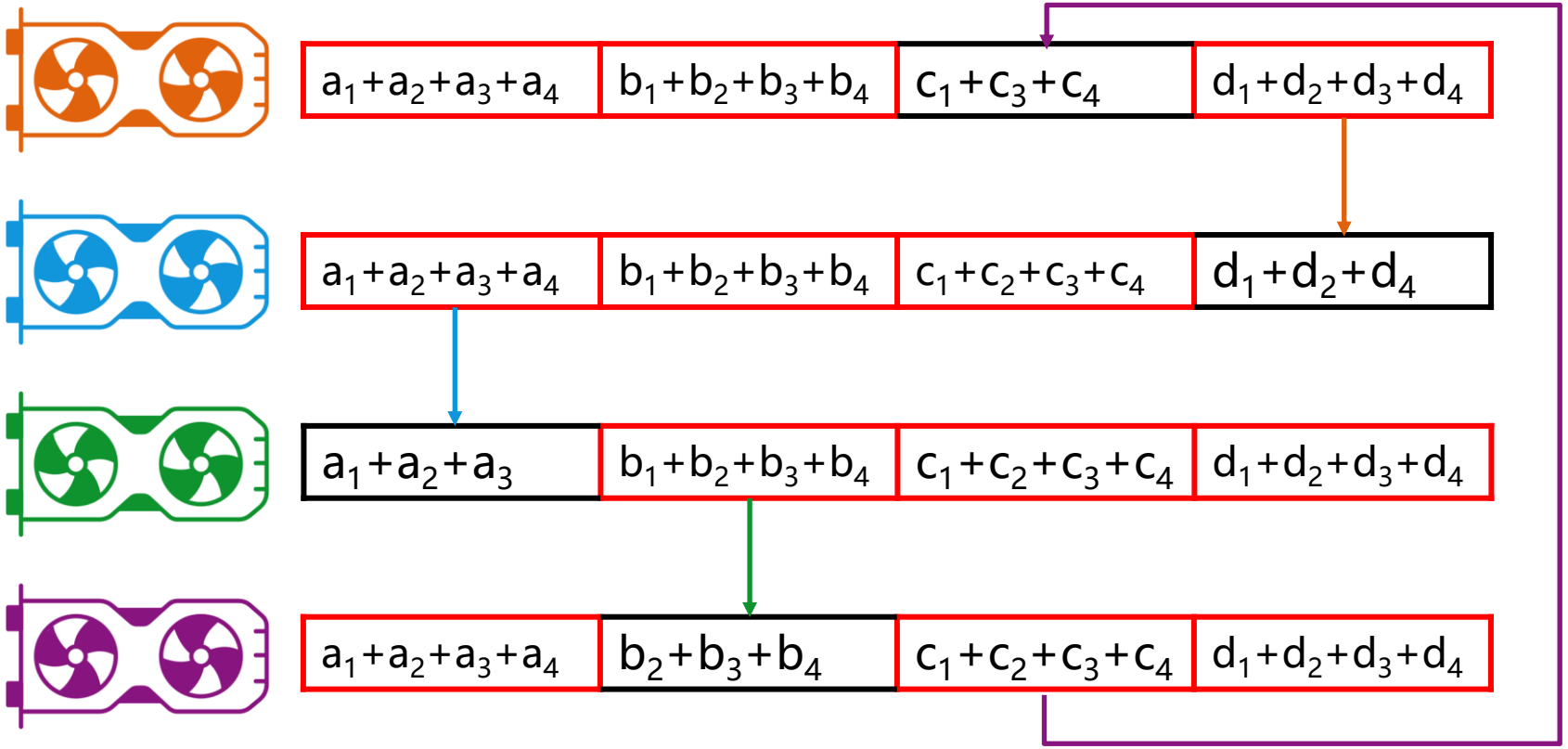
每个GPU只和“下一个”GPU通信



序列并行：环注意力

GPU间的通信瓶颈

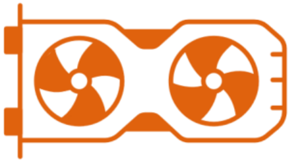
每个GPU只和“下一个”GPU通信



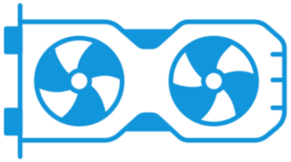
序列并行：环注意力

GPU间的通信瓶颈

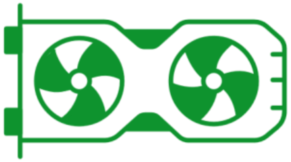
每个GPU只和“下一个”GPU通信



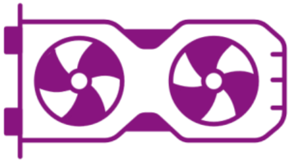
$a_1+a_2+a_3+a_4$	$b_1+b_2+b_3+b_4$	$c_1+c_2+c_3+c_4$	$d_1+d_2+d_3+d_4$



$a_1+a_2+a_3+a_4$	$b_1+b_2+b_3+b_4$	$c_1+c_2+c_3+c_4$	$d_1+d_2+d_3+d_4$

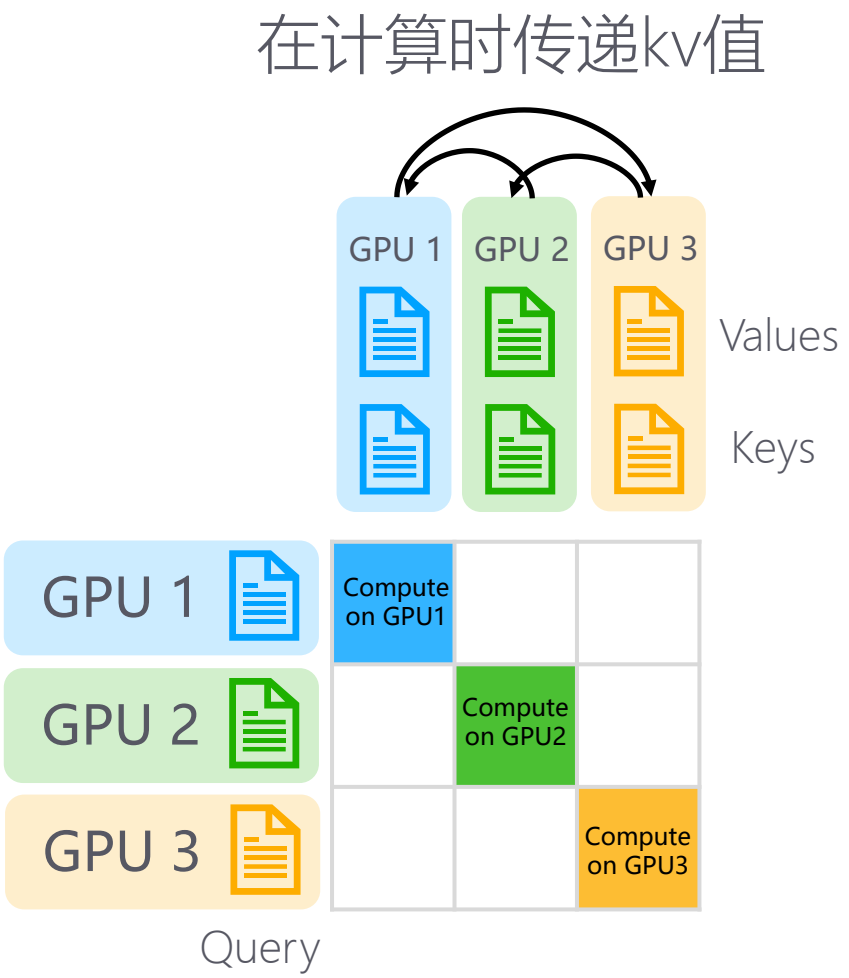


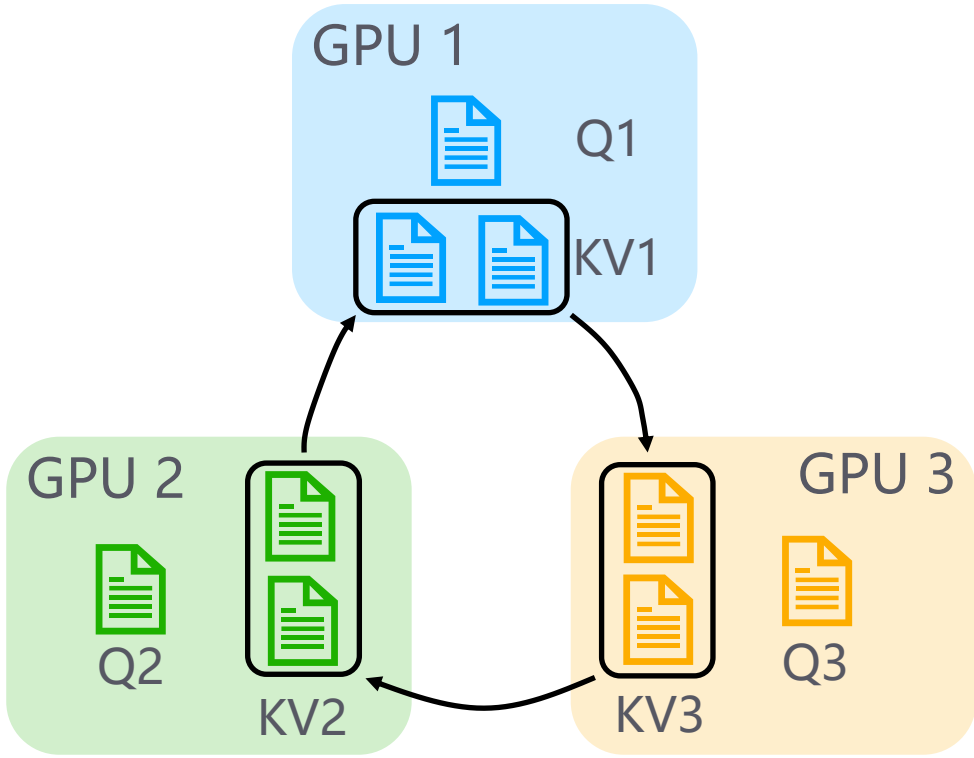
$a_1+a_2+a_3+a_4$	$b_1+b_2+b_3+b_4$	$c_1+c_2+c_3+c_4$	$d_1+d_2+d_3+d_4$



$a_1+a_2+a_3+a_4$	$b_1+b_2+b_3+b_4$	$c_1+c_2+c_3+c_4$	$d_1+d_2+d_3+d_4$

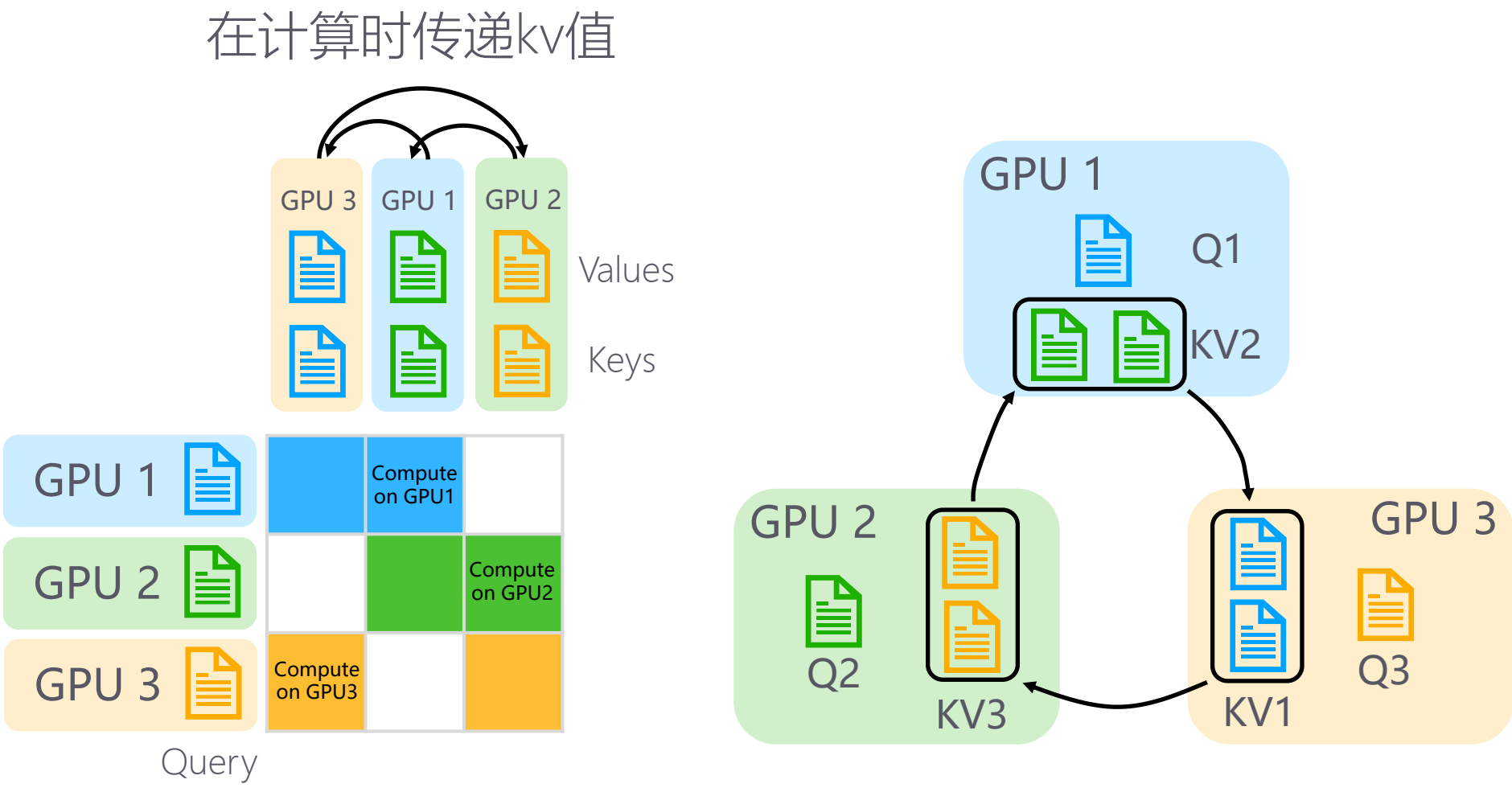
序列并行：环注意力





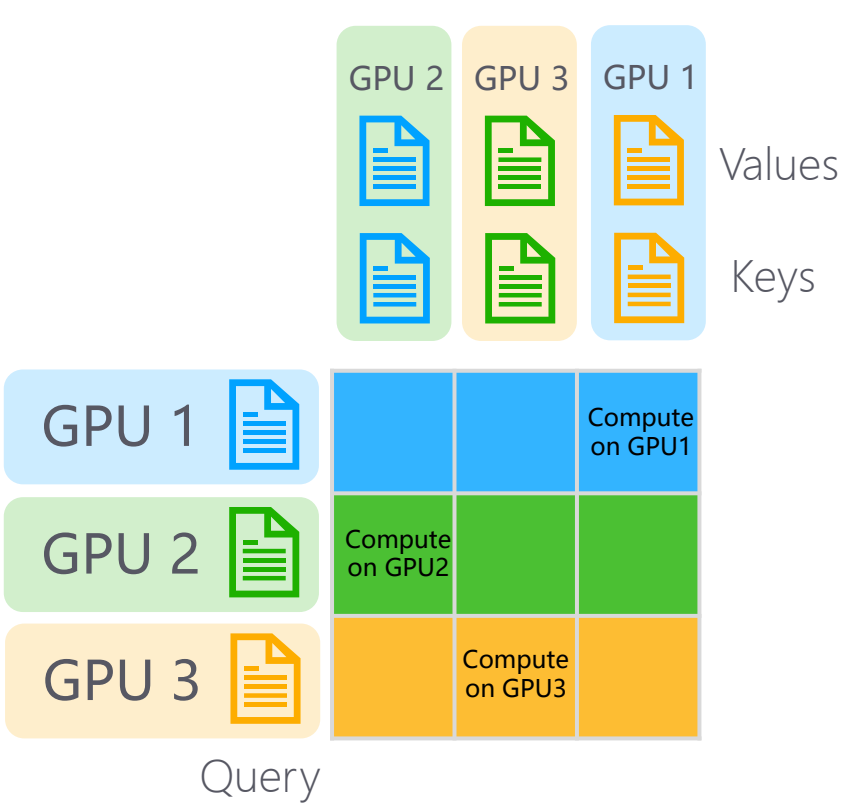
https://arxiv.org/abs/2310.01889

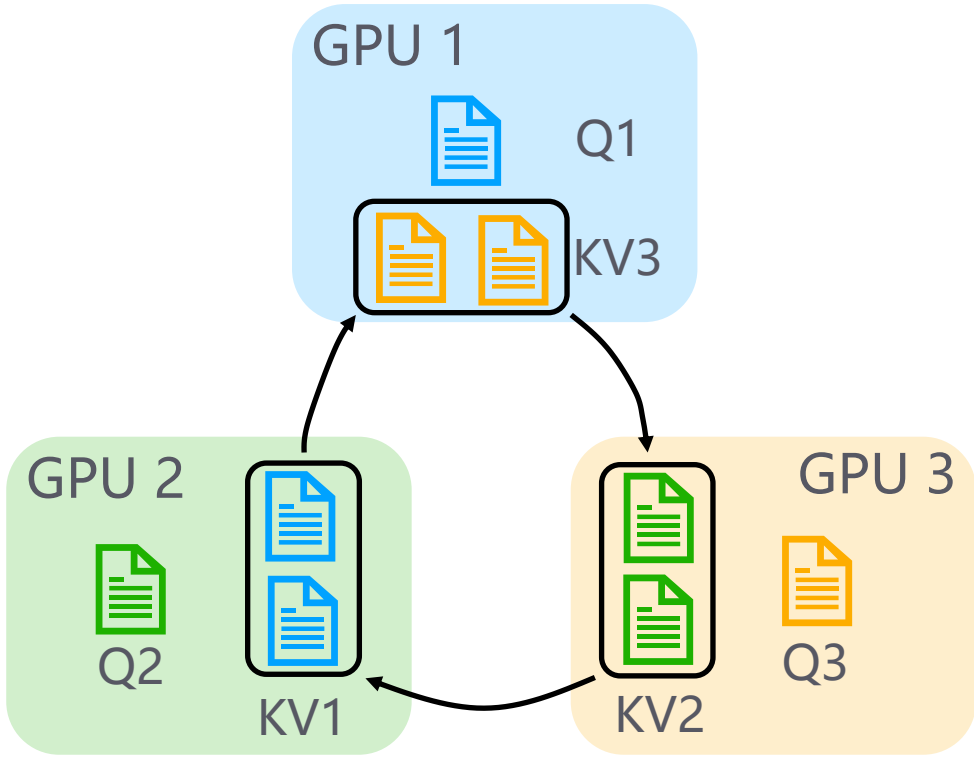
序列并行：环注意力



https://arxiv.org/abs/2310.01889

序列并行：环注意力

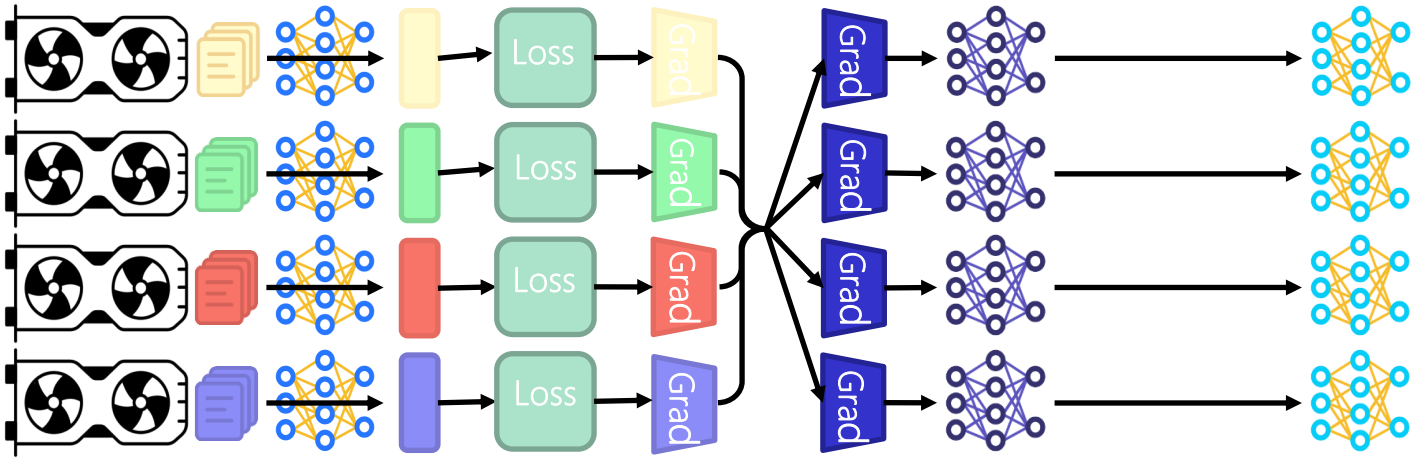




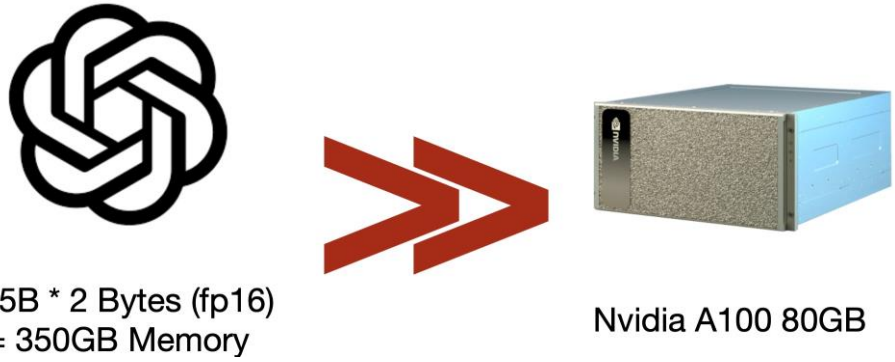
https://arxiv.org/abs/2310.01889

目录

ZeRO-1/2/3及FSDP



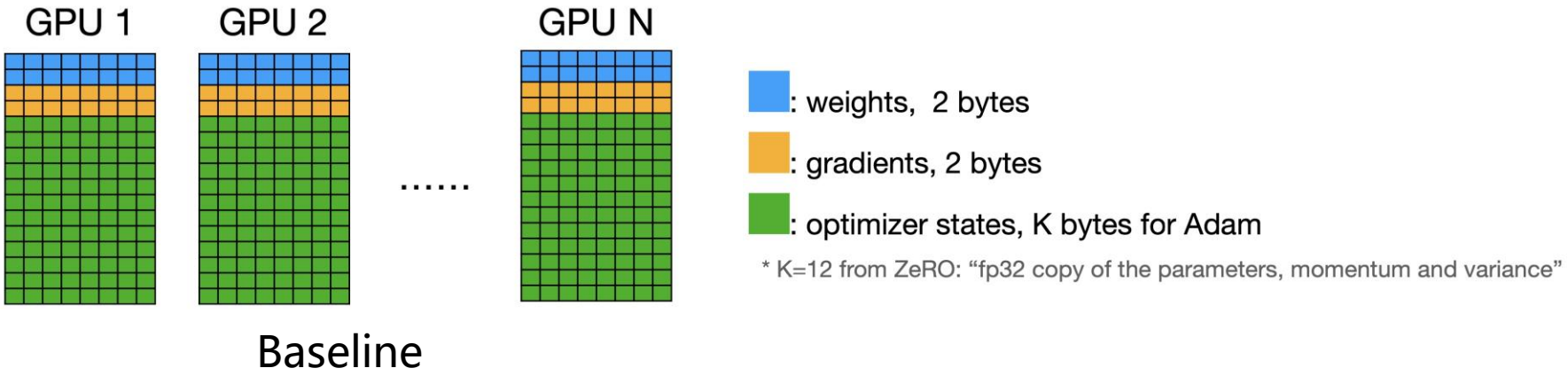
在数据并行框架下，如果我们想要训练一个超大规模模型（例如 GPT-3 175B）：



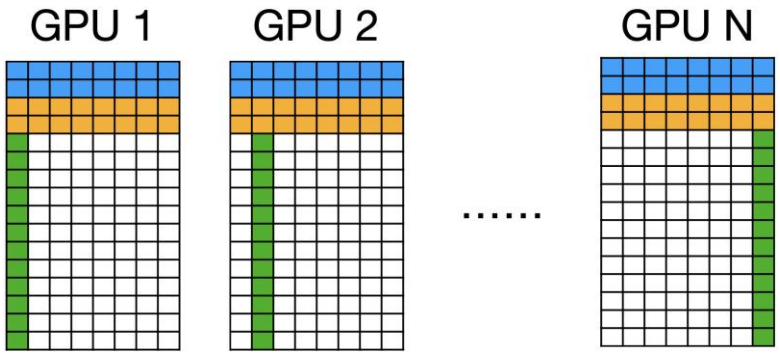
- 即使是最好的 GPU 也无法将模型权重完全加载到显存中！
- 此外，训练还需要存储梯度和优化器状态。

ZeRO-1/2/3及FSDP

微软提出ZeRO（零冗余优化），是DeepSpeed分布式训练框架的核心，来解决大模型训练中的显存开销问题。ZeRO的核心思想：**将优化器状态、梯度与模型参数切片，划分到多个设备上，**通过按需重构数据进一步减少存储占用，用通讯换显存。



Baseline

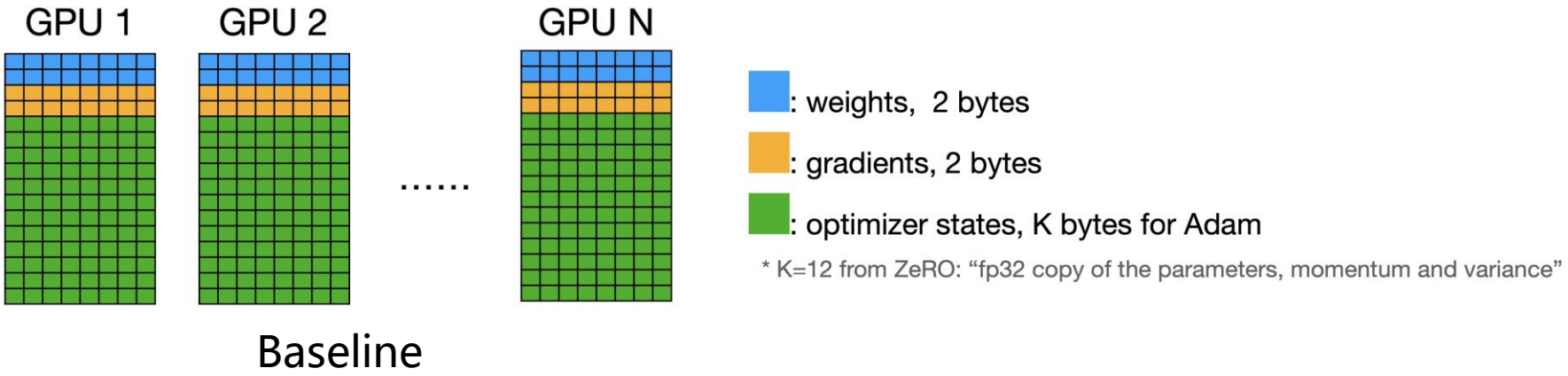


ZeRO-1

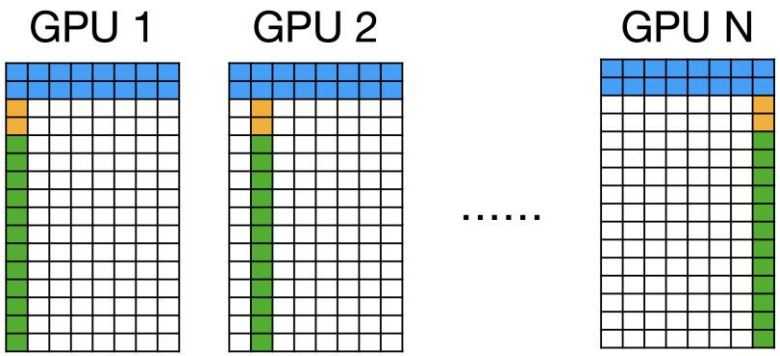
- ZeRO-1 提出对**优化器状态**进行分区/切片。

ZeRO-1/2/3及FSDP

微软提出ZeRO（零冗余优化），是DeepSpeed分布式训练框架的核心，来解决大模型训练中的显存开销问题。ZeRO的核心思想：**将优化器状态、梯度与模型参数切片，划分到多个设备上，**通过按需重构数据进一步减少存储占用，用通讯换显存。



Baseline

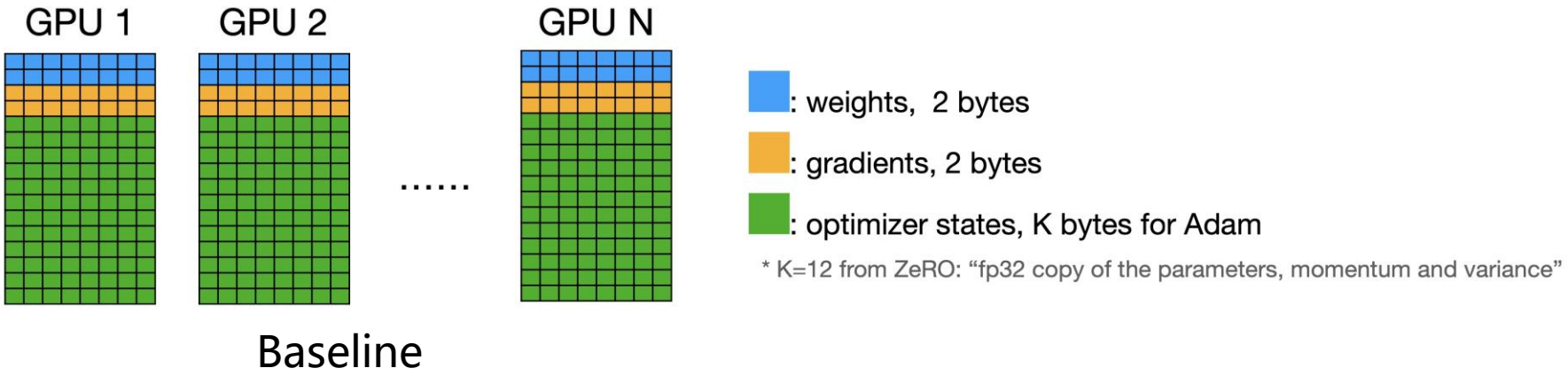


ZeRO-2

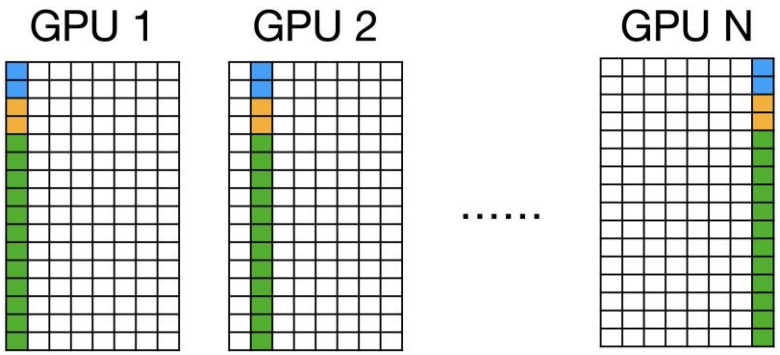
- ZeRO-2 提出对**优化器状态**和**梯度**进行分区/切片。

ZeRO-1/2/3及FSDP

微软提出ZeRO（零冗余优化），是DeepSpeed分布式训练框架的核心，来解决大模型训练中的显存开销问题。ZeRO的核心思想：**将优化器状态、梯度与模型参数切片，划分到多个设备上，**通过按需重构数据进一步减少存储占用，用通讯换显存。



Baseline



ZeRO-3

- ZeRO-3 提出对**优化器状态**、**梯度**和**权重**进行分区/切片。
- 在 PyTorch 中，ZeRO-3 通过 FullyShardedDataParallel（简称**FSDP**）实现。

多维混合并行

- 多维混合并行指将数据并行、模型并行和流水线并行等多种并行技术结合起来进行分布式训练。
- 通常，在进行超大规模模型的预训练和全参数微调时，都需要用到多维混合并行。