LLM 101

一起入门大语言模型

https://llm101.top

哔哩哔哩@一万篇论文笔记

一万篇论文笔记

微信扫描二维码，关注我的公众号

2025.02.02

LLM Pre-training and Beyond

- GPT-1 && GPT-2
  - NLP中的预训练-微调范式: CoVe、ELMo、ULMFiT、GPT-1、BERT
  - GPT-1 && GPT-2: Transformer LM + Large scale pre-training ==> zero-shot
  - 编程实践：阅读gpt-1/gpt-2代码；训练124M GPT-2

llm.c **Modded-NanoGPT**

- (Train-time Compute) Scaling Laws for LM, Empirically
- LLM预训练之分布式训练(Distributed Training)
- GPT-3 and Beyond
  - 涌现、幻觉、位置编码、合成数据、提示工程、SLMs ...

LLM预训练之分布式训练(Distributed Training)

- 数据并行(Data Parallelism)
  - DDP(PyTorch), Sharded Data Parallelism: DeepSpeed-ZeRO
  - 分布式通信 (Distributed Communication)
  - 混合精度训练(Mixed Precision Training)和常见数据格式
- 模型并行(Model Parallelism)
  - 张量并行(Tensor Parallelism, TP)
  - 流水线并行(Pipeline Parallelism, PP)
  - 序列并行(Sequence Parallelism, SP)
  - 上下文并行(Context Parallelism, CP)
  - 专家并行(Expert Parallelism, EP)

picotron

LLM预训练

之

混合精度训练

分布式通信

数据并行

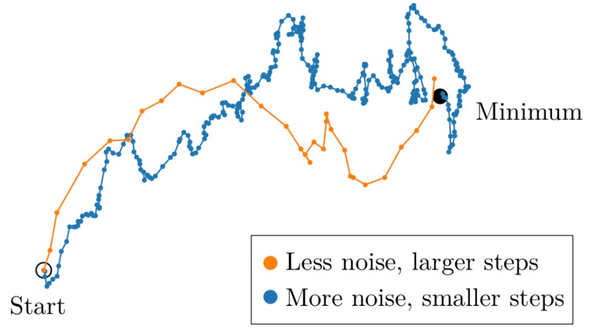
模型并行

LLM预训练之分布式训练(Distributed Training)

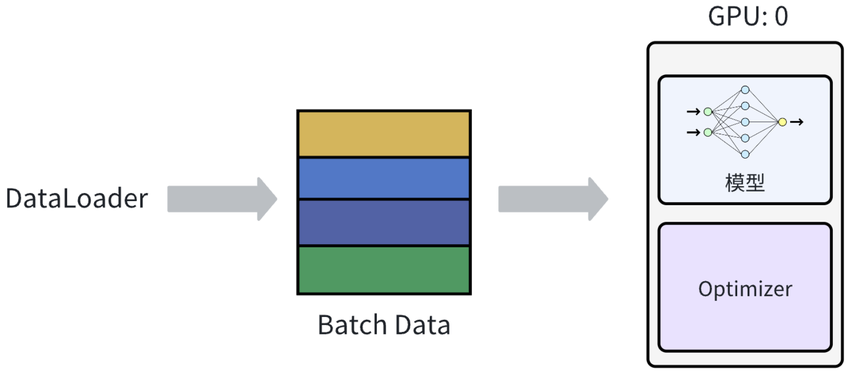
- 数据并行(Data Parallelism)
  - DDP(PyTorch), Sharded Data Parallelism: DeepSpeed-ZeRO
  - 分布式通信 (Distributed Communication)
  - 混合精度训练(Mixed Precision Training)和常见数据格式
- 模型并行(Model Parallelism)
  - 张量并行(Tensor Parallelism, TP)
  - 流水线并行(Pipeline Parallelism, PP)
  - 序列并行(Sequence Parallelism, SP)
  - 上下文并行(Context Parallelism, CP)
  - 专家并行(Expert Parallelism, EP)

picotron

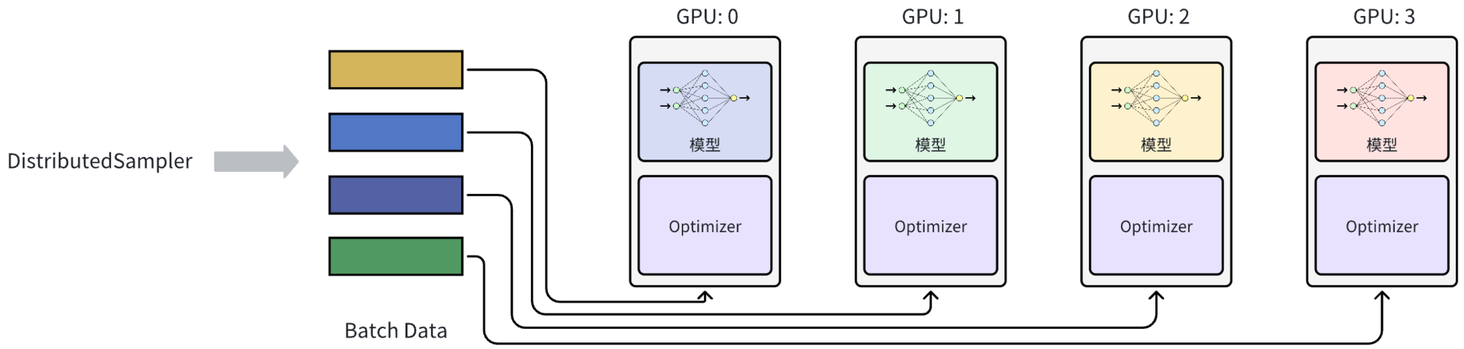
数据并行(Data Parallelism)



An Empirical Model of Large-Batch Training



单卡训练



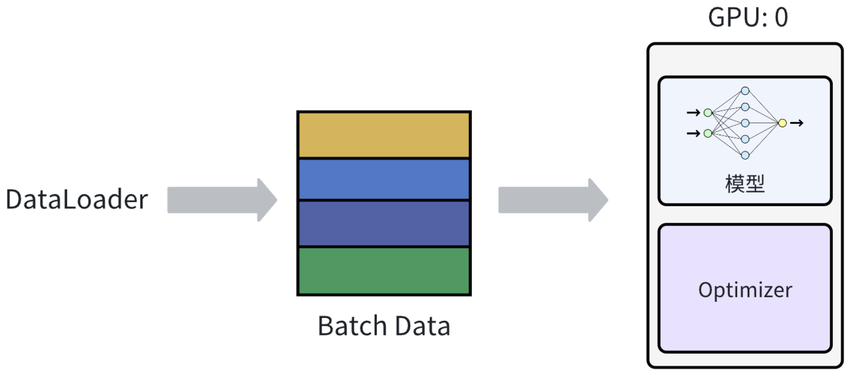
多卡训练

Note： 1) model size不大，能够用单卡训练

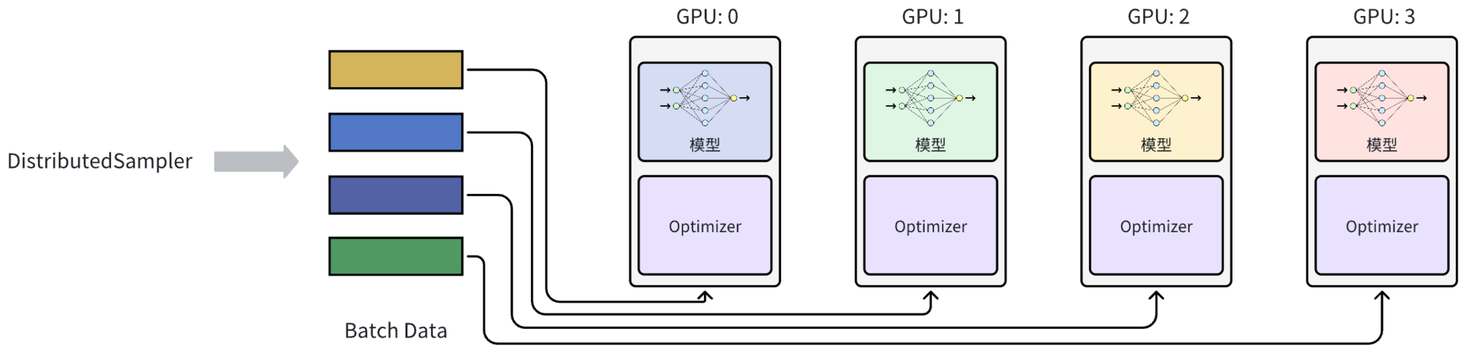
2) 每张卡的模型参数和opt状态相同

数据并行(Data Parallelism)

如何更新参数？



单卡训练



多卡训练

Note： 1) model size不大，能够用单卡训练

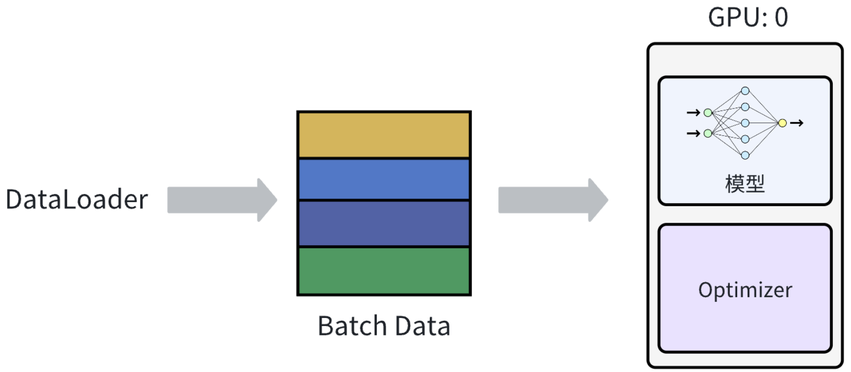
2) 每张卡的模型参数和opt状态相同

数据并行(Data Parallelism)

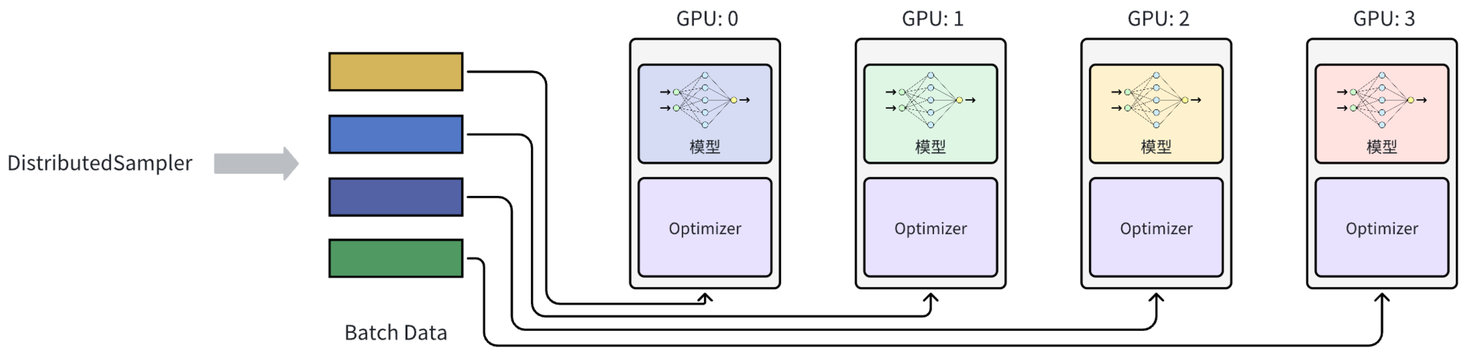
如何更新参数？

1. Compute the gradient of the loss function using a minibatch on each GPU.
2. Compute the mean of the gradients by inter-GPU communication.
3. Update the model.

Technologies behind Distributed Deep Learning: AllReduce



单卡训练



多卡训练

Note： 1) model size不大，能够用单卡训练

2) 每张卡的模型参数和opt状态相同

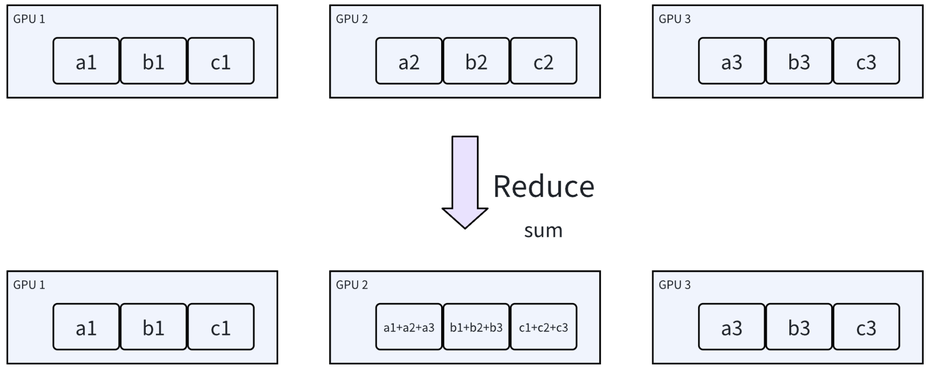
分布式通信(Distributed Communication)

- 点对点通信(Point-to-Point Communication, P2P)
- 集合/集群通信(Collective Communication)
  - 通信原语(Communication Primitives): Reduce、Scatter、Gather、Broadcast、ReduceScatter

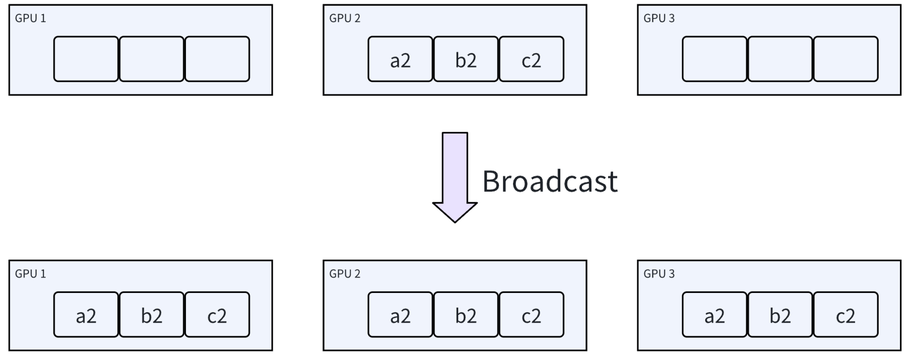
    AllReduce、AllGather …

分布式通信(Distributed Communication)

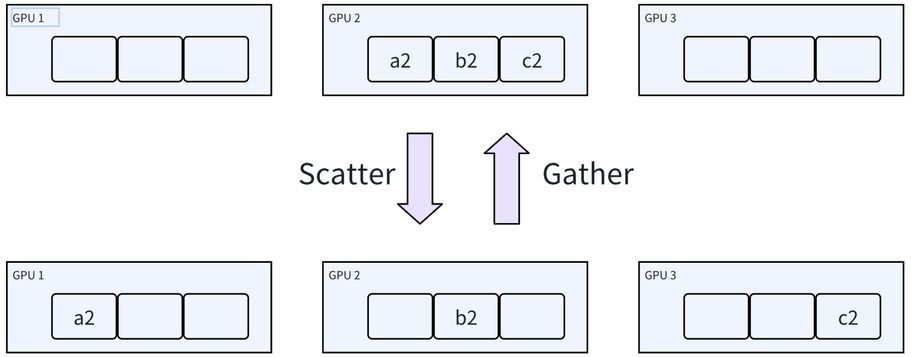
- Reduce



- Broadcast

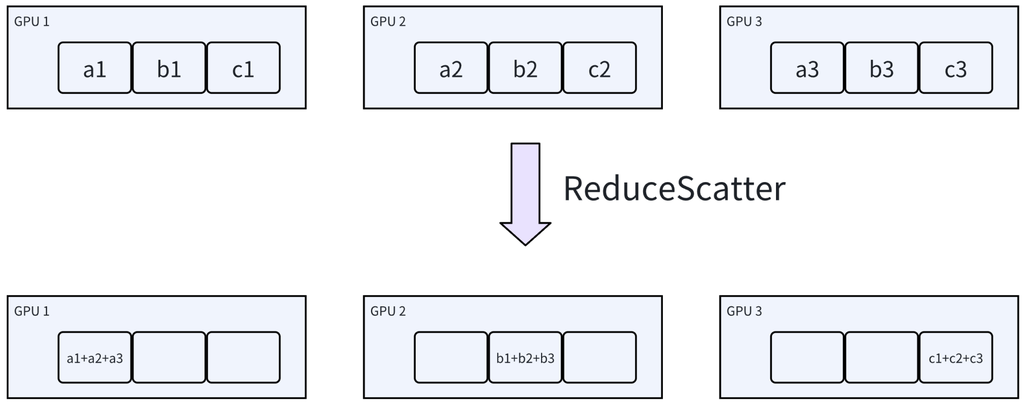


- Scatter和Gather

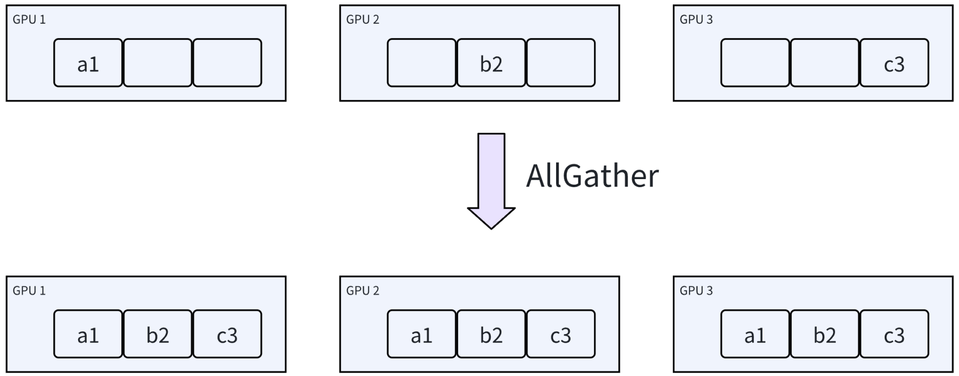


分布式通信(Distributed Communication)

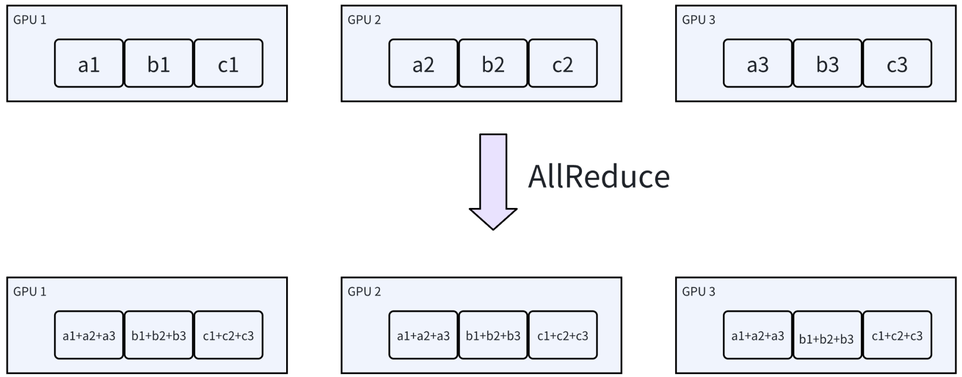
- ReduceScatter



- AllGather

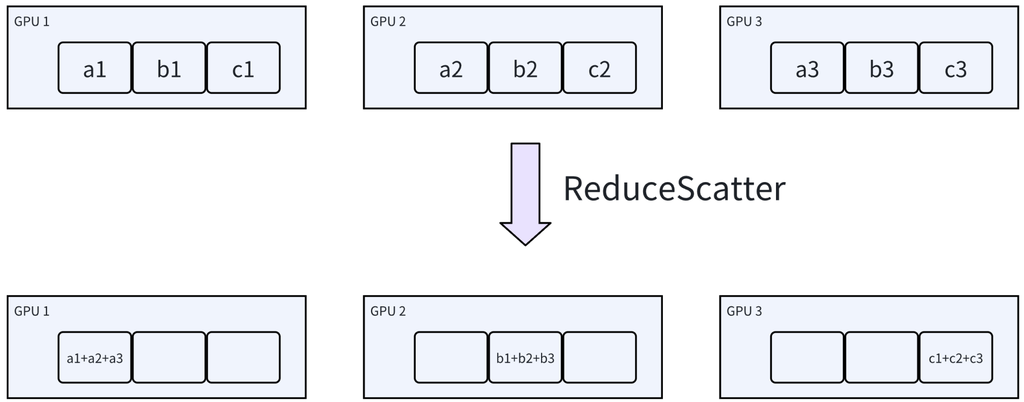


- AllReduce

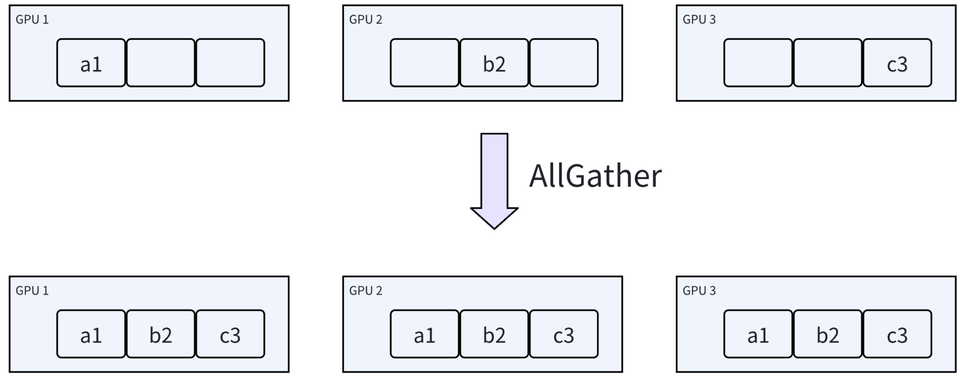


分布式通信(Distributed Communication)

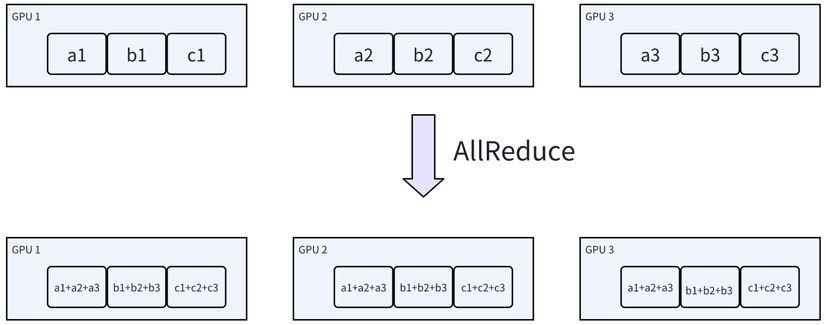
- ReduceScatter



- AllGather



- AllReduce



AllReduce = ReduceScatter + AllGather

AllReduce = Reduce + Broadcast

...

NVIDIA/**nccl**

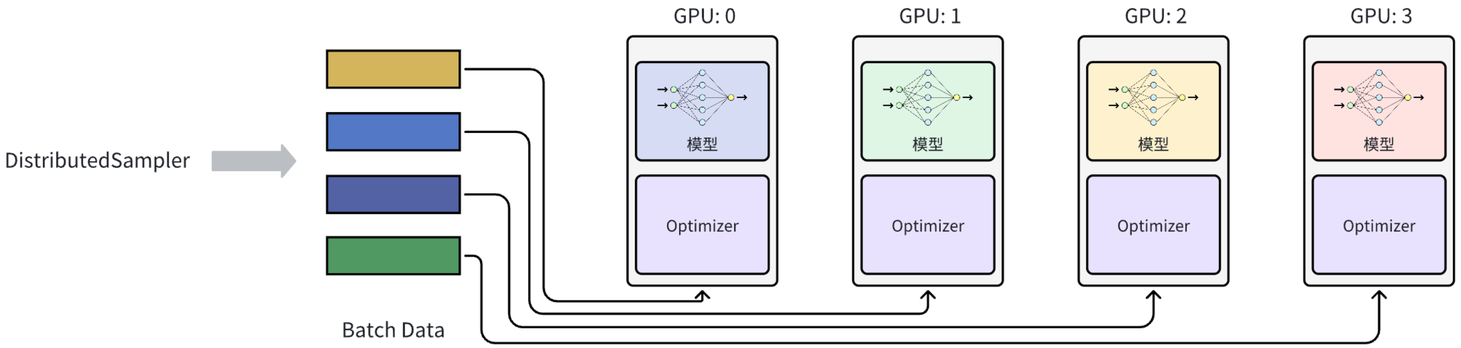
Optimized primitives for collective multi-GPU communication

分布式通信(Distributed Communication)

- Ring(环状) AllReduce　　Ring AllReduce Demo
  - 基于Ring(环状)通信的集群通信算法执行时间几乎和设备数无关，但总通信量和设备数成正比

手把手推导Ring All-reduce的数学性质

数据并行(Data Parallelism)



1. Compute the gradient of the loss function using a minibatch on each GPU.
2. Compute the mean of the gradients by inter-GPU communication.
3. Update the model.

Sharded Data Parallelism: DeepSpeed-ZeRO

- 数据并行(DP)场景，每张显卡都存储一份模型参数和一份optimizer状态
  - Adam在SGD基础上，为每个参数梯度增加了一阶动量（momentum）和二阶动量

$$\mathrm{m}_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \qquad \hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \boldsymbol{\eta} \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \boldsymbol{\epsilon}}$$

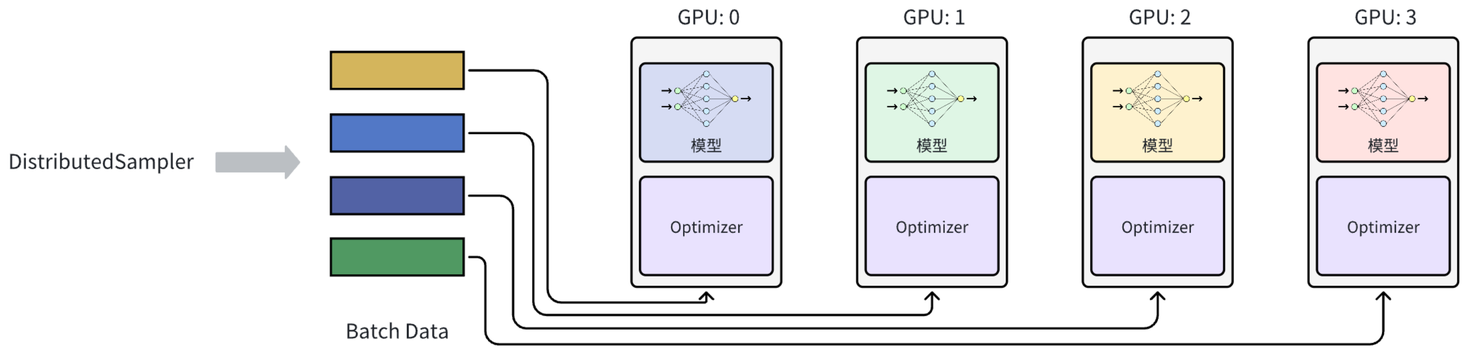
Sharded Data Parallelism: DeepSpeed-ZeRO

- 数据并行(DP)场景，每张显卡都存储一份模型参数和一份optimizer状态
  - Adam在SGD基础上，为每个参数梯度增加了一阶动量（momentum）和二阶动量

$$\mathrm{m}_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \qquad \hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\boldsymbol{\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}}$$

2倍参数量



N份模型参数

N份optimizer状态

混合精度训练(Mixed Precision Training)

- BF16-FP32
- FP16-FP32
- FP8-BF16-FP32

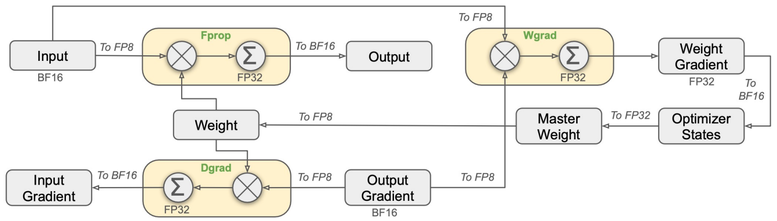


Figure 6 | The overall mixed precision framework with FP8 data format. For clarification, only the `Linear` operator is illustrated.

DeepSeek-V3 Technical Report

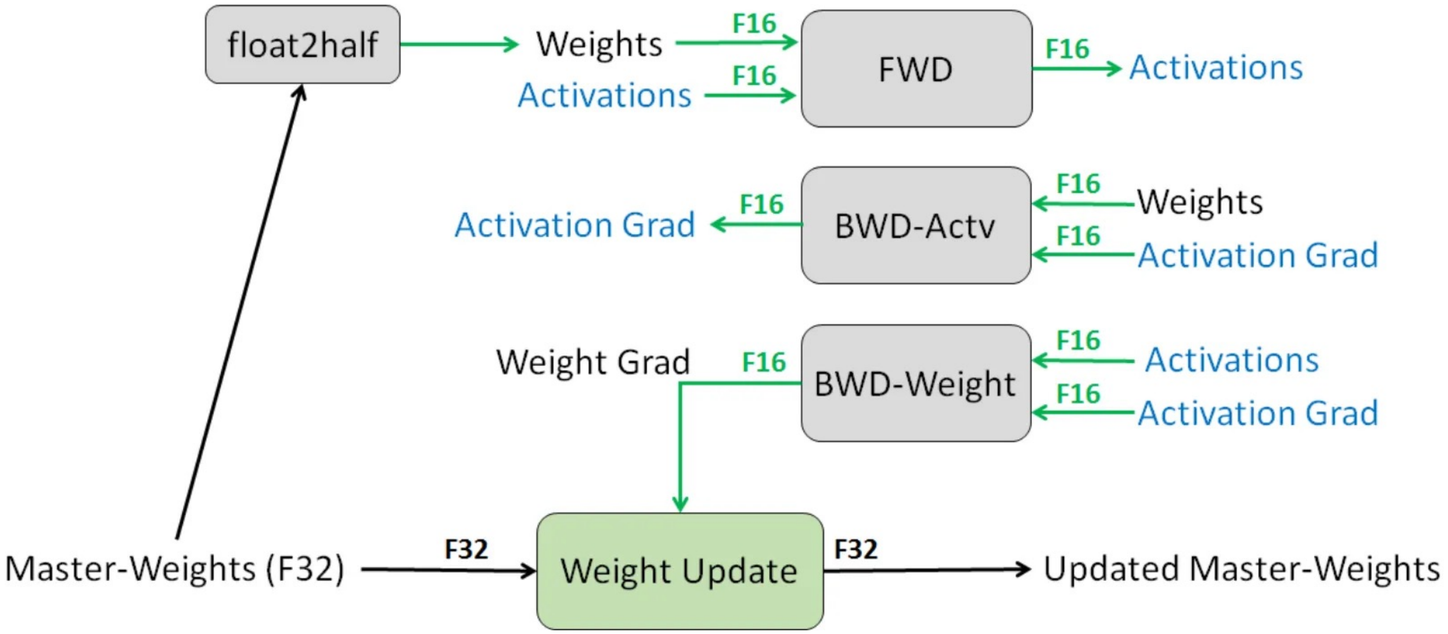


Figure 1: Mixed precision training iteration for a layer.

Mixed Precision Training

Data types

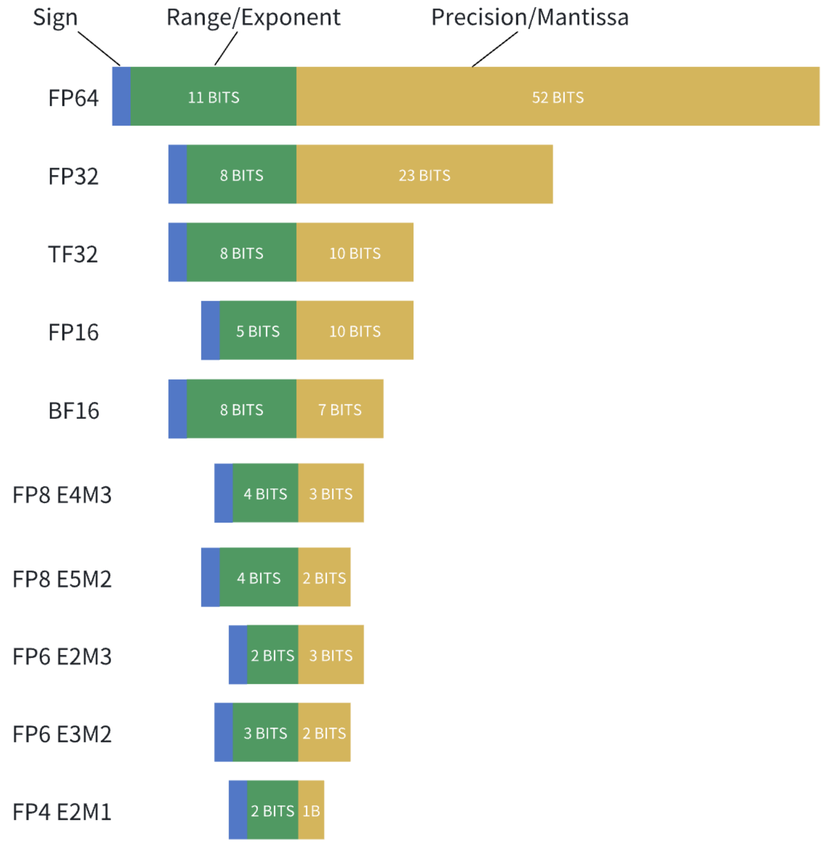
- FP64, FP32, TF32, FP16, BF16, FP8, FP6, FP4, …

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-127} \times 1.\text{fraction}$$

FP32 wiki

	Nvidia GPU Architectures				
	Blackwell	**Hopper**	**Ada Lovelace**	**Ampere**	**Turing**
Supported CUDA* Core precision	FP16, FP32, FP64, bfloat16	FP16, FP32, FP64, bfloat16	FP16, FP32, FP64, bfloat16	FP16, FP32, FP64	FP16, FP32, FP64
Supported Tensor Core precision	FP16, FP64, bfloat16, TF32, FP8, FP6, FP4	FP16, FP64, bfloat16, TF32 FP8	FP16, FP64, bfloat16, TF32 FP8	FP16, FP64, bfloat16, TF32	FP16

Standalone 16-bit Training



混合精度训练(Mixed Precision Training)

- BF16-FP32
- FP16-FP32
- FP8-BF16-FP32

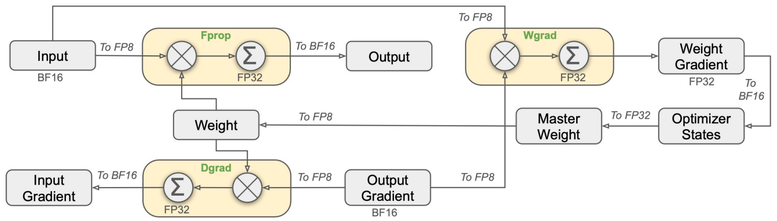


Figure 6 | The overall mixed precision framework with FP8 data format. For clarification, only the `Linear` operator is illustrated.

DeepSeek-V3 Technical Report

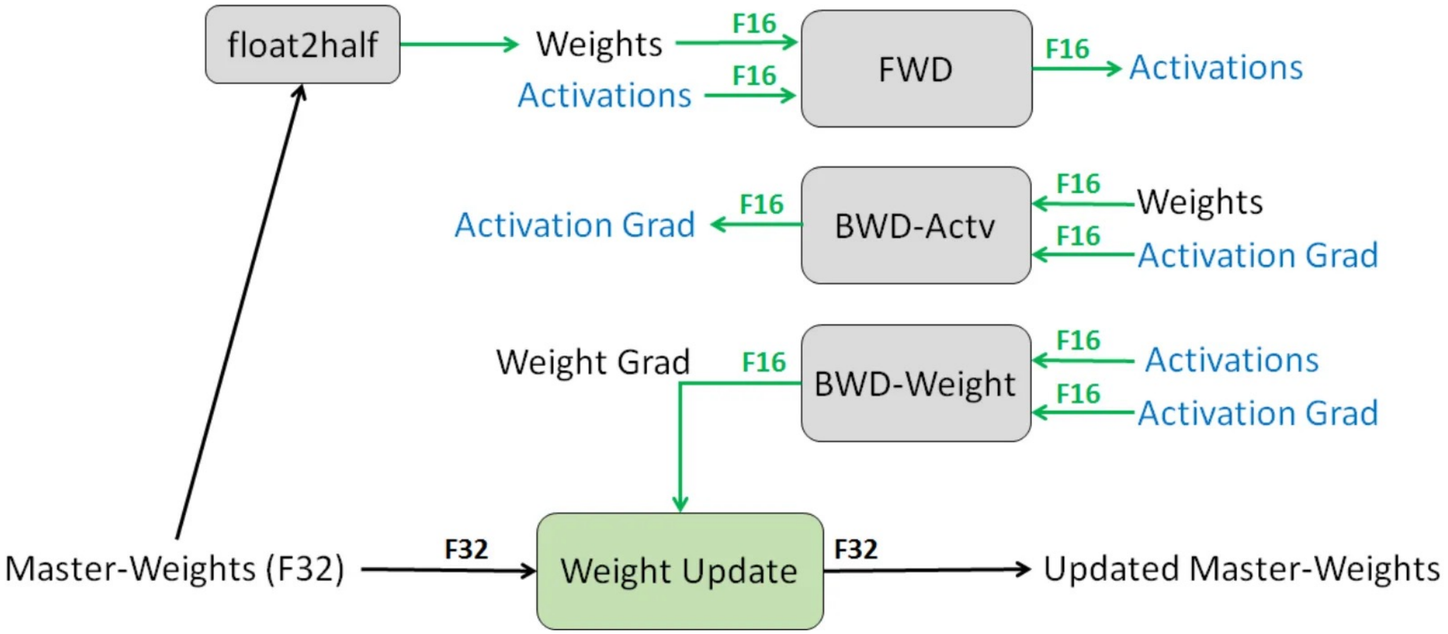
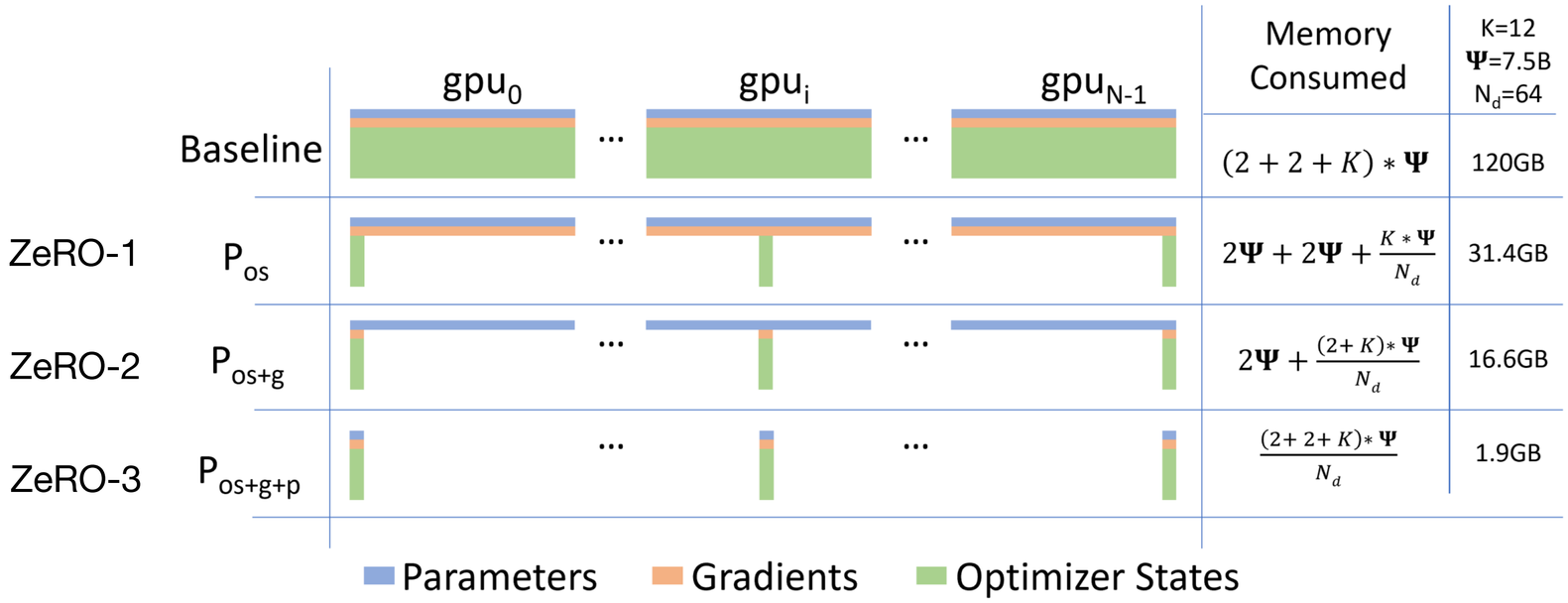


Figure 1: Mixed precision training iteration for a layer.

Mixed Precision Training

Sharded Data Parallelism: DeepSpeed-ZeRO

- ZeRO(Zero Redundancy Optimizer)



数据并行(Data Parallelism)

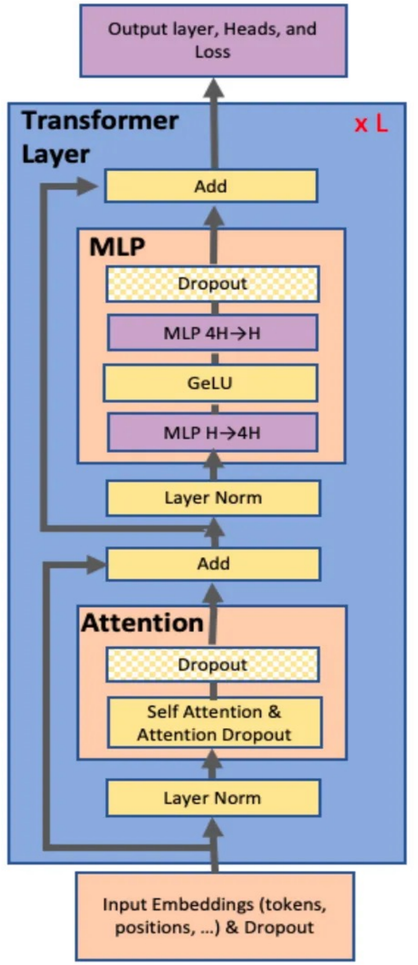
- DDP (PyTorch)

模型并行(Model Parallelism)

- 张量并行(Tensor Parallelism, TP)
- 流水线并行(Pipeline Parallelism, PP)
- 序列并行(Sequence Parallelism, SP)
- 上下文并行(Context Parallelism, CP)
- 专家并行(Expert Parallelism)

张量并行(Tensor Parallelism, TP)

- FFN(MLP) sublayer
- MHSA sublayer

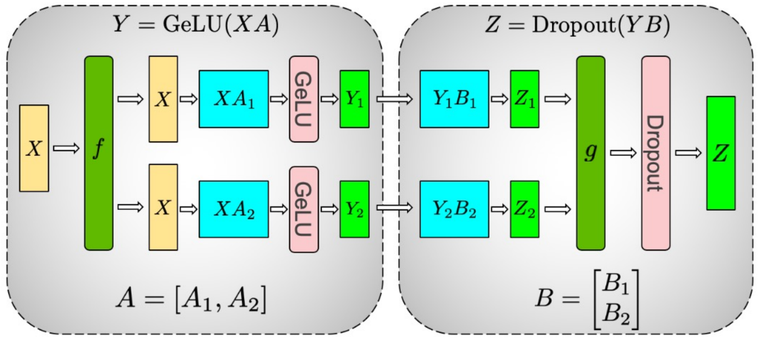


GPT-2

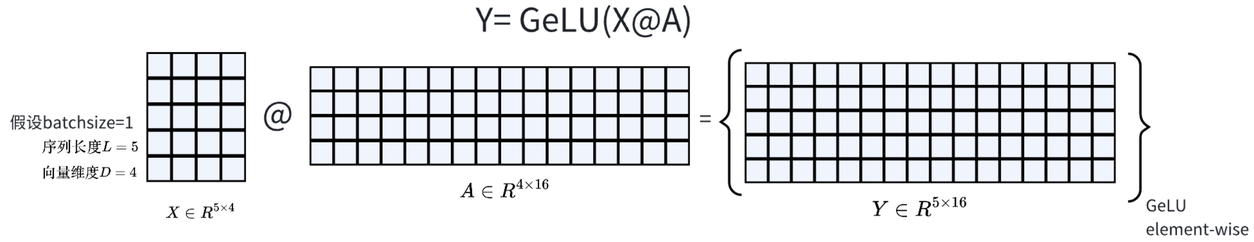
Megatron-LM, 2019

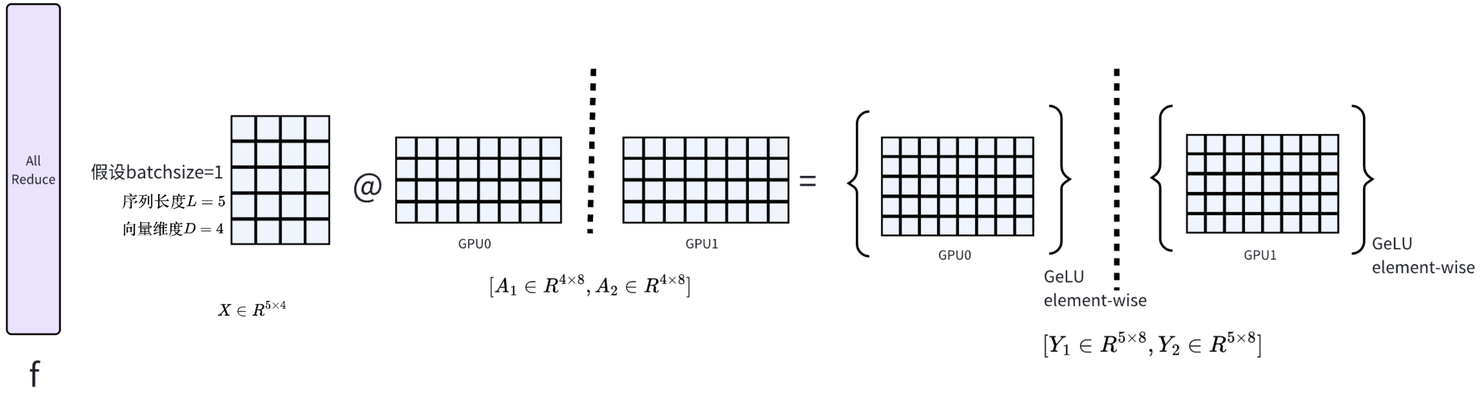
张量并行(Tensor Parallelism, TP)

- FFN(MLP) sublayer
  - Y=GeLU(XA)
  - Z=Dropout(YB)



Megatron-LM, 2019

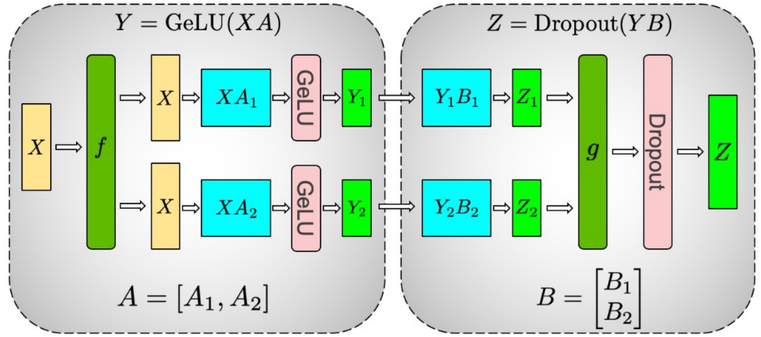





```
class f(torch.autograd.Function):
    def forward(ctx, x):
        return x
    def backward(ctx, gradient):
        all_reduce(gradient)
        return gradient
```

张量并行(Tensor Parallelism, TP)

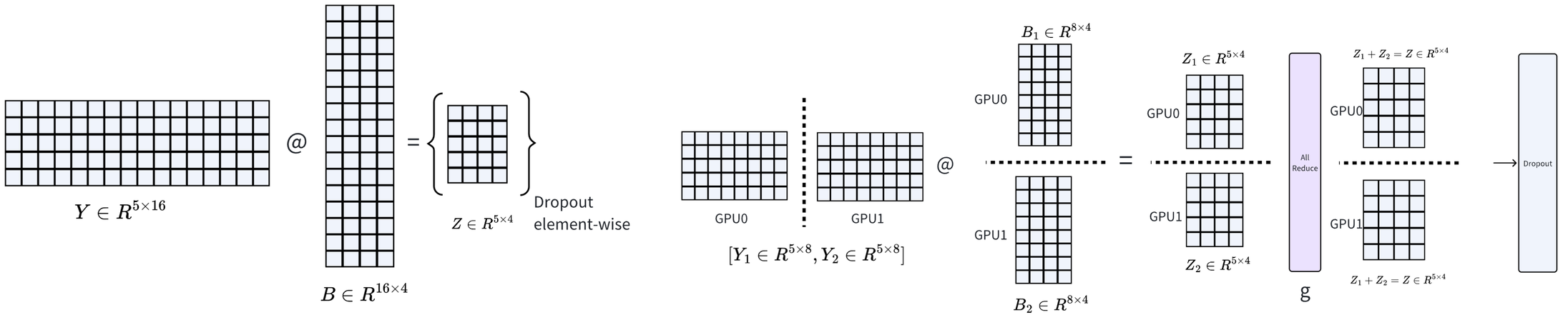
- FFN(MLP) sublayer
  - Z=Dropout(YB)



(a) MLP

Megatron-LM, 2019

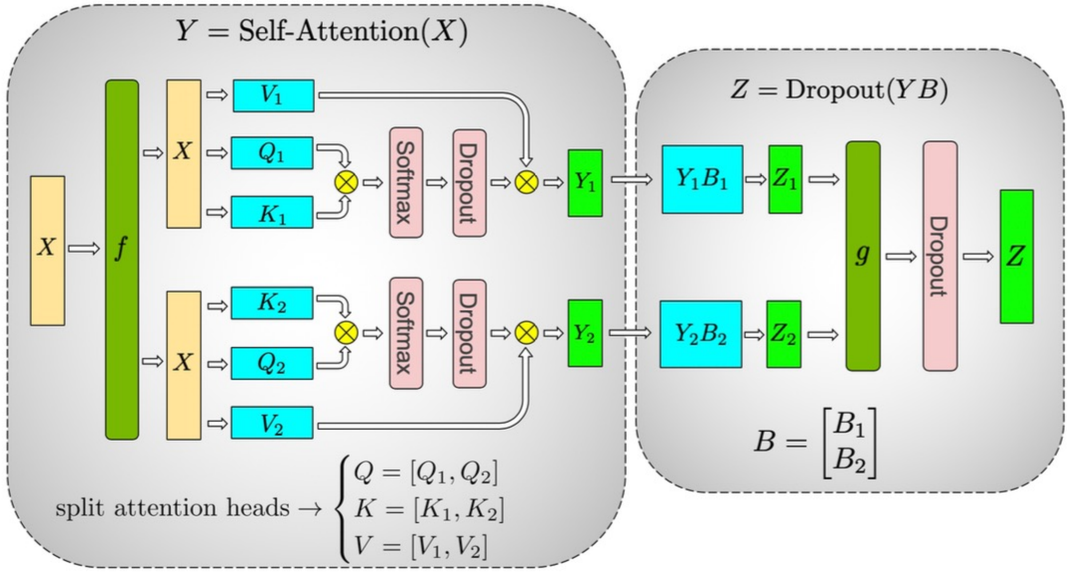
Z= Dropout(Y@B)



张量并行(Tensor Parallelism)

- MHSA sublayer
  - 利用多头自身的并行属性
  - 先column parallel再row parallel

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

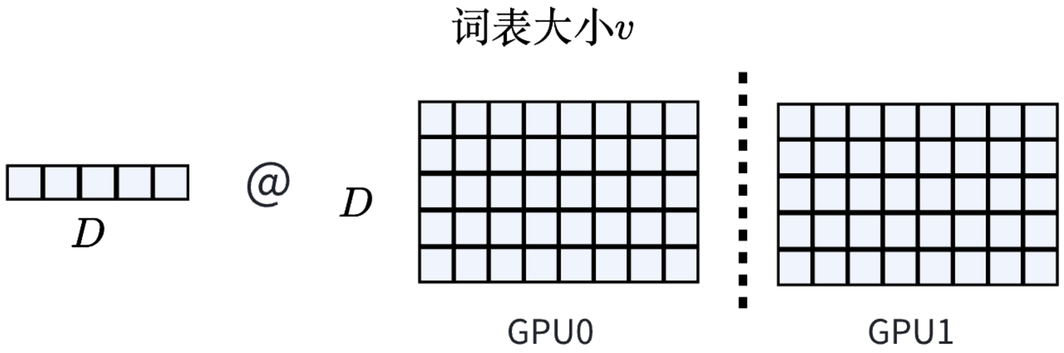


(b) Self-Attention

Megatron-LM, 2019

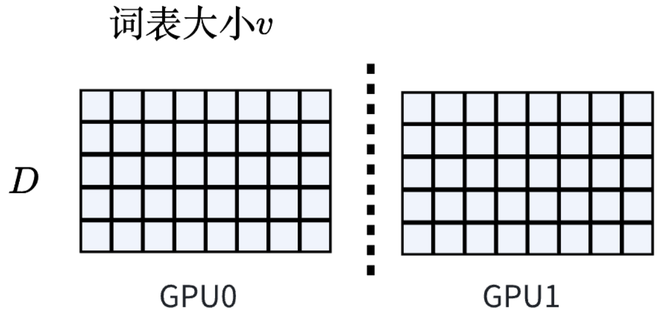
张量并行(Tensor Parallelism)

- Output embedding layer



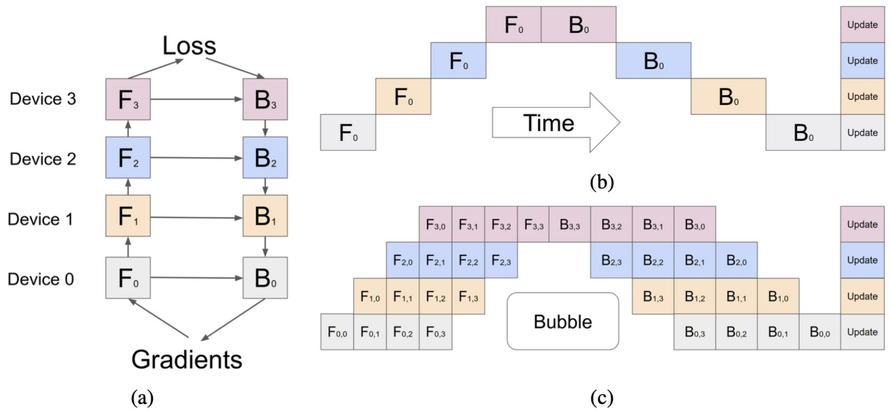
AllGather + softmax

- Embedding

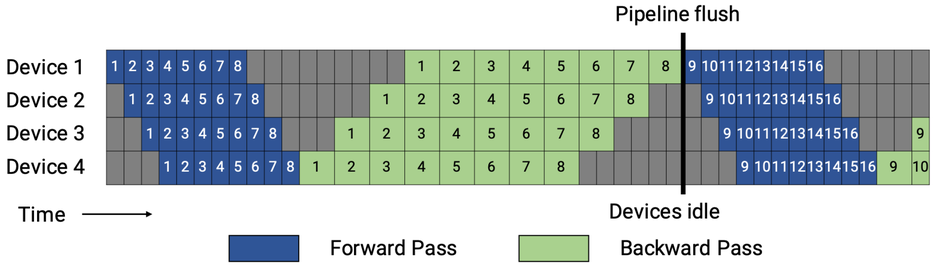


AllReduce

流水线并行(Pipeline Parallelism)



GPipe, 2018



Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, 2021

将一个batch分成 m 个microbatch

pipeline stage个数是 p, 也就是有 p 个GPU

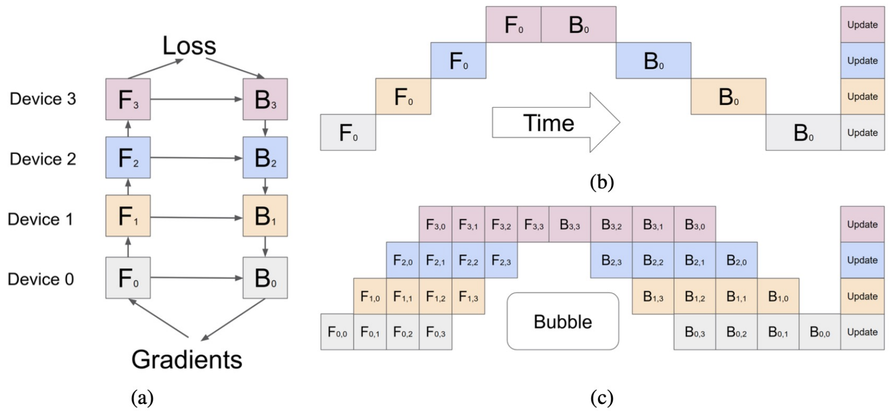
GPU执行一个microbatch数据的forward和backward时间 t_f 和 t_b

每个step/iteration，理想情况下，每个GPU的总时间是$t_{id} = m \cdot (t_f + t_b)$

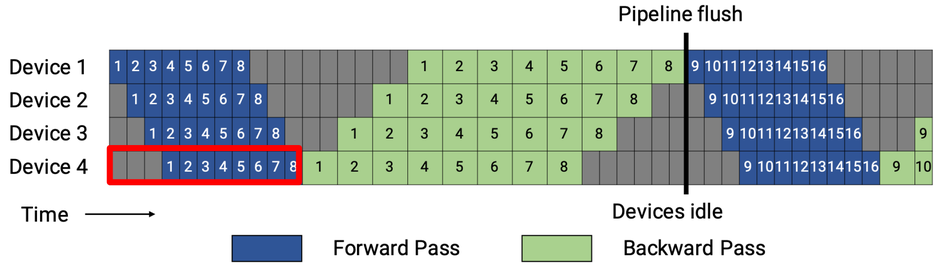
空闲时间(pipeline bubble) $t_{pb} = (p-1) \cdot (t_f + t_b)$

空闲时间占比 $\frac{t_{pb}}{t_{id}} = \frac{(p-1) \cdot (t_f + t_b)}{m \cdot (t_f + t_b)} = \frac{p-1}{m}$

流水线并行(Pipeline Parallelism)



GPipe, 2018



Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, 2021

将一个batch分成 m 个microbatch

pipeline stage个数是 p, 也就是有 p 个GPU

GPU执行一个microbatch数据的forward和backward时间 t_f 和 t_b

每个step/iteration，理想情况下，每个GPU的总时间是$t_{id} = m \cdot (t_f + t_b)$

空闲时间(pipeline bubble) $t_{pb} = (p-1) \cdot (t_f + t_b)$

空闲时间占比 $\frac{t_{pb}}{t_{id}} = \frac{(p-1)\cdot(t_f+t_b)}{m\cdot(t_f+t_b)} = \frac{p-1}{m}$

$m >> p$

显存？

😭

m份activations

流水线并行(Pipeline Parallelism)

- 减小显存占用：PipeDream-Flush schedule

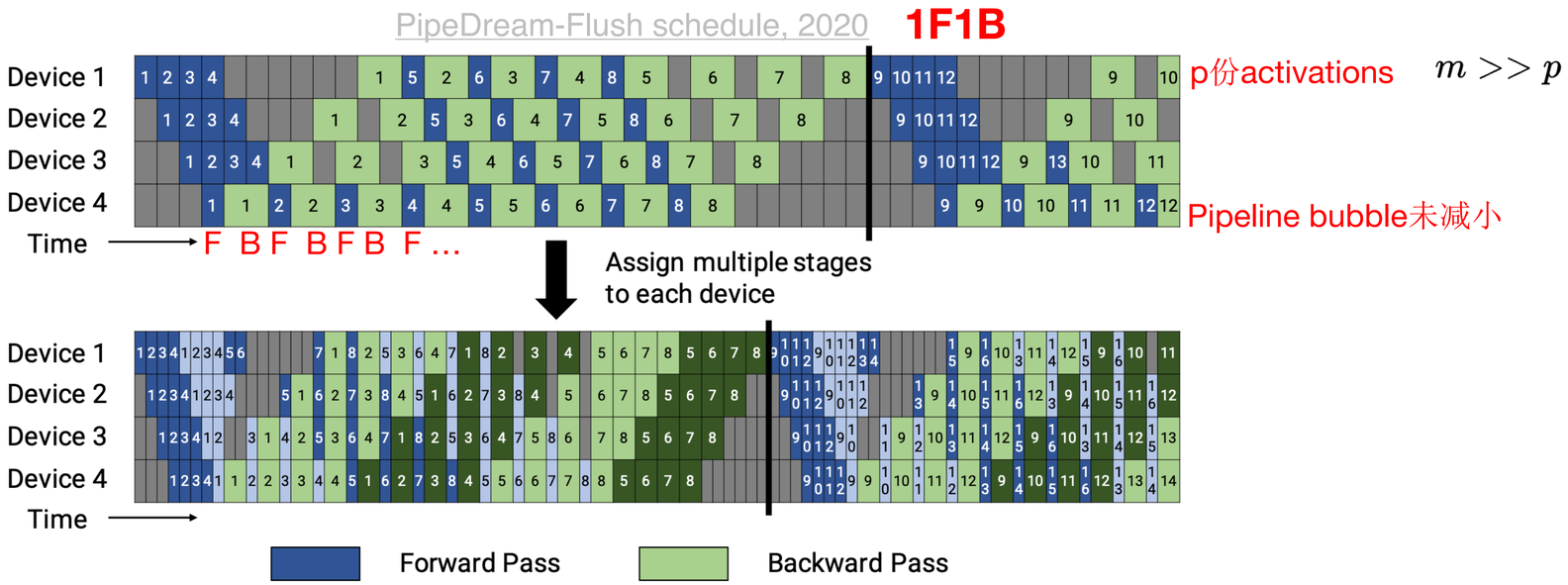


Figure 4: Default and interleaved 1F1B pipeline schedules. The top figure shows the default non-interleaved 1F1B schedule. The bottom figure shows the interleaved 1F1B schedule, where each device is assigned multiple chunks (in this case, 2). Dark colors show the first chunk and light colors show the second chunk. The size of the pipeline bubble is smaller (the pipeline flush happens sooner in the interleaved timeline).

Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, 2021

流水线并行(Pipeline Parallelism)

- 减小显存占用：PipeDream-Flush schedule
- 减小GPU空闲时间：Schedule with Interleaved Stages

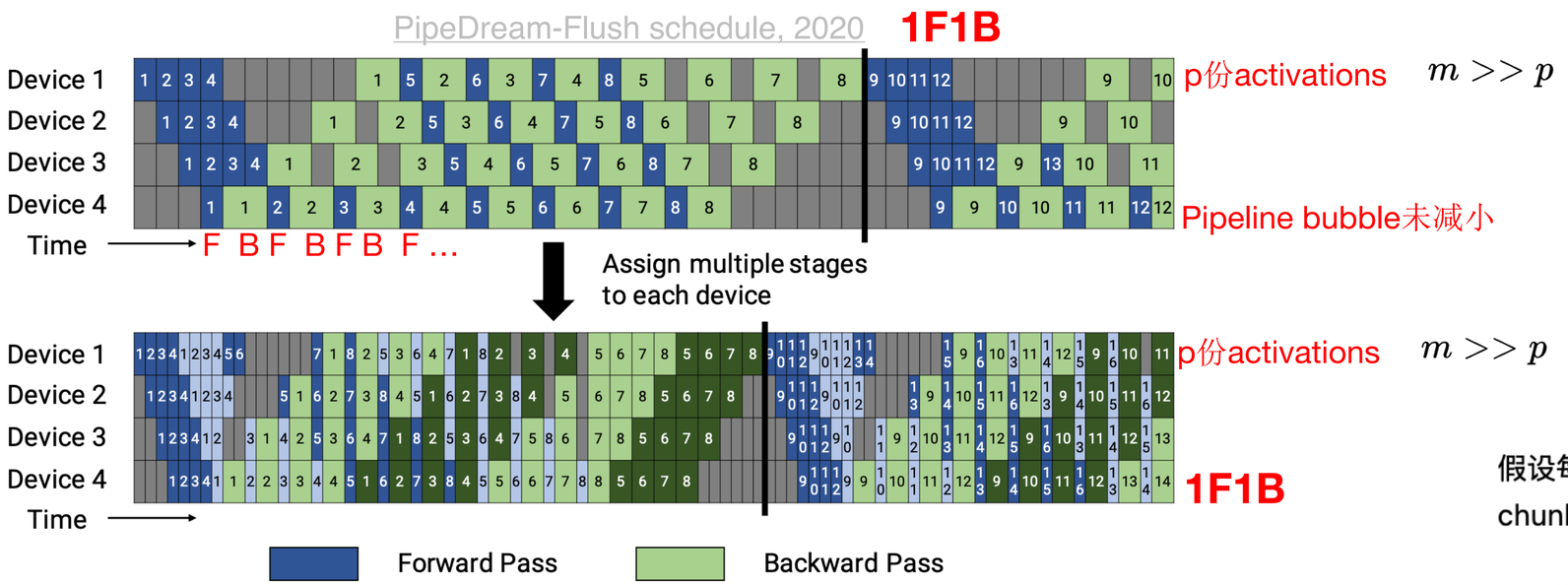


Figure 4: Default and interleaved 1F1B pipeline schedules. The top figure shows the default non-interleaved 1F1B schedule. The bottom figure shows the interleaved 1F1B schedule, where each device is assigned multiple chunks (in this case, 2). Dark colors show the first chunk and light colors show the second chunk. The size of the pipeline bubble is smaller (the pipeline flush happens sooner in the interleaved timeline).

连续2层为一个model chunk

假设有一个16层的模型，使用4个设备：

- 传统方式：
  - 设备1：层 1-4
  - 设备2：层 5-8
  - 设备3：层 9-12
  - 设备4：层 13-16
- Interleaved 方式：
  - 设备1：层 1,2,9,10
  - 设备2：层 3,4,11,12
  - 设备3：层 5,6,13,14
  - 设备4：层 7,8,15,16

假设每张GPU包含v个model chunk, 每个model chunk的forward和backward时间分别是t_f/v 和 t_b/v

pipeline bubble 时间 $t_{pb}^{int} = \frac{(p-1)\cdot(t_f+t_b)}{v}$

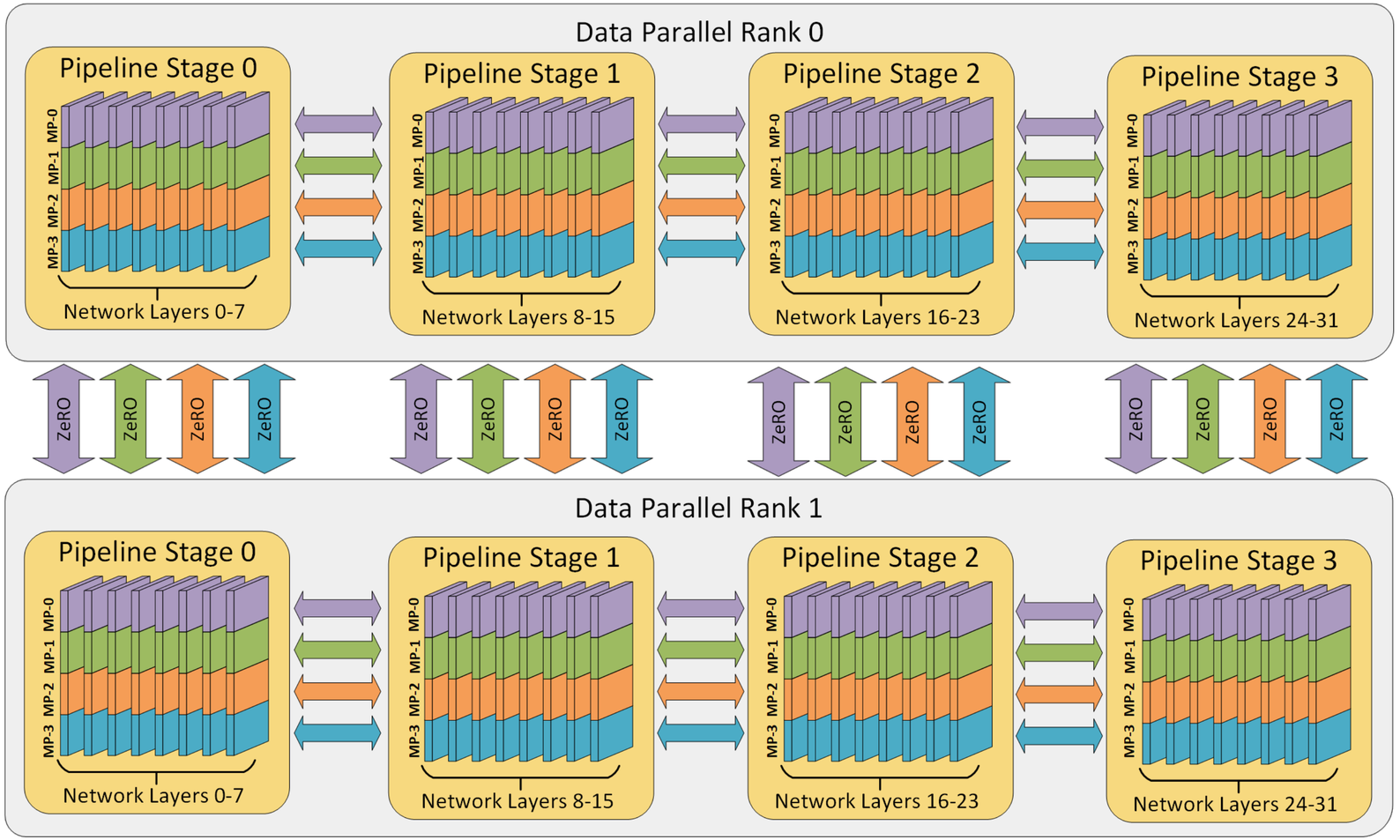
空闲时间占比 $\frac{t_{pb}^{int}}{t_{id}} = \frac{(p-1)\cdot(t_f+t_b)}{v\cdot m\cdot(t_f+t_b)} = \frac{1}{v}\cdot\frac{p-1}{m}$

The amount of communication also increases by v

Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, 2021

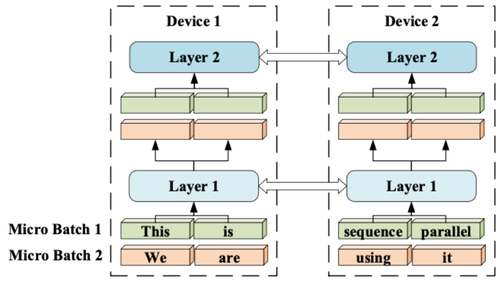
3D并行: DP + PP + TP

4D并行?

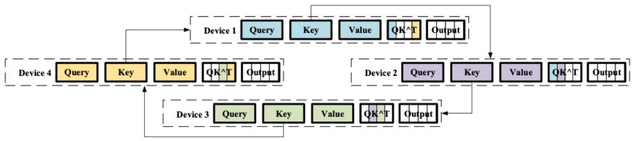


DeepSpeed

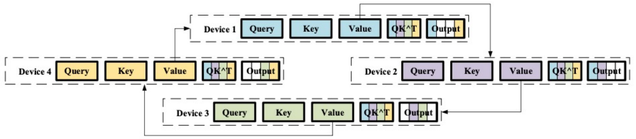
序列并行(Sequence Parallelism, SP)



(c) Sequence parallelism (Ours)



(a) Transmitting key embeddings among devices to calculate attention scores

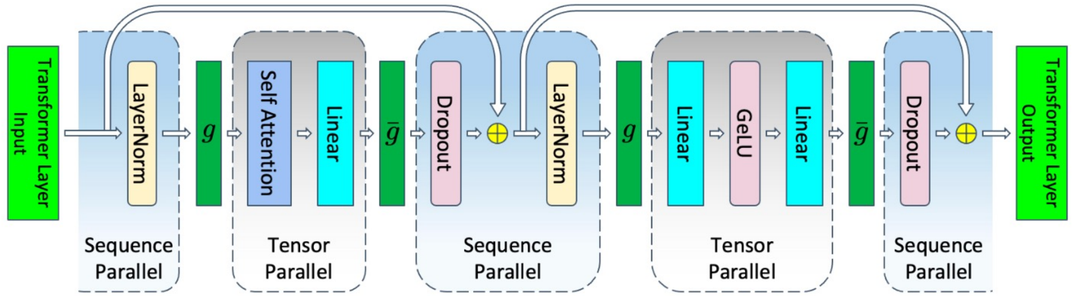


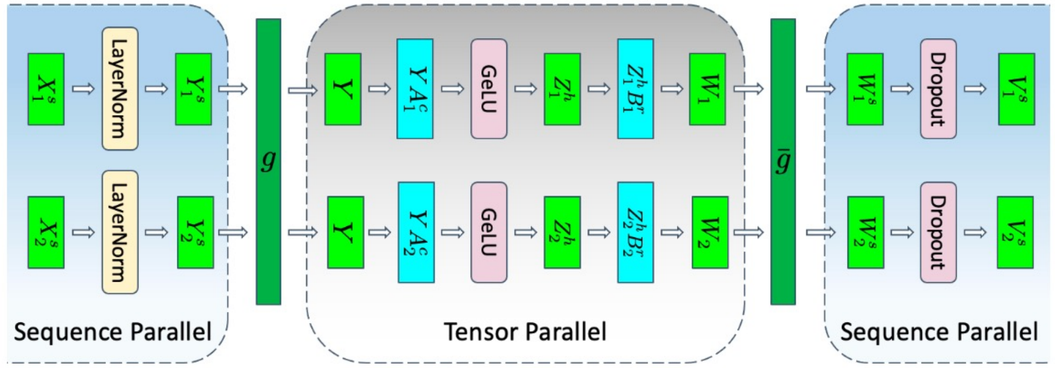
(b) Transmitting value embeddings among devices to calculate the output of attention layers

Figure 2: Ring Self-Attention

Sequence Parallelism, 2021

- 将TP扩展到LayerNorm和Dropout





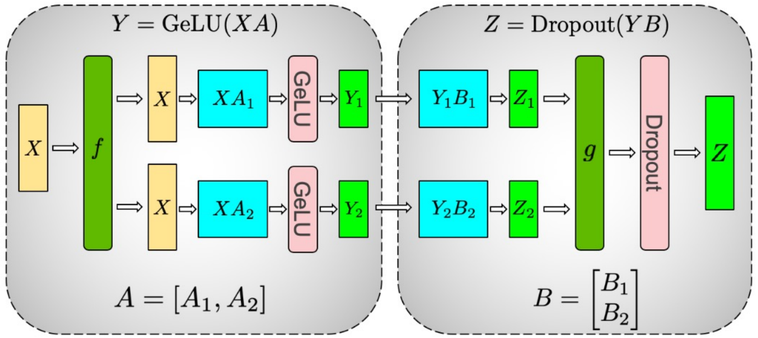
Reducing Activation Recomputation in Large Transformer Models, 2022

LayerNorm position-wise

Dropout element-wise

不依赖序列

序列并行(Sequence Parallelism, SP)



Only TP

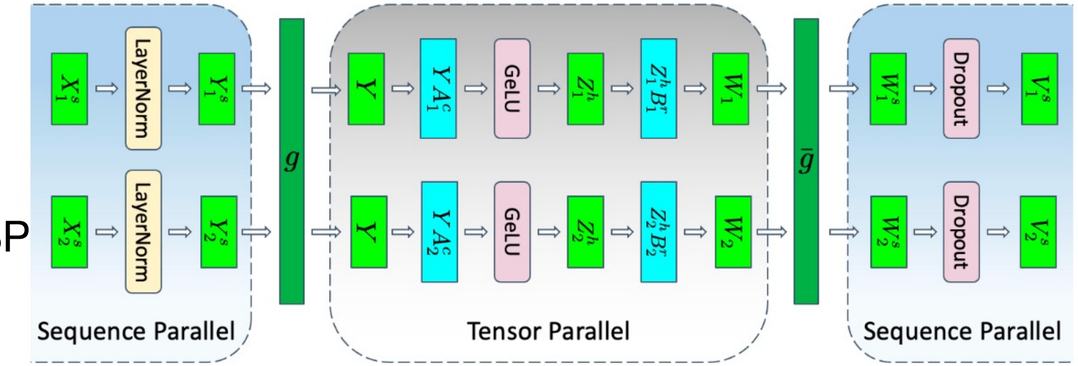
(a) MLP

Megatron-LM, 2019

f: Identity in forward, AllReduce in backward

g: AllReduce in forward, Identity in backward

TP + SP



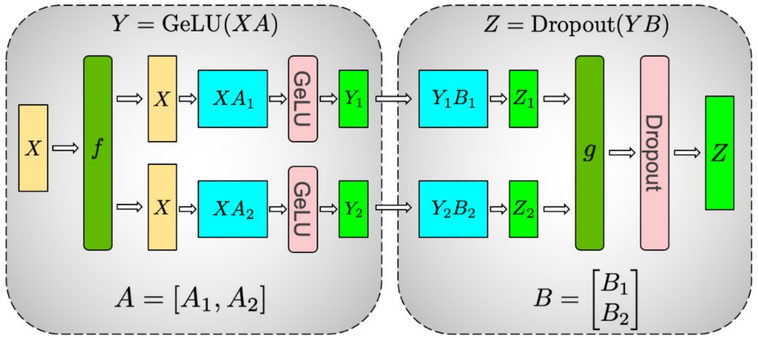
Reducing Activation Recomputation in Large Transformer Models, 2022

$$
\begin{aligned}
[Y_1^s, Y_2^s] &= \mathrm{LayerNorm}([X_1^s, X_2^s]), \\
Y &= g(Y_1^s, Y_2^s), \\
[Z_1^h, Z_2^h] &= [\mathrm{GeLU}(YA_1^c),\ \mathrm{GeLU}(YA_2^c)], \\
W_1 &= Z_1^h B_1^r \ \text{ and } \ W_2 = Z_2^h B_2^r, \\
[W_1^s, W_2^s] &= \bar{g}(W_1, W_2), \\
[V_1^s, V_2^s] &= [\mathrm{Dropout}(W_1^s),\ \mathrm{Dropout}(W_2^s)].
\end{aligned}
$$

g: AllGather in forward, Reduce-Scatter in backward

$\bar{g}$: Reduce-Scatter in forward, AllGather in backward

序列并行(Sequence Parallelism, SP)



(a) MLP

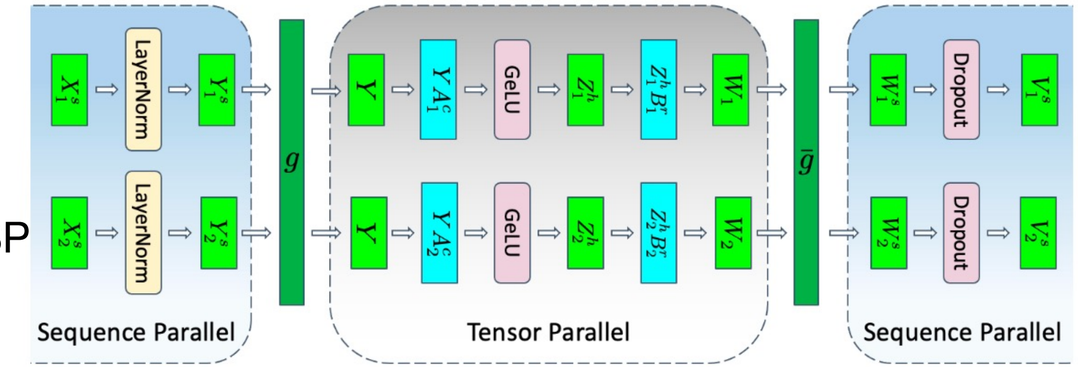
Megatron-LM, 2019

Only TP

f: Identity in forward, AllReduce in backward

g: AllReduce in forward, Identity in backward

TP + SP



Reducing Activation Recomputation in Large Transformer Models, 2022

$$
\begin{aligned}
[Y_1^s, Y_2^s] &= \mathrm{LayerNorm}([X_1^s, X_2^s]), \\
Y &= g(Y_1^s, Y_2^s), \\
[Z_1^h, Z_2^h] &= [\mathrm{GeLU}(YA_1^c),\ \mathrm{GeLU}(YA_2^c)], \\
W_1 &= Z_1^h B_1^r \quad \text{and} \quad W_2 = Z_2^h B_2^r, \\
[W_1^s, W_2^s] &= \bar{g}(W_1, W_2), \\
[V_1^s, V_2^s] &= [\mathrm{Dropout}(W_1^s),\ \mathrm{Dropout}(W_2^s)].
\end{aligned}
$$

=

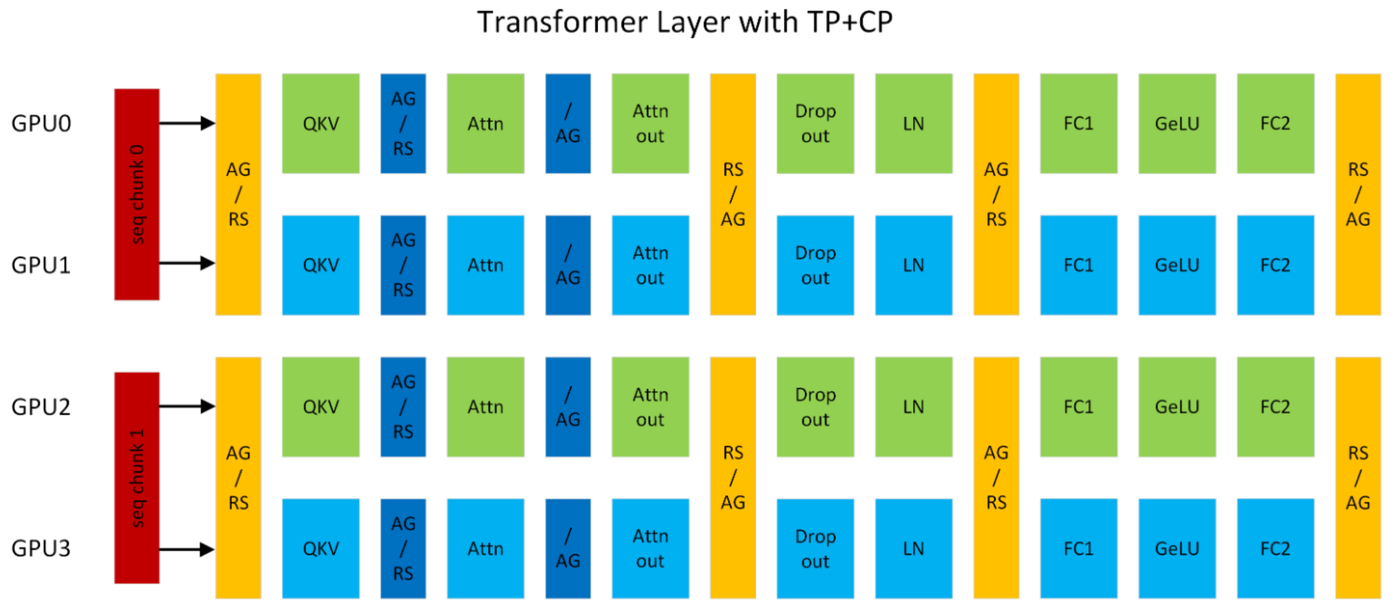
Ring AllReduce = ReduceScatter + AllGather

g: AllGather in forward, Reduce-Scatter in backward

$\bar{g}$: Reduce-Scatter in forward, AllGather in backward

上下文并行(Context Parallelism, CP)

- Another Sequence Parallelism, CP partitions the network inputs and all activations along sequence dimension
- CP ~ Ring Attention

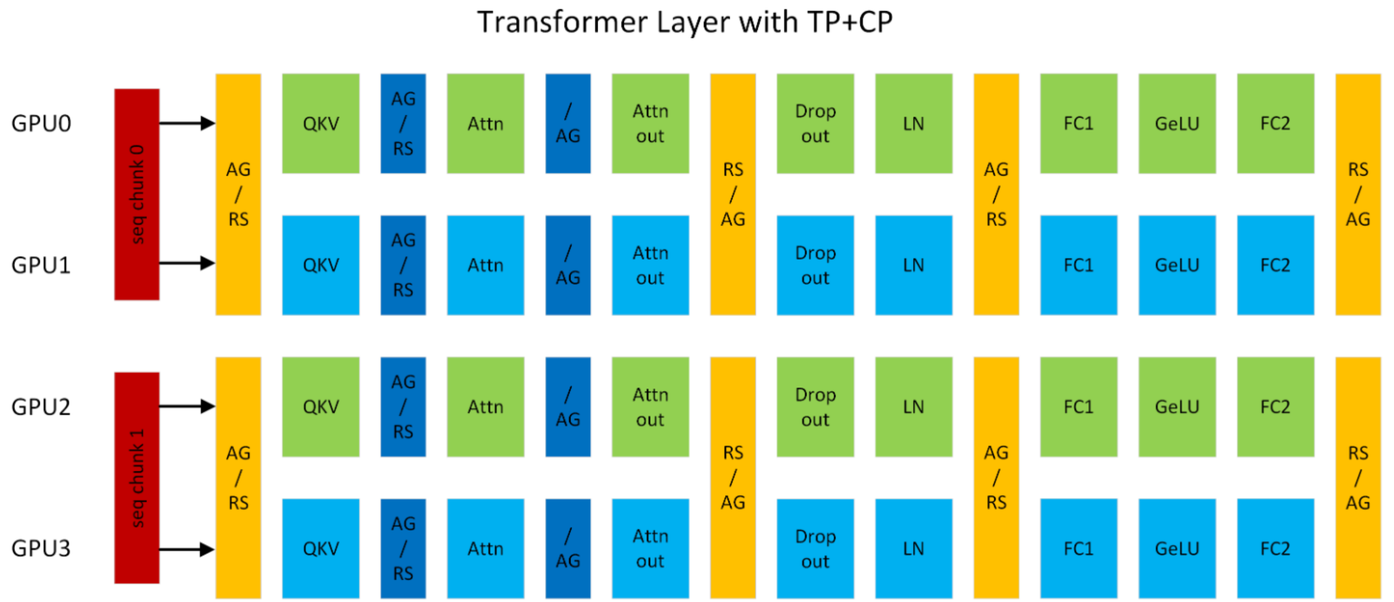


Megatron Core context_parallel

上下文并行(Context Parallelism, CP)

- Another Sequence Parallelism, CP partitions the network inputs and all activations along sequence dimension
- CP ~ Ring Attention

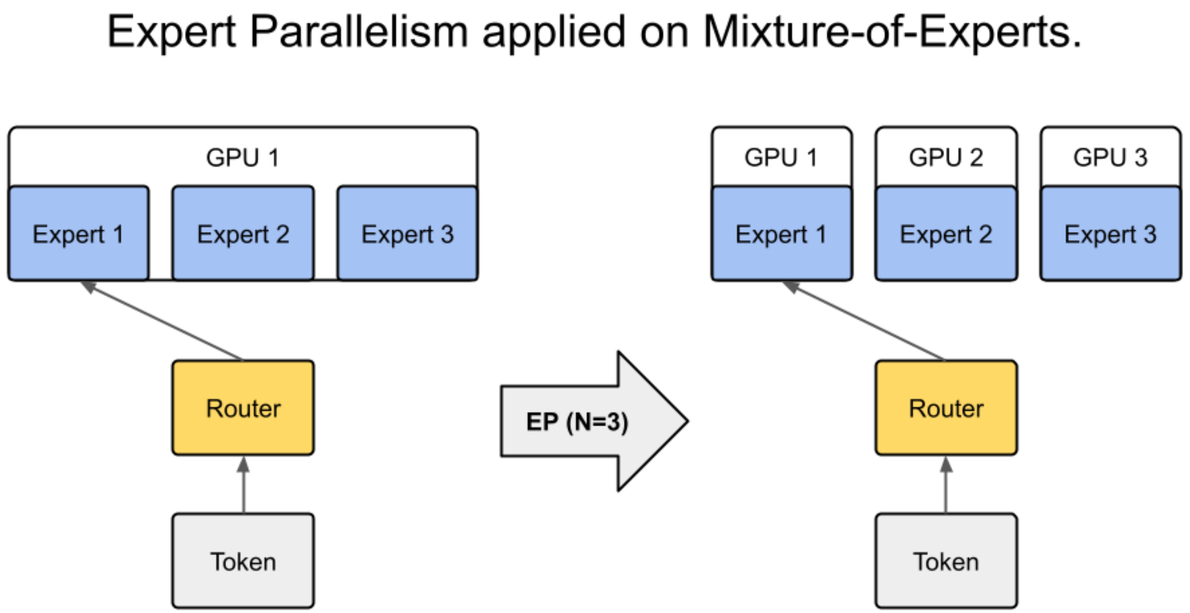
4D Parallelism: DP, TP, PP, CP



Megatron Core context_parallel

专家并行 (Expert Parallelism, EP)

- 作用于MoE模型的专家层
  - 利用专家之间的并行属性



NeMo, Expert Parallelism