北京智源人工智能研究院
BEIJING ACADEMY OF ARTIFICIAL INTELLIGENCE

大规模分布式训练优化技术

北京智源人工智能研究院AI框架研发组

2025年8月16日

目录

01 大规模分布式训练概念

02 常见分布式训练并行方法

03 FlagScale分布式训练优化

04 FlagScale异构混合训练技术

1. 大规模分布式训练概念

大规模分布式训练概述

什么是大规模分布式训练

大规模分布式训练是指利用多个计算节点，并行处理大规模数据集，以加速机器学习和深度学习模型的训练过程。

大规模分布式训练的发展历程

从早期的简单指令级并行计算，发展到如今的高效分布式训练框架，技术不断演进，性能不断提升。

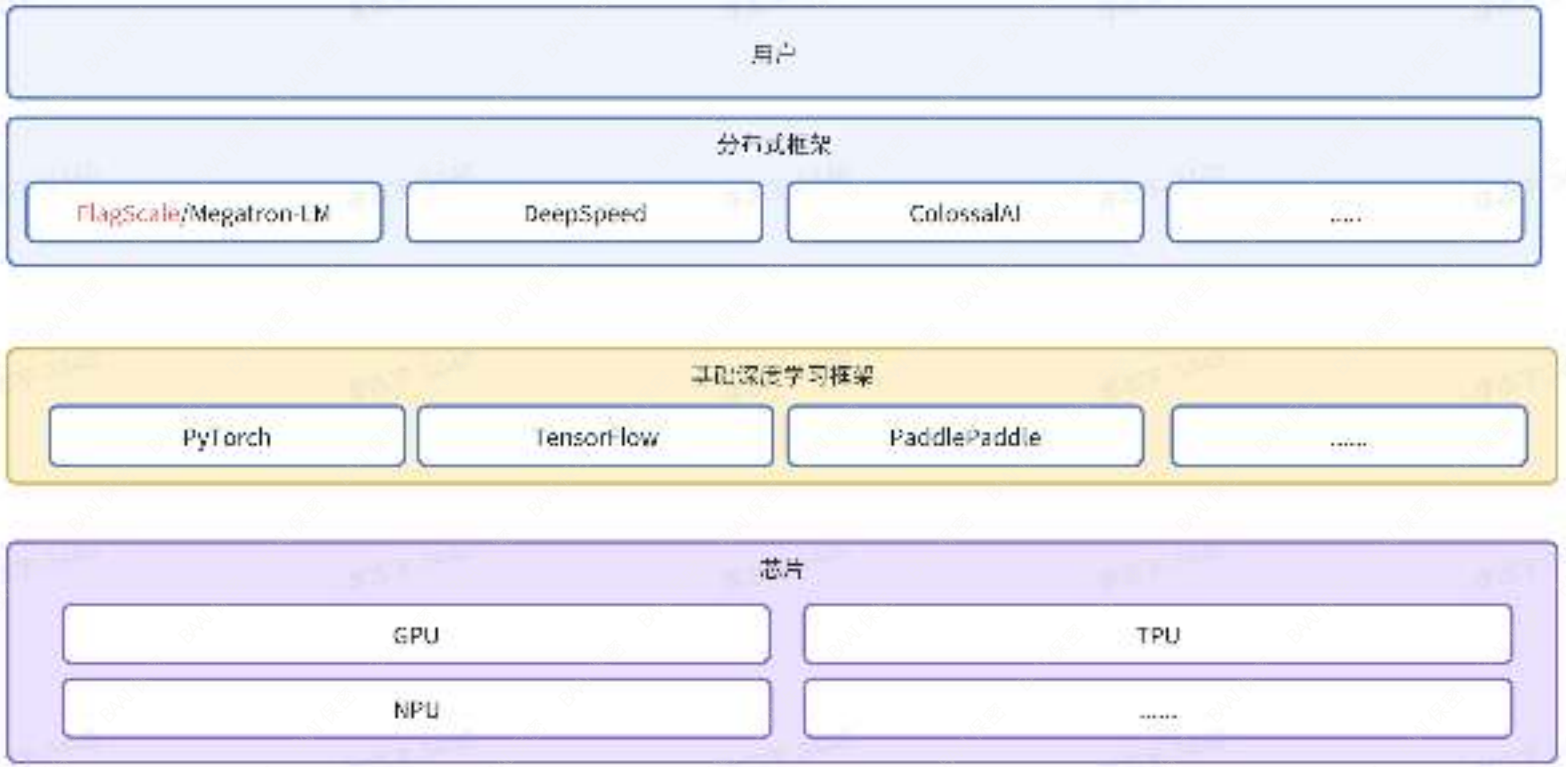
为什么需要大规模分布式训练

随着数据集和模型规模的不断增加，单个计算节点已无法满足训练需求，分布式训练可以有效缩短训练时间，提高计算资源的利用率。

大规模分布式训练的应用场景

广泛应用于自然语言处理、推荐系统等领域，如训练大语言模型、多模态模型等。

大规模分布式架构



分布式框架层：分布式框架层支持在多设备或多节点上部署和运行AI应用，实现资源的高效利用和协同工作，如FlagScale(Megatron-LM)、DeepSpeed等

框架层：基础框架层提供构建、训练和部署AI模型的工具集，如TensorFlow、PyTorch、PaddlePaddle等。

芯片层: 芯片层是AI技术的基础，提供计算能力，包括GPU、TPU、NPU等专用芯片。

2. 常见分布式训练并行方法

大语言模型

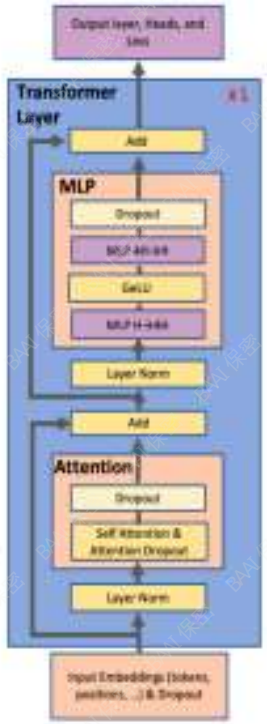


Figure 2. Transformer Architecture. Purple blocks correspond to fully connected layers. Each blue block represents a single transformer layer that is replicated N times.

Reference: https://arxiv.org/pdf/1909.08053

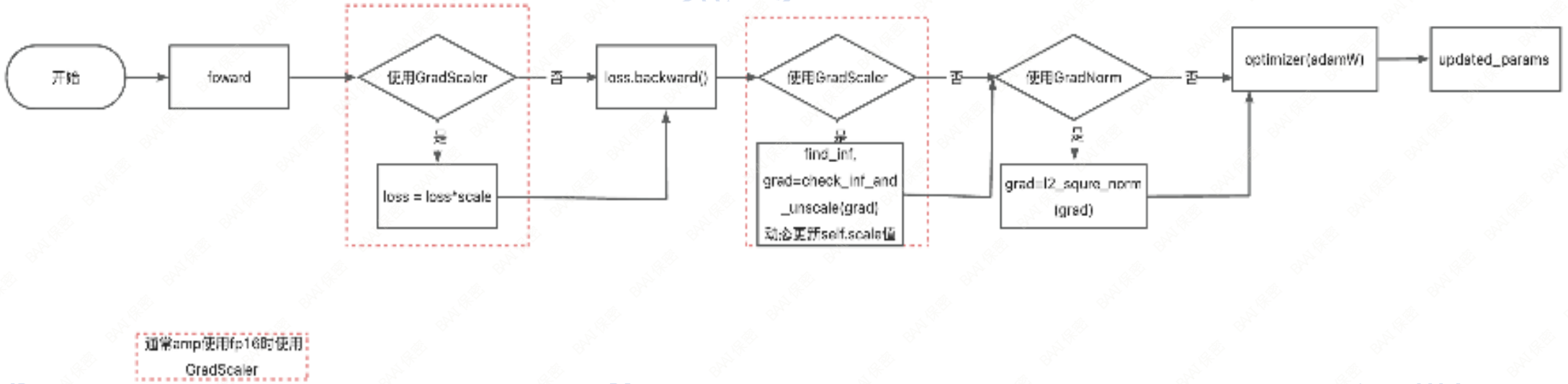
从整体到局部进行分解：

- 整体上分为Embedding，TransformerBlock，lm_head，通常将TransformerBlock称为backbone，负责对数据进行特征提取。
- TransformerBlock由多个TransformerDecoderLayer组成
- TransformerDecoderLayer由Attention和MLP组成

大语言模型的扩展可以从宽度“Attention-Head-Dim”和深度“TransformerDecoderLayer”两个角度进行。

如DeepSeek-v3参数为671B-A37B，hidden_size为7168，layer数为61， 而最近的GLM4.5从深度扩展，参数为355B-A32B，但layer层数多达92

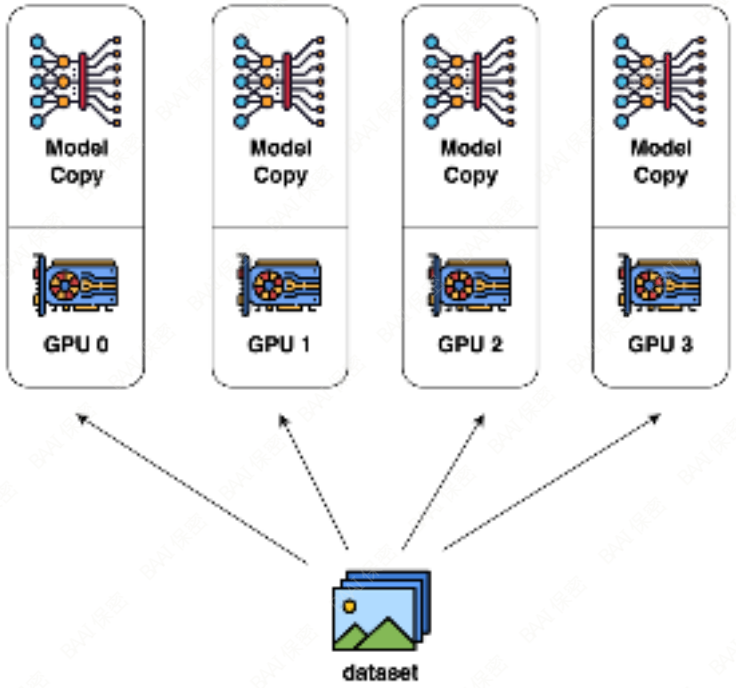
模型训练执行流程



- 前向传播（Forward）：参数与输入数据计算得到激活变量
- 反向传播（Backward）：根据激活变量与loss计算梯度
- 参数更新（Optimizer）：优化器使用梯度来更新参数

数据并行

朴素数据并行



- 数据不同，参数相同：在每块计算GPU上都拷贝一份完整的模型参数，把一份数据均匀分给不同的GPU
- 各自获得梯度：每块GPU做完Fwd和Bwd后，算得一份梯度G
- 梯度同步：所有GPU对梯度G使用all_reduce进行同步，同步后的结果用于更新模型参数W，保证参数一致性
- 通信量：由于all_reduce操作可以看做reduce_scatter+all_gather，所以约定all_reduce通信量是2φ，朴素数据并行的通信量为2φ*N

数据并行

零冗余数据并行

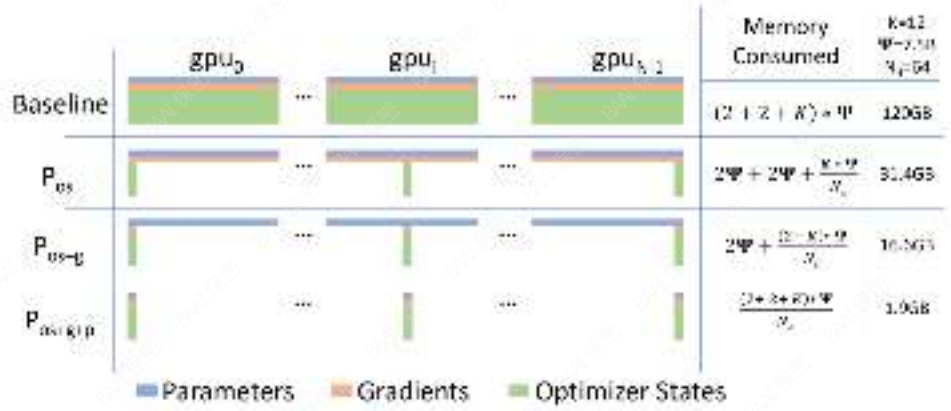


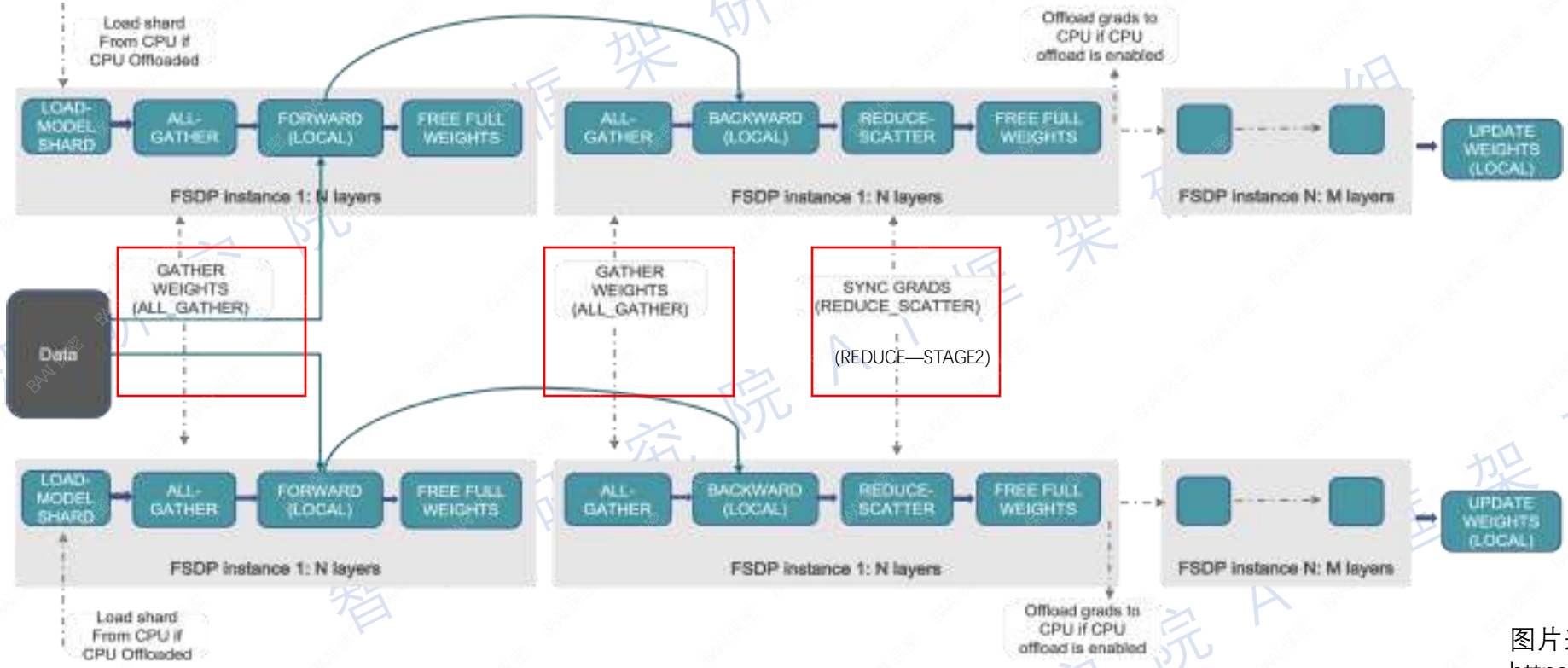
Figure 1: Comparing the per-device memory consumption of model states, with three stages of ZeRO-DP optimizations. Ψ denotes model size (number of parameters), K denotes the memory multiplier of optimizer states, and N_d denotes DP degree. In the example, we assume a model size of $\Psi = 7.5B$ and DP of $N_d = 64$ with $K = 12$ based on mixed-precision training with Adam optimizer.

Reference: https://arxiv.org/pdf/1910.02054

- 朴素数据并行问题：所有的GPU各自维护了一份模型参数、梯度、优化器状态，无法进行有效的scaling训练超大模型
- 零冗余数据并行：不同的GPU维护部分的参数、梯度、优化器状态，理想状态下每个GPU所维护的训练状态互不重叠，大大提高了分布式训练的scaling能力
- 零冗余策略：将零冗余数据并行划分为三层级
  - Stage1: 将优化器状态进行切分
  - Stage2: 将优化器状态+梯度进行切分
  - Stage3: 将优化器状态+梯度+参数进行切分

数据并行

零冗余数据并行



图片来源:
https://docs.pytorch.org/tutorials/intermediate/FSDP_tutorial.html

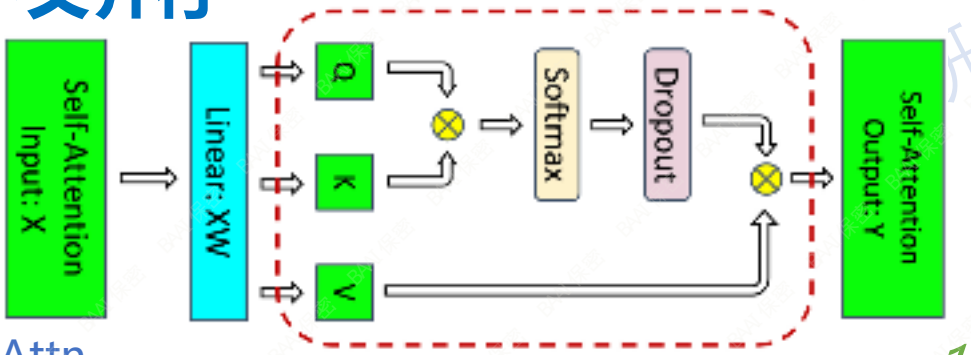
ZeRO实现由多个版本，如DeepSeed、FSDP，这里以FSDP(Stage3)描述执行过程:

- Forward通过all_gather得到完整模型参数，执行对应计算后丢弃不属于自己的参数
- Backward通过all_gather得到完整模型参数，执行对应计算后丢弃不属于自己的参数
- Backward得到参数的Grad后，使用reduce_scatter进行同步，最后优化器更新参数

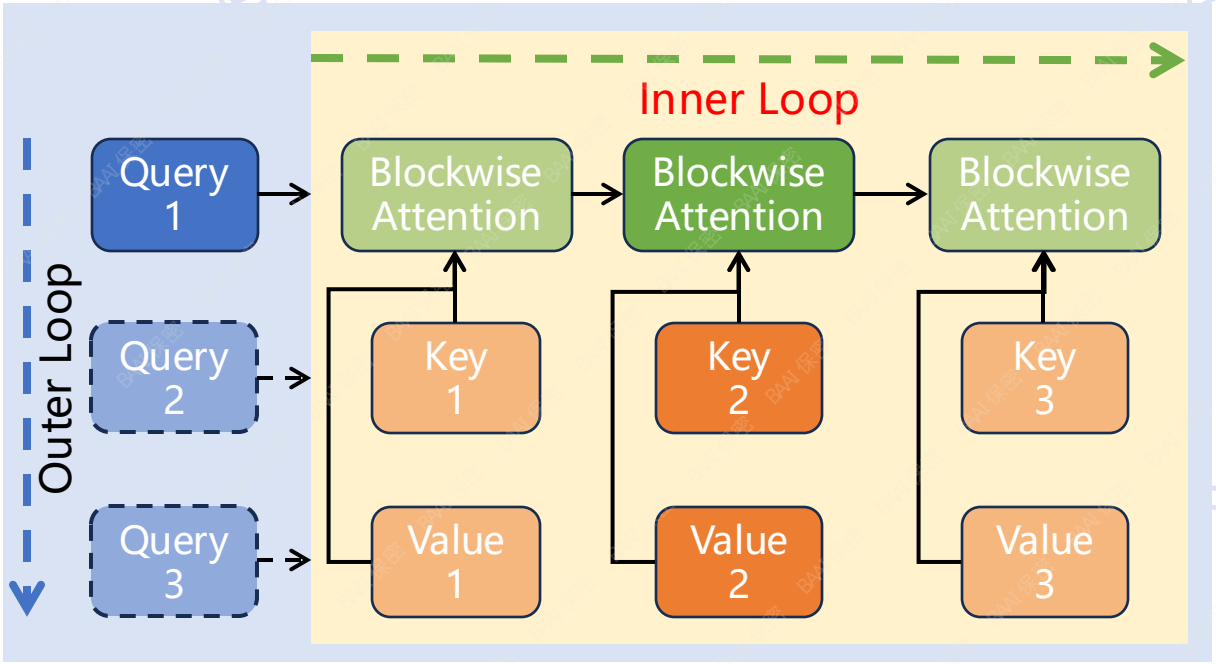
如果是Stage2，需要保留梯度同步的reduce_scatter和Forward中参数同步的all_gather
如果是Stage1，梯度同步使用reduce和Forward中参数同步的all_gather
相比与朴素数据并行的通信量，stage1和stage2的没有增多，stage3变为1.5倍

数据并行

上下文并行

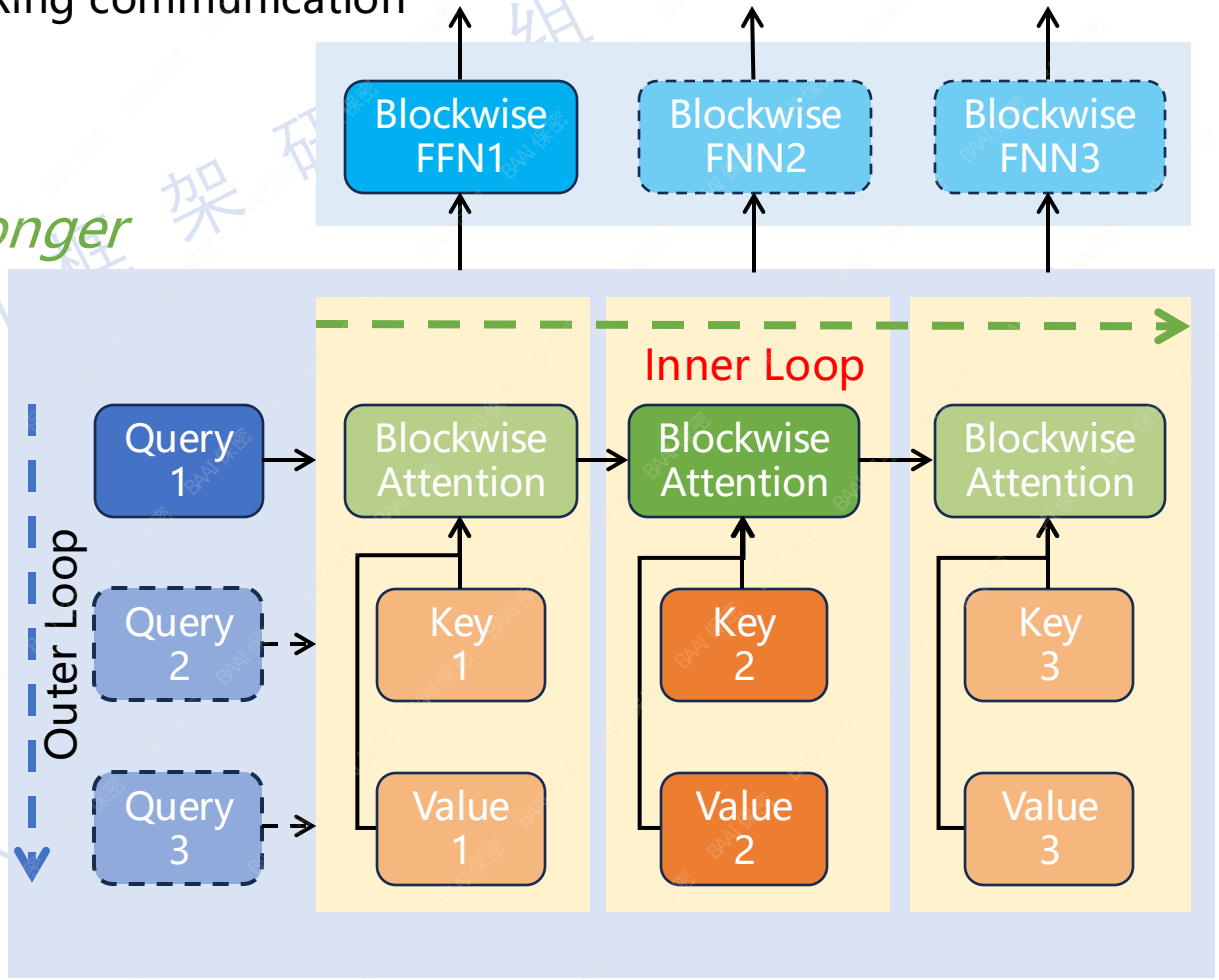


FlashAttn
- Online softmax
- Blockwise attention

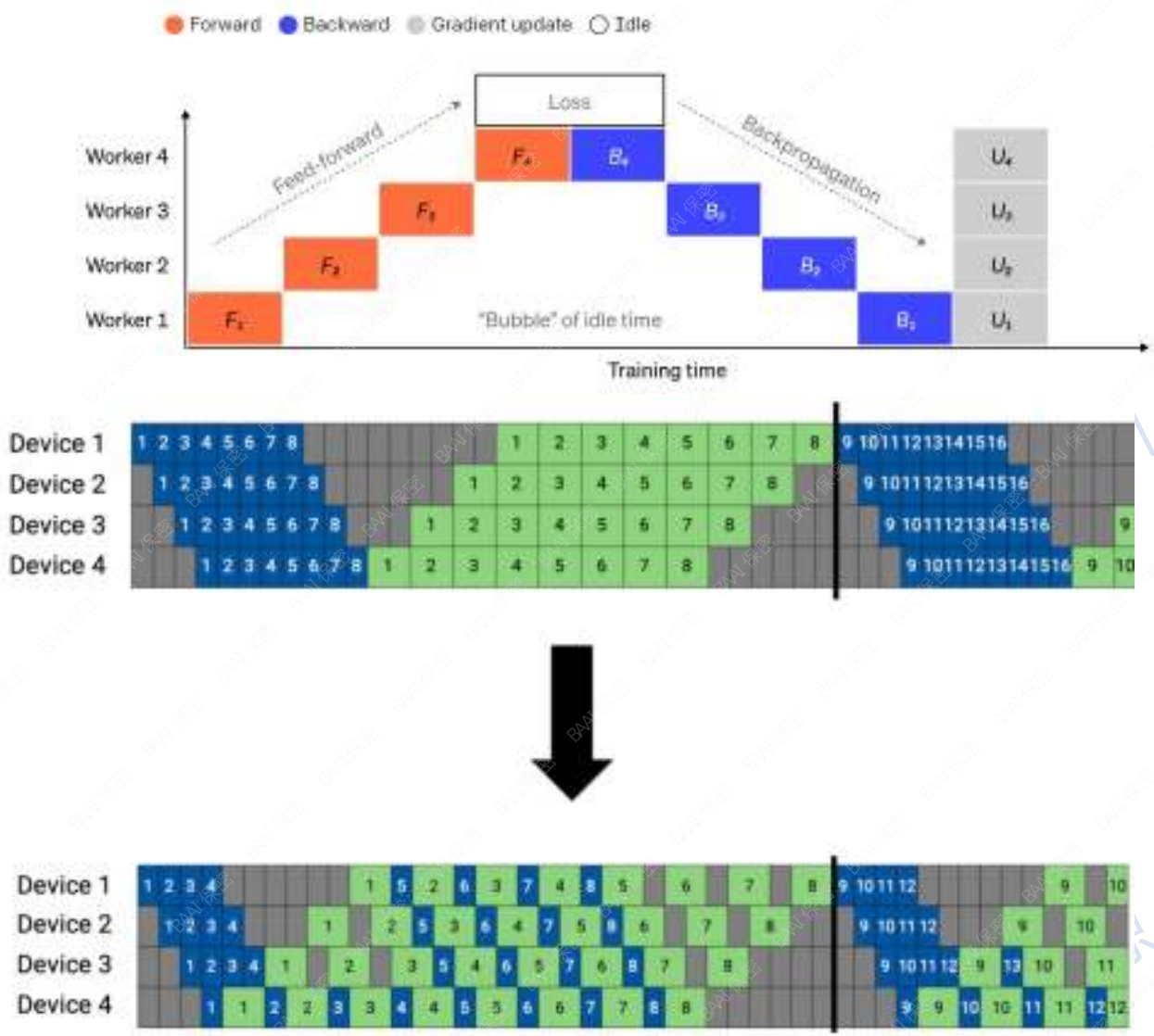


1000x longer

RingAttn
- Online softmax
- Blockwise attention and blockwise feedforward (FFN)
- Ring communication



流水线并行



图片来源：https://arxiv.org/pdf/2104.04473

- 朴素流水线并行

将模型横向切分，每次仅迭代一个批次数据

假设分为p个stages，bubble时间为：(p-1)*(Tf+Tb)
Bubble率为：(p-1)/p，资源利用率极低

- F-Then-B

增加多个批次的数据提高并行度，所有Forward全部执行完毕后再执行Backward

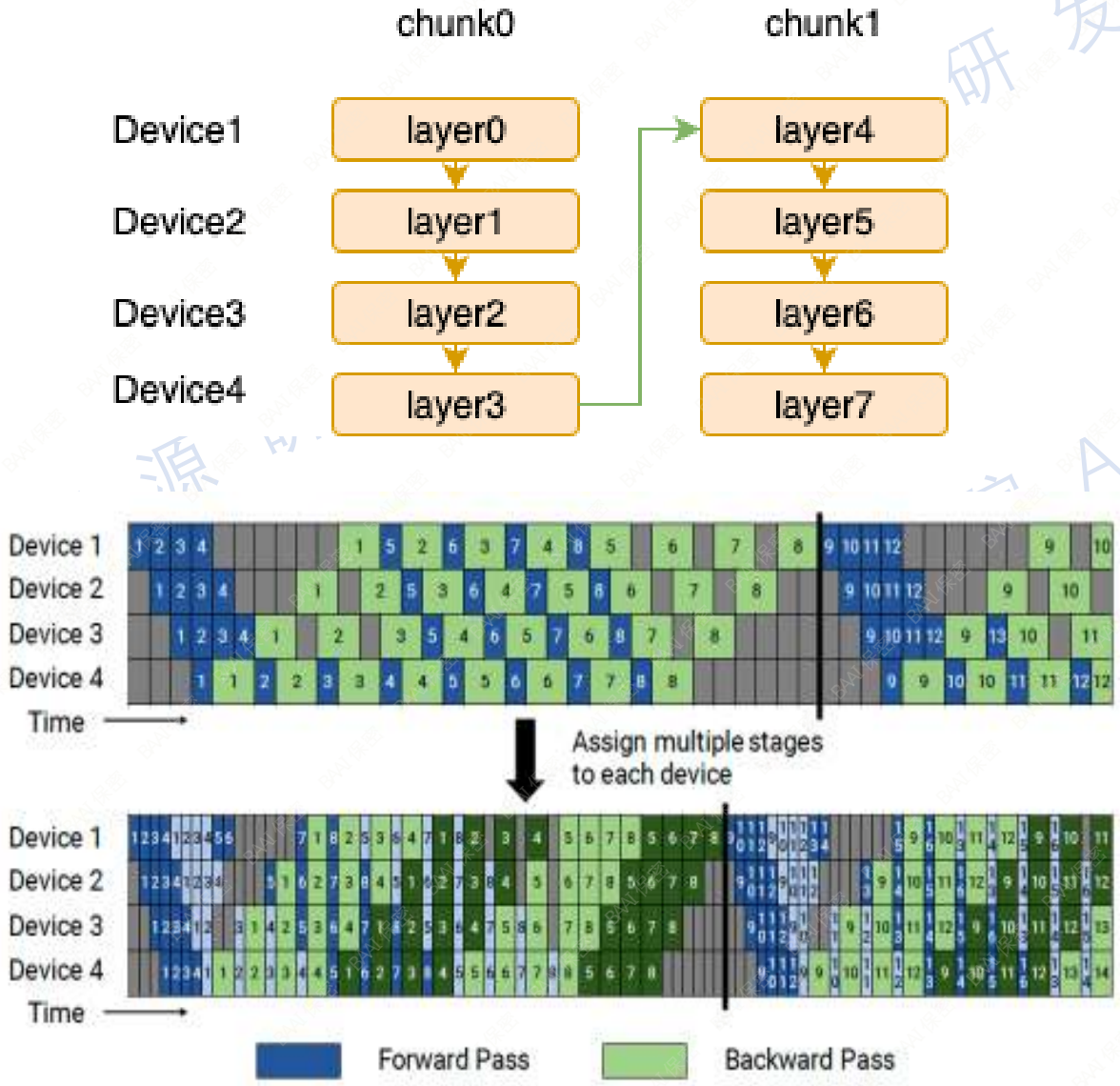
假设分为p个stages，m个批次的数据，bubble时间为：(p-1)*(Tf+Tb)
Bubble率为：(p-1)/(m*p)，降为朴素流水线并行的1/m，提高数据批次可以降低bubble rate
峰值显存出现在warmup结束时刻，总共累计了m个forward的中间变量

- 1F1B：降低峰值显存

最后一个stage的第一个Forward执行完毕后立刻执行其Backward

假设分为p个stages，m个批次的数据，bubble时间为：(p-1)*(Tf+Tb)
Bubble率为：(p-1)/(m*p)，跟F-Then-B一致
峰值显存出现在stage0的warmup结束时刻，总共累计了p个forward的中间变量，但是最后一个stage仅需累计1个forward的中间变量，峰值显存较低，不同stage的显存占用不均衡

流水线并行

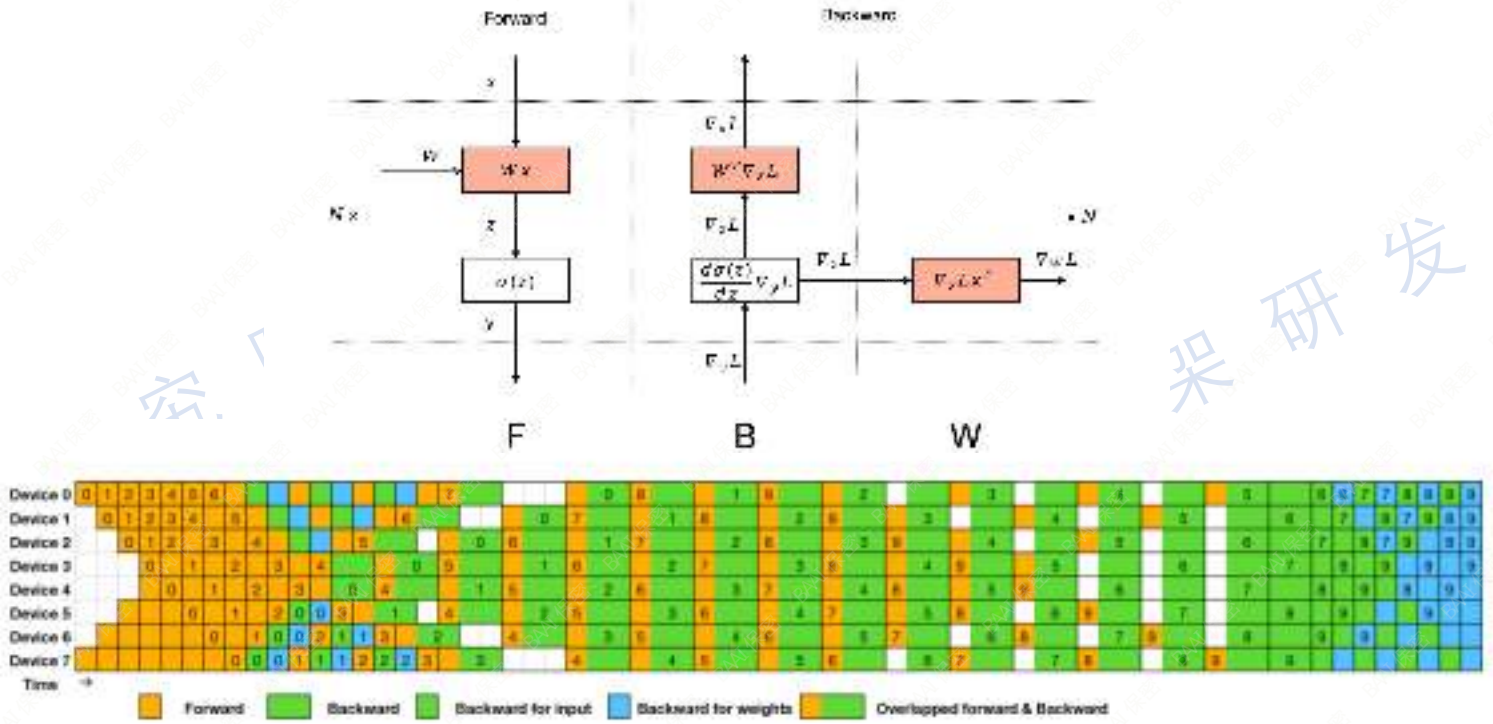


图片来源：https://arxiv.org/pdf/2104.04473

- Interleaved 1F1B：进一步降低bubble

- 在1F1B的基础上，将模型进一步切分为chunk，每个device上分配多个chunks
- 假设分为p个stages每个stages有c个chunks，m个批次的数据，bubble时间为：(p-1)*(Tf+Tb)/c
- Bubble率为：(p-1)/(c*m*p)，减低为1F1B的1/c，显然提高chunks数量将模型切分的更加细粒度可以降低bubble rate，但是同时会引入更多的send/recv通信。
- 峰值显存出现在stage0的warmup结束时刻，总共累计了10个forward的中间变量，但是最后一个stage仅需累计5个forward的中间变量，峰值显存较低，不同stage的峰值显存依旧不均衡

流水线并行

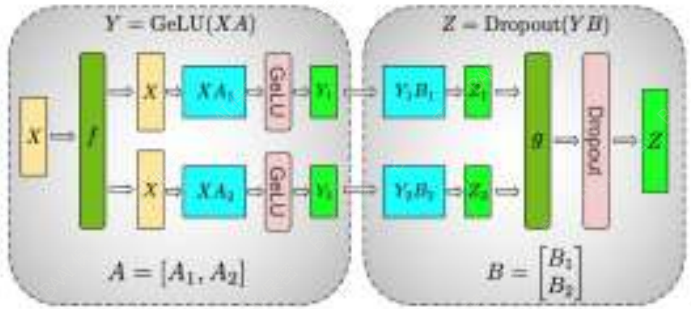


图片来源：https://arxiv.org/html/2412.19437v1

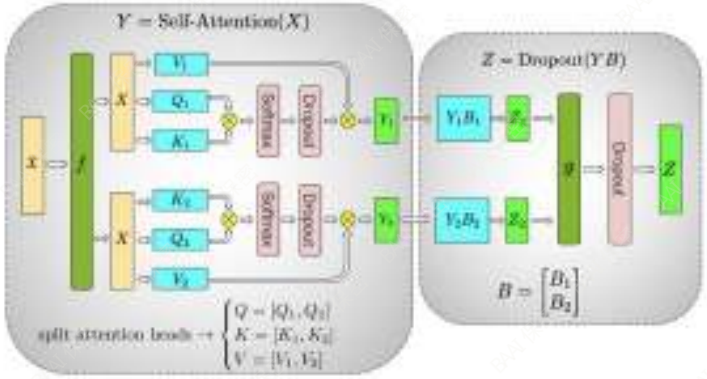
- DeepSeek-DualPipe：降低bubble&显存均衡

- 在1F1B和split_BW基础上，模型切分为F,B_a,B_w
- 在每个Device上均有两个独立的执行流
- 细粒度调度计算和通信达到在steady阶段彻底的overlapping
- Bubble可以近似降低为1F1B的一半
- 每个device的显存保持均衡
- DualPipe需要针对特定模型进行细粒度的调优才能达到最佳效果

张量并行



(a) MLP



(b) Self-Attention

Figure 3. Blocks of Transformer with Model Parallelism. f and g are conjugate. f is an identity operator in the forward pass and all reduce in the backward pass while g is an all reduce in the forward pass and identity in the backward pass.

图片来源：https://arxiv.org/pdf/1909.08053

以MLP为例：

第一个GEMM后面紧跟一个非线性操作GeLU，$Y = GeLU(XA)$，如果我们对A的row进行split，则有

$$X = [X_1, X_2], A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$
$$XA = X_1A_1 + X_2A_2$$

由于GeLU是个非线性操作，$GeLU(X_1A_1 + X_2A_2) \neq GeLU(X_1A_1) + GeLU(X_2A_2)$，所以需要在GeLU前增加一个All-reduce操作，才能输入GeLU。

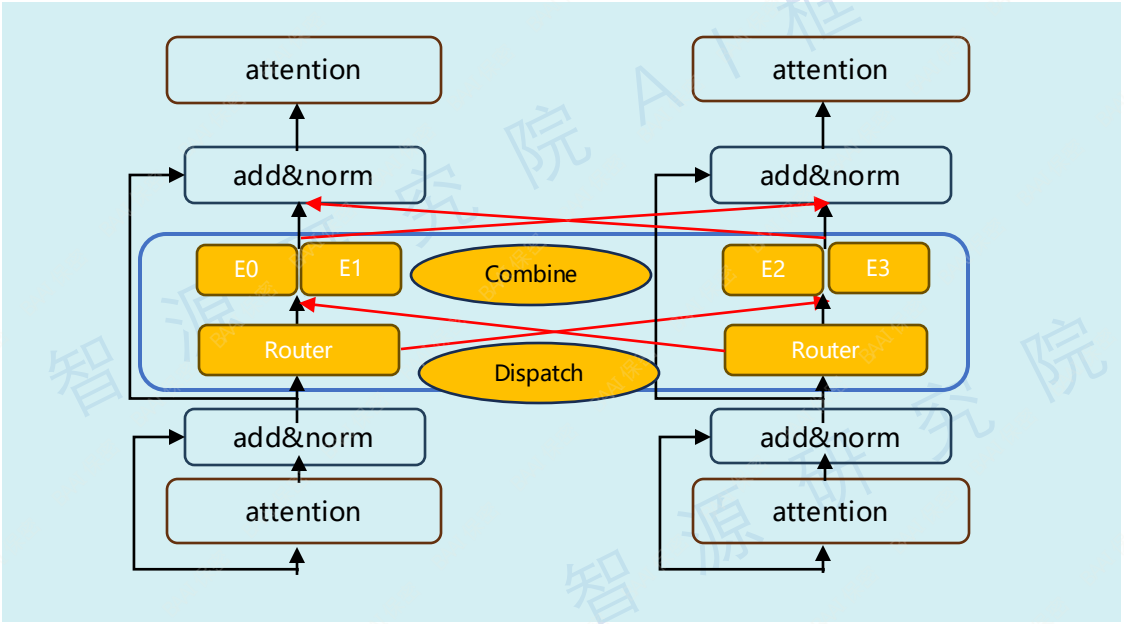
反之，我们对A沿着columns进行切分。

$$A = [A_1, A_2]$$
$$XA = [XA_1, XA_2]$$

这样可以直接输入GeLU，不需要通信操作。

由于MLP含有两个GEMM所以我们在第一个GEMM采用column parallel，第二个GEMM采用row parallel，减少通信操作。

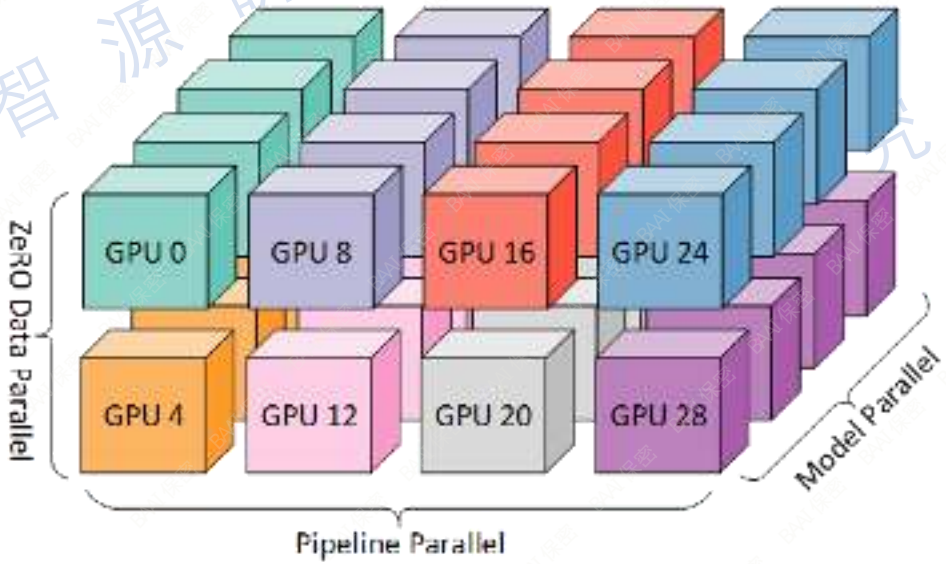
专家并行

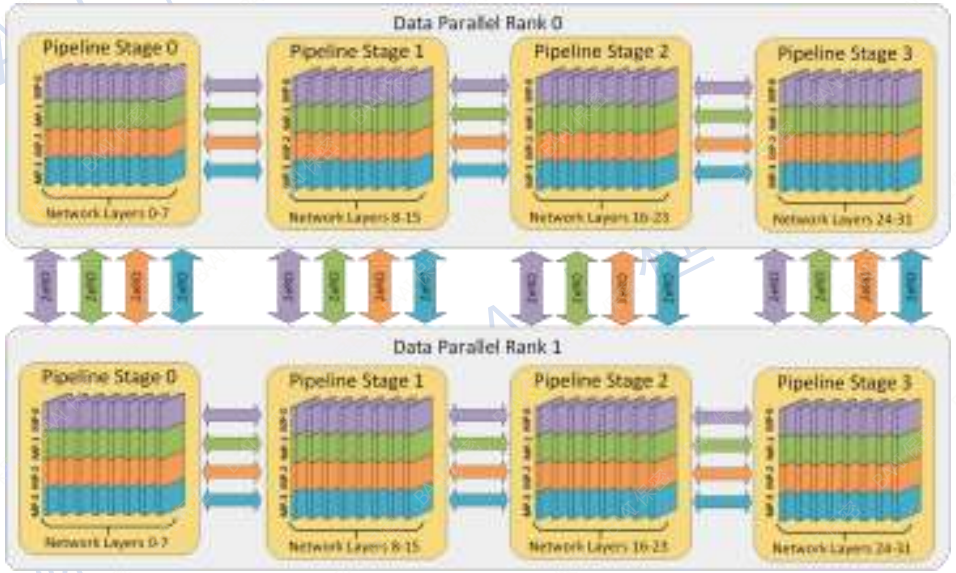


- MoE(Mixture-of-Experts)替代了原来简单MLP，可以简单理解为带有router的多MLP结构，每个MLP对应一个Experts
- EP并行是指将不同的Experts分配到不同的GPU上
- EP并行在router时需要进行Dispatch通信，通常是all2all操作
- 在对应专家执行完毕后，需要Combine操作将结果同步到相应的device上，通常也是all2all操作，但是数据分配方式跟Dispatch相反

混合并行示例

- Stage内部：Fwd和Bwd都需要需要数次TP.sync
- Stage之间：Fwd和Bwd都需要一次PP.sync
- 完成所有Stage的Fwd和Bwd之后，需要一次DP.sync





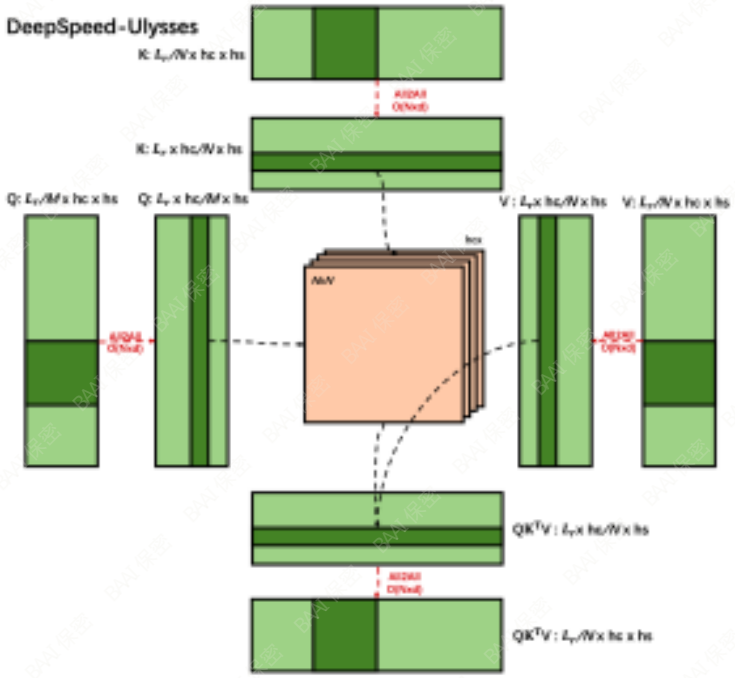
3. FlagScale并行训练优化

统一序列并行

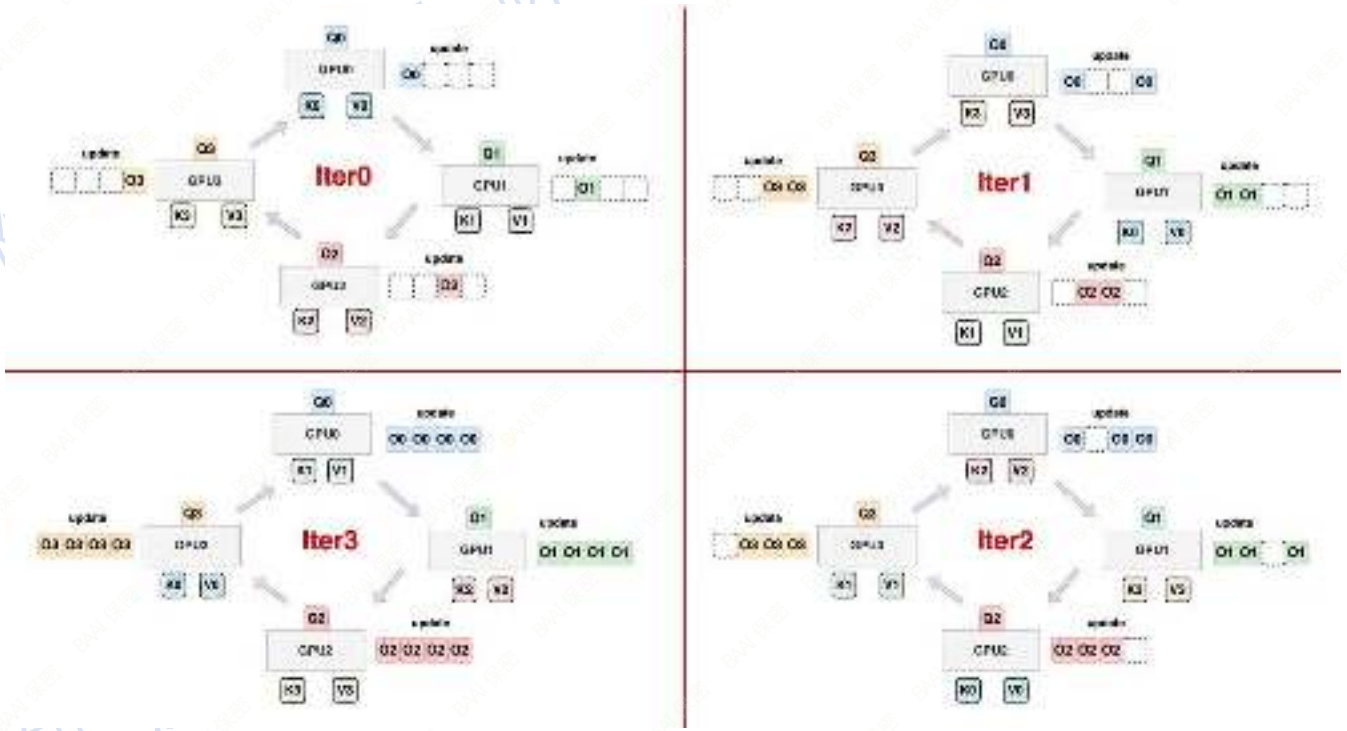
将transformer划分为core-attn和其他组件，只有core-attn中需要完整sequence的KV，其他部分都是序列无关(sequence independent)，所以只要解决split sequence给core-attn带来的问题，就可以实现基本的序列并行方案：

$$\text{Core}-\text{attn: } \mathrm{O} = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

目前主流的两种序列并行的方法为Ulysses-SP，另一个就是Megatron-LM基于ring-attention的context-parallel



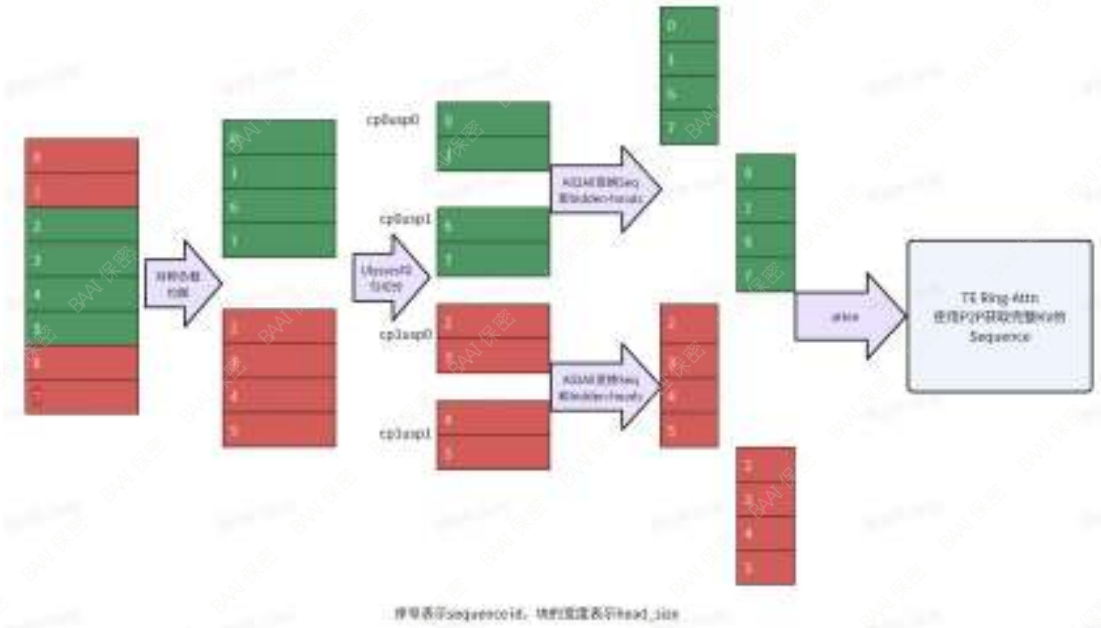
借助All2All获取完整Seq的KV

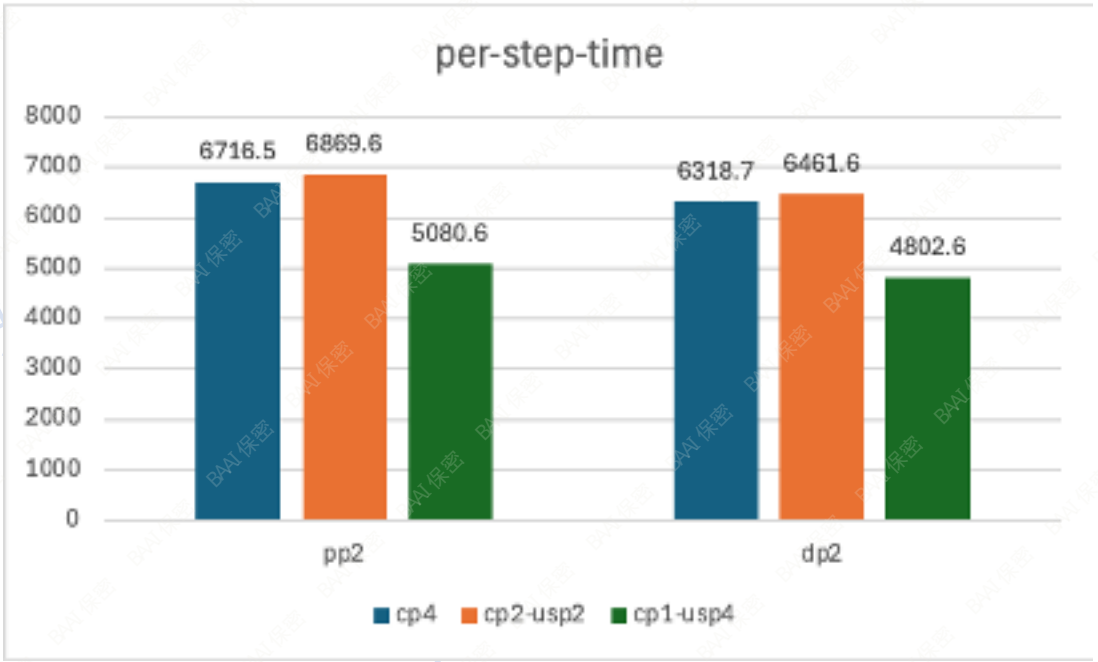


借助P2P获取完整Seq的KV

图片来源：https://arxiv.org/abs/2405.07719
https://zhuanlan.zhihu.com/p/5502876106

统一序列并行





CP=2,Ulysses-sp=2(外部是CP-内部是USP)：

- 由于我们最常用的是Causal model，在cor-atten中是下三角，为了平衡算力，CP切分sequence时，采用对称切分
- Ulysses-sp不需要进行二次平衡，因为进入core-attn之前会通过All2All恢复CP切分后局部sequence完整的KV
- 借助TE的ring-attn(send/recv)，恢复全局sequence完整的KV，计算最后的score
- Backward过程中数据执行相反的通信过程

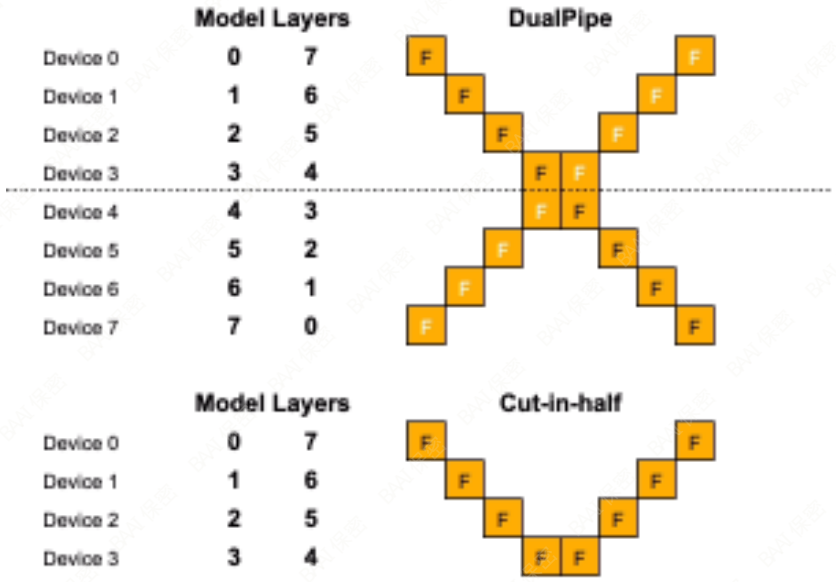
- Ulysses Sequence Parallel性能更高
- 两种序列并行结合，解决Ulysses-SP维度受num_heads的限制，同时解决国产硬件不支持TE下无法开启长序列训练的问题

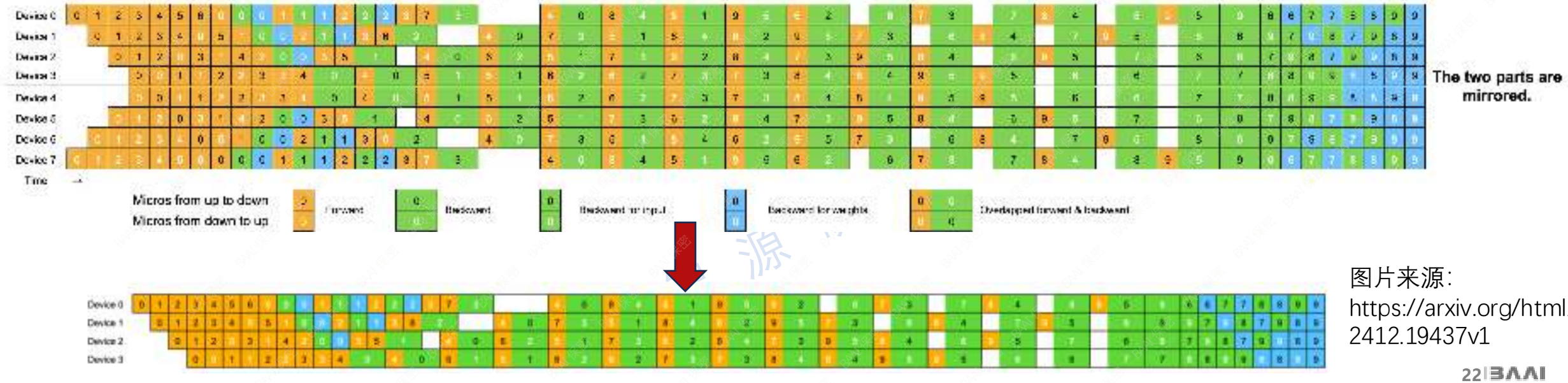
PR:Support ulysses sequence parallism

双向流水线并行

由于DeepSeek-DualPipe需要维持两份模型参数，实现起来负担较重。通过分析可以看到DualPipe本质上就镜像的，将双向流水线改为V形流水线，只需要一半的卡即可实现相似的调度效果。

PR:https://github.com/FlagOpen/FlagScale/pull/644





图片来源：
https://arxiv.org/html/2412.19437v1

案例：多模态模型Qwen2.5-VL分布式训练

智源研究院具身团队依托FlagScale完成高效训练RoboBrain2.0(base模型是Qwen2.5-VL)的7B和32B模型，FlagScale在多模态训练的主要优化和创新点：

- 数据预处理和加载：基于原生Megatron-Energon数据加载，整体耗时大幅缩减，在预处理320k的样本数据时，耗时从2小时降低10分钟内
- 显存预分配机制：为了应对数据sample长度变化范围大，导致OOM问题，在首次迭代预分配最大显存，支持稳定高效训练
- 分离recompute：为了应对超长图片token，对VIT和LLM部分采用不同的reompute策略

使用FlagScale相比原始Llama-factory端到端性能提升超150%，为具身团队在同样资源和时间下进行更多组训练实验提供了方便，最终RoboBrain2.0的7B和32B在多项评测指标达到SOTA：

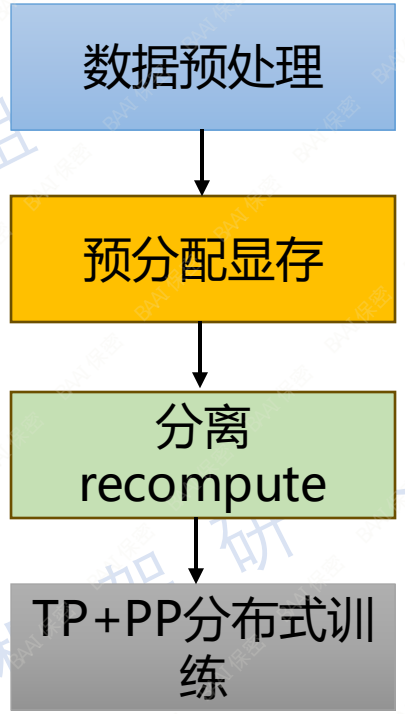


Table 2 Performance across five spatial reasoning benchmarks. The best results among different models are highlighted in **bold**, while the second-best results are underlined.

Models / Metrics	BLINK			CV-Bench	EmbSpatial	RoboSpatial	RefSpatial-Bench		
	Dep.	Spa.	All ↑	All ↑	All ↑	All ↑	Loc.	Pla.	All ↑
General Baselines									
Gemini-2.5-Pro-preview-05-06 [16]	79.03	84.62	81.83	84.59	**78.74**	59.87	44.58	31.73	38.16
Gemini-2.5-Flash-preview-04-17 [16]	77.42	79.02	78.22	84.03	74.75	54.10	37.50	23.00	30.25
GPT-o4-mini-2025-05-16 [45]	79.03	**88.11**	83.57	85.21	78.29	51.25	15.00	19.58	17.29
GPT-4o-2024-11-20 [22]	72.58	83.22	77.90	78.63	71.92	44.42	8.00	9.55	8.78
Claude-Sonnet-4-2025-05-14 [3]	75.81	80.42	78.12	78.43	64.26	51.26	5.00	10.37	7.69
Qwen2.5-VL-32B-Instruct [53]	77.42	85.31	81.37	81.59	74.45	52.10	10.83	10.60	13.72
Qwen2.5-VL-72B-Instruct [53]	74.19	78.32	76.26	82.68	73.30	48.33	23.50	15.83	19.67
Embodied Baselines									
Cosmos-Reason1-7B [4]	63.71	73.43	68.57	74.71	65.22	38.81	0.84	1.04	5.44
VeBrain-8B [36]	78.23	81.12	79.68	78.57	70.52	42.48	0.00	0.57	0.30
Magma-8B [74]	65.32	66.43	65.88	60.98	64.59	33.71	1.00	8.00	4.50
RoboBrain-7B-1.0 [23]	75.81	78.32	77.07	76.22	68.13	51.51	14.43	5.41	9.92
RoboBrain-7B-2.0	**84.68**	83.22	**83.95**	**85.75**	76.32	54.23	36.00	29.00	32.50
RoboBrain-32B-2.0	79.84	87.41	83.63	83.92	78.57	**72.43**	**54.00**	**54.00**	**54.00**

相关链接：https://superrobobrain.github.io/

案例：多模态模型Qwen2.5-VL分布式训练

我们也在FlagScale中开源了RoboBrain2.0的训练代码，这是使用FlagScale训练步骤：

1. 下载安装FlagScale
    1. 下载FlagScale仓库
    2. 应用patch文件
2. 准备checkpoint
    1. 从huggingface或魔塔download权重
    2. 使用checkpoint转换工具进行转换
3. 预处理数据
    1. 从huggingface下载示例数据
    2. 使用energon预处理多模态数据
4. 自定义configuration
5. 将checkpoint转回HuggingFace格式评测

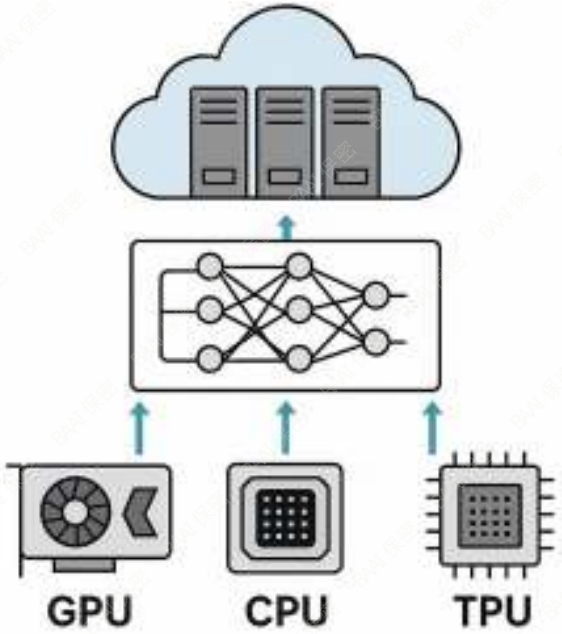
更加详细的可以参考：
https://github.com/FlagOpen/FlagScale/blob/main/flagscale/train/models/qwen2_5_vl/QuickStart.md

PS：如果最新的FlagScale运行有问题，可以将代码回滚到：commit-id(**dc6e824**)

4. FlagScale异构混合训练技术

异构混训概念

异构集群混合训练是指在由不同类型计算设备组成的集群中（如GPU、CPU、TPU、NPU等，以及不同代际/厂商的GPU），**协同进行大模型训练**的技术方法。其核心目标是充分利用异构计算资源，在保证训练效率与成本效益的前提下，支撑百亿乃至万亿参数规模的模型训练。



异构混训方案挑战

✗ **挑战1:** 性能会被最慢的芯片设备拖累。

✓ **解决方案:** 通过负载均衡任务划分来发挥不同类型芯片的算力潜能。

✗ **挑战2:** 在不同芯片之间进行通信是非常有挑战的。

✓ **解决方案:** 只需要在节点之间基于标准网络协议（IB/RoCE）进行通信。

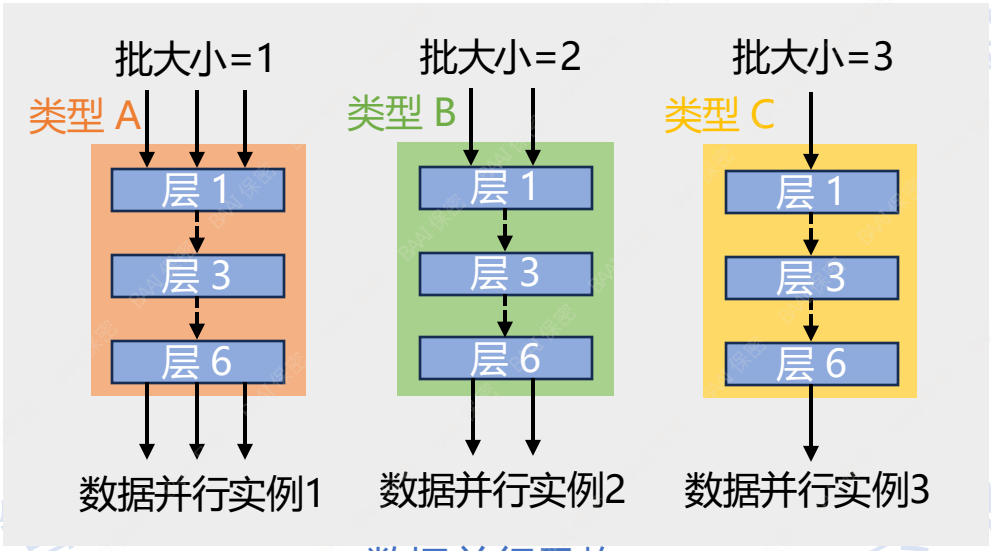
✗ **挑战3:** 不同芯片算力不同导致很难进行算力调度。

✓ **解决方案:** 从以芯片为中心的调度转换成以算力透明的调度。

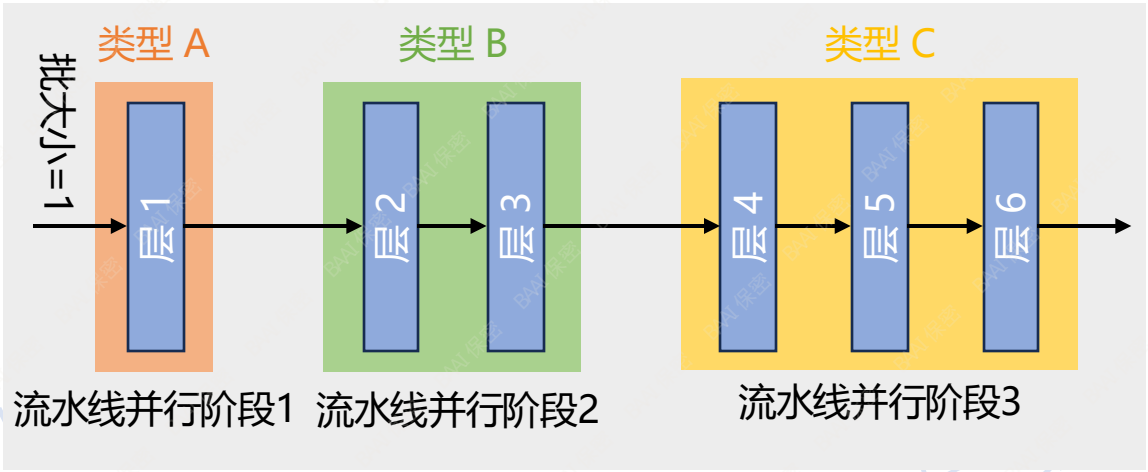
✗ **挑战4:** 异构训练的模型效果可能没法保证。

✓ **解决方案:** 对精度不明感，而且如果模型效果严重没对齐，可能芯片软硬件栈存在兼容性问题。

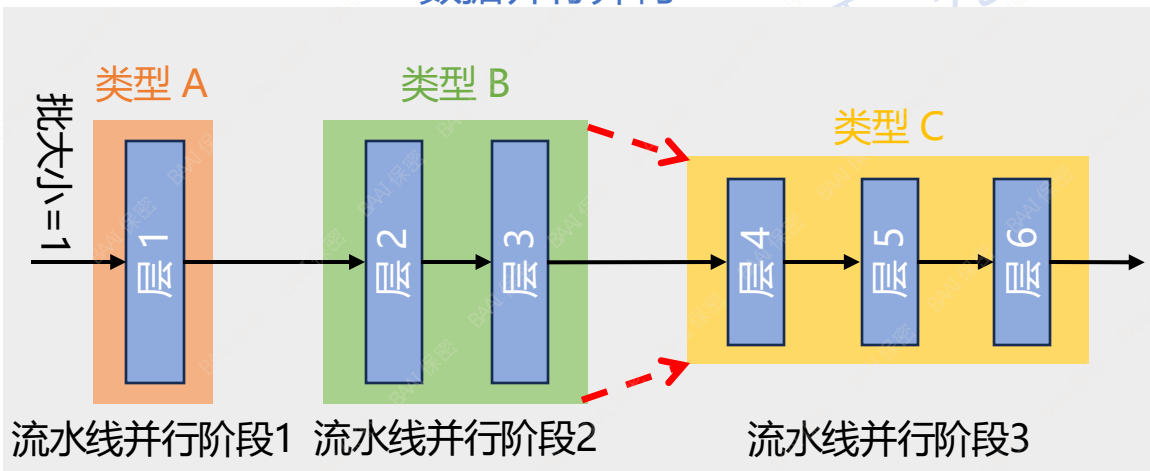
单一异构并行方案



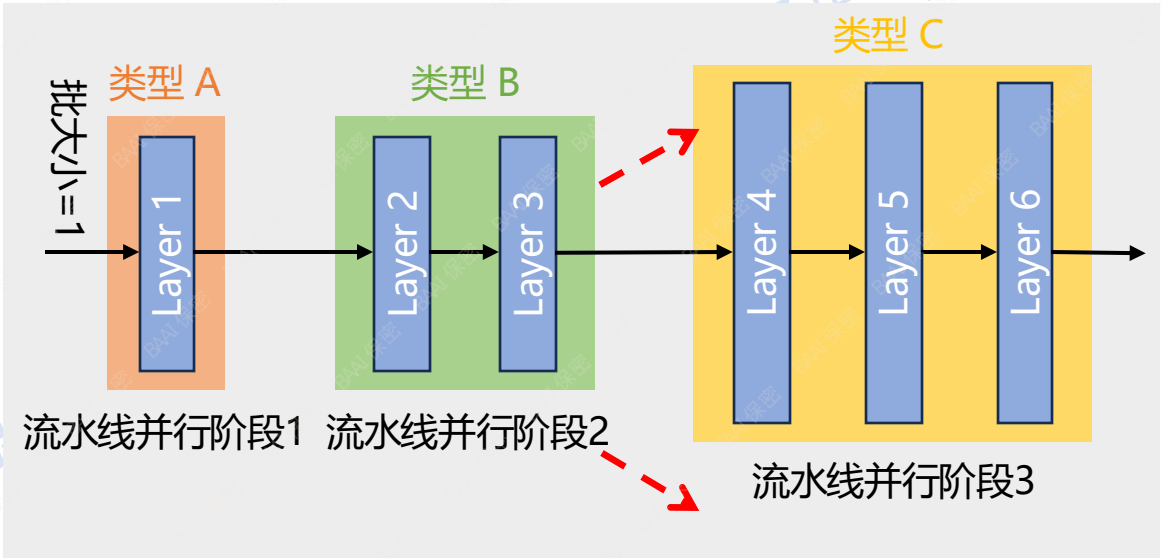
数据并行异构



流水线并行异构

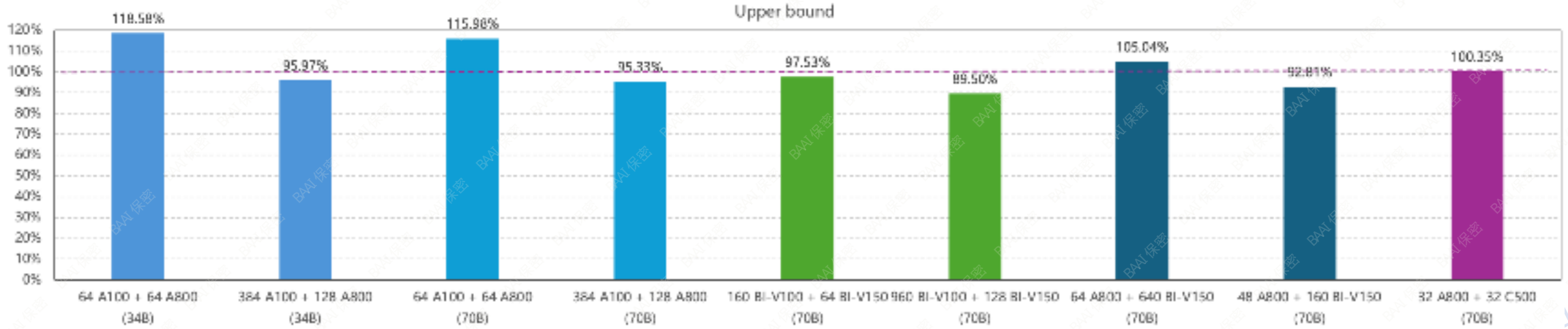


模型并行异构（减小）



模型并行异构（增大）

单一异构混训模型效果

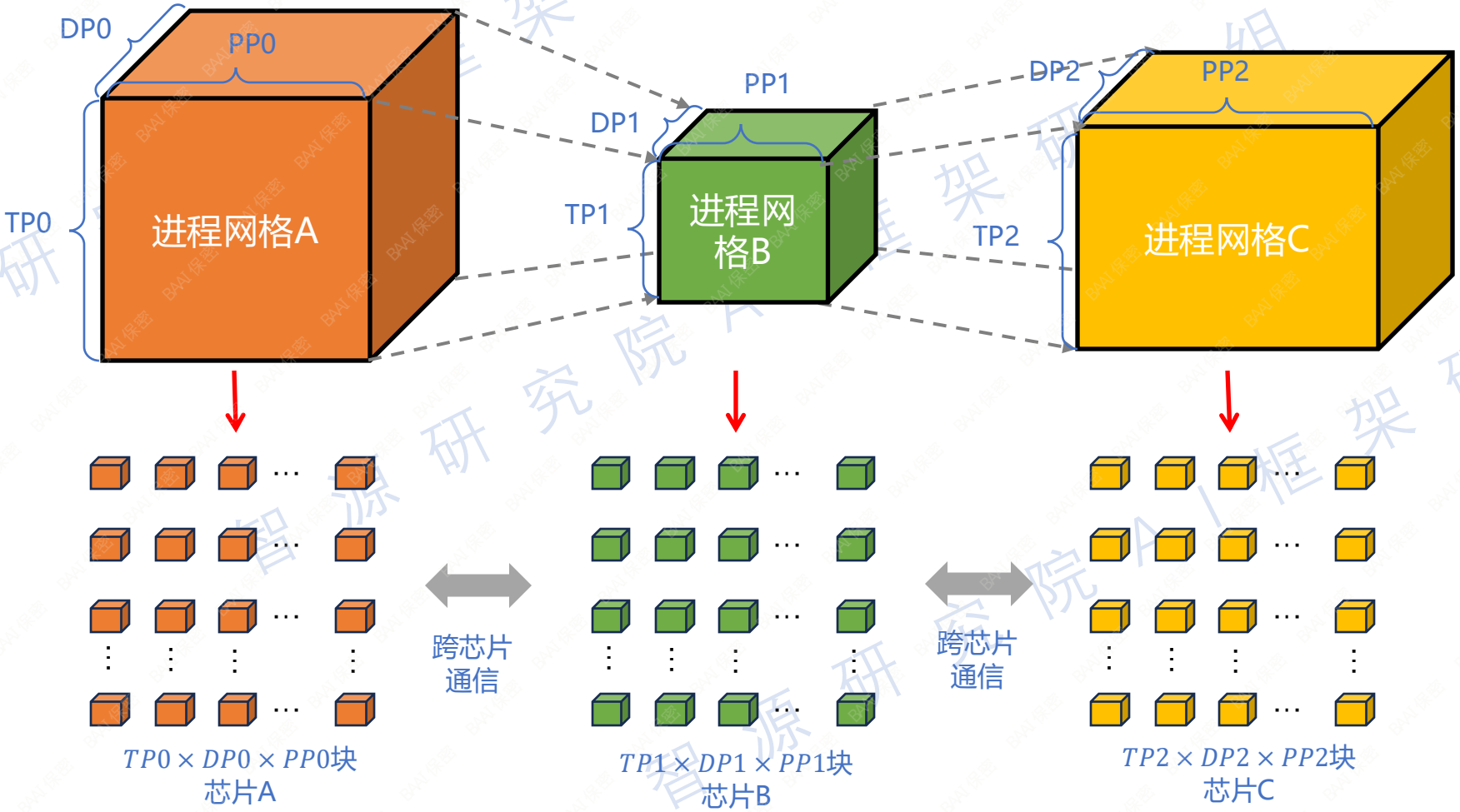


Model	Diff Average	CEVAL (5shot)	MMLU (5shot)	PIQA (0shot)	Hellaswag (10shot)	ARC-C (25shot)	BOOLQ (0shot)
Aquila-34B (A800+A100, Hetero-DP)	-0.74%	-0.25%	-1.25%	0.84%	-0.31%	-3.76%	0.32%
Aquila-34B (A800+A100, Hetero-PP)	-0.66%	0.10%	-0.11%	-0.77%	0.13%	-2.92%	-0.40%
Aquila-70B (A800+A100, Hetero-PP)	0.44%	-0.17%	0.67%	0.22%	-0.06%	1.90%	0.07%
Aquila-70B (BI-V100+BI-V150, Hetero-PP)	-0.62%	-1.81%	0.28%	-0.44%	0.19%	-1.26%	-0.65%
Aquila-70B (BI-V150+NVIDIA, Hetero-PP)	-0.04%	-3.02%	-0.81%	0.14%	0.37%	4.21%	-1.16%
Aquila-70B (C500+NVIDIA, Hetero-PP)	-0.13%	-2.91%	0.22%	0.88%	0.05%	0.60%	0.39%

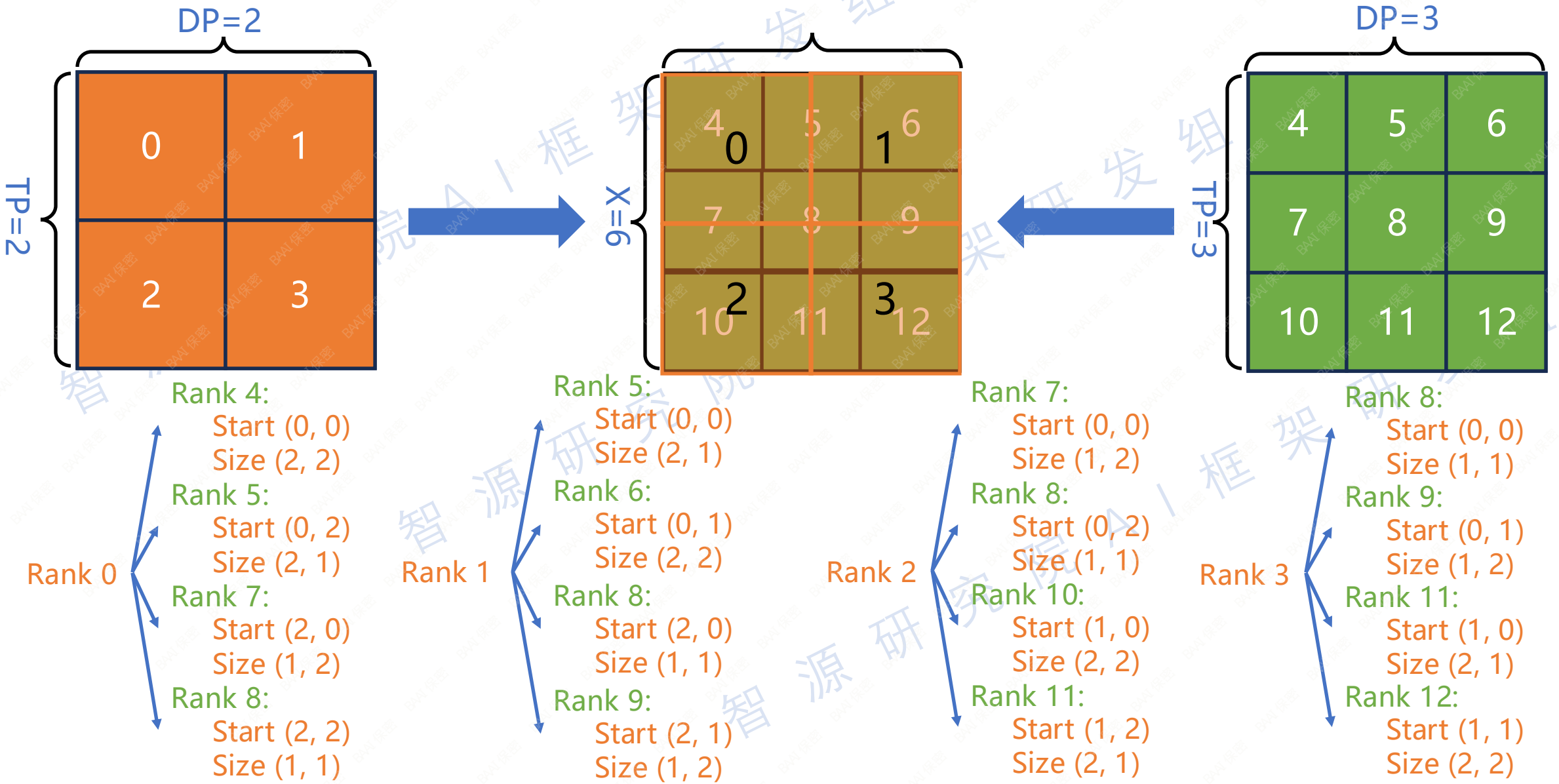
- 在不同模型和配置下，异构混合训练都能获得较高性能水平
- 通过异构合池训练，获得100%+的异构训练性能，比如能够使用更大的micro-batch和避免不必要的重计算
- 在不同模型大小和异构配置下，异构模型效果与同构训练保持一致

通用多维混合异构并行策略

统一各种异构并行策略，支持多款芯片跨芯，实现最灵活的异构混训硬件配比



通信优化

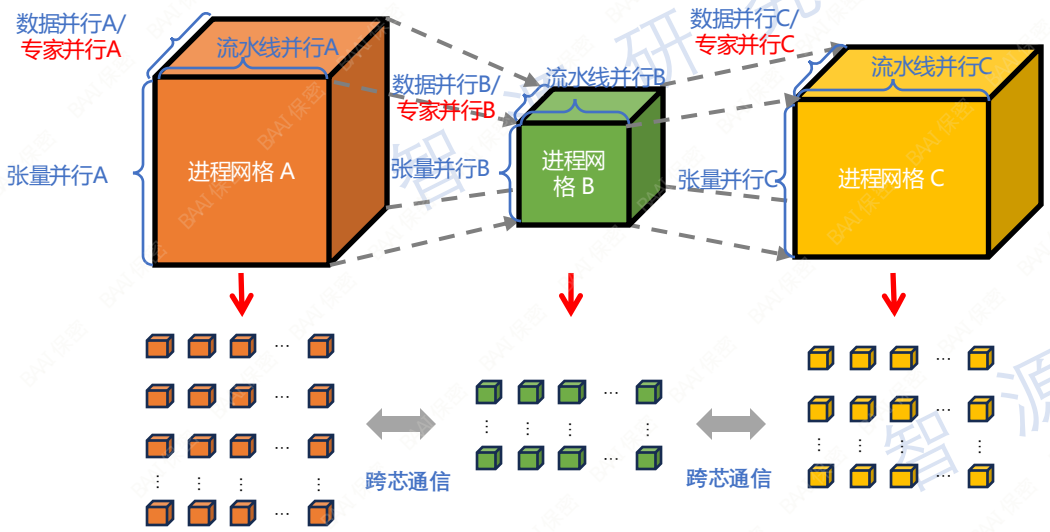

DP=2
TP=2
0
1
2
3
DP=3
TP=3
4
5
6
7
8
9
10
11
12
6=X
Rank 0
Rank 4:
Start (0, 0)
Size (2, 2)
Rank 5:
Start (0, 2)
Size (2, 1)
Rank 7:
Start (2, 0)
Size (1, 2)
Rank 8:
Start (2, 2)
Size (1, 1)
Rank 1
Rank 5:
Start (0, 0)
Size (2, 1)
Rank 6:
Start (0, 1)
Size (2, 2)
Rank 8:
Start (2, 0)
Size (1, 1)
Rank 9:
Start (2, 1)
Size (1, 2)
Rank 2
Rank 7:
Start (0, 0)
Size (1, 2)
Rank 8:
Start (0, 2)
Size (1, 1)
Rank 10:
Start (1, 0)
Size (2, 2)
Rank 11:
Start (1, 2)
Size (2, 1)
Rank 3
Rank 8:
Start (0, 0)
Size (1, 1)
Rank 9:
Start (0, 1)
Size (1, 2)
Rank 11:
Start (1, 0)
Size (2, 1)
Rank 12:
Start (1, 1)
Size (2, 2)

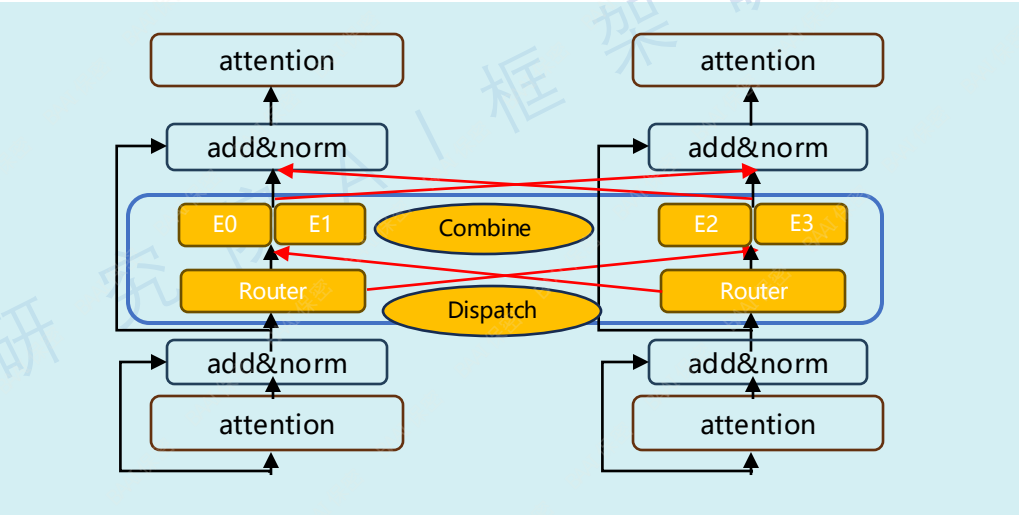
通用异构混合并行全场景

- 全场景的异构混训支持

模型结构	GDR-based Communication	CPU-based Communication
Dense结构	✅	✅
MoE结构	✅	✅

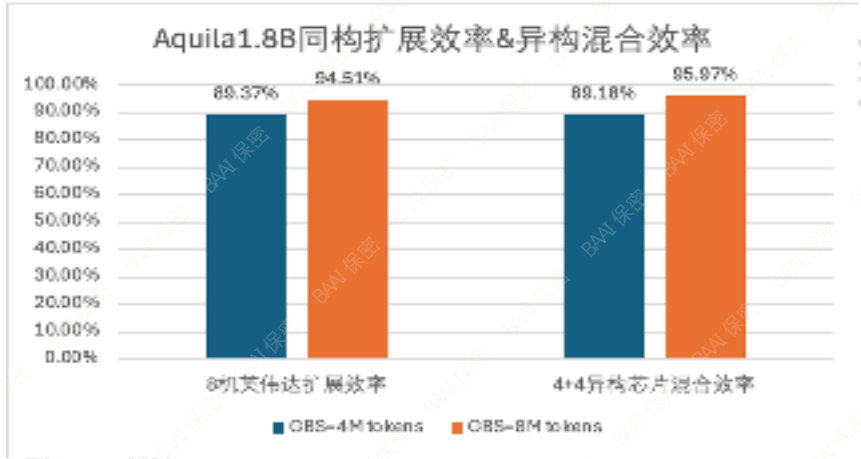
- 支持DeepSeek-V3异构混训
  - 基于进程网格(ProcessMesh)，延续异构并行基本原则“在同一个进程网格内保持EP均匀切分，不同进程网格之间可以设置不同EP切分”，将EP复杂的Dispatch和Combine通信限制在同构集群，异构集群之间仅存在跨Pipeline Stage的相关通信，实现PP、DP、TP、EP多维混合异构并行策略。



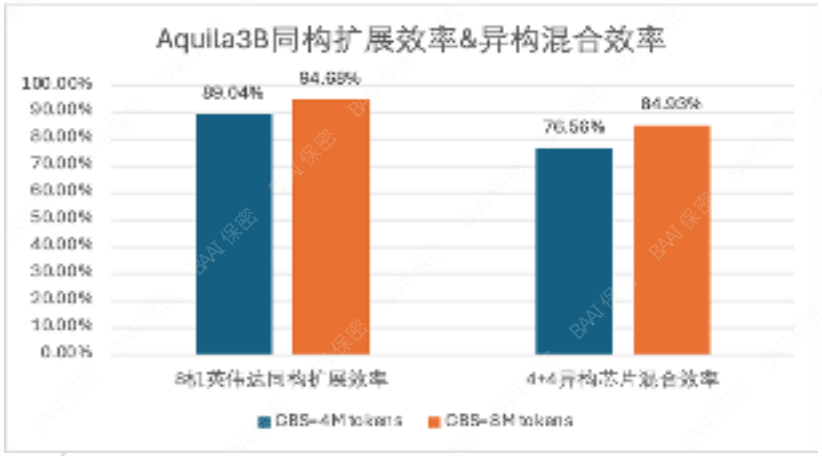


通用异构并行效率

- 异构混训Dense模型性能数据

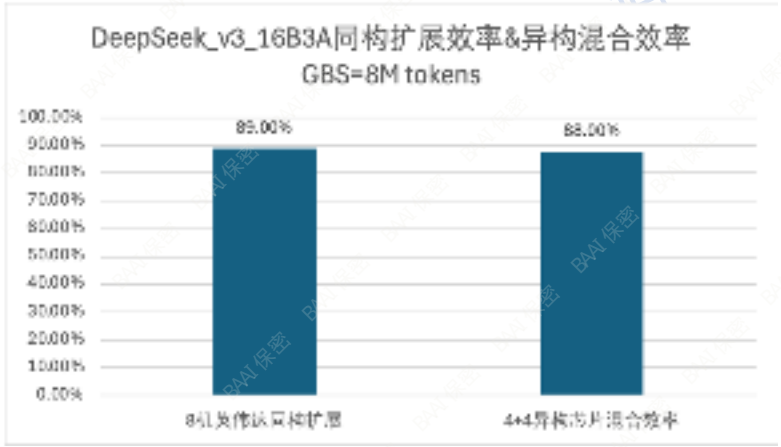


- 非共享Embedding，通信占比低，异构混合效率高
- 异构混合效率最高达96%



- 共享Embedding(593MB)，通信占比长，异构混合效率受限
- 异构混合效率达85%

- 异构混训MoE模型性能数据



DeepSeek_v3结构模型(总参数16B 激活参数3B):

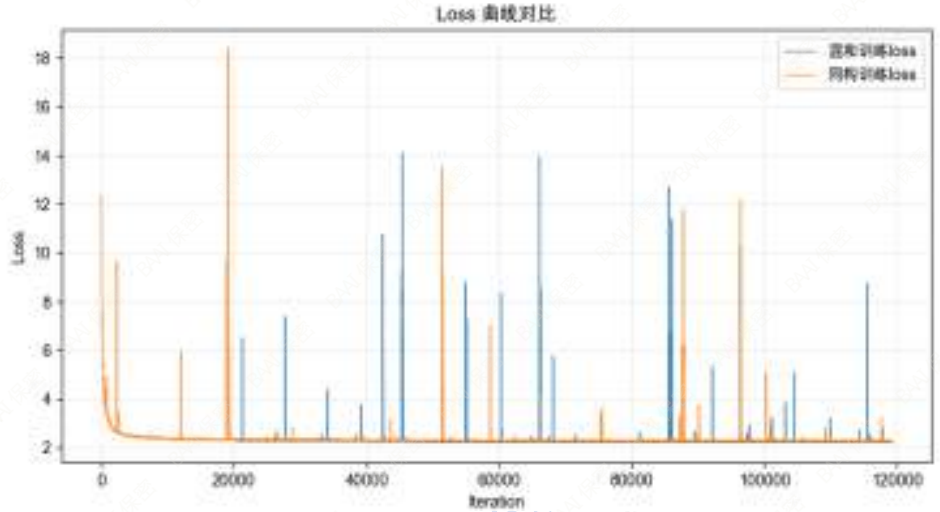
- 集群通信带宽限制异构混合效率
- 异构混合效率达88%，跟英伟达集群扩展效率(89%)基本一致

结论：在异构机器加速卡数量不同、单卡算力不同、显存不同、机内和机间通信带宽均不同的复杂异构场景下，基于通用异构并行策略的混训性能在不同类型的模型上可以达到跟英伟达同构机器基本一致的混合效率

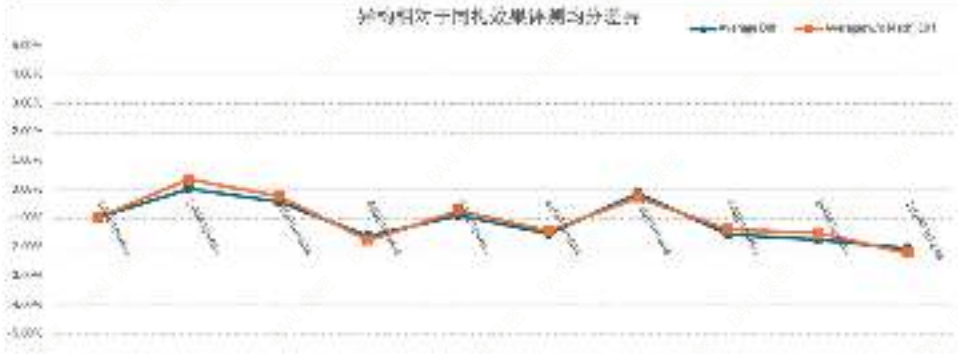
异构混训端到端模型

- 在4台英伟达CUDA+4台天数BI-V150端到端混合预训练Aquila-3B模型
- 共计消耗1TB的高质量文本数据，并且使用8台英伟达CUDA执行相同的预训练过程进行对比

Loss收敛对比：全程监督loss收敛趋势和绝对误差，二者趋势完全一致



评测效果对比：每100B进行效果评测，相对于英伟达同构训练效果差异在-2.05%-+0.04%之间，符合行业误差范围



感 谢 聆 听

欢迎大家使用：

FlagScale: https://github.com/FlagOpen/FlagScale

FlagCX: https://github.com/FlagOpen/FlagCX