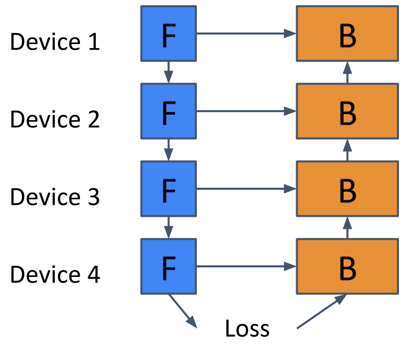
大模型训练流水线并行四部曲：吞吐、内存、线性扩展与负载均衡

主讲人：万信逸

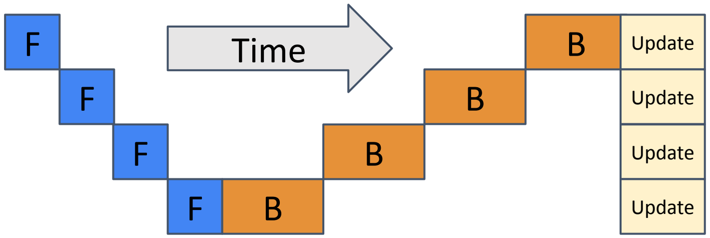
Sea AI Lab

01 流水线并行概述

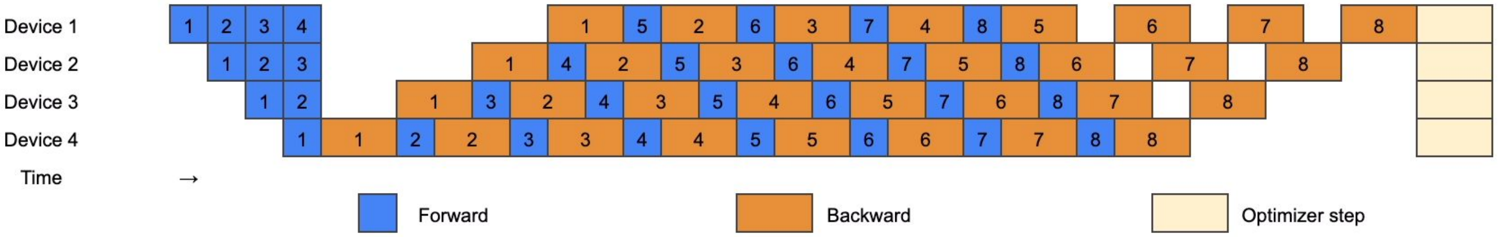
大语言模型中的流水 线并行



按层切分模型



按时序排列



叠加多个microbatch

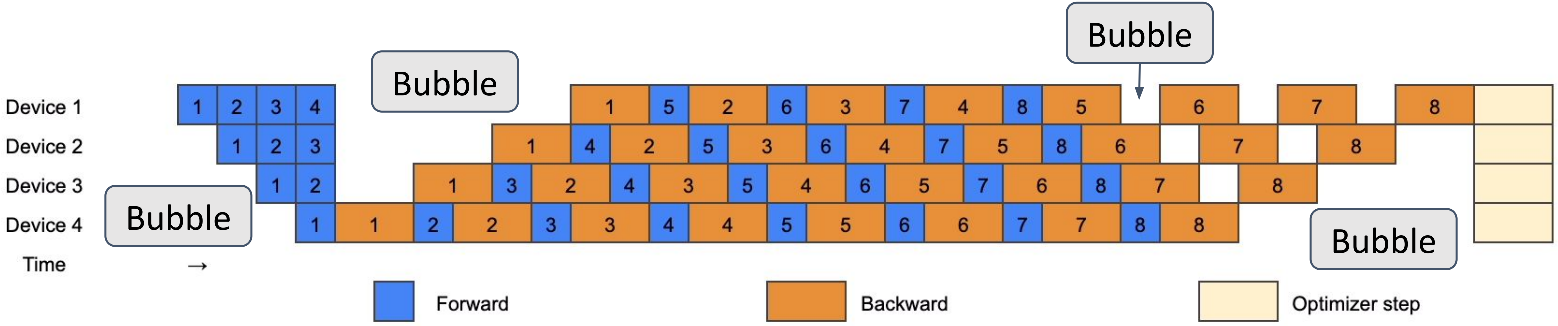
* PipeDream: Generalized pipeline parallelism for DNN training

流水线并行为何重要？

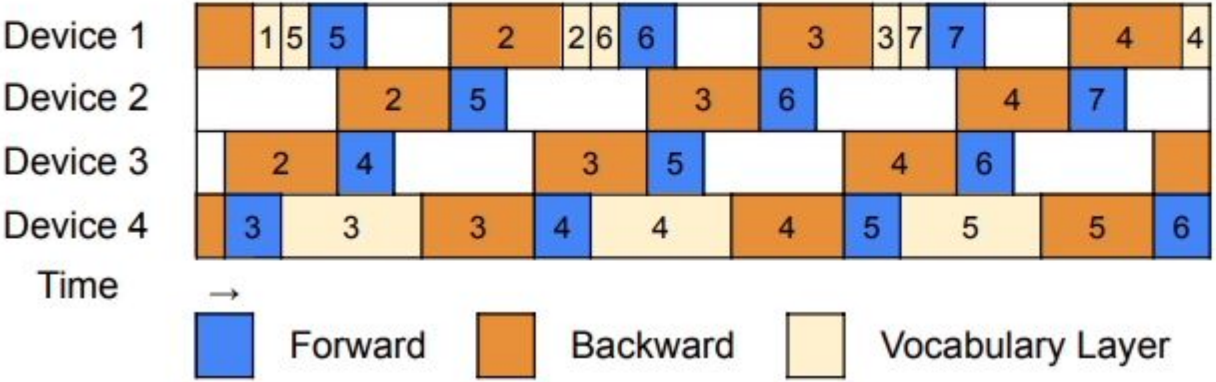
- 流水线并行是跨机并行的首选策略
  - 🤔 Data Parallel (DP) 简单可靠，但是无法支持内存使用超过单卡的模型
  - 🤔 Tensor Parallel (TP) 高通信载量，无法跨节点
  - 🤔 ZeRO / FSDP 参数频繁通信，在节点数量多时性能受限
  - ✅ Pipeline Parallel (PP) 低通信载量，适合多节点训练

流水线并行的缺陷 - Pipeline Bubbles

流水线气泡

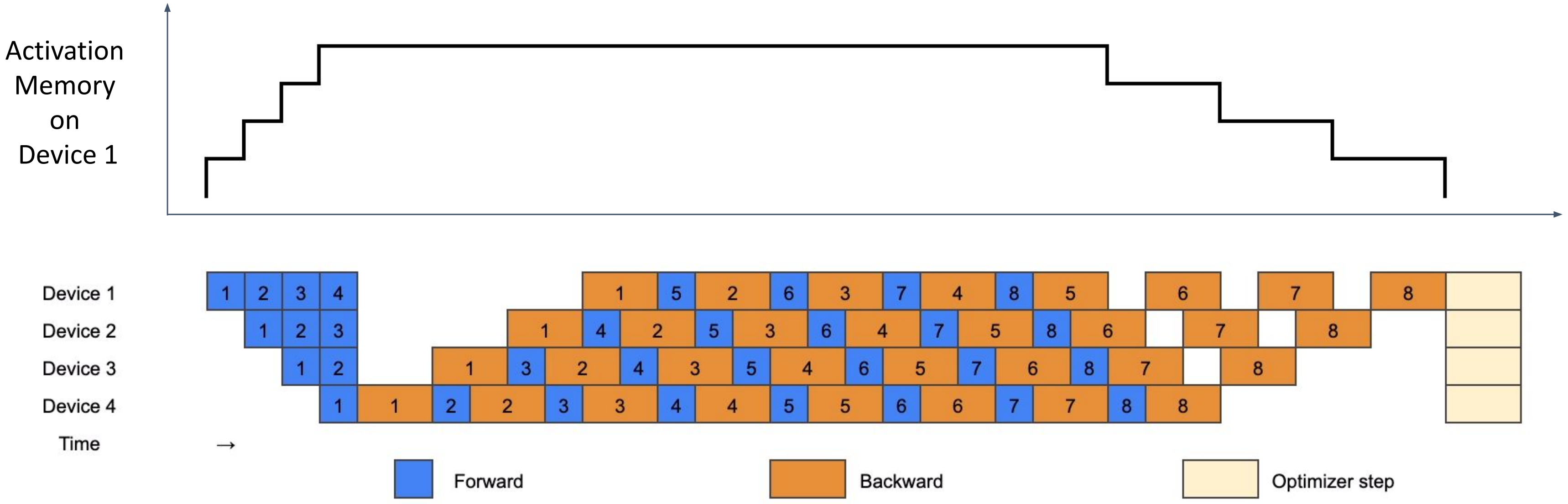


不均衡气泡



流水线并行的缺陷 - Activation Memory

- 每个 F 占用一份Activation Memory，供 B 使用并释放



流水线并行的缺陷 - Activation Memory

GPT Model	GPU 数量	每个GPU的 Parameter Memory (GB)	每个GPU的 Activation Memory (GB)
1.5B	1	24	15
3.4B	2	27	26
7.2B	4	29	41
18B	8	36	68
39B	16	38	110
76B	32	37	171

在纯PP场景下，大模型的Activation Memory随模型增大爆炸式增长

* sequence length=4096; micro bs=2

* 每个 NVIDIA A100/H100 GPU 配备有80GB 内存

我们的主要贡献

Zero Bubble Pipeline Parallelism
ICLR’ 24
Extreme Throughput

✅ 消除流水线气泡

PP with Controllable Memory
NeurIPS’24
Minimum Memory

✅ 最低内存使用

Vocabulary Parallelism
MLSys’ 25
Balanced Workload

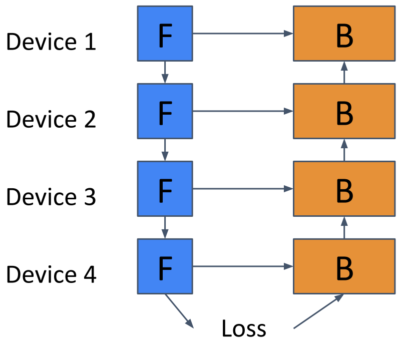
✅ 消除不均衡气泡

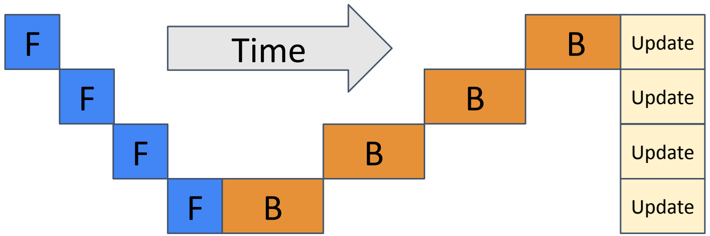
PipeOffload
ICML’ 25
Scalability

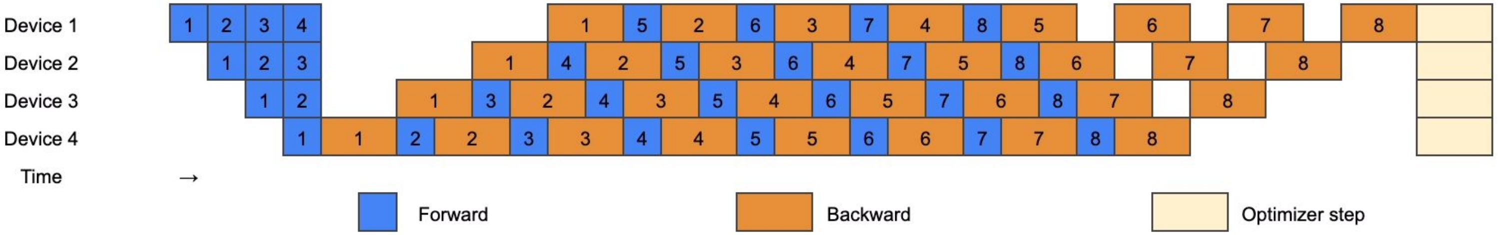
✅ 线形扩展

01 Zero Bubble Pipeline Parallelism

对1F1B的细节观察







一般Backward计算是Forward计算的两倍，why?

* PipeDream: Generalized pipeline parallelism for DNN training

B-W 分离

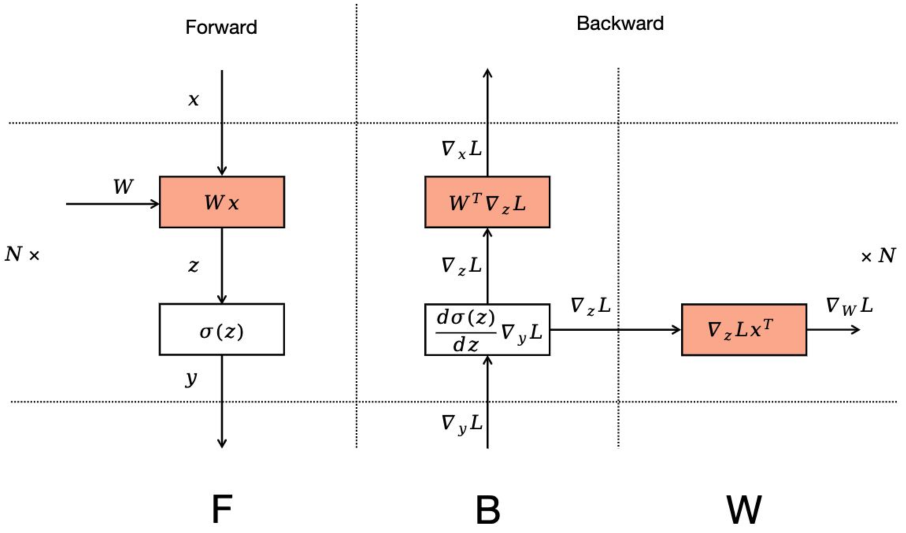


Figure 1: Computation Graph for MLP.

- 以MLP为例，此处三个矩阵乘法计算量相同
- 在一般实现中，B与W被绑定在同一个.backward()函数调用中

B-W 分离

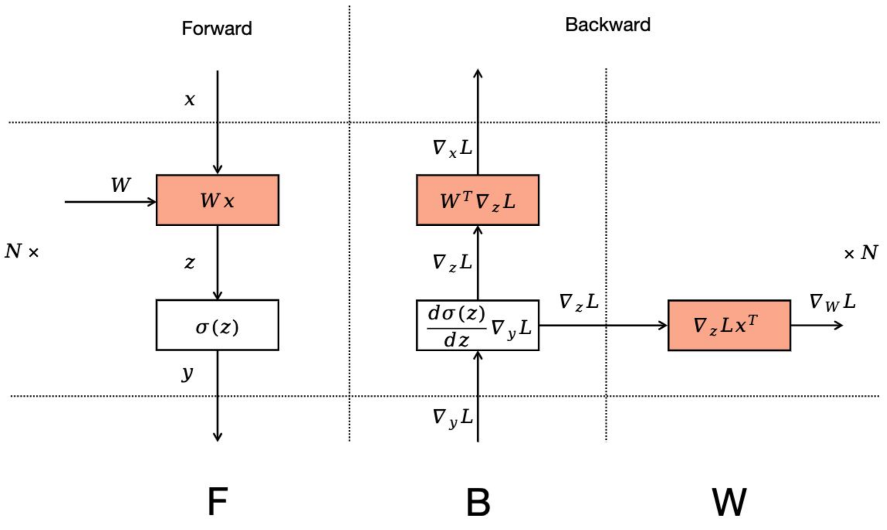
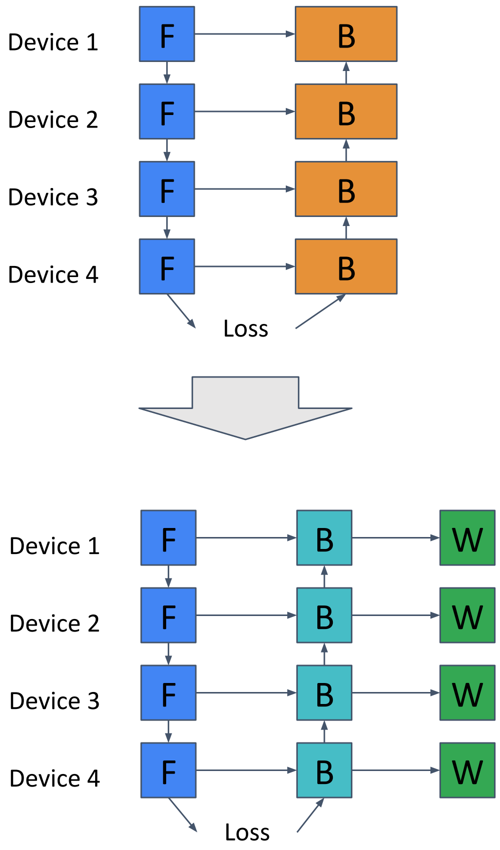


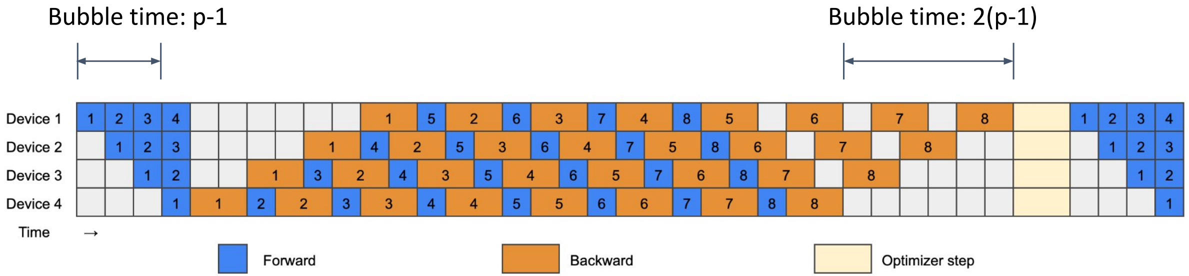
Figure 1: Computation Graph for MLP.



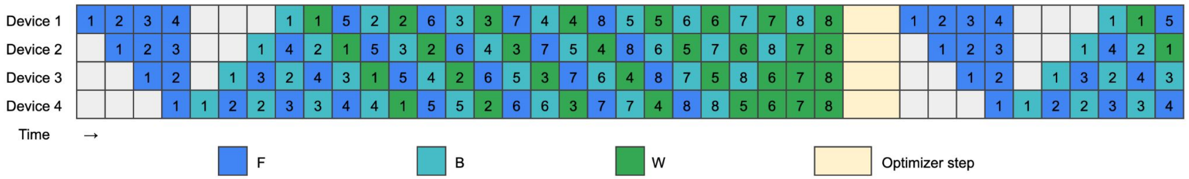
- 其他设备仅依赖B的输出，不依赖W
- B和W可以单独调度

手动调度策略 - ZB-H1

1F1B

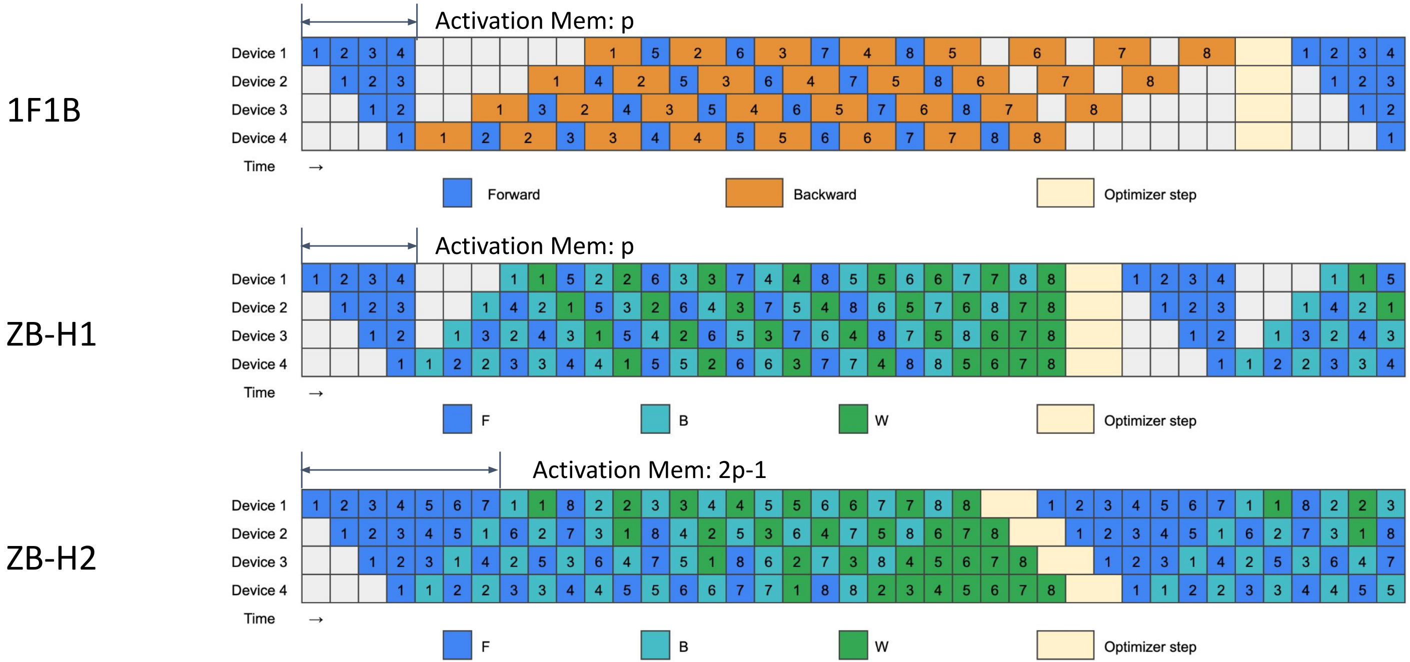


ZB-H1



流水线气泡 3(p-1) -> (p-1)
保持与1F1B一样的最高内存占用！

手动调度策略 - ZB-H2



完全消除流水线气泡!

自动调度策略

- 为何我们需要自动调度策略？
  - F, B, W时间不一致，且与模型/序列长度相关
  - 需要考虑通信时间
  - 气泡率往往与内存使用直接相关，用户需要寻求气泡率与内存之间的平衡
- 基于贪心的自动调度算法
  - 使用真实F/B/W时间
  - 可配置内存限制

自动调度的一个实例
F/B/W/C=10.2/12.9/8/0.5; Memlimit = 2p
Result bubble rate=1.98%

实验评估

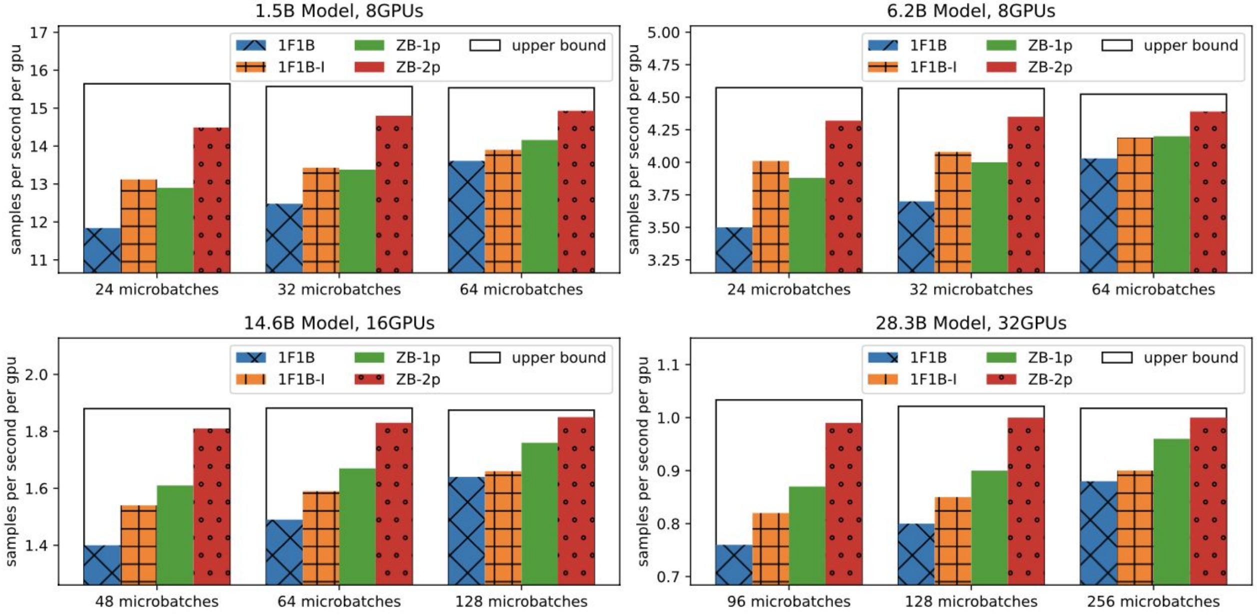


Figure 5: Comparison of throughput across different pipeline schedules.

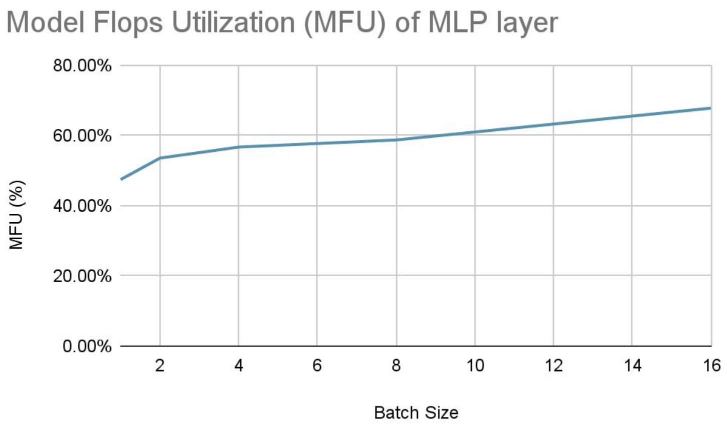
最多加速LLM训练30%

02

Pipeline Parallelism with Controllable Memory

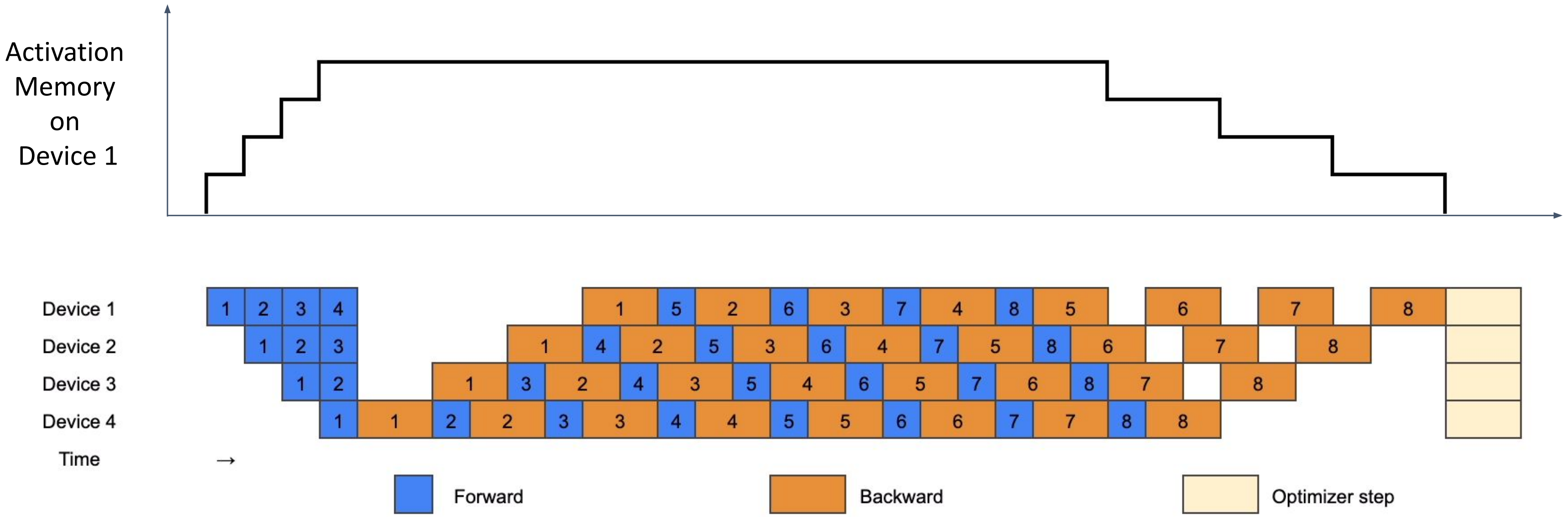
流水线并行的内存 问题

- 消除了流水线并行的气泡之后，我们问了自己几个问题
  - ZB-H2需要两倍activation memory，是否可以避免？
  - 是否有可能有更低的activation memory使用？
  - 在大模型训练中，省内存=提升吞吐，因为省下的内存可以
    - 提升batch size以提升计算密度
    - 减少使用如TP这样的内存低，但是通信载量高的策略



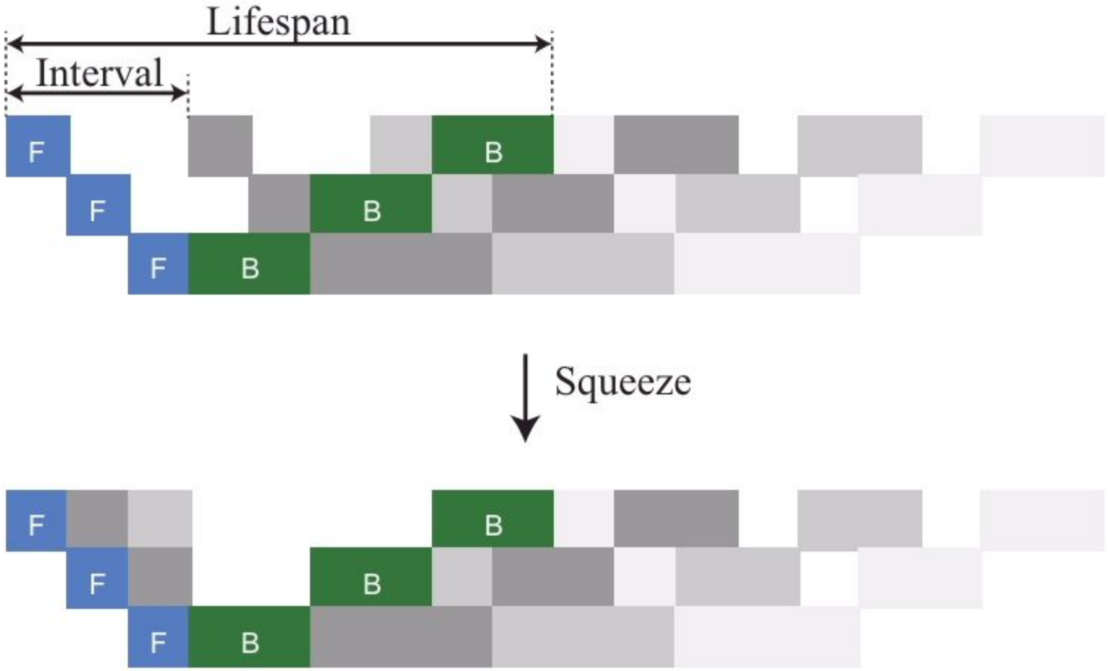
流水线并行的内存 问题

- 在PP中，不同的microbatch的activation memory互相重叠
- 什么决定了activation memory的峰值？



核心Insight: 生存周期(lifespan)决定峰值内存

把流水线并行看作重复若干个基本构造单元(Building Block)



用 l 代表生存周期，T 代表重复间隔，m 代表每个microbatch的内存占用

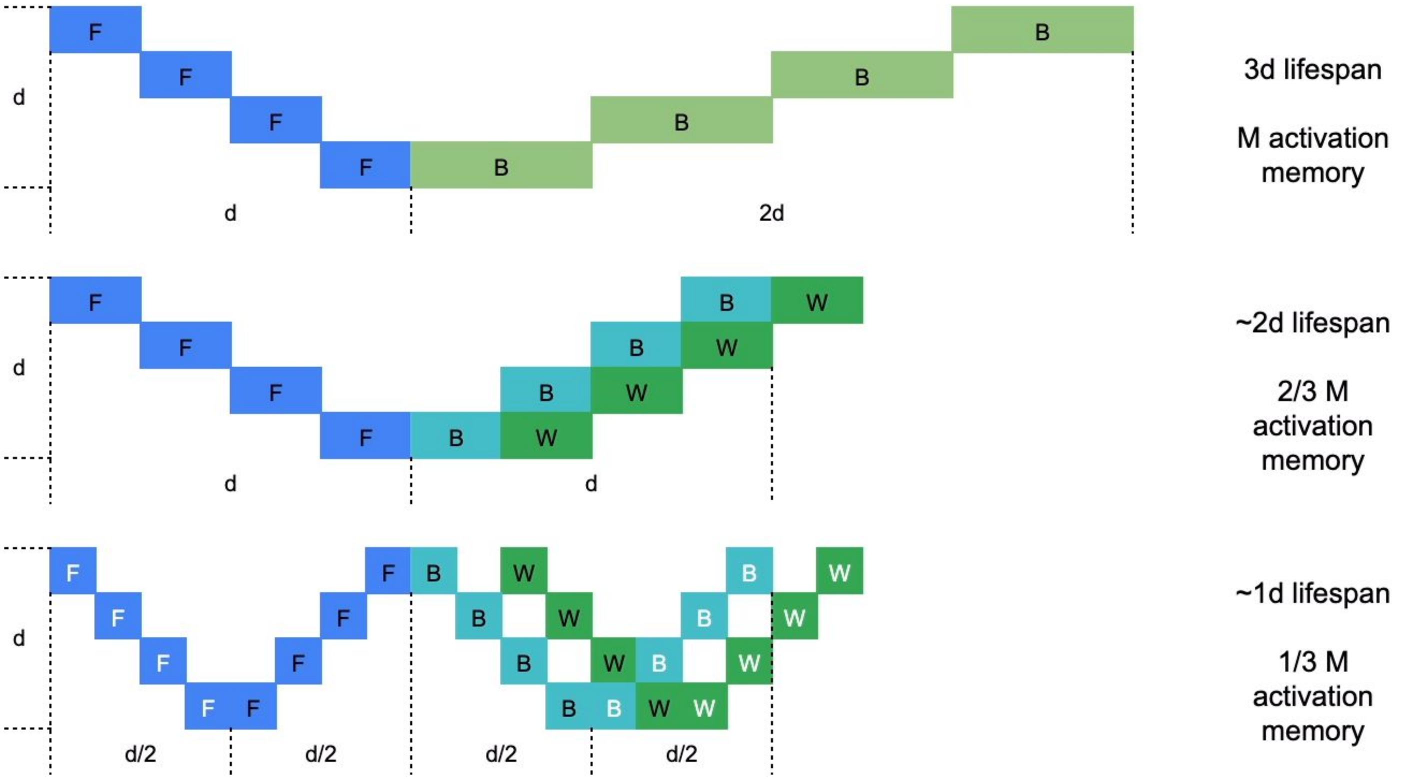
那么

$$\text{peak memory} \le \lceil \frac{l}{T} \rceil m$$

使用V字形构造单元来减少lifespan

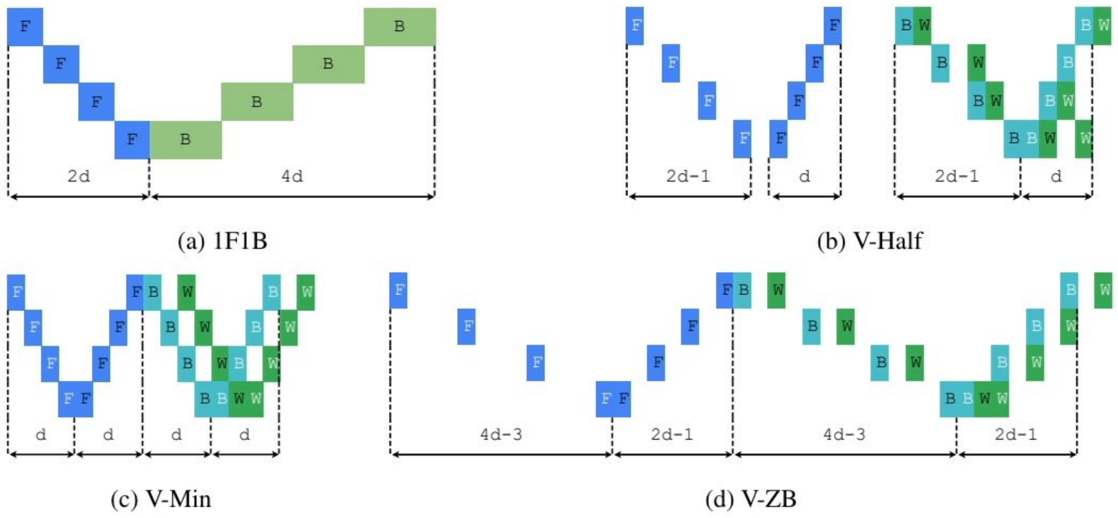
使用B-W分离来缩短关键路径

将模型分割成两个分片，并以V字形状排布

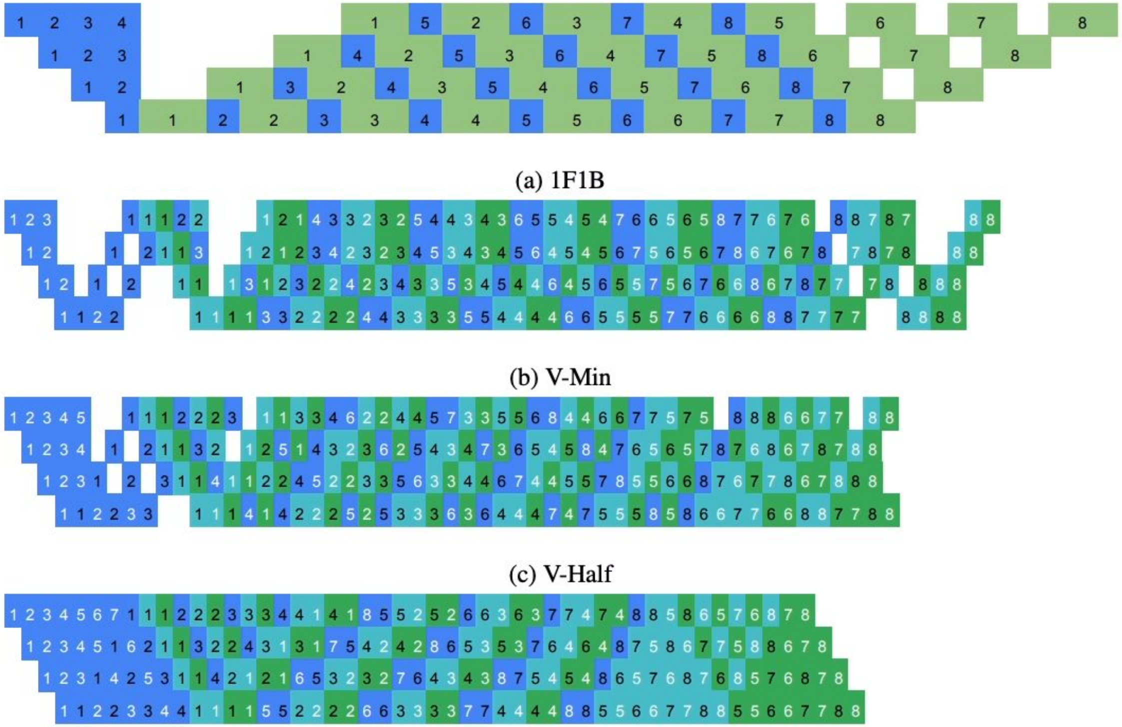


通过控制生存周期控制内存使用

- 在实际场景中，我们可以通过控制V字型构建单元的生存周期来调配内存使用与气泡率



最终调度策略



Memory: M

(a) 1F1B

Memory: M/3

(b) V-Min

Memory: M/2

(c) V-Half

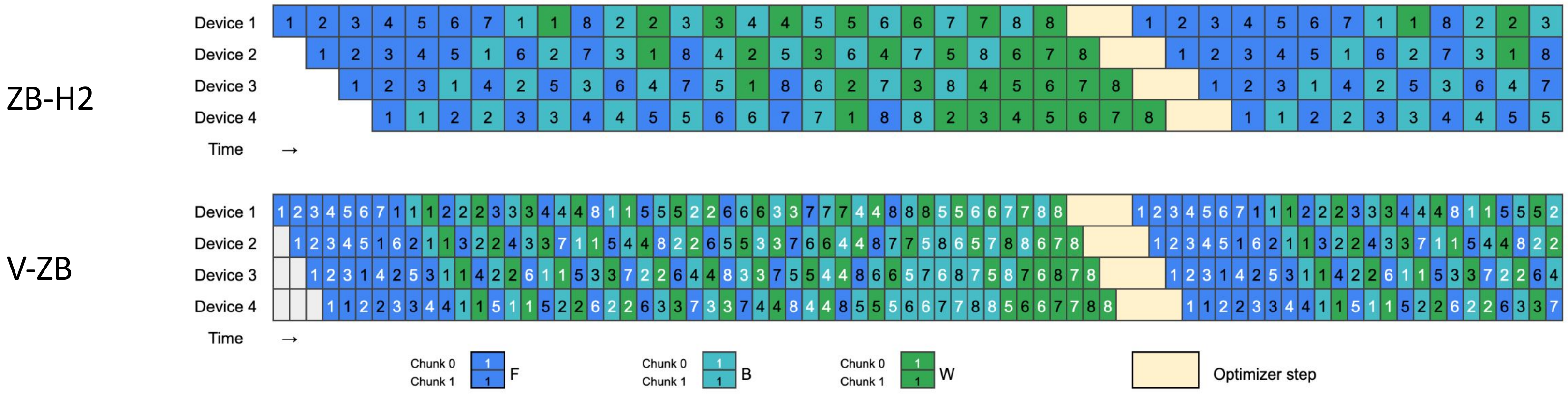
Memory: M

(d) V-ZB

基于V字型的调度策略不仅比基准有更高吞吐，同时也有更低的内存使用

Zero Bubble策略

- 对比V-ZB与ZB-H2，两种方法均zero bubble
- 但V-ZB仅需ZB-H2的一半内存，与基准线一致



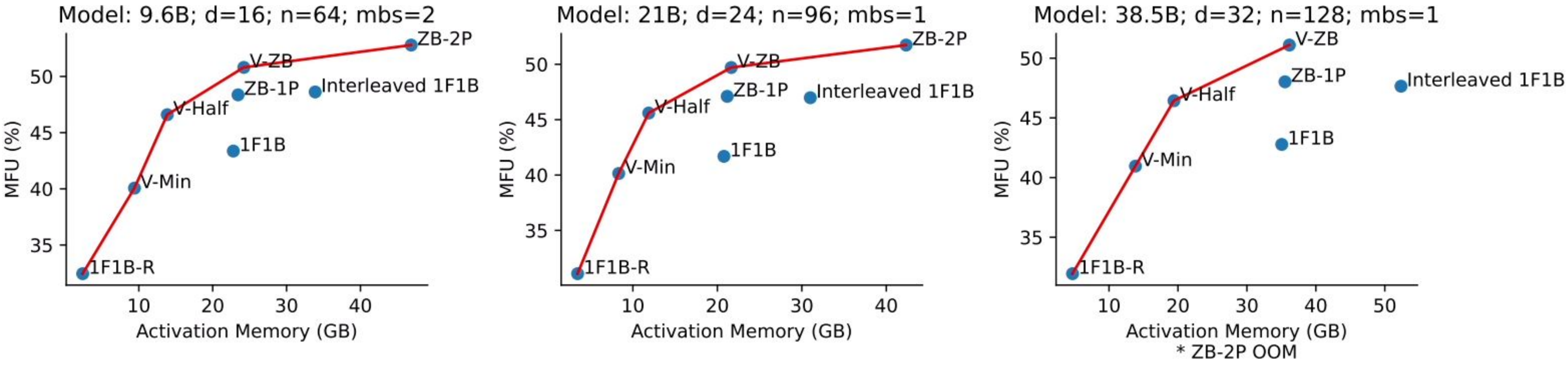
策略比较

流水线策略	气泡率	峰值内存	特性
1F1B	$\frac{p-1}{p+n-1}$ = R	M	基准线方法，被广泛应用
Interleaved 1F1B	R/v	M*(1+1/v)	基准线方法，被广泛应用
ZB-H1	R/3	M	降低气泡率
ZB-H2	0	2M	Zero Bubble
V-Min	2R/3	M/3	可证明的最小内存*
V-Half	R/2	M/2	平衡内存与气泡率
V-ZB	0	M	内存友好的Zero Bubble

* 限定纯PP，无offload/recompute
* 符号：
- p: number of devices
- n: number of microbatches

M: activation memory of the entire model
v: number of model chunks on each devices for interleaved 1F1B

推进吞吐量与内存的 Pareto Frontier



一些思考

- 既然已经达到了极致吞吐和最小内存，流水线并行应该已经足够好了
- 既然TP的通信载量那么大（一般认为TP8有20-30%的overhead），为何社区里还是优先使用TP而不是PP？
- 为何社区里不使用纯PP（或者PP+DP）？比如，为何不能用64 PP训练一个有64个layer的模型？
  - PP的activation memory仍然是很大的问题 - TP8的单机activation只有⅛，但PP8就算用v-min也有~⅓
  - Vocabulary layer导致的负载/内存不均衡会成为非常大的问题
- 我们需要解决这两个问题

03

PipeOffload

流水线并行的可扩展性

- 在分布式训练的语境下，可扩展性意味着
  - ①扩展数据 - 能否通过使用更多的GPU训练更多的数据？
  - ②扩展内存 - 能否通过使用更多的GPU训练更大的模型？

- 考虑几种典型策略

- DP & CP : ①OK, ②无法满足
- TP : 不管是①还是②, 都不能超过8个GPU
- PP : ①OK, ②受制于Activation Memory

流水线并行的Activation Memory困境

$$\text{peak memory} \leq \lceil \frac{l}{T} \rceil m$$

- 当使用更多设备时，生存周期 (l)不变，T/m同比变小，因此内存不变

- 能否让PP的Activation Memory 也变得可扩展？---- 借助CPU offloading 缩短生存周期

Activation Offloading (到CPU内存)

PP的Activation问题来自于它每份activation都有很长的生命周期，但长生命周期也意味着中间有相当长的时间做Offloading

Offloading的最大问题 - CPU-GPU 通信是否足够快？

Offloading Bound

如果我们定义

$$k = \frac{T_o}{T_c} = \frac{10}{3(6h+s)} * \frac{B_c}{B_o}$$

其中To/Tc分别是一个 transformer layer的Offload时间与计算时间，Bo/Bc是GPU的带宽/计算吞吐

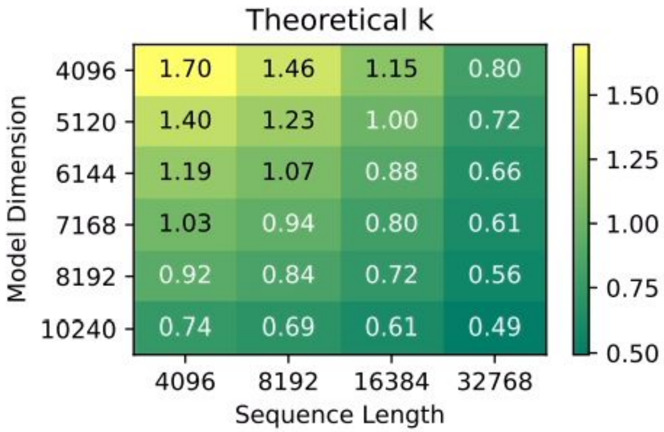
那么k是一个标志我们能够Offload多少内存的关键指标

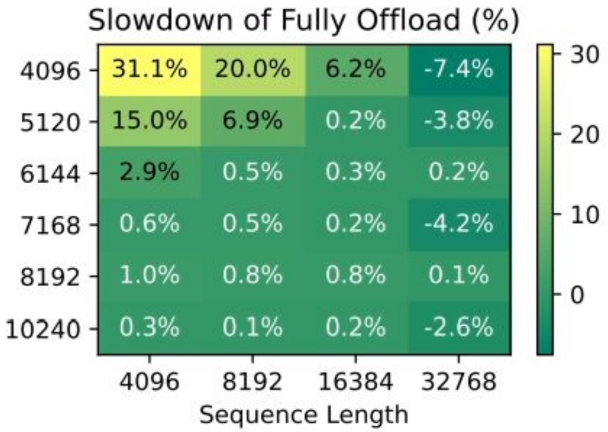
而且，k反比于h (model dimension)和s (sequence length)

关键结论：模型越大，序列越长，越容易offload

k到底有多小？

理论k值
B_c= 220T/s
B_o=15G/s



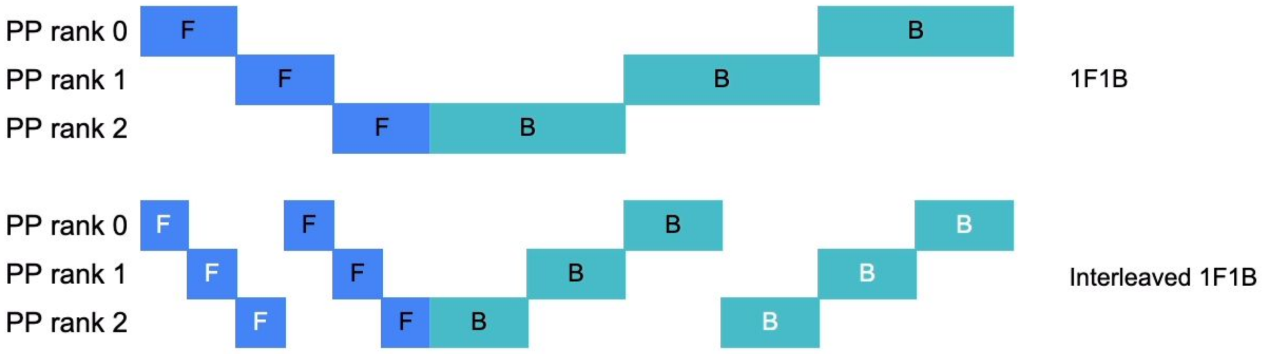


实验验证

- 对于中等规模(h>=8k)或者长序列(s>=32k)模型，k<1
  - 带宽支持offload全部内存
- 对于所有情况，k<2
  - 带宽支持offload一半内存
- Offload overlap计算带来的性能损失可以忽略不计(<1%)

选择性Offload

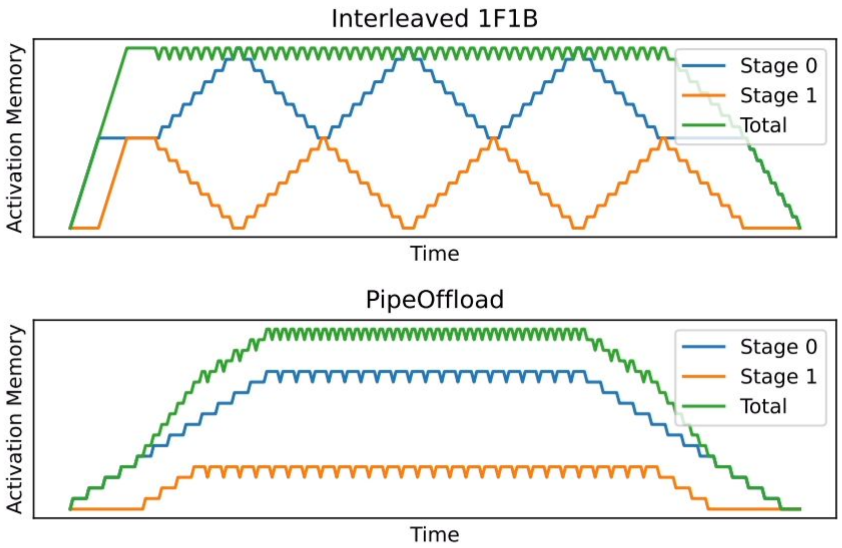
当k>1时，我们没有足够带宽offload所有内存，因此我们需要选择性offload一半内存

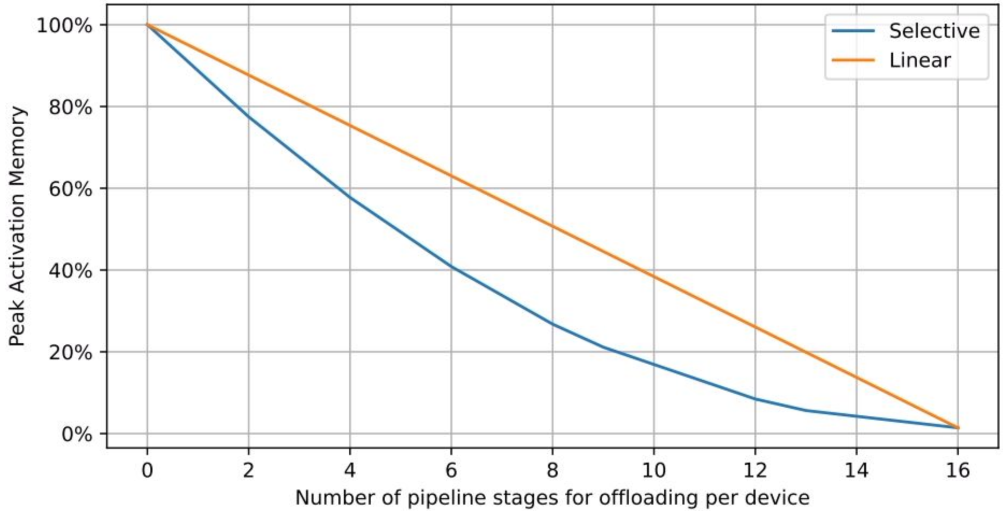


在社区最常用的Interleave 1F1B这种流水线策略中，一个GPU上有多个模型分片，因此我们可以选择性offload一半的模型分片

- 多个模型分片的生存周期不一致，选择性offload可以获得超线性收益

选择性Offload





选择性Offload可以获得超线性收益。Offload一半的模型分片可以降低内存至约1/4

流水线并行的线性扩展

方法	每个GPU上的Activation Memory
DP	M
TP	M / D
PP (1F1B)	M
PP (V-Min)	~M/3
PipeOffload (Selective)	~(1+2/v) * M / 8
PipeOffload (Full)	O(M / VD)

M: activation memory of the entire model
D: number of GPUs
V: number of model chunks on a GPU

流水线并行的线性扩展

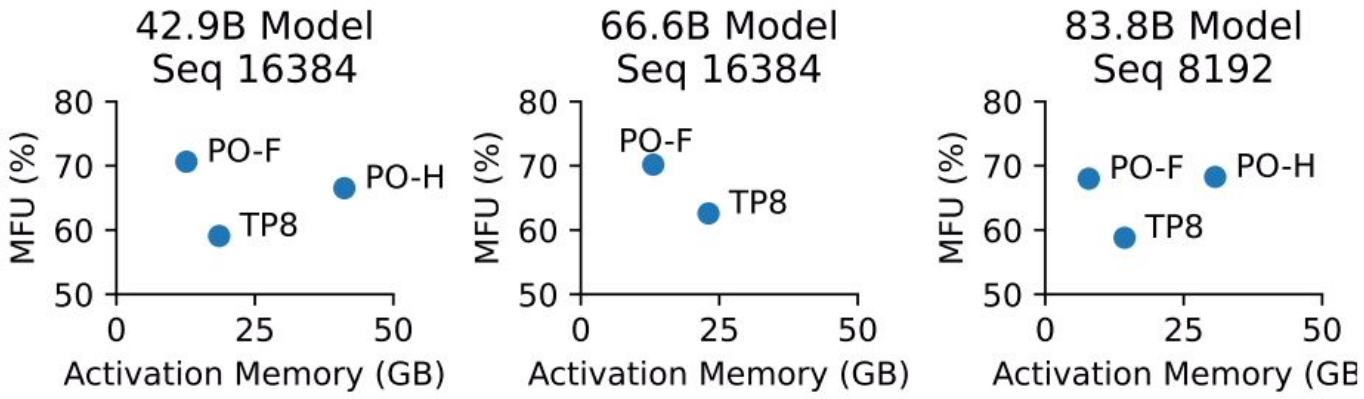
方法	每个GPU上的Activation Memory
DP	M
TP	M / D
PP (1F1B)	M
PP (V-Min)	~M/3
PipeOffload (Selective)	~(1+2/v) * M / 8
PipeOffload (Full)	O(M / VD)

↖️

M: activation memory of the entire model
D: number of GPUs
V: number of model chunks on a GPU

PP的内存甚至可以比TP更低!

与TP的对比

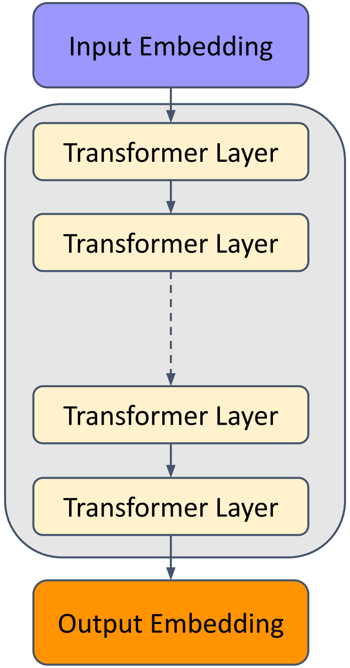


和TP相比，全量Offload (PO-F)比TP又快又省内存，选择性Offload (PO-H)则提供了一个有竞争力的TP替代方案。

04

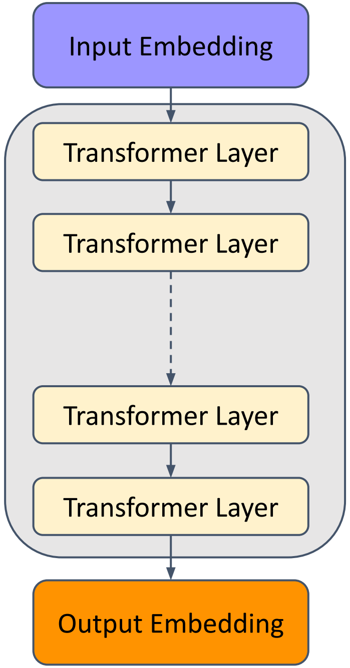
Vocabulary Parallelism

Motivation



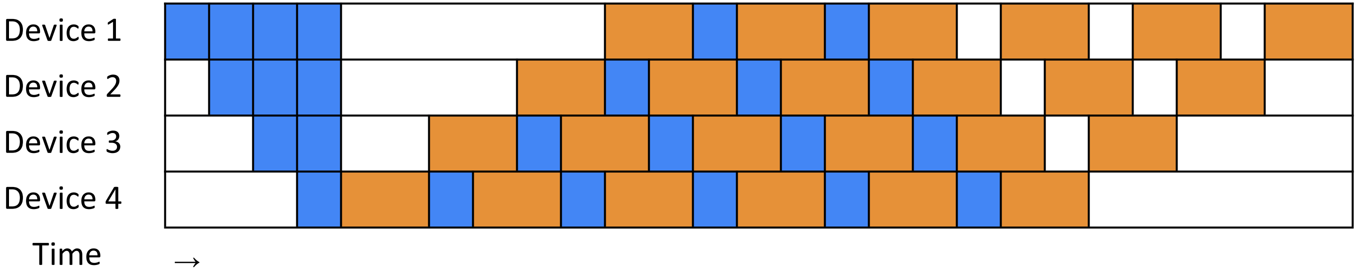
LLM架构

Motivation

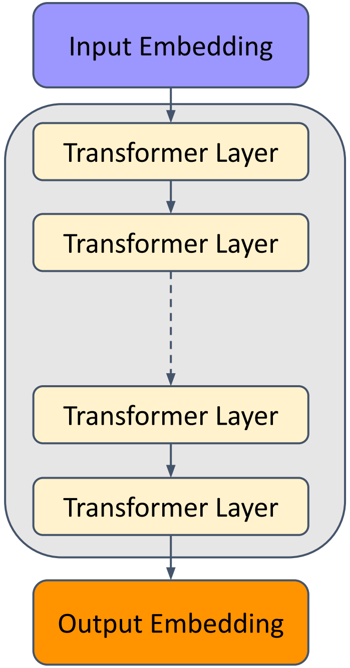


LLM架构

理想中的流水线并行

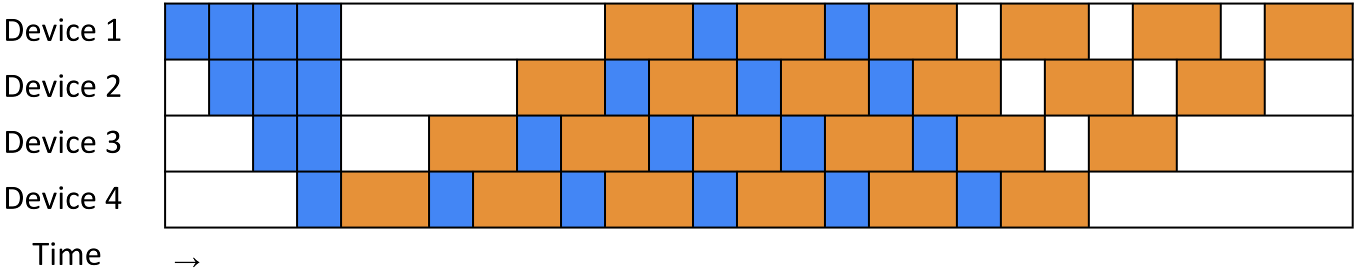


Motivation

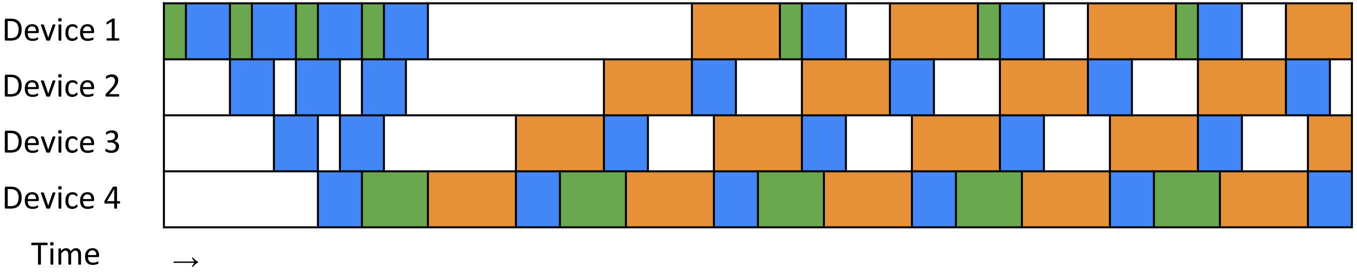


LLM架构

理想中的流水线并行



有词表的流水线并行



Motivation

- 随着多语言LLM的普及，词表大小进一步增大

Model	Activated Parameter Per Transformer Layer (M)	Hidden Size	Vocabulary Size	Ratio of activated parameters between output and transformer layer
Gemma2-9B	198	3584	256000	4.63
Gemma2-27B	566	4608	256000	2.08
Deepseek-v3-671B	607	7168	129280	1.53
Qwen2-7B	231	3584	152064	2.36
Qwen2-72B	884	8192	152064	1.41
Llama3-7B	234	4096	128000	2.24
Llama3-70B	862	8192	128000	1.22
Llama3-405B	3198	16384	128000	0.66
Mixtral-8x22B	355	6144	32768	0.57
Mixtral-8x7B	212	4096	32768	0.63

Motivation

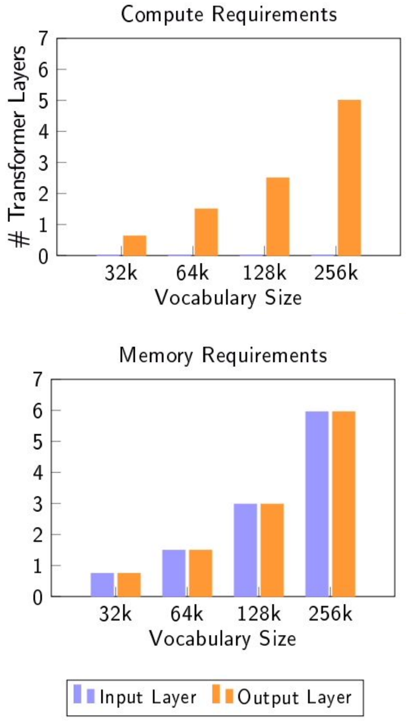
- 随着多语言LLM的普及，词表大小进一步增大

Model	Activated Parameter Per Transformer Layer (M)	Hidden Size	Vocabulary Size	Ratio of activated parameters between output and transformer layer
Gemma2-9B	198	3584	256000	4.63
Gemma2-27B	566	4608	256000	2.08
Deepseek-v3-671B	607	7168	129280	1.53
Qwen2-7B	231	3584	152064	2.36
Qwen2-72B	884	8192	152064	1.41
Llama3-7B	234	4096	128000	2.24
Llama3-70B	862	8192	128000	1.22
Llama3-405B	3198	16384	128000	0.66
Mixtral-8x22B	355	6144	32768	0.57
Mixtral-8x7B	212	4096	32768	0.63

Motivation

- 随着多语言LLM的普及，词表大小进一步增大

Model	Activated Parameter Per Transformer Layer (M)	Hidden Size	Vocabulary Size	Ratio of activated parameters between output and transformer layer
Gemma2-9B	198	3584	256000	4.63
Gemma2-27B	566	4608	256000	2.08
Deepseek-v3-671B	607	7168	129280	1.53
Qwen2-7B	231	3584	152064	2.36
Qwen2-72B	884	8192	152064	1.41
Llama3-7B	234	4096	128000	2.24
Llama3-70B	862	8192	128000	1.22
Llama3-405B	3198	16384	128000	0.66
Mixtral-8x22B	355	6144	32768	0.57
Mixtral-8x7B	212	4096	32768	0.63

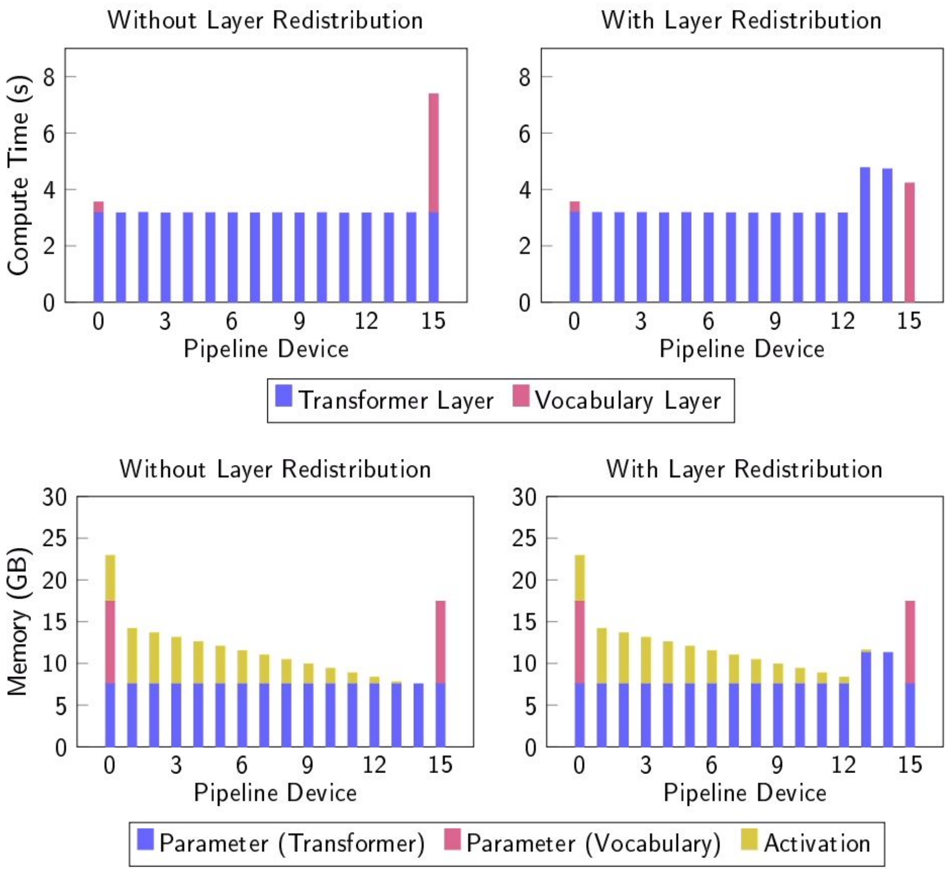


Ratio for Gemma2-9B

基准线方法

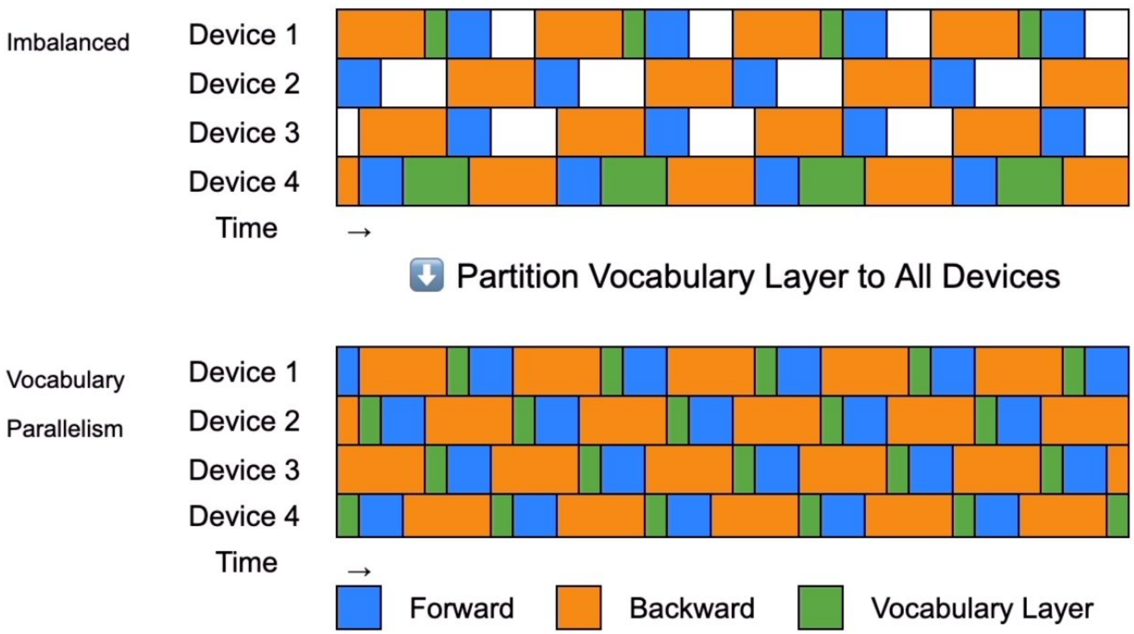
- 我们能否通过重排其他layer来达到负载均衡？

- 可以，但是效果有限



词表并行

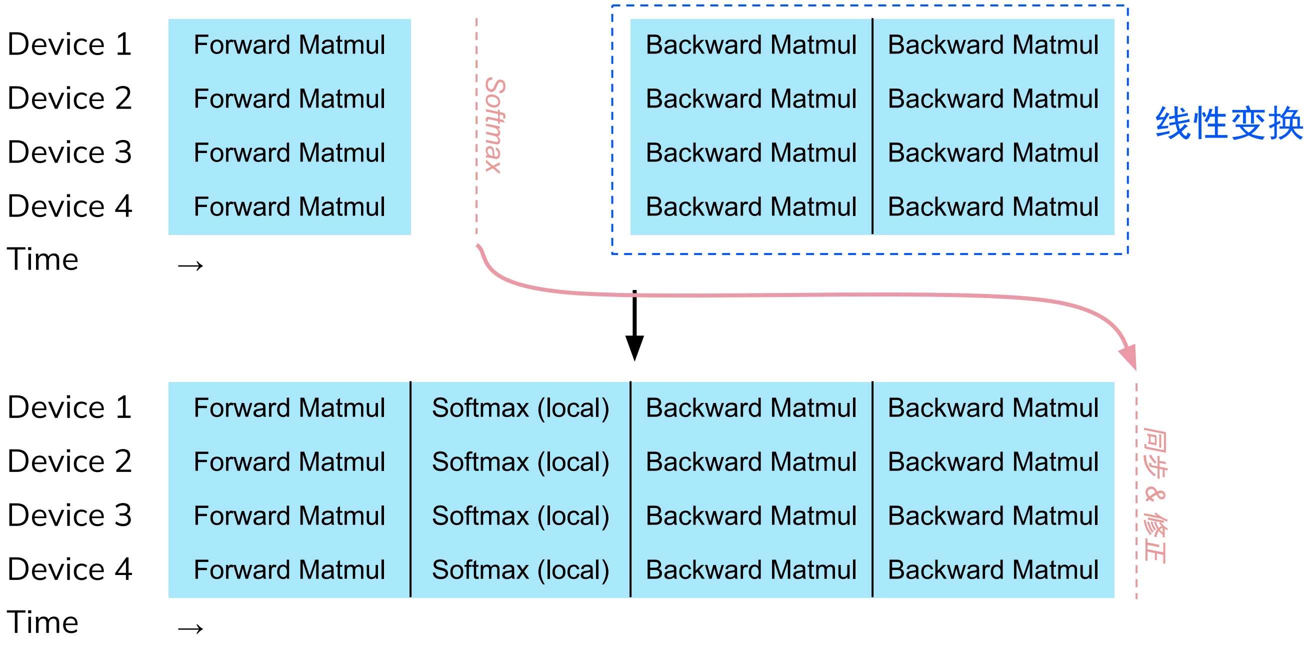
- 我们的目标是将词表分配至不同流水线设备，以实现负载均衡
- 为何这个问题non-trivial?
- 词表计算中有softmax，如果分配至不同设备，会引入同步



Softmax优化

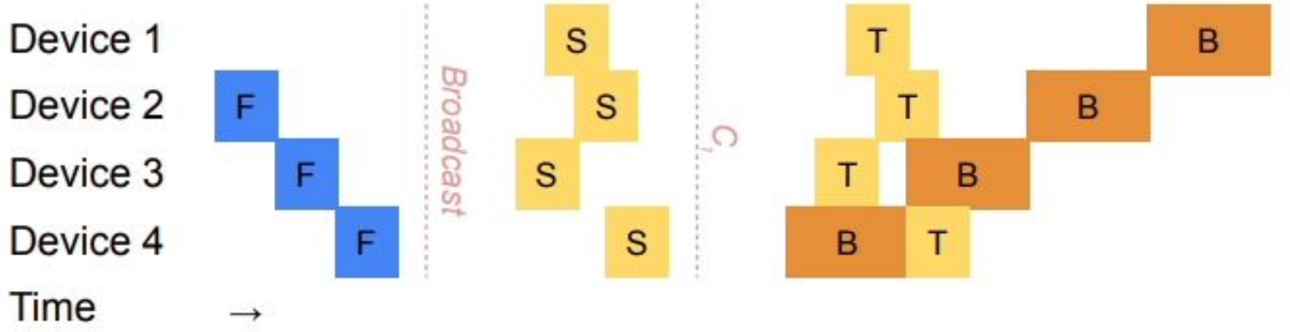
Online-softmax (Milakov & Gimelshein, 2018):

- 将原本在关键路径上的softmax移出关键路径

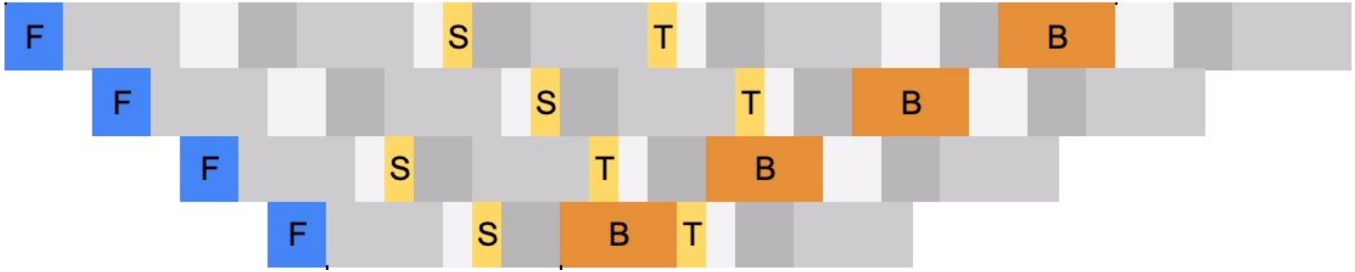


将词表计算并入流水线调度

我们将词表层的主要计算称为S pass，同步修正计算称为T pass，那么我们可以得到他们与其他流水线计算的依赖关系



根据此依赖关系，将S/T pass并入流水线中



05 社区影响&讨论

社区影响

- 前三个工作开源于 GitHub - sail-sg/zero-bubble

Starred 411

- Vocabulary Parallelism开源于GitHub - sail-sg/VocabularyParallelism

- 我们已知的使用了我们方案的社区玩家

deepseek   PyTorch   llama

torchtitan   Megatron-DeepSpeed   飞桨 PaddlePaddle

Colossal-AI   SAILOR2: Sailing in South-East Asia with Inclusive Multilingual LLMs

- 欢迎更多社区交流

Deepseek-v3中的应用

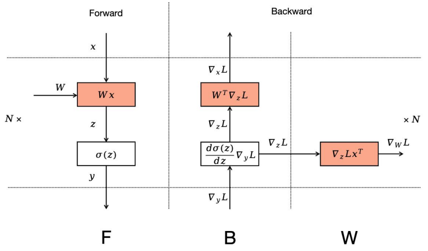
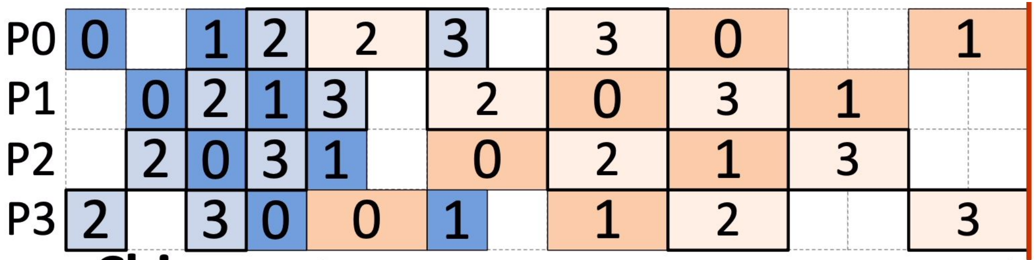
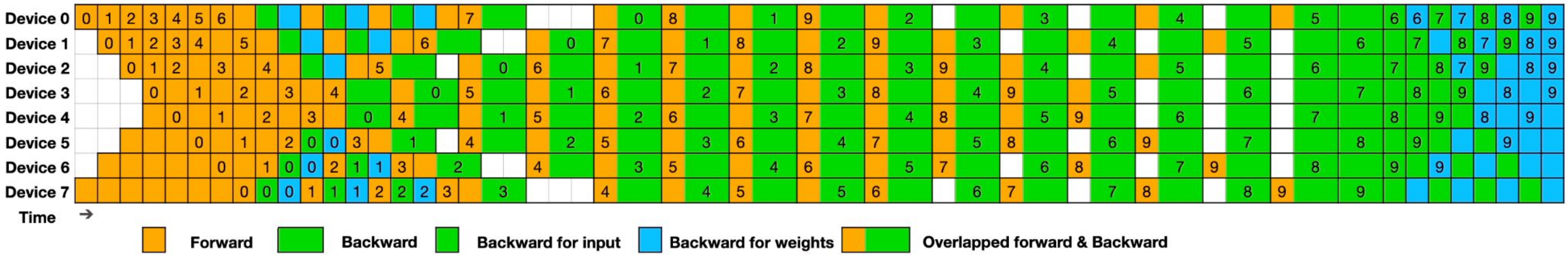


Figure 1: Computation Graph for MLP.

B-W split (Zero Bubble)

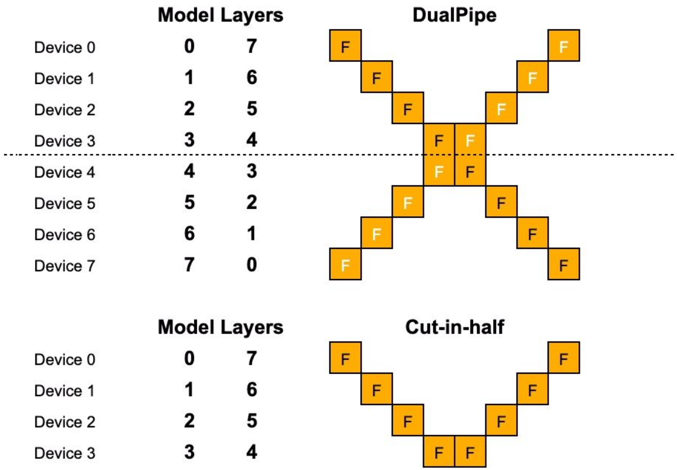


Chimera (backward is **2×** workload of forward)



DualPipe (Deepseek-v3)

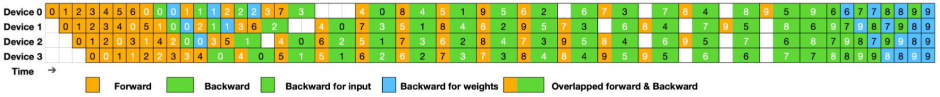
双流并行没有双流会更好



DualPipeV

DualPipeV is a concise V-shape schedule derived from DualPipe using a "cut-in-half" procedure, introduced by Sea AI Lab as "Cut-in-half" in their blog post. Thanks to them for this efficient schedule!

Schedules



* https://github.com/deepseek-ai/DualPipe

DualPipe开源后，我们跟进了一篇博文，指出

- DualPipe其实可以可以对半裁剪（Cut-in-half），避免在PP内有一份参数拷贝
- Cut-in-half其实是V-ZB在EP场景下的特例

Cut-in-half被Deepseek接受并命名为DualPipeV

https://zhuanlan.zhihu.com/p/26915547331

PP的其他问题

- 可扩展性：GPU数量仍然不能超过模型层数
- 即使对于Zero Bubble, PP仍然需要至少2D个微批次来满足效率需求，这在长序列训练场景中有可能会是一个瓶颈
- 在LLM Inference中，PP的高延迟，高内存使用的缺陷仍然存在。因为这些原因，Inference中PP很少有使用。

关注我们

Q&A

Zero Bubble Pipeline Parallelism
ICLR' 24
Extreme Throughput

PP with Controllable Memory
NeurIPS'24
Minimum Memory

Vocabulary Parallelism
MLSys' 25
Balanced Workload

PipeOffload
ICML' 25
Scalability

Paper Links:

- arxiv.org/abs/2401.10241
- arxiv.org/abs/2405.15362
- arxiv.org/pdf/2411.05288
- arxiv.org/abs/2503.01328

Open source: github.com/zero-bubble

Email: wanxy@sea.com

WeChat: ufotalent