



Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning

Lianmin Zheng^{1,*} Zhuohan Li^{1,*} Hao Zhang^{1,*} Yonghao Zhuang⁴
Zhifeng Chen³ Yanping Huang³ Yida Wang² Yuanzhong Xu³ Danyang Zhuo⁶
Eric P. Xing⁵ Joseph E. Gonzalez¹ Ion Stoica¹

¹UC Berkeley ²Amazon Web Services ³Google ⁴Shanghai Jiao Tong University
⁵MBZUAI, Carnegie Mellon University ⁶Duke University

Existing methods and shortcomings

- 传统痛点：
 - 目前业界主要有三种并行方式：**数据并行 (Data Parallelism)**、**张量并行 (Tensor Parallelism)** 和 **流水线并行 (Pipeline Parallelism)**。像 Megatron-LM 这样的顶尖系统，需要资深工程师根据特定模型（如 Transformer）的结构，极其繁琐地手动组合这三种并行方式。如果换一个**新模型**，或者换一种**集群硬件**，之前的策略就失效了。
 - **对模型不泛化**：如果你的模型不是纯粹的**均质 Transformer**（比如引入了混合专家模型 MoE，或者像 Wide-ResNet 那样特征图和权重比例在不同层级剧烈变化），原有的均质切分策略会导致严重的内存不平衡和计算气泡。
 - **对集群硬件不泛化**：论文举了个非常具体的例子：同样是训练一个大型的 MoE 模型，如果你在 Google 的 TPU 集群上跑，你需要为每一层**手动调整分割轴 (Partitioning axis)**；但如果你换到 AWS 的 GPU 集群上跑，由于网络带宽拓扑变了，你必须重新设计一套全新的流水线方案。

Limitations of Existing Auto-parallel Systems

- 在 Alpa 之前，学术界也有人在尝试做自动化并行（比如 **Tofu**、**FlexFlow**、**DAPPLE** 等），但它们都卡在了组合空间的爆炸上。
- 为了让算法跑得出来，这些系统做了严重的自我阉割：它们通常只允许“**数据并行**”加上最多**一种**其他的模型并行方式（要么只加张量并行，要么只加流水线并行）。这种妥协导致它们根本无法触及真正的全局最优解，白白错失了大量的性能优化机会。

- Alpha 没有沿用传统的“数据/张量/流水线”分类，而是提出了一个极具启发性的新视角，将并行方式分为两个层级：

- **算子内并行** (Intra-operator Parallelism): 对某一个具体的算子（比如矩阵乘法）在张量的维度上进行切分。特点是：计算效率高，但显卡之间需要极其频繁、大量的数据通信（如 All-Reduce）。
- **算子间并行** (Inter-operator Parallelism): 不切分算子，而是把整个模型的计算图“拦腰斩断”分成不同的阶段（Stages），不同的显卡负责不同的阶段（即流水线并行）。特点是：阶段之间通信量小（只需点对点传输中间结果），但由于存在数据依赖，会产生设备等待的空闲时间（Bubble）。

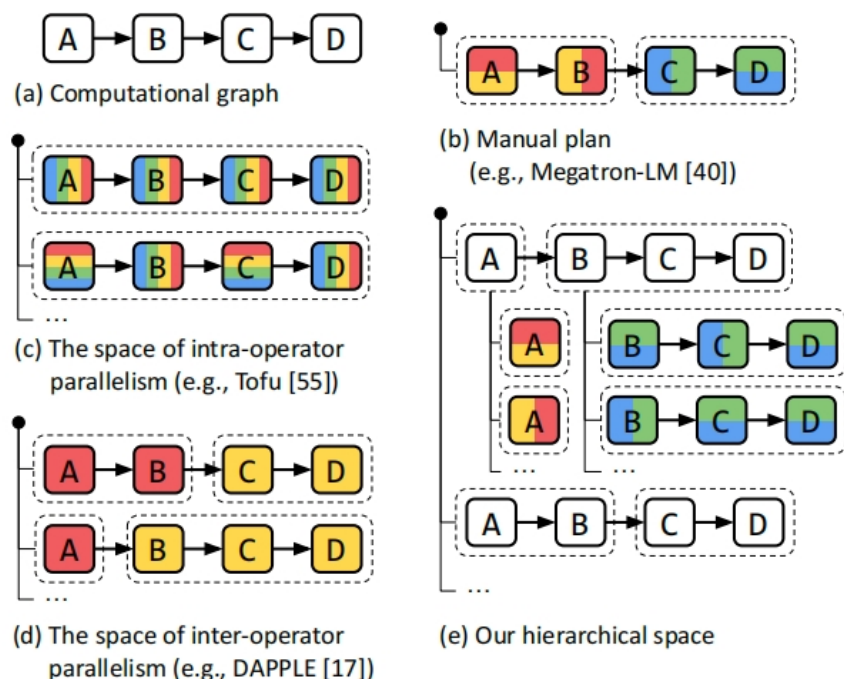


Figure 1: Generation of parallelization plans for a computational graph shown in (a). Different colors represent different devices, dashed boxes represent pipeline stages. (b) creates the plan manually. (c) and (d) automatically generate plans using only one of intra- and inter-operator parallelisms. (e) shows our approach that creates a hierarchical space to combine intra- and inter-operator parallelisms.

算子内并行 (Intra-operator Parallelism)

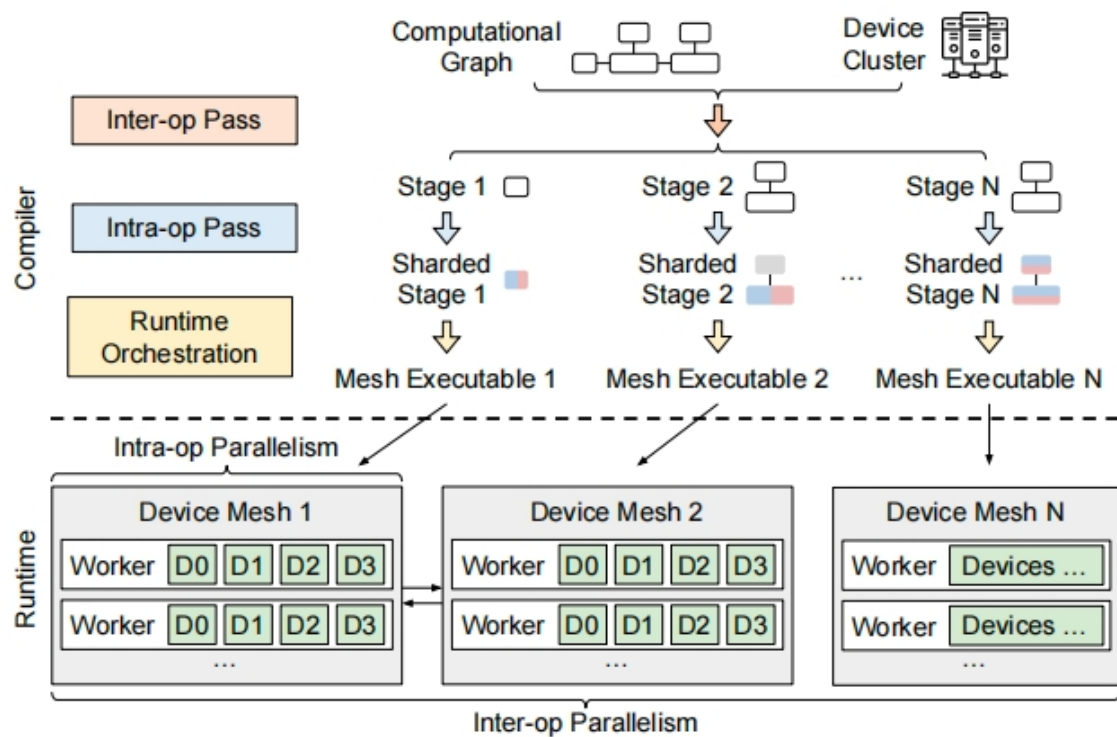


Figure 3: Compiler passes and runtime architecture. A sharded stage is a stage annotated with the sharding specs generated by intra-op pass.

输入：位于图表最上方的是用户定义的完整计算图（Computational Graph）和可用的物理设备集群（Device Cluster）

1. 顶层：算子间编译（Inter-op Pass）
2. 中层：算子内编译（Intra-op Pass）
3. 底层：运行时统筹（Runtime Orchestration）

Table 2: Several cases of resharding. $all-gather(x, i)$ means an all-gather of x bytes along the i -th mesh axis. M is the size of the tensor. (n_0, n_1) is the mesh shape.

#	Src Spec	Dst Spec	Communication Cost
1	RR	S^0S^1	0
2	S^0R	RR	$all-gather(M, 0)$
3	S^0S^1	S^0R	$all-gather(\frac{M}{n_0}, 1)$
4	S^0R	RS^0	$all-to-all(\frac{M}{n_0}, 0)$
5	S^0S^1	$S^{01}R$	$all-to-all(\frac{M}{n_0 \cdot n_1}, 1)$

Table 3: Several parallel algorithms for a batched matmul $C_{b,i,j} = \sum_k A_{b,i,k} B_{b,k,j}$. The notation $all-reduce(x, i)$ means an all-reduce of x bytes along the i -th mesh axis. M is the size of the output tensor. (n_0, n_1) is the mesh shape.

#	Parallel Mapping	Output Spec	Input Specs	Communication Cost
1	$i \rightarrow 0, j \rightarrow 1$	RS^0S^1	RS^0R, RRS^1	0
2	$i \rightarrow 0, k \rightarrow 1$	RS^0R	RS^0S^1, RS^1R	$all-reduce(\frac{M}{n_0}, 1)$
3	$j \rightarrow 0, k \rightarrow 1$	RRS^0	RRS^1, RS^1S^0	$all-reduce(\frac{M}{n_0}, 1)$
4	$b \rightarrow 0, i \rightarrow 1$	S^0S^1R	S^0S^1R, S^0RR	0
5	$b \rightarrow 0, k \rightarrow 1$	S^0RR	S^0RS^1, S^0S^1R	$all-reduce(\frac{M}{n_0}, 1)$
6	$i \rightarrow \{0, 1\}$	$RS^{01}R$	$RS^{01}R, RRR$	0
7	$k \rightarrow \{0, 1\}$	RRR	$RRS^{01}, RS^{01}R$	$all-reduce(M, \{0, 1\})$

- 对于一个 N 维张量，它的切分状态用 $X_0X_1...X_{n-1}$ 来表示，其中 $X_i \in \{S, R\}$:
 - S (Sharded): 代表这个维度被切分了。
 - R (Replicated): 代表这个维度没有被切分，是完全复制的。
- 为了把切分维度和上面的 2D 设备网格对应起来，Alpa 还在 S 上加了上标:
 - RR : 一个二维矩阵在所有显卡上都有完整副本。
 - S^0R : 矩阵按行切分（切到网格的第 0 维上），列保持复制。
 - $S^{01}R$: 矩阵按行切分，并且同时分摊到了网格的第 0 维和第 1 维上。

$$\min_s \sum_{v \in V} s_v^T (c_v + d_v) + \sum_{(v,u) \in E} s_v^T R_{vu} s_u, \quad (1)$$

算子间并行 (Inter-operator Parallelism)

- 类似于流水线并行，但是对象从层划分变成了算子的划分
- 第一部分：这是流水线的“启动时间”。也就是第一批数据 (Microbatch) 从头走到尾，历经所有 S 个阶段所需的总时间。
- 第二部分：这是剩下 $B-1$ 批数据的“排队时间”。在流水线跑满之后，整个工厂的产出速度完全取决于“最慢的那个阶段”。

$$T^* = \min_{\substack{s_1, \dots, s_S; \\ (n_1, m_1), \dots, (n_S, m_S)}} \left\{ \sum_{i=1}^S t_i + (B-1) \cdot \max_{1 \leq j \leq S} \{t_j\} \right\}. \quad (2)$$

- 动态规划 (Dynamic Programming, DP) 求出最优的策略划分方案。

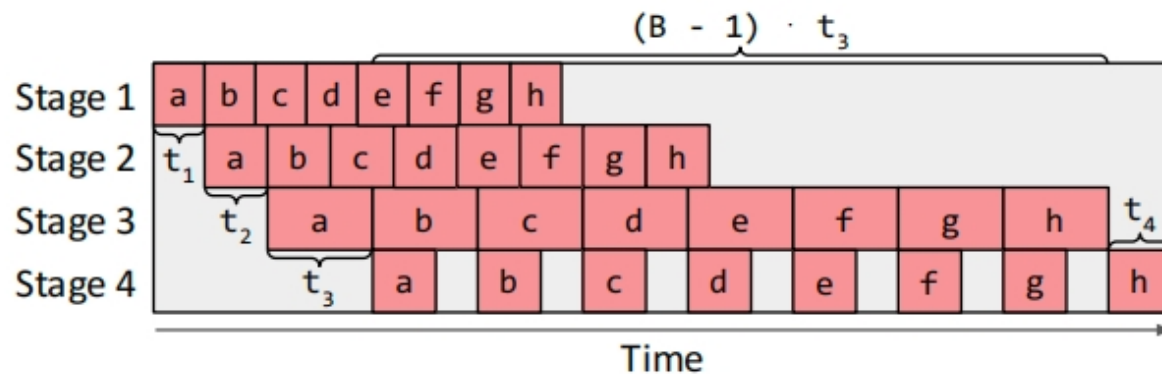


Figure 5: Illustration of the total latency of a pipeline, which is determined by two parts: the total latency of all stages ($t_1 + t_2 + t_3 + t_4$) and the latency of the slowest stage ($(B-1) \cdot t_3$).

Evaluation

- 硬件测试环境 (Hardware Environment)
 - 集群规模：总共 8 个节点 (Nodes)，包含 64 张 GPU。
 - 节点配置：使用的是 Amazon EC2 的 p3.16xlarge 实例。每个节点配备 8 张 NVIDIA V100 (16 GB) GPU、64 个 vCPU 以及 488 GB 内存。
 - 网络拓扑 (关键点)：
 - 节点内 (高速)：单台机器内的 8 张 GPU 通过 NVLink 高速互联。
 - 节点间 (低速)：8 个节点被放置在一个置放群组 (Placement group) 中，跨节点之间的网络带宽仅为 25 Gbps。

Table 4: Models used in the end-to-end evaluation. LM = language model. IC = image classification.

Model	Task	Batch size	#params (billion)	Precision
GPT-3 [10]	LM	1024	0.35, 1.3, 2.6, 6.7, 15, 39	FP16
GShard MoE [31]	LM	1024	0.38, 1.3, 2.4, 10, 27, 70	FP16
Wide-ResNet [59]	IC	1536	0.25, 1.0, 2.0, 4.0, 6.7, 13	FP32

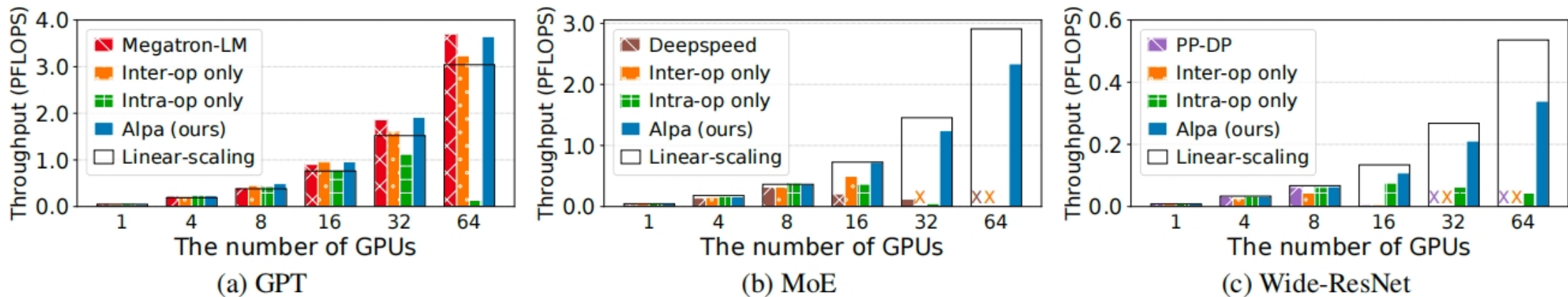


Figure 7: End-to-end evaluation results. “x” denotes out-of-memory. Black boxes represent linear scaling.

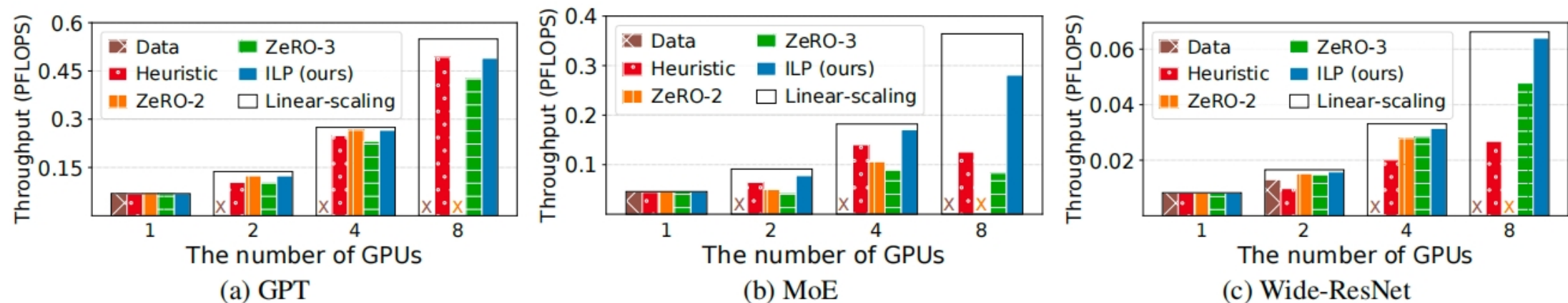


Figure 8: Intra-operator parallelism ablation study. “x” denotes out-of-memory. Black boxes represent linear scaling.

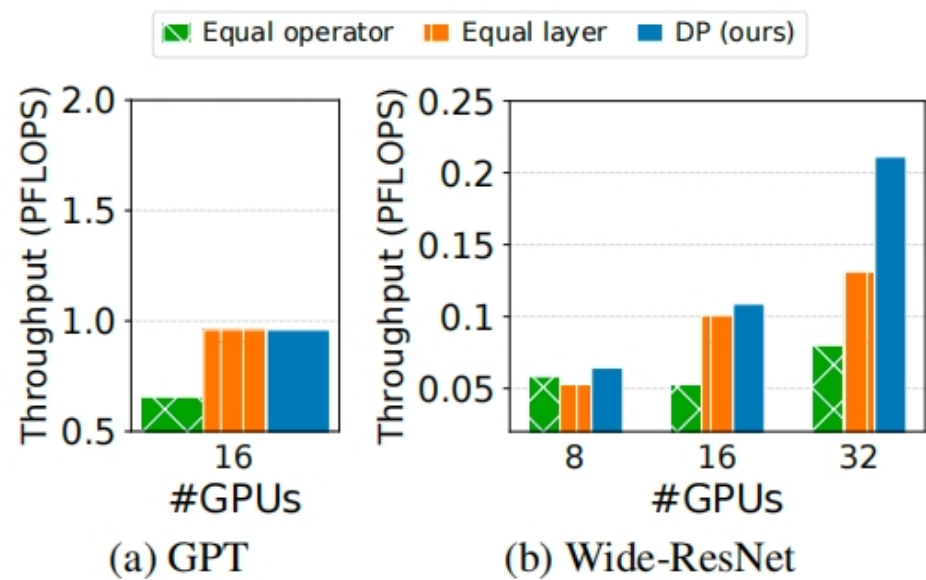


Figure 9: Inter-operator parallelism ablation study.

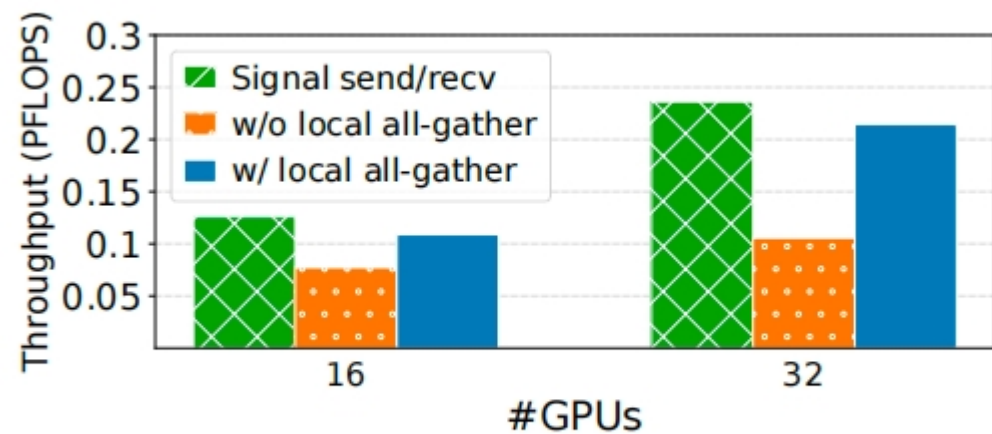


Figure 11: Cross-mesh resharding on Wide-ResNet.