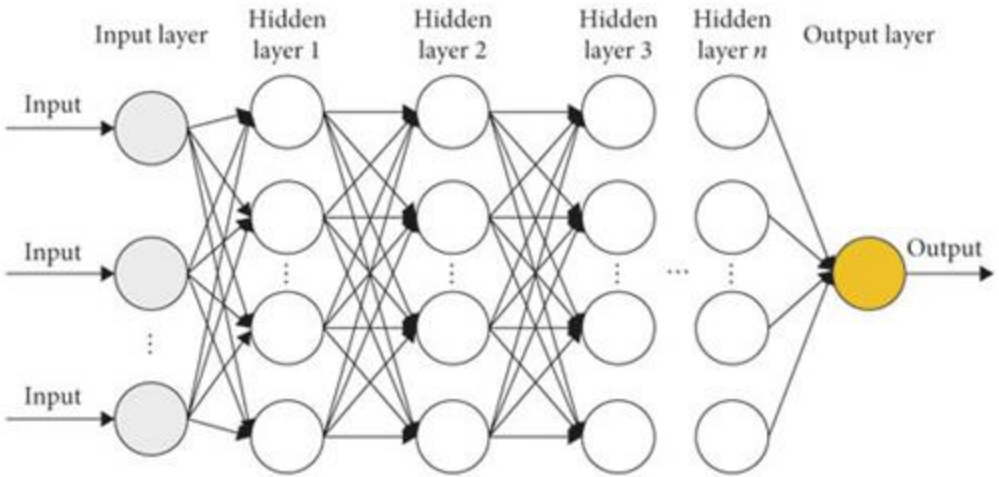
Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning

Authors: Lianmin Zheng,* Zhuohan Li,* Hao Zhang,* Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica

Presenter: Yihan Zhang

Background:

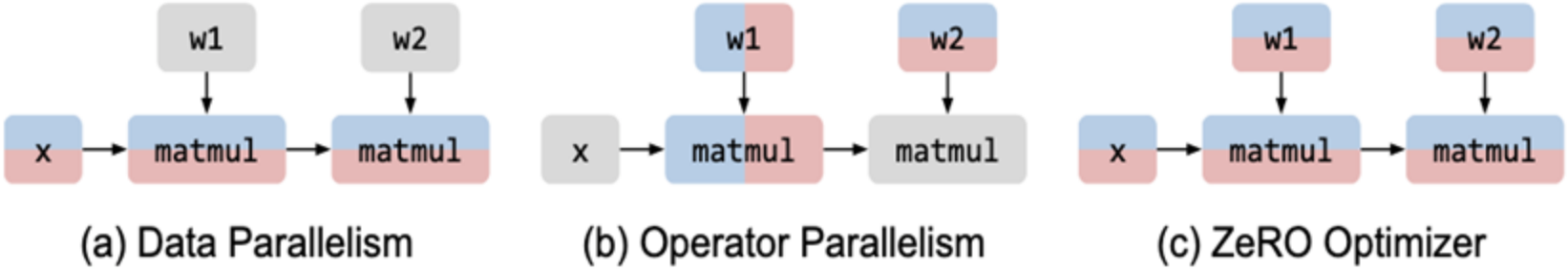
1. Training DL models tend to be distributed, which means parallelism



Background

1. Data Parallelism
2. Operator Parallelism
3. Pipeline Parallelism

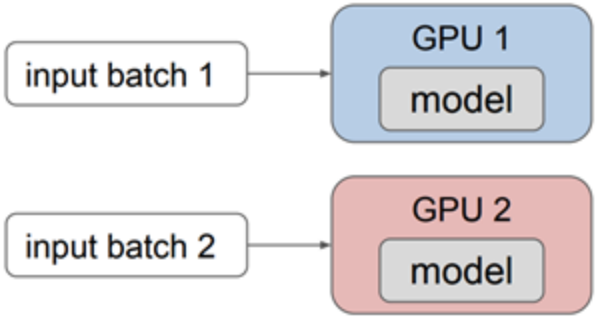
Figuring out those combination is HARD!!!



What are System Challenges?

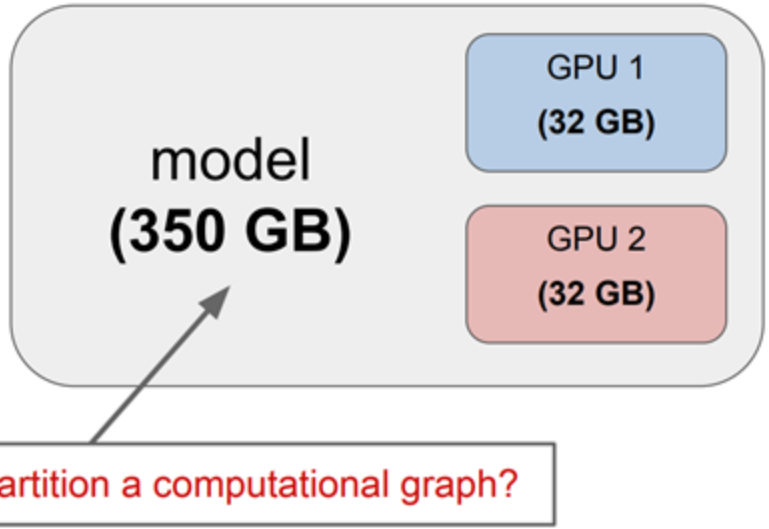
1. What if the input dataset is very large?

😃 **Easy.**
Use data parallelism: partition input data and replicate the model

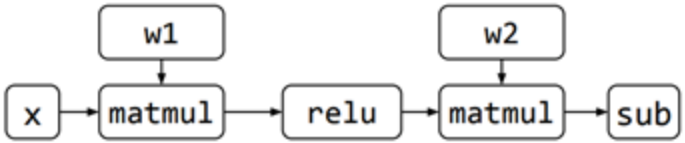


2. What if the model is very large?

😖 **Hard !!**



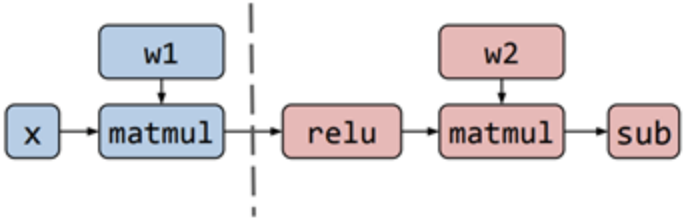
Partition Computational Graphs



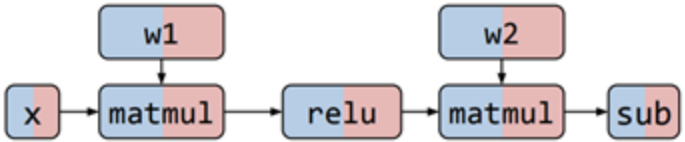
Device 1

Device 2

Strategy 1: Inter-operator Parallelism



Strategy 2: Intra-operator Parallelism



Trade-off

	Inter-operator Parallelism	Intra-operator Parallelism
Communication	Less	More
Device Idle Time	More	Less

Alpa Compiler:

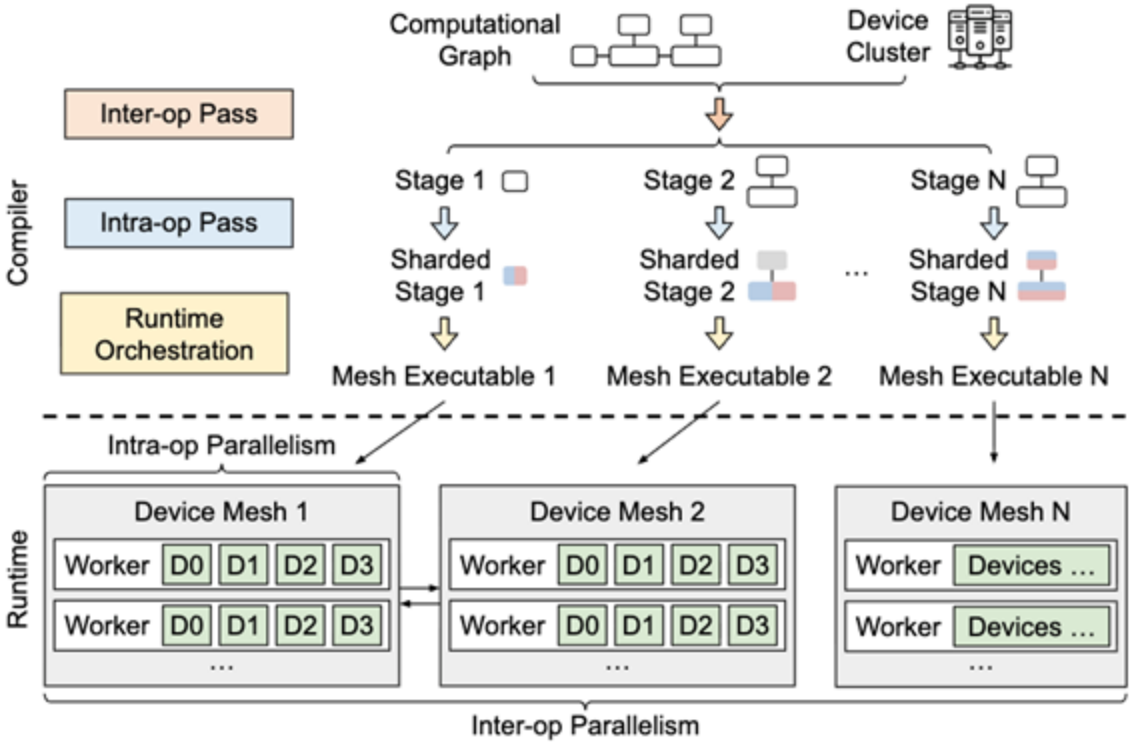
A unified compiler that automatically finds and executes the best Inter-op and Intra-op parallelism for large deep learning models

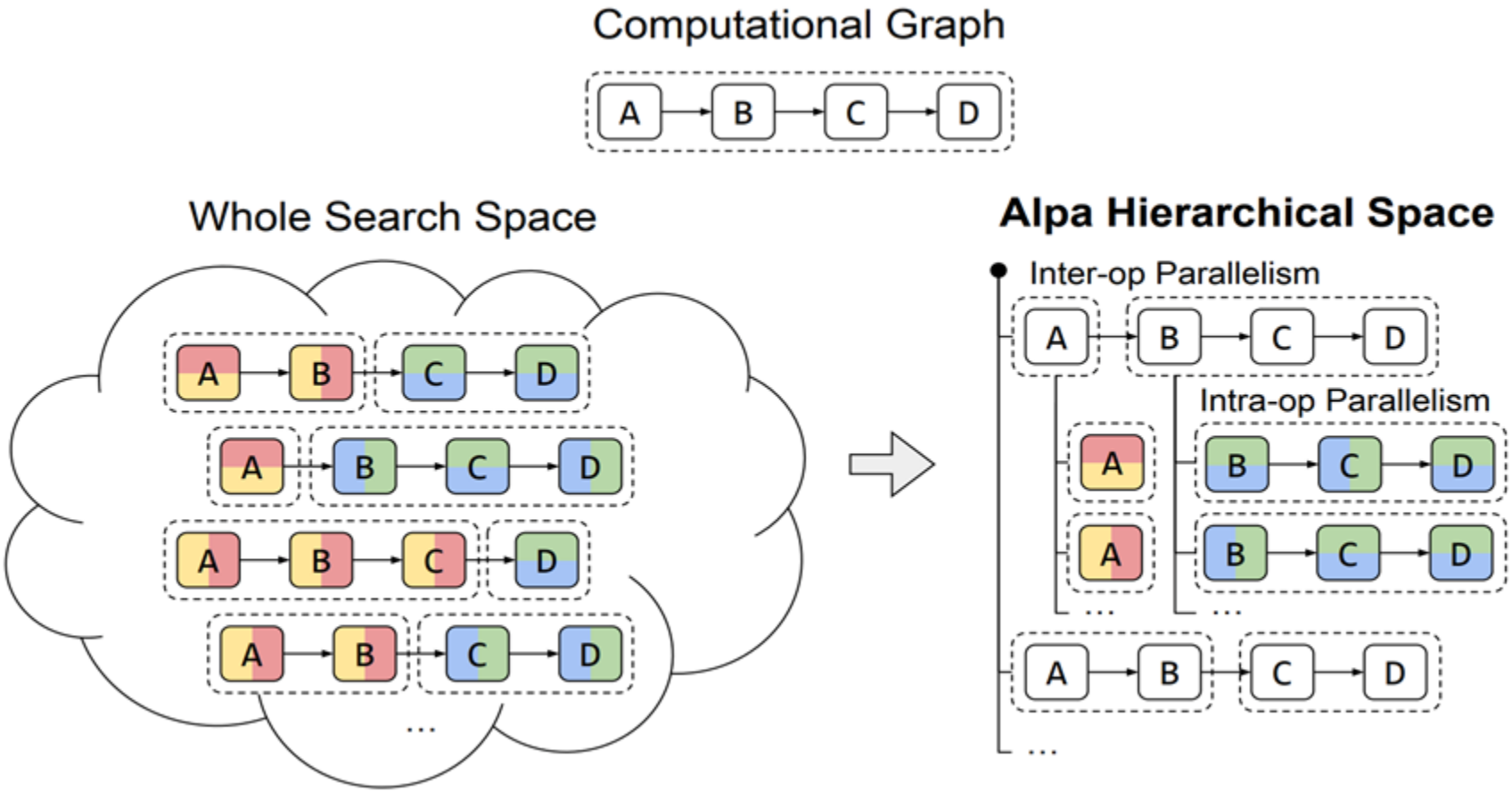
Two-level hierarchical space of parallelism techniques.

Effective optimization algorithms at each level.

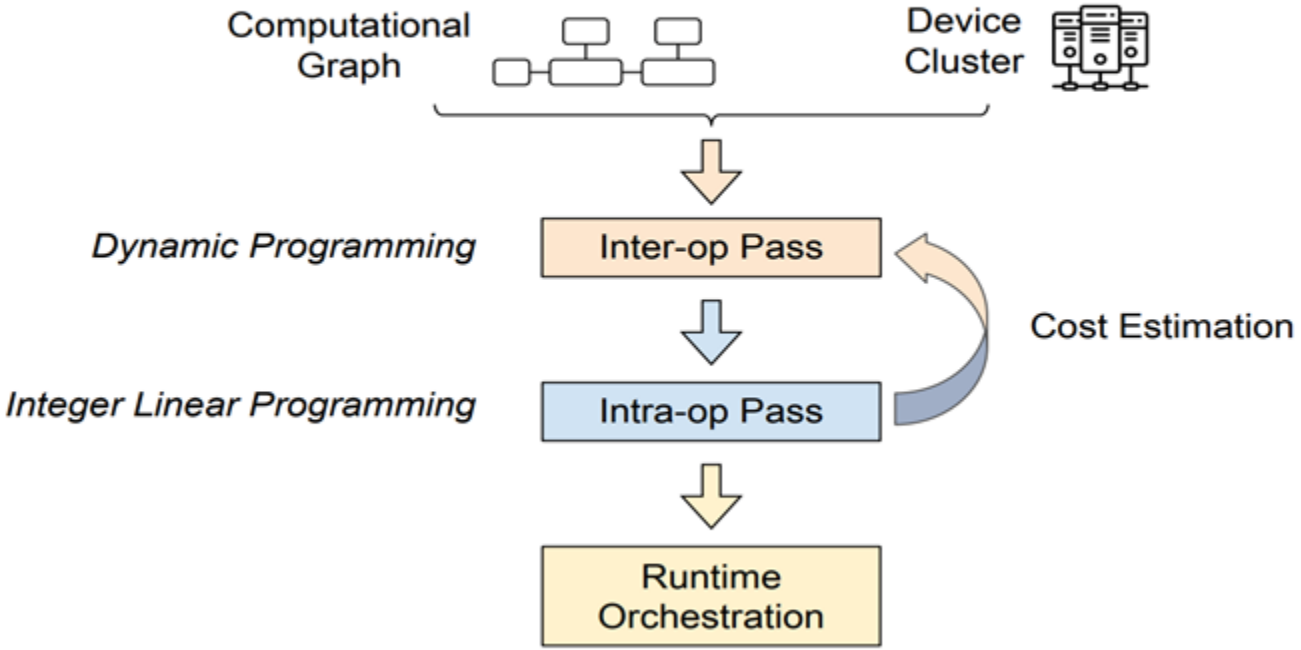
Efficient compiler and runtime system implementation.

Overview


Computational Graph
Device Cluster
Compiler
Inter-op Pass
Intra-op Pass
Runtime Orchestration
Stage 1
Stage 2
Stage N
Sharded Stage 1
Sharded Stage 2
Sharded Stage N
Mesh Executable 1
Mesh Executable 2
Mesh Executable N
Intra-op Parallelism
Runtime
Device Mesh 1
Device Mesh 2
Device Mesh N
Worker
D0
D1
D2
D3
Devices ...
Inter-op Parallelism

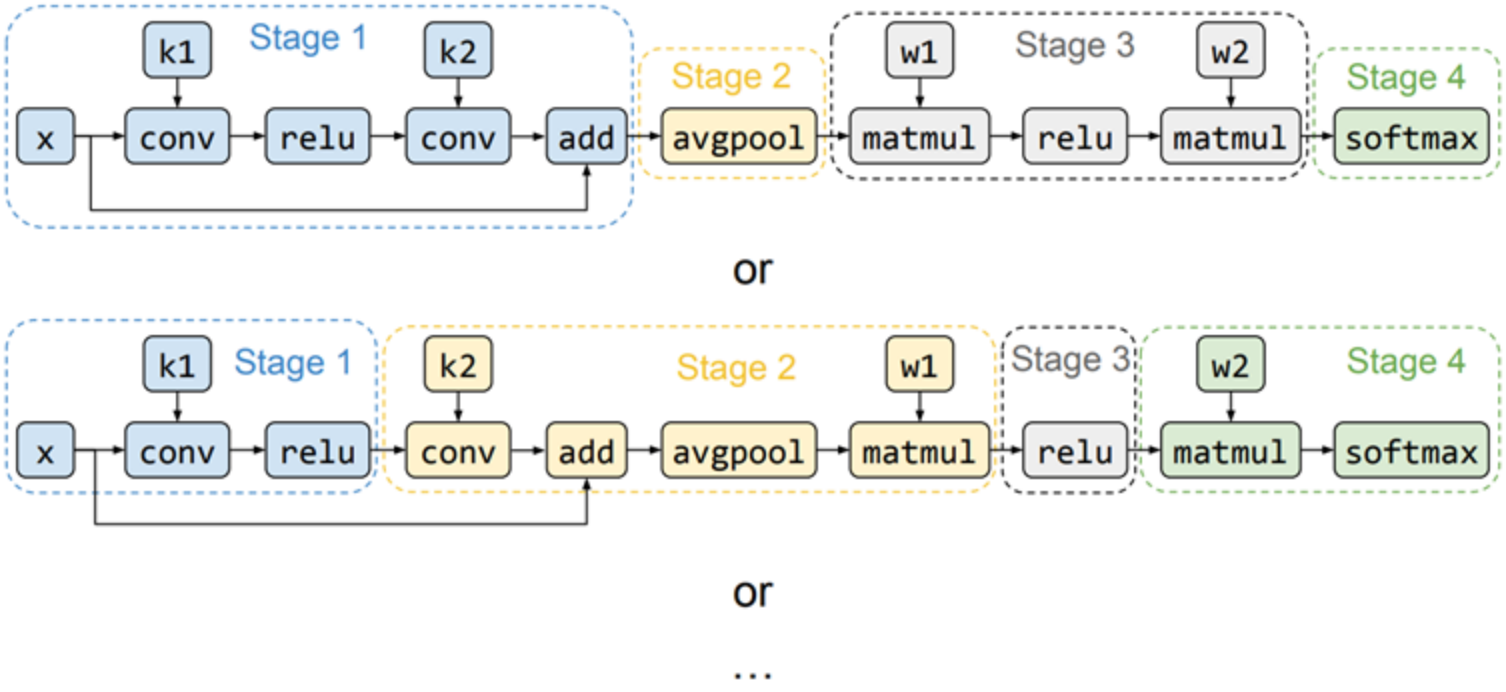

Computational Graph
A
B
C
D
Whole Search Space
Alpa Hierarchical Space
Inter-op Parallelism
Intra-op Parallelism

Alpa Compiler: Hierarchical Optimization

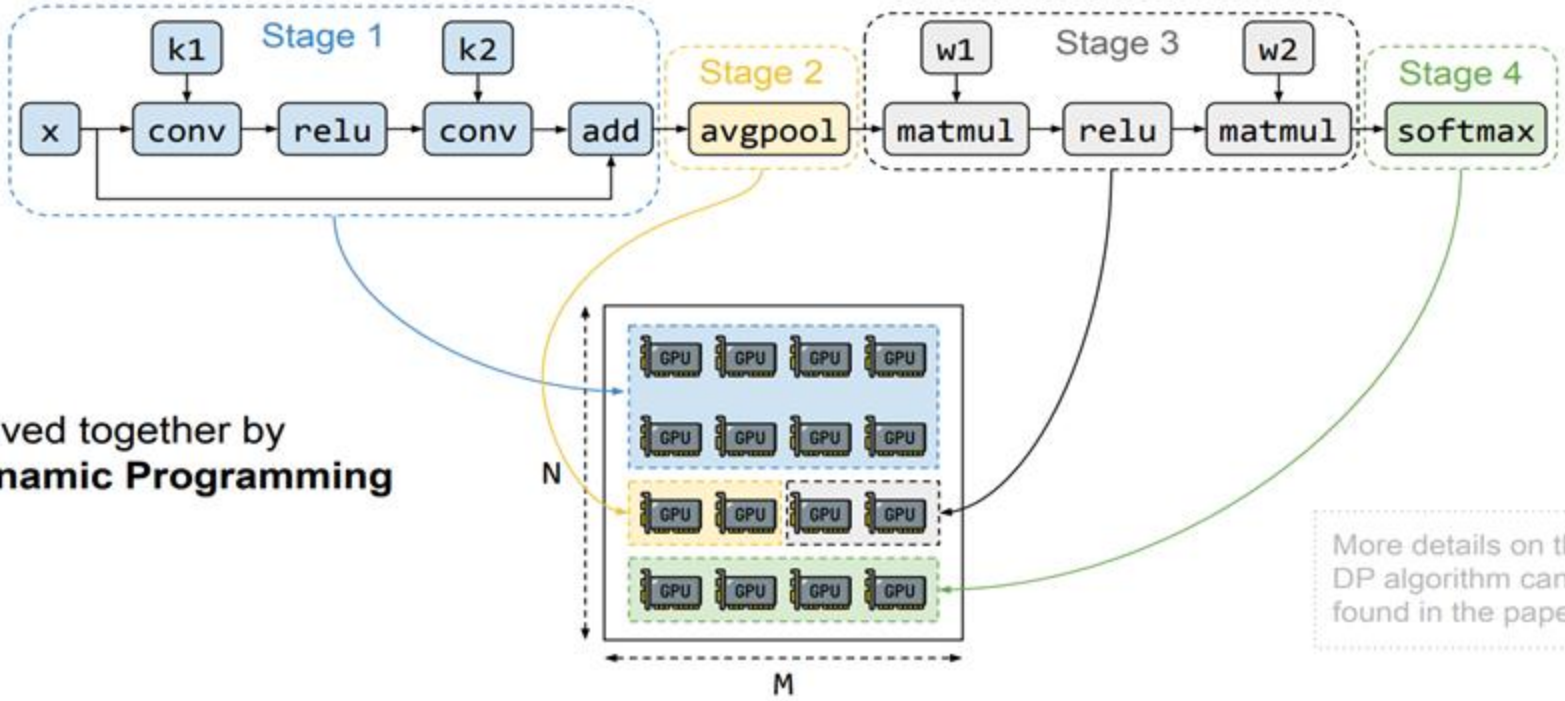


Inter-op Pass

Graph Partitioning



Inter-op Pass



Solved together by
Dynamic Programming

More details on the DP algorithm can be found in the paper.

Inter-op Steps:

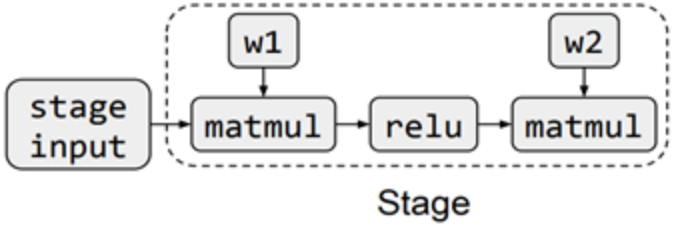
1. Minimize 1F1B iteration latency by partitioning the model into stages and mapping stages to device meshes under memory and device constraints.

1. Use dynamic programming to choose stage boundaries and meshes to minimize the serial sum, adding boundary comm/resharding costs and enforcing device non-overlap.

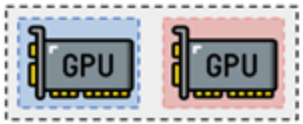
$$T^* = \min_{\substack{s_1,\dots,s_S;\\ (n_1,m_1),\dots,(n_S,m_S)}} \left\{ \sum_{i=1}^{S} t_i + (B-1) \cdot \max_{1 \le j \le S} \{t_j\} \right\}.$$

1. Output the ordered stage–mesh plan with cross-mesh comm specs, compile per-mesh executables, and run with a 1F1B pipeline.

Intra-op Pass

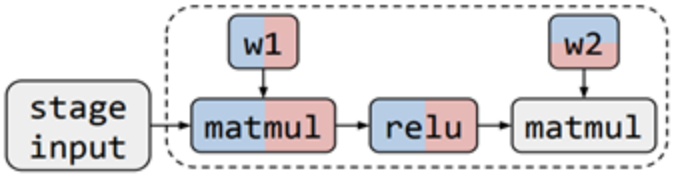


Stage



Submesh

Solved by
Integer Linear Programming



Stage with intra-operator parallelization

Intra-op Steps

1. Get output from internode stage

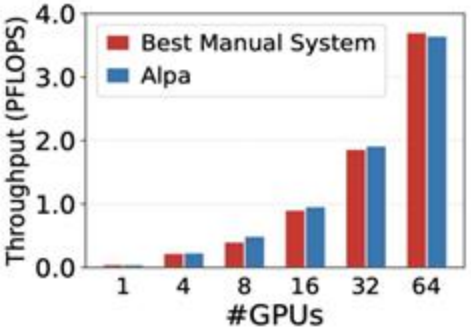
1. Create the search space by enumerating a set of tensor-sharding options with their required collectives (all-reduce/all-gather/all-to-all), and define the resulting shard layouts. Estimate per-candidate communication time using mesh bandwidth.

1. Solve a compact ILP to pick one candidate per op that minimizes total (op communication + re-sharding), outputing sharding and collective plan for the whole stage.

$$\min_{s} \sum_{v \in V} s_v^{\mathsf{T}} (c_v + d_v) + \sum_{(v,u) \in E} s_v^{\mathsf{T}} R_{vu} s_u,$$

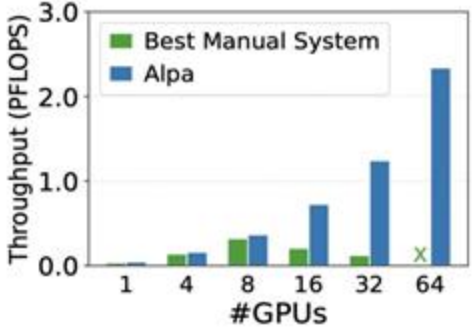
Evaluation: Comparing with Previous Works

GPT (up to 39B)



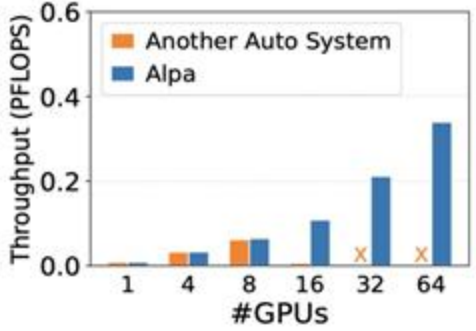
Match specialized manual systems.

GShard MoE (up to 70B)



Outperform the manual baseline by up to 8x.

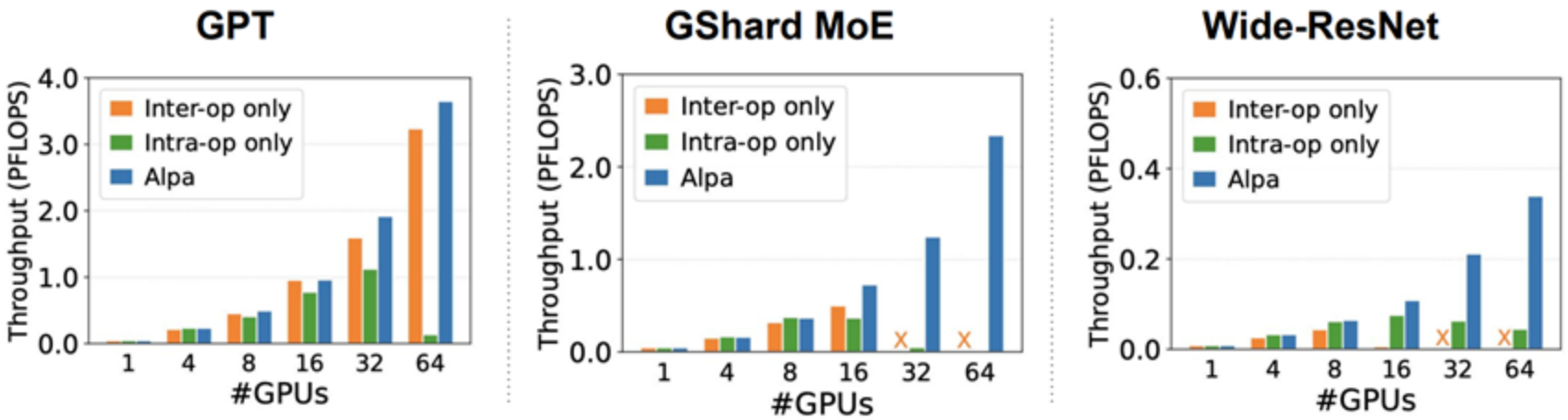
Wide-ResNet (up to 13B)



Generalize to models without manual plans.

Weak scaling results where the model size grow with #GPUs.
Evaluated on 8 AWS EC2 p3.16xlarge nodes with 8 16GB V100s each (64 GPUs in total).

Evaluation: Ablation Study with Inter-op and Intra-op Only



Combining inter- and intra-operator parallelism scales to more devices.

THANKS!!!