Parallelize One Operator

2D Convolution

Simple spatial loops. Can be arbitrarily split.

Stencil computation loops. Splitting these requires careful boundary handling.

Reduction loop. Need to accumulate partial results.

Reduction loops. But usually too small (<= 5) for parallelization.

```python
for n in range(0, N):
  for co in range(0, CO):
    for h in range(0, H):
      for w in range(0, W):
        for ci in range(0, CI):
          for kh in range(0, KH):
            for kw in range(0, KW):
              C[n,co,h,w] += A[n,co,h+kh,w+kw] x B[kh,kw,co,ci]
```

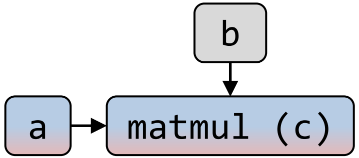
Simple case: Parallelize loop n, co, ci, then the parallelization strategies are almost the same as matmul's.

Complicated case: Parallelize loop h and w

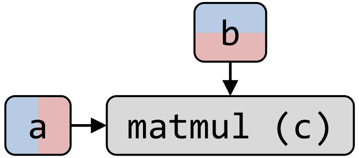
Data Parallelism as A Case of Intra-op Parallelism

Replicated Row-partitioned Column-partitioned

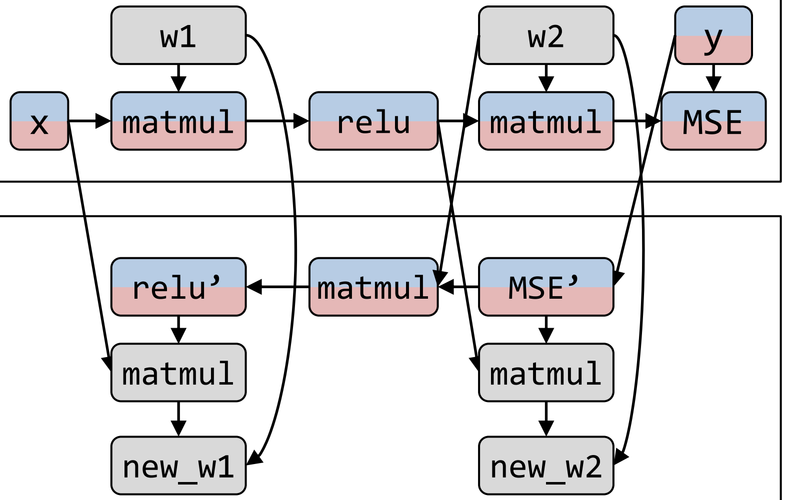
Matmul Parallelization Type 1
communication cost = 0



Matmul Parallelization Type 2
communication cost = all-reduce(c)



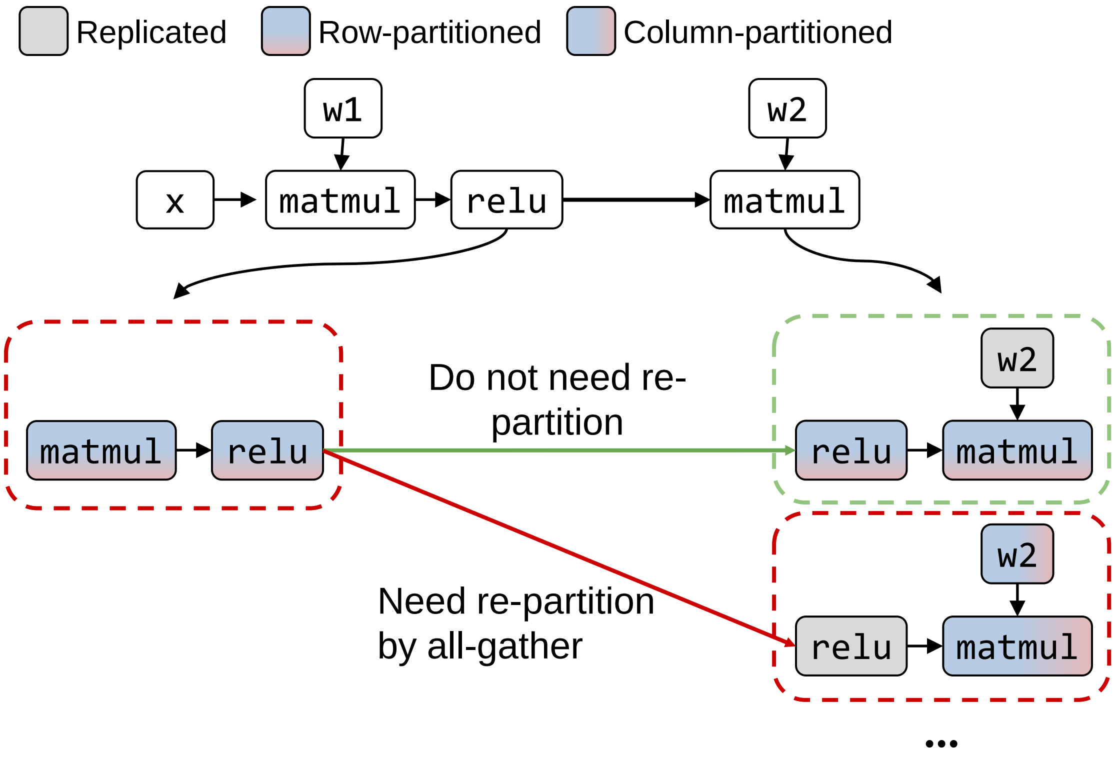
Forward Pass
Two “Type 1” matmuls: no communication



Backward Pass
One “Type 1” matmul: no communication
Two “Type 2” matmuls: require all-reduce

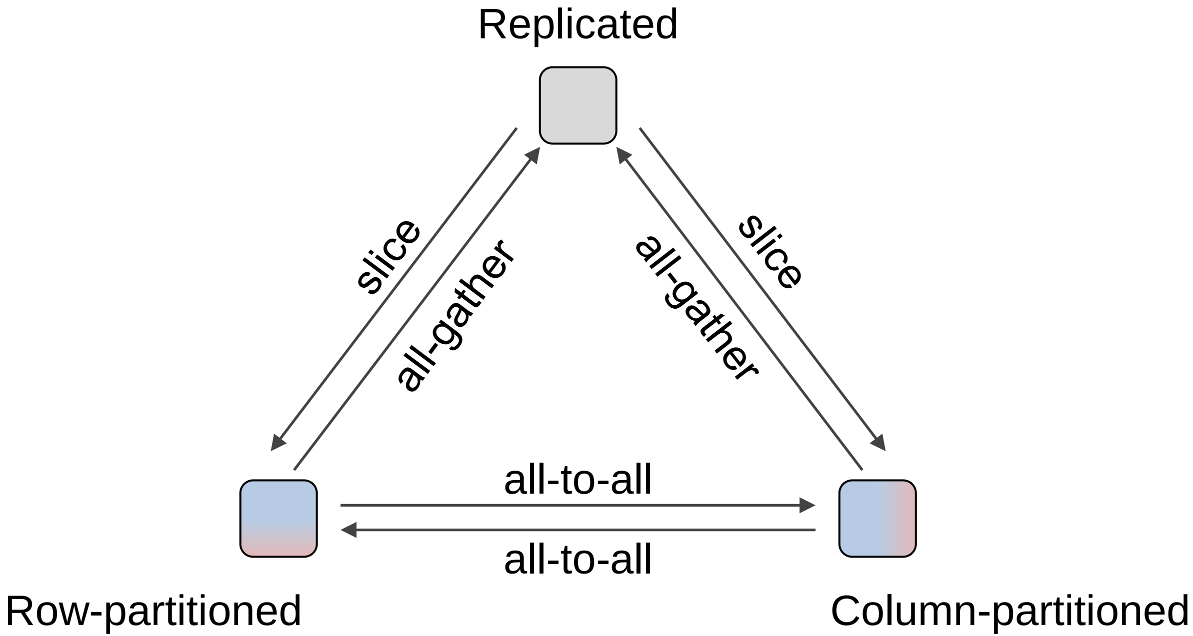
Re-partition Communication Cost

Different operators' parallelization strategies require different partition format of the same tensor



Re-partition Communication Cost

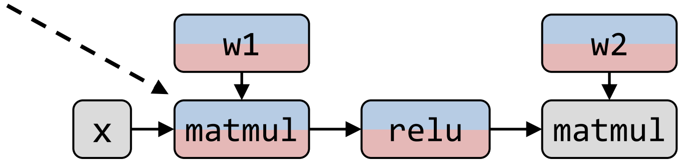
Different operators' parallelization strategies require different partition format of the same tensor



Parallelize All Operators in a Graph

Problem

Pick a parallel strategy
of each operator



Minimize Node costs (computation + communication) + Edge costs (re-partition communication)

Solution

Manual design
Randomized search
Dynamic programming
Integer linear programming

Important Projects

Model-specific Intra-op Parallel Strategies

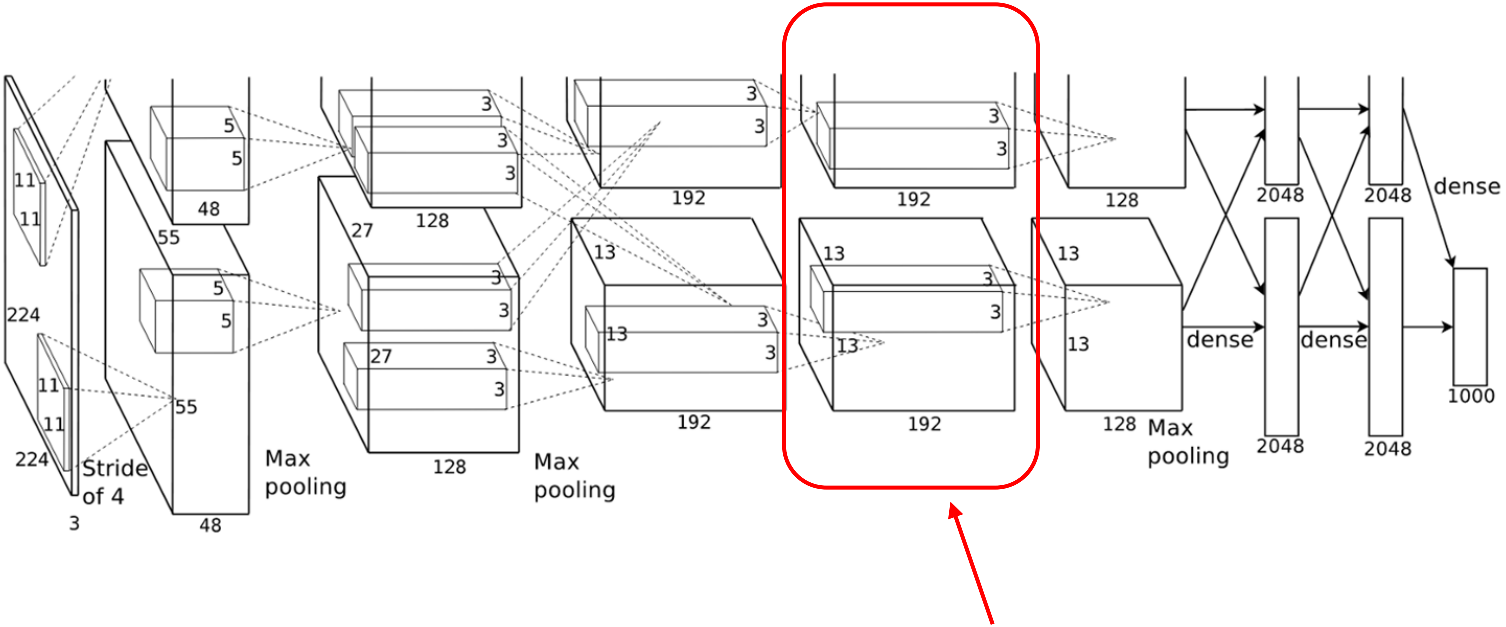
- AlexNet
- Megatron-LM
- GShard MoE

Systems for Intra-op Parallelism

- ZeRO
- Mesh-Tensorflow
- GSPMD
- Tofu
- FlexFlow

AlexNet

Result: increase top-1 accuracy by 1.7%



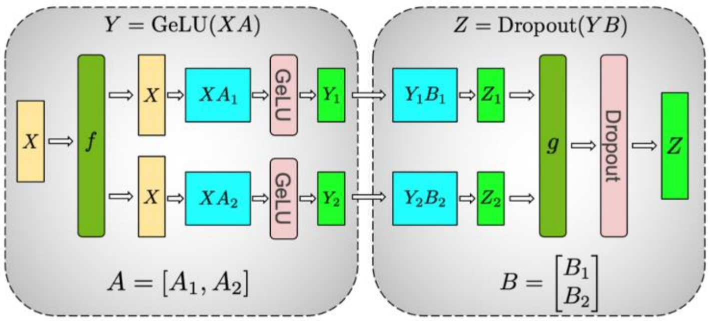
Assign a group convolution layer to 2 GPUs

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *NeurIPS 2012*

Megaton-LM

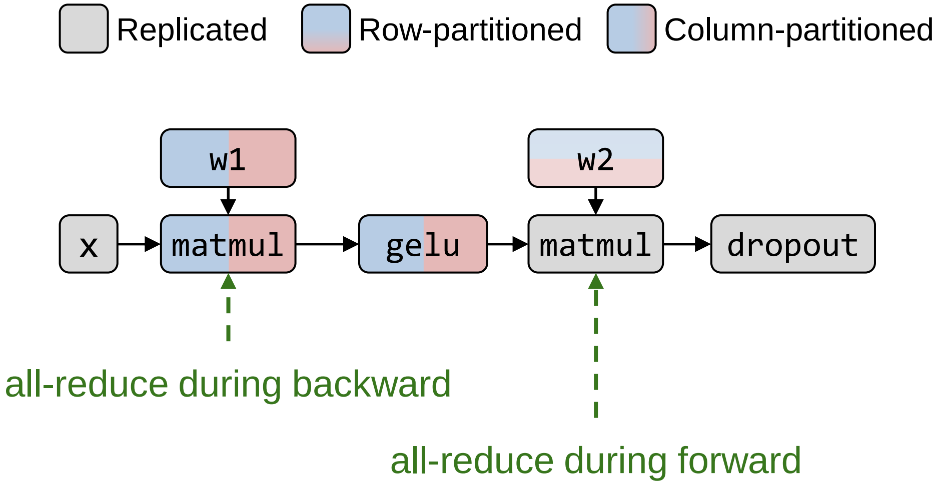
Result: a large language model with 8.3B parameters that outperforms SOTA results

Figure 3 from the paper:
How to partition the MLP in the transformer.



(a) MLP

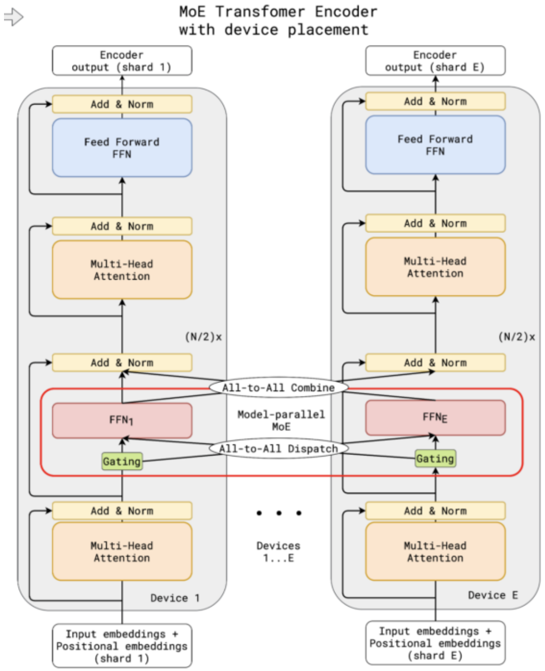
Illustrated with the notations in this tutorial



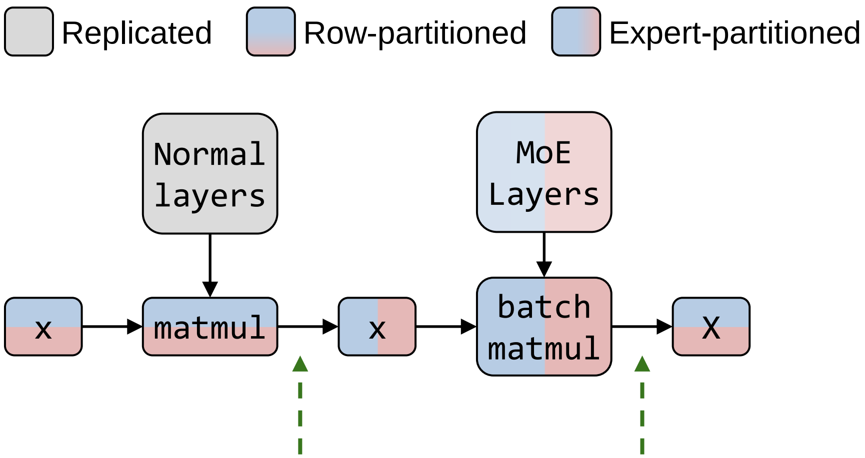
Shoeybi, Mohammad, et al. "Megatron-LM: Training multi-billion parameter language models using model parallelism."

GShard MoE

Result: a multi-language translation model with 600B parameters that outperforms SOTA



Illustrated with the notations in this tutorial



Lepikhin, Dmitry, et al. "GShard: Scaling giant models with conditional computation and automatic sharding." *ICLR 2021*

ZeRO Optimizer

Problem
Data parallelism replicates gradients, optimizer states and model weights on all devices.

Idea
Partition gradients, optimizer states and model weights.

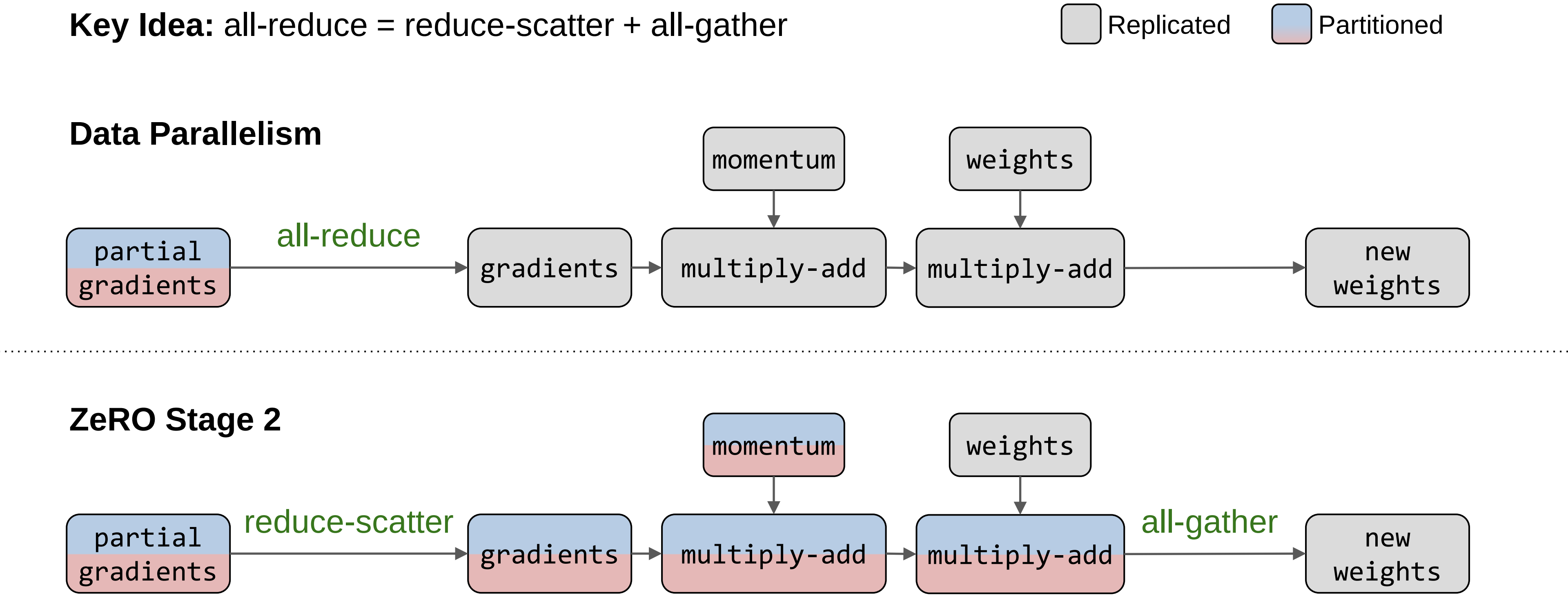
M is the number of parameters, N is the number of devices.

	Optimizer States (12M)	Gradients (2M)	Model Weights (2M)	Memory Cost	Communication Cost
Data Parallelism	Replicated	Replicated	Replicated	$16M$	all-reduce(2M)
ZeRO Stage 1	Partitioned	Replicated	Replicated	$4M + \frac{12M}{N}$	all-reduce(2M)
ZeRO Stage 2	Partitioned	Partitioned	Replicated	$2M + \frac{14M}{N}$	all-reduce(2M)
ZeRO Stage 3	Partitioned	Partitioned	Partitioned	$\frac{16M}{N}$	1.5 all-reduce(2M)

Rajbhandari, Samyam, et al. "Zero: Memory optimizations toward training trillion parameter models." *SC 2020*

ZeRO Stage 2

Key Idea: all-reduce = reduce-scatter + all-gather

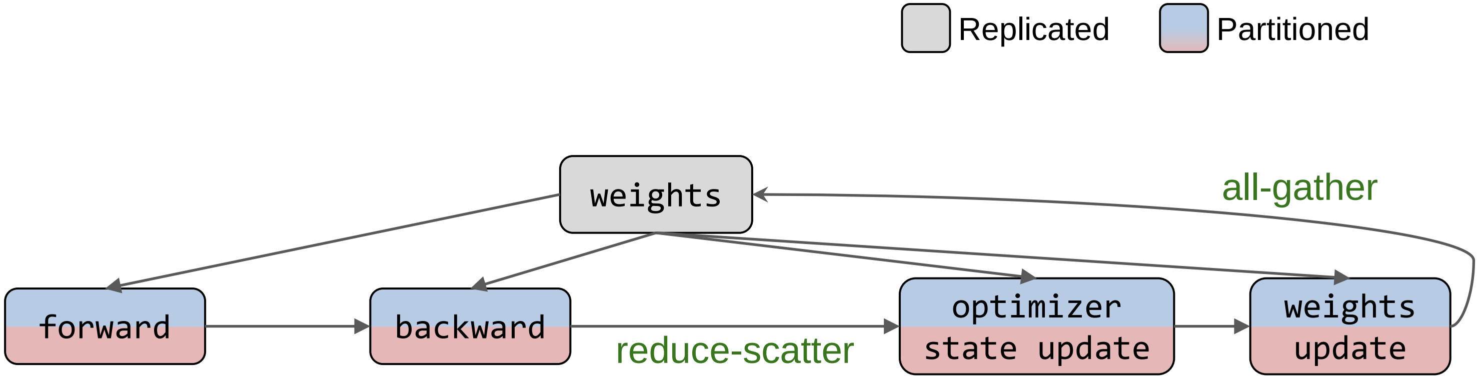


Same communication cost but save memory by partitioning more tensors

ZeRO Stage 3

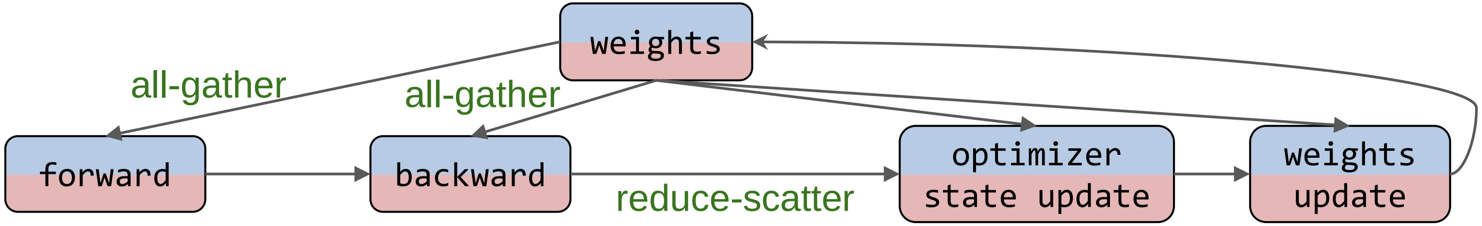
ZeRO Stage 2

communication cost
= all-reduce



ZeRO Stage 3

communication cost
= 1.5 all-reduce



Mesh-Tensorflow

Map tensor dimension to mesh dimension for parallelism

```
...
batch = mtf.Dimension("batch", b)
io = mtf.Dimension("io", d_io)
hidden = mtf.Dimension("hidden", d_h)
# x.shape == [batch, io]
w = mtf.get_variable("w", shape=[io, hidden])
bias = mtf.get_variable("bias", shape=[hidden])
v = mtf.get_variable("v", shape=[hidden, io])
h = mtf.relu(mtf.einsum(x, w, output_shape=[batch, hidden]) + bias)
y = mtf.einsum(h, v, output_shape=[batch, io])
...
```

Tensor dimension

Mesh dimension

```
mesh_shape = [("rows", r), ("cols", c)]
computation_layout = [("batch", "rows"), ("hidden", "cols")]
```

Mapping

Shazeer, Noam, et al. "Mesh-tensorflow: Deep learning for supercomputers." *NeurIPS* 2018.

GSPMD

- Use annotations to specify partition strategy
- Propagate the annotations to whole graph
- Use compiler to generate SPMD (Single Program Multiple Data) parallel executables

```
  # Partition inputs along group (G) dim.
+ inputs = split(inputs, 0, D)
  # Replicate the gating weights
+ wg = replicate(wg)
  gates = softmax(einsum("GSM,ME->GSE", inputs, wg))
  combine_weights, dispatch_mask = Top2Gating(gating_logits)
  dispatched_expert_inputs = einsum(
    "GSEC,GSM->EGCM", dispatch_mask, reshaped_inputs)
  # Partition dispatched inputs along expert (E) dim.
+ dispatched_expert_inputs = split(dispatched_expert_inputs, 0, D)
  h = einsum("EGCM,EMH->EGCH", dispatched_expert_inputs, wi)
  ...
```

Xu, Yuanzhong, et al. "GSPMD: general and scalable parallelization for ML computation graphs." *arXiv 2021*

Tofu

Tensor description language for automatic parallelization analysis

```
@tofu.op
def conv1d(data, filters):
  return lambda b, co, x:
    Sum(lambda ci, dx: data[b, ci, x+dx]*filters[ci, co, dx]
```

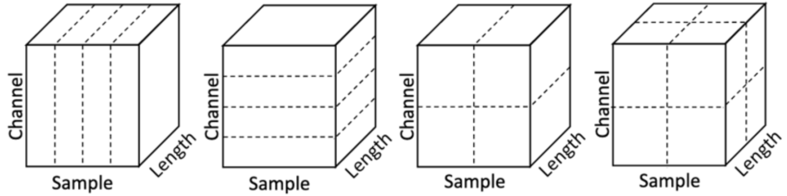
Dynamic programming for graph-level optimization

- Use graph coarsening to merge operators (e.g., elementwise-ops)
- Use dynamic programming with recursive partitioning

Wang, Minjie, Chien-chin Huang, and Jinyang Li. "Supporting very large models using automatic dataflow graph partitioning." *EuroSys 2019*

FlexFlow

Operator	Parallelizable Dimensions		
	(S)ample	(A)ttribute	(P)arameter
1D pooling	sample	length, channel	
1D convolution	sample	length	channel
2D convolution	sample	height, width	channel
Matrix multiplication	sample		channel



Data Parallelism (S) **Model Parallelism (P)** **Hybrid Parallelism (S, P)** **Hybrid Parallelism (S, A, P)**

SOAP parallelism space

- Sample, Operator, Attribute, Parameter

Intra-op Parallelism

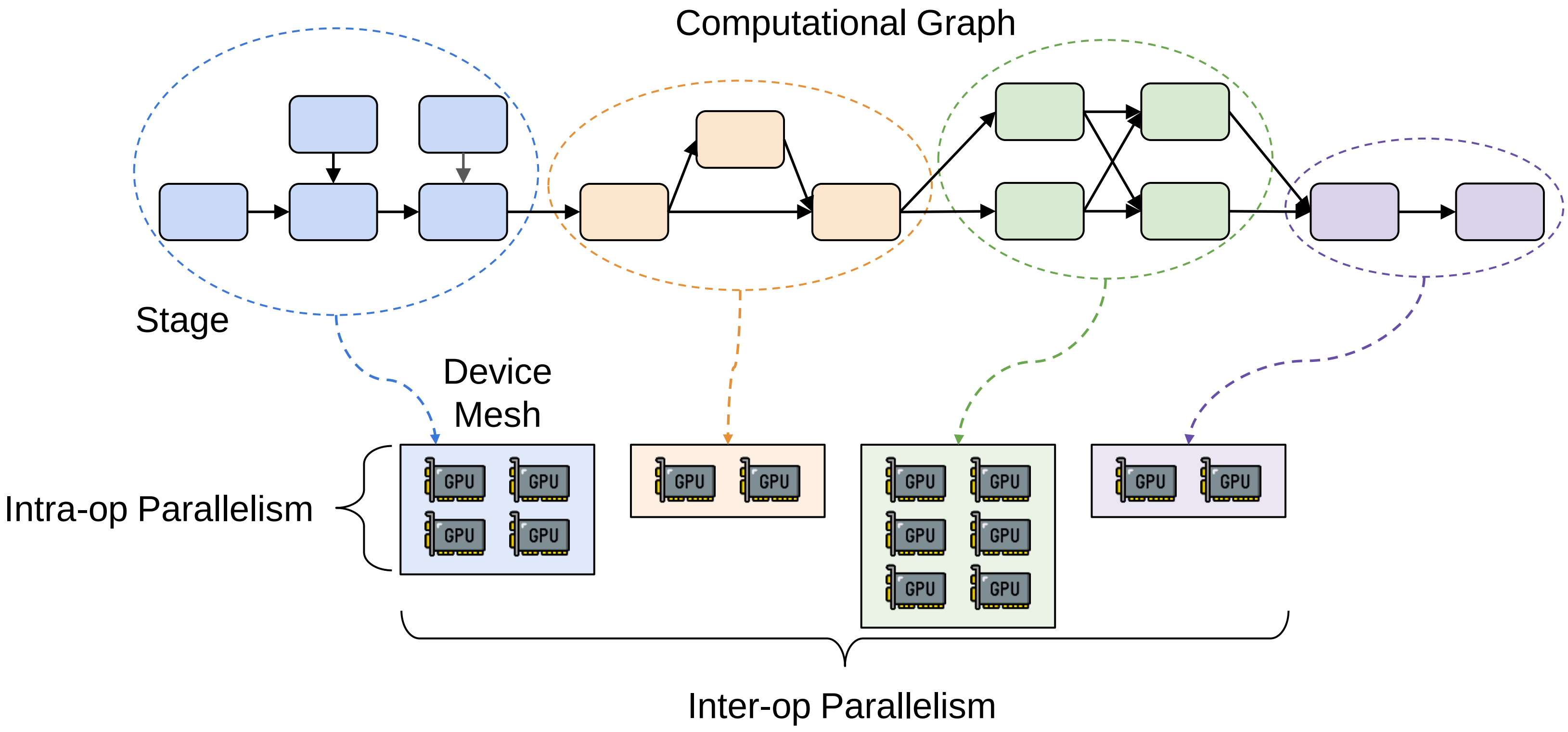
Inter-op Parallelism (w/o pipeline)

Simulator + MCMC for finding parallel strategies

- Details will be discussed later

Jia, Zhihao, Matei Zaharia, and Alex Aiken. "Beyond Data and Model Parallelism for Deep Neural Networks." *MLSys 2019*

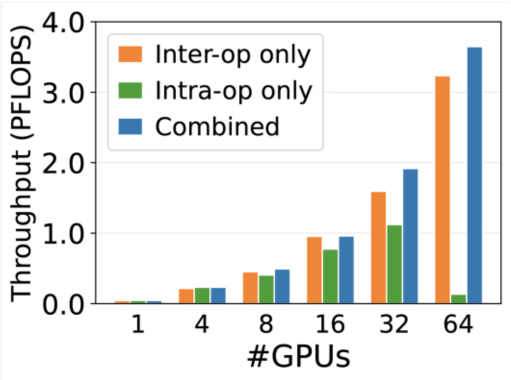
Combine Intra-op Parallelism and Inter-op Parallelism



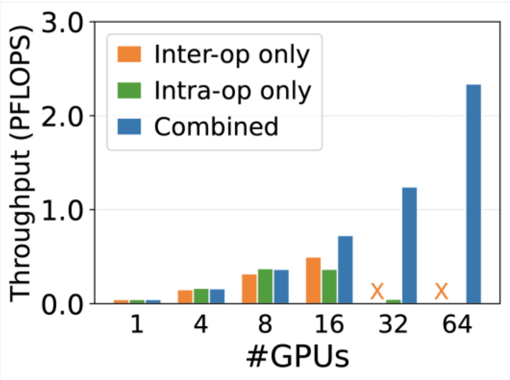
Narayanan, Deepak, et al. "Efficient large-scale language model training on gpu clusters using megatron-lm." *SC 2021*
Zheng, Lianmin, et al. "Alpa: Automating Inter-and Intra-Operator Parallelism for Distributed Deep Learning." *OSDI 2022*

Combine Intra-op Parallelism and Inter-op Parallelism

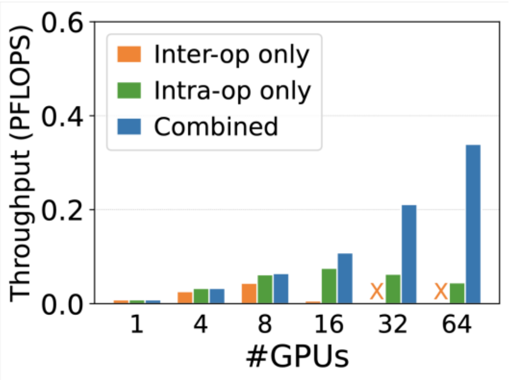
GPT



GShard MoE



Wide-ResNet



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

Zheng, Lianmin, et al. "Alpa: Automating Inter-and Intra-Operator Parallelism for Distributed Deep Learning." *OSDI 2022*

Intra-operator Parallelism Summary

- We can parallelize a single operator by exploiting its internal parallelism
- To do this for a whole computational graph, we need to choose strategies for all nodes in the graph to minimize the communication cost
- Intra-op and inter-op can be combined

Other Techniques for Training Large Models

System-level Memory Optimizations

- Rematerialization/Gradient Checkpointing
- Swapping

ML-level Optimizations

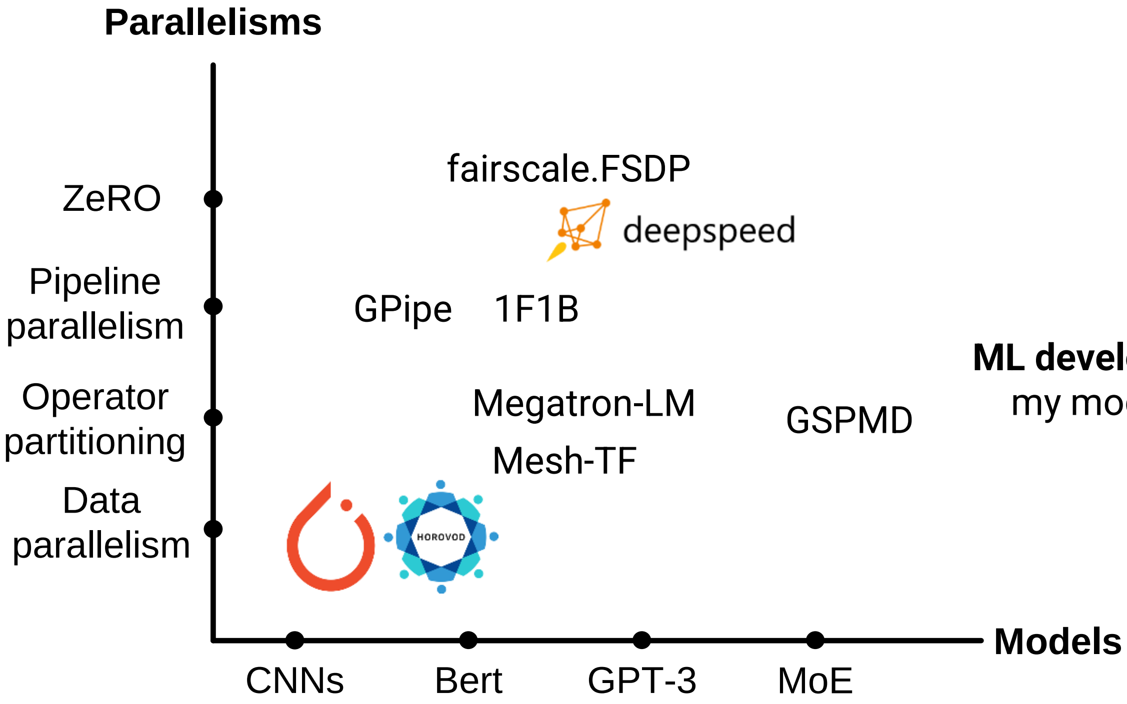
- Quantization
- Sparsification
- Low-rank approximation

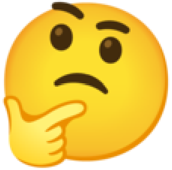
Chen, Tianqi, et al. "Training deep nets with sublinear memory cost." *arXiv 2016*
Rajbhandari, Samyam, et al. "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning." *SC* 2021.
Tang, Hanlin, et al. "1-bit adam: Communication efficient large-scale training with adam's convergence speed." *ICML* 2021.
Shazeer, Noam, and Mitchell Stern. "Adafactor: Adaptive learning rates with sublinear memory cost." *ICML* 2018.

Where We Are

- Motivation
- History
- Parallelism Overview
- Data parallelism
- Model parallelism
    - Inter-op parallelism
    - Intra-op parallelism
- **Auto-parallelization**

Auto-parallelization: Motivation





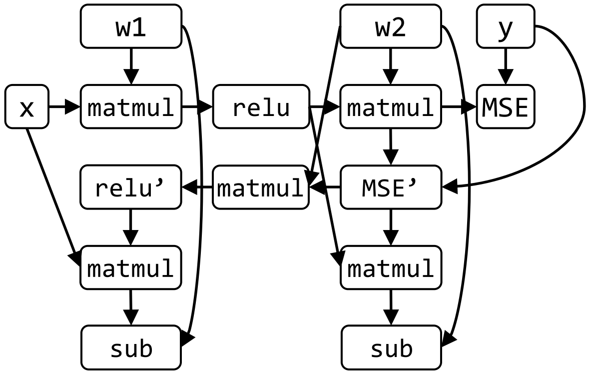
ML developer: which one is for my model and my cluster?

Auto-parallelization: Problem

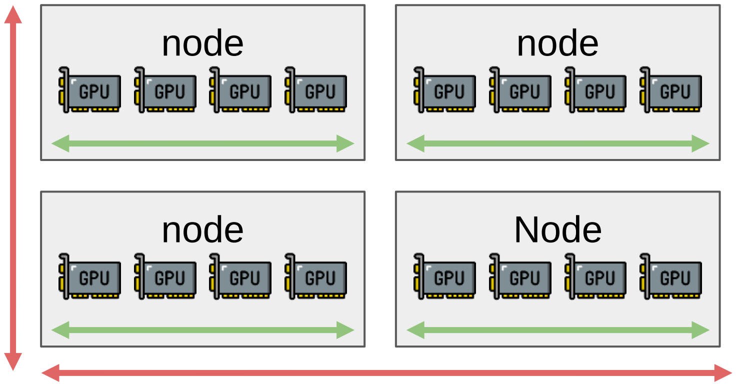
$$\max_{\text{strategy}} \ \text{Performance}(\text{Model}, \text{Cluster})$$
$$s.t. \ \text{strategy} \in \text{Inter-op} \cup \text{Intra-op}$$

Auto-parallelization: Problem

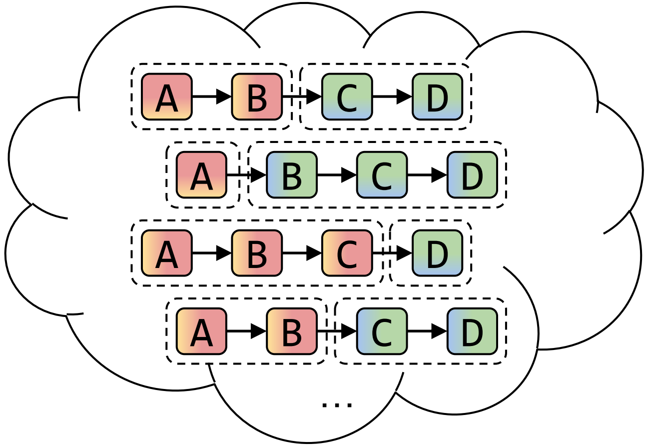
Model



Cluster



Strategy



The Search Space is Huge

#ops in a real model (nodes to color)	#op types (type of nodes)	#devices on a cluster (available colors)
100 - 10K	**80 - 200+**	**10s - 1000s**

Automatic Parallelization Methods

Search-based methods

- MCMC:
  - ➔ [Jia et al., 2018]
  - ➔ [Jia et al., 2019]
- Heuristics
  - ➔ [Fan et al., 2021]

Learning-based methods

- Reinforcement Learning:
  - ➔ [Mirhoseini et al., 2017]
  - ➔ [Mirhoseini et al., 2018]
  - ➔ [Addanki, et al., 2019]
- ML-based cost model:
  - ➔ [Chen et al., 2018],
  - ➔ [Zhou et al., 2020],
  - ➔ [Zhang, 2020]
- Bayesian optimization:
  - ➔ [Sergeev et al., 2018],
  - ➔ [Peng et al., 2019]

Optimization-based methods

- Dynamic programming
  - ➔ [Wang, et al., 2018]
  - ➔ [Narayanan, et al., 2019]
  - ➔ [Li, et al., 2021]
  - ➔ [Narayanan, et al., 2012]
  - ➔ [Tarnawski, et al., 2020]
  - ➔ [Tarnawski, et al., 2021]
- Integer linear programming
  - ➔ [Tarnawski, et al., 2020]
- Hierarchical Optimization
  - ➔ [Zheng, et al., 2022]

The complete list of references is available on the tutorial website

Automatic Parallelization Methods

Search-based methods

- MCMC:
  - ➔ [Jia et al., 2018]
  - ➔ **[Jia et al., 2019]**
- Heuristics
  - ➔ [Fan et al., 2021]

Learning-based methods

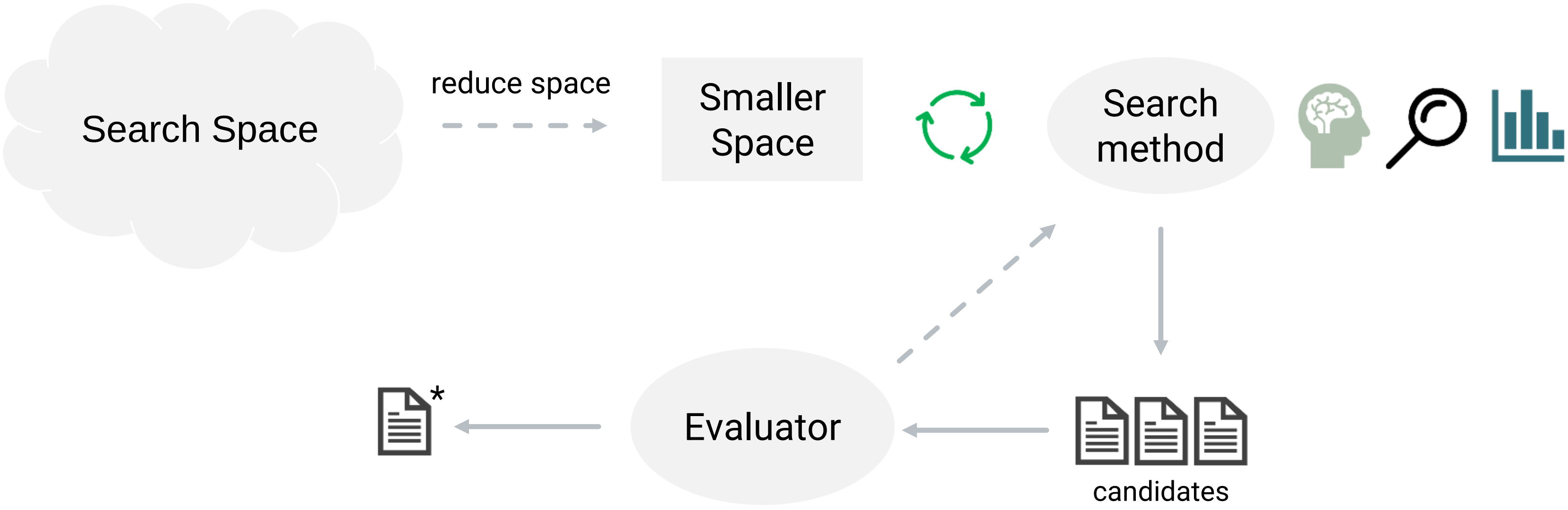
- Reinforcement Learning:
  - ➔ **[Mirhoseini et al., 2017]**
  - ➔ [Mirhoseini et al., 2018]
  - ➔ [Addanki, et al., 2019]
- ML-based cost model:
  - ➔ [Chen et al., 2018],
  - ➔ [Zhou et al., 2020],
  - ➔ [Zhang, 2020]
- Bayesian optimization:
  - ➔ [Sergeev et al., 2018],
  - ➔ [Peng et al., 2019]

Optimization-based methods

- Dynamic programming
  - ➔ [Wang, et al., 2018]
  - ➔ [Narayanan, et al., 2019]
  - ➔ [Li, et al., 2021]
  - ➔ [Narayanan, et al., 2012]
  - ➔ [Tarnawski, et al., 2020]
  - ➔ [Tarnawski, et al., 2021]
- Integer linear programming
  - ➔ [Tarnawski, et al., 2020]
- Hierarchical optimization
  - ➔ **[Zheng, et al., 2022]**

The complete list of references is available on the tutorial website

General Recipe



Automatic Parallelization Methods

Search-based methods

- MCMC:
  - ➔ [Jia et al., 2018]
  - ➔ **[Jia et al., 2019]**
- Heuristics
  - ➔ [Fan et al., 2021]

Learning-based methods

- Reinforcement Learning:
  - ➔ **[Mirhoseini et al., 2017]**
  - ➔ [Mirhoseini et al., 2018]
  - ➔ [Addanki, et al., 2019]
- ML-based cost model:
  - ➔ [Chen et al., 2018],
  - ➔ [Zhou et al., 2020],
  - ➔ [Zhang, 2020]
- Bayesian optimization:
  - ➔ [Sergeev et al., 2018],
  - ➔ [Peng et al., 2019]

Optimization-based methods

- Dynamic programming
  - ➔ [Wang, et al., 2018]
  - ➔ [Narayanan, et al., 2019]
  - ➔ [Li, et al., 2021]
  - ➔ [Narayanan, et al., 2012]
  - ➔ [Tarnawski, et al., 2020]
  - ➔ [Tarnawski, et al., 2021]
- Integer linear programming
  - ➔ [Tarnawski, et al., 2020]
- Hierarchical optimization
  - ➔ **[Zheng, et al., 2022]**

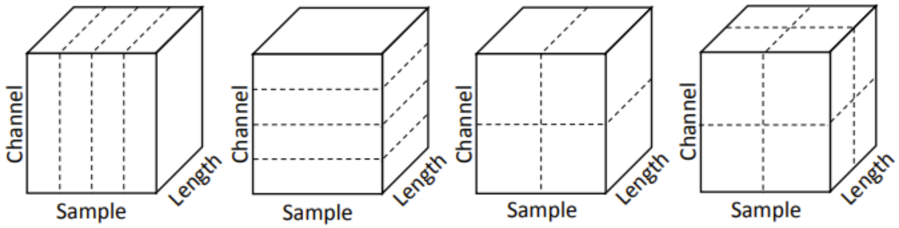
The complete list of references is available on the tutorial website

FlexFlow: Search Space

SOAP Space

- S (Sample) - sample dimension
- O (Operation) – operator placement
- P (Parameter) – split the parameter
- A (Attribute) – the rest

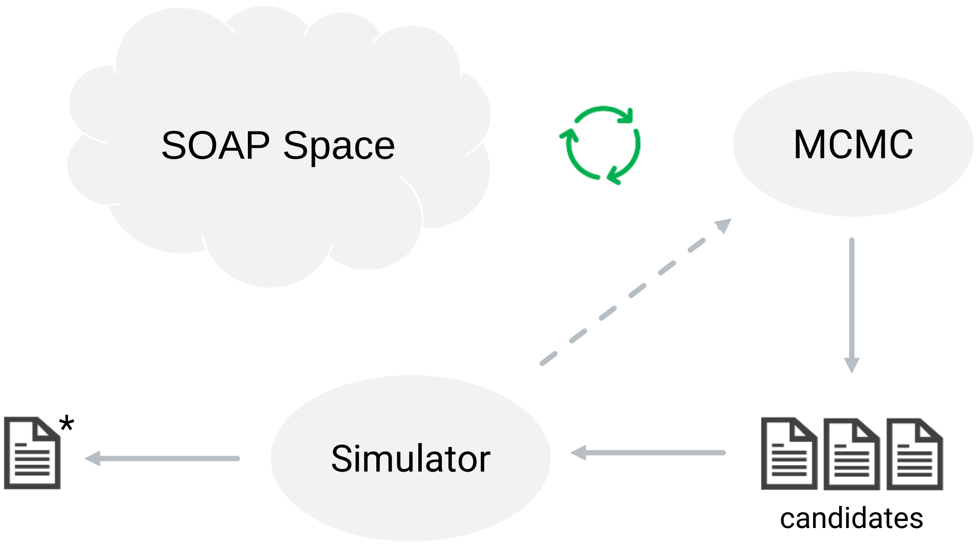
Operator	**Parallelizable Dimensions**		
	(S)ample	**(A)ttribute**	**(P)arameter**
1D pooling	sample	length, channel	
1D convolution	sample	length	channel
2D convolution	sample	height, width	channel
Matrix multiplication	sample		channel



Data Parallelism (S) **Model Parallelism (P)** **Hybrid Parallelism (S, P)** **Hybrid Parallelism (S, A, P)**

Jia, et al. "Beyond Data and Model Parallelism for Deep Neural Networks". *MLSys* 2019.

FlexFlow: Workflow



Jia, et al. "Beyond Data and Model Parallelism for Deep Neural Networks". *MLSys* 2019.

Results Discussion

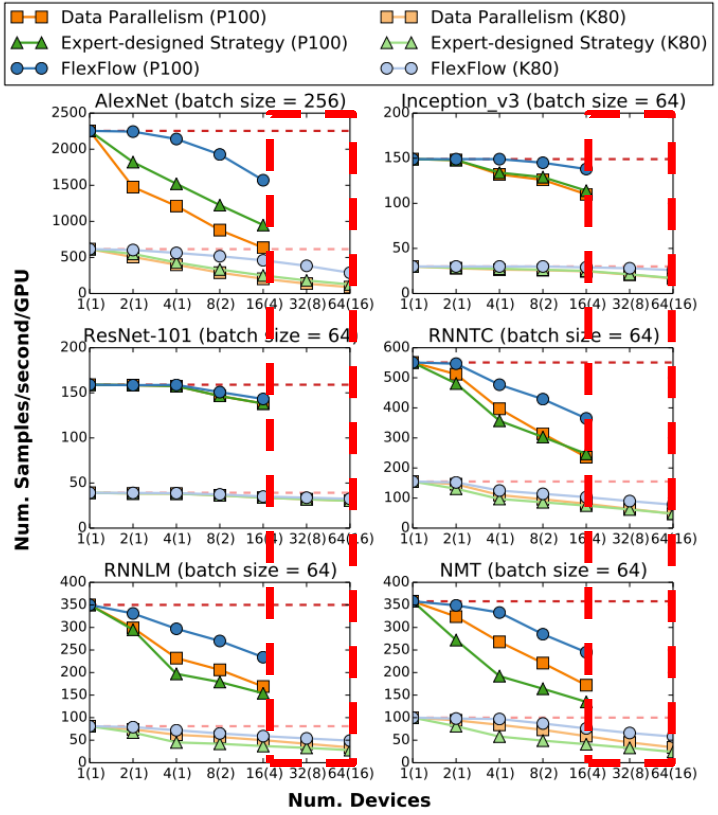


Figure from [Jia et al., MLsys 2019]

Jia, et al. "Beyond Data and Model Parallelism for Deep Neural Networks". *MLSys* 2019.

Automatic Parallelization Methods

Search-based methods

- MCMC:
  - ➔ [Jia et al., 2018]
  - ➔ [Jia et al., 2019]
- Heuristics
  - ➔ [Fan et al., 2021]

The complete list of references is available on the tutorial website

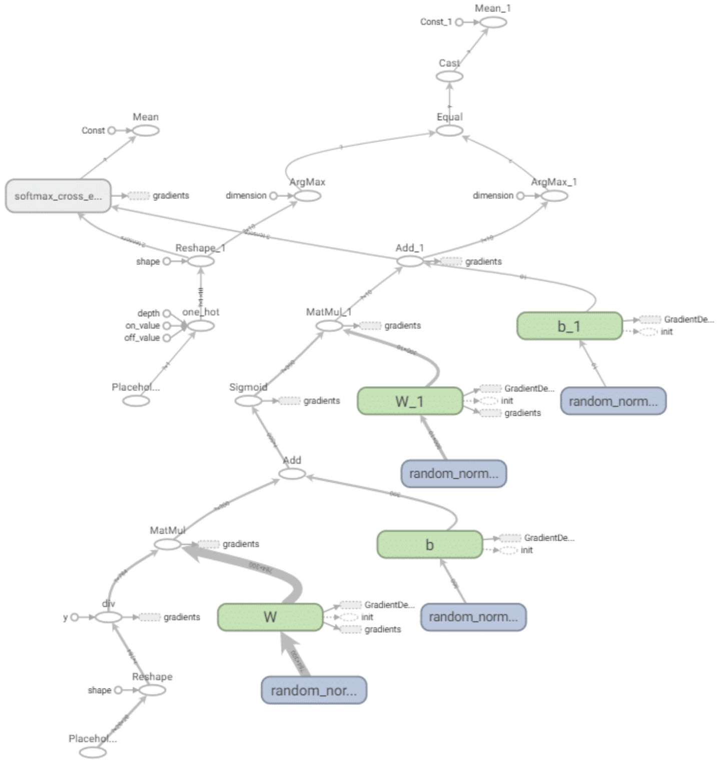
Learning-based methods

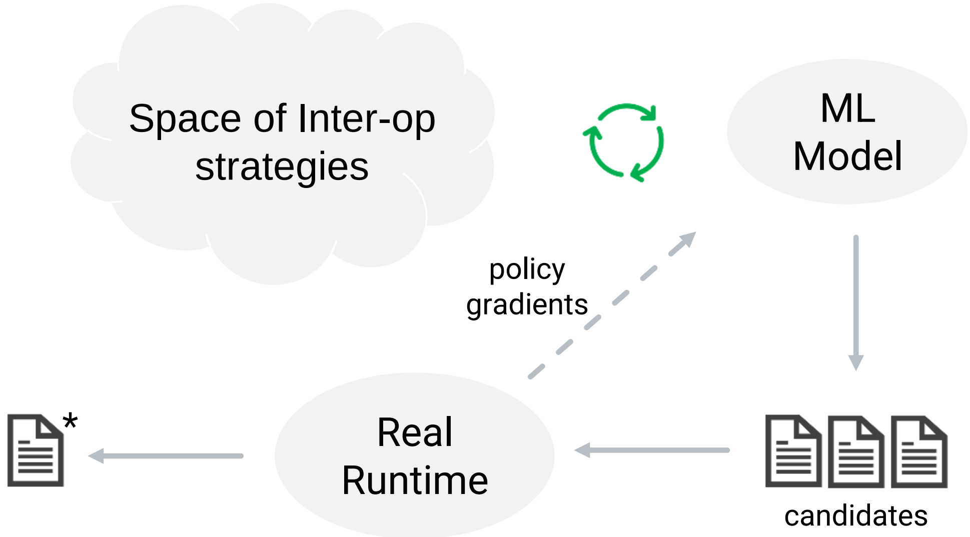
- Reinforcement Learning:
  - ➔ **[Mirhoseini et al., 2017]**
  - ➔ [Mirhoseini et al., 2018]
  - ➔ [Addanki, et al., 2019]
- ML-based cost model:
  - ➔ [Chen et al., 2018],
  - ➔ [Zhou et al., 2020],
  - ➔ [Zhang, 2020]
- Bayesian optimization:
  - ➔ [Sergeev et al., 2018],
  - ➔ [Peng et al., 2019]

Optimization-based methods

- Dynamic programming
  - ➔ [Wang, et al., 2018]
  - ➔ [Narayanan, et al., 2019]
  - ➔ [Li, et al., 2021]
  - ➔ [Narayanan, et al., 2012]
  - ➔ [Tarnawski, et al., 2020]
  - ➔ [Tarnawski, et al., 2021]
- Integer linear programming
  - ➔ [Tarnawski, et al., 2020]
- Hierarchical optimization
  - ➔ **[Zheng, et al., 2022]**

ColocRL (a.k.a. Device Placement Optimization)





Mirhoseini, et al. "Device Placement Optimization with Reinforcement Learning." *ICML* 2017.

ColocRL: Model

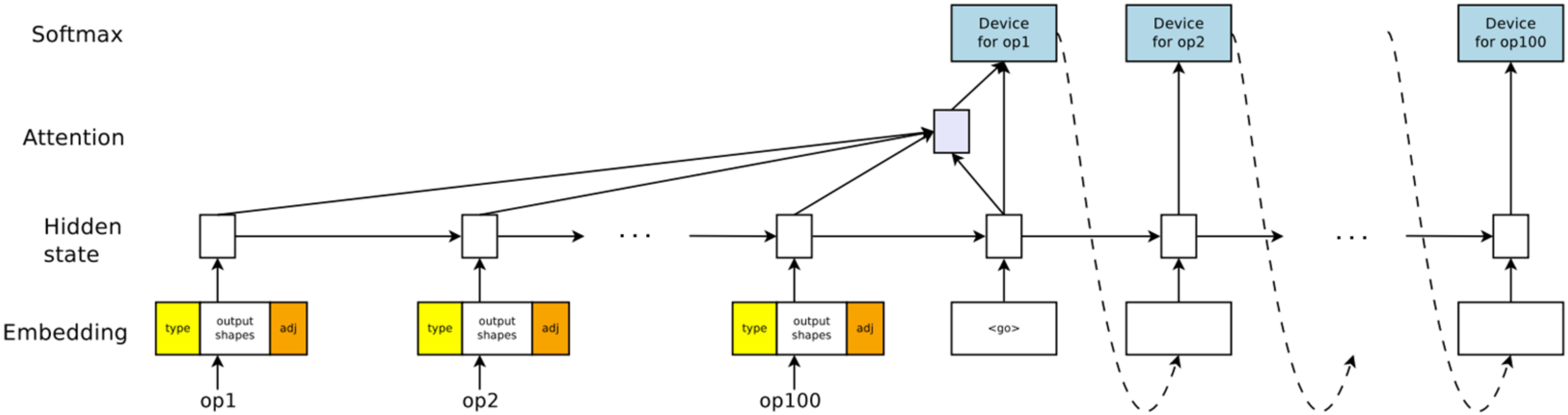


Figure from [Mirhoseini et al., ICML 2017]

Mirhoseini, et al. "Device Placement Optimization with Reinforcement Learning." *ICML* 2017.

ColocRL: Training

$$\mathbb{E}_{\mathcal{P}\sim\pi(\mathcal{P}\mid\mathcal{G};\theta)}[R(\mathcal{P})||\mathcal{G}]$$

$\mathcal{G}$: computational graph

$\mathcal{R}(\mathcal{P})$: Real runtime of a placement

$\pi(\cdot)$: output distributed of the RNN

Mirhoseini, et al. "Device Placement Optimization with Reinforcement Learning." *ICML* 2017.

ColocRL: Other Improvement

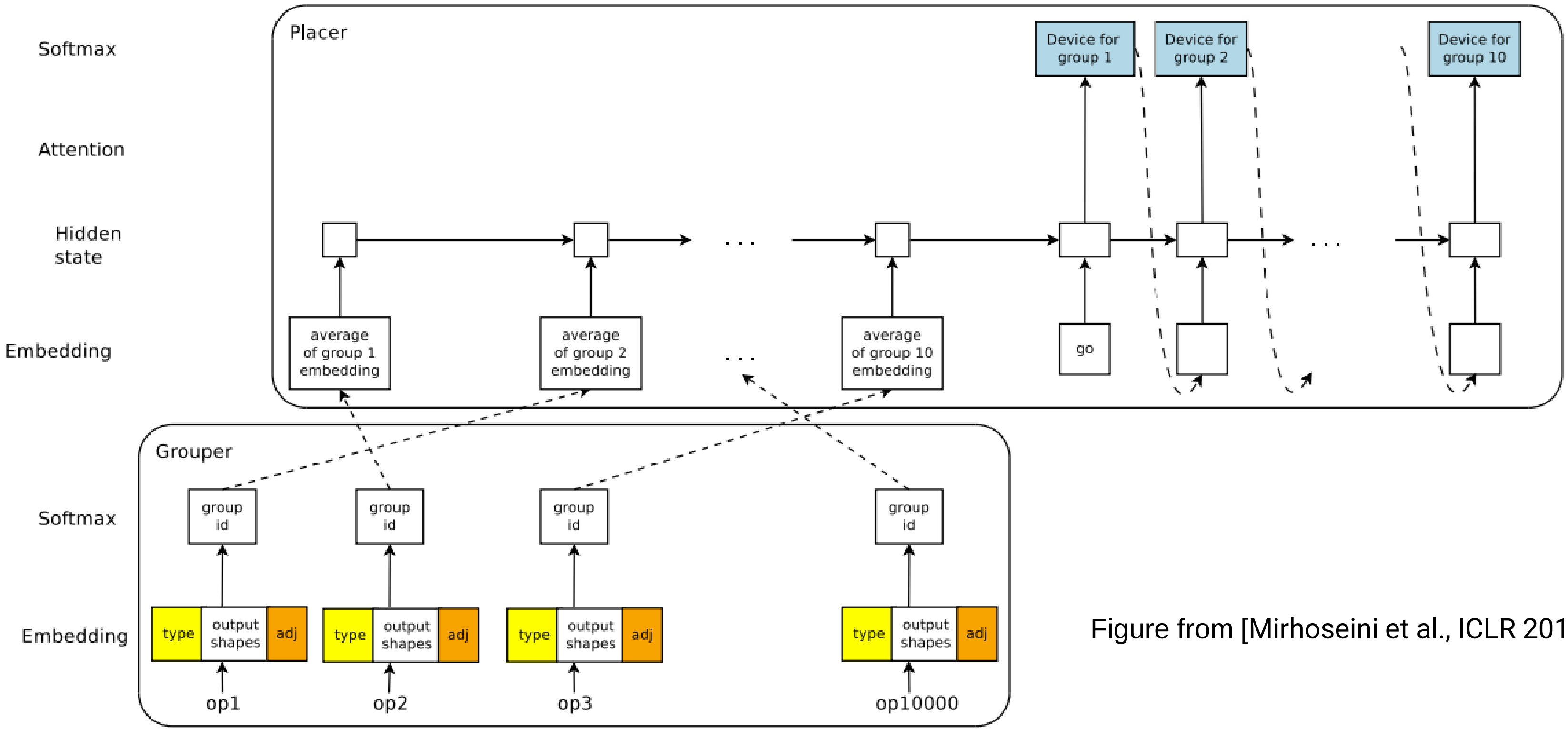
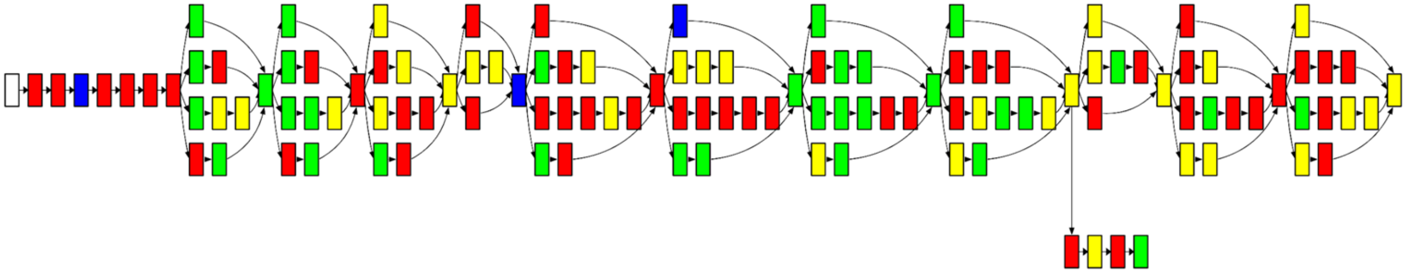


Figure from [Mirhoseini et al., ICLR 2018]

Mirhoseini, et al. "A Hierarchical Model for Device Placement." *ICLR* 2018.

Results Discussion



Tasks	Single-CPU	Single-GPU	#GPUs	Scotch	MinCut	Expert	RL-based	Speedup
RNNLM (batch 64)	6.89	**1.57**	2	13.43	11.94	3.81	**1.57**	0.0%
			4	11.52	10.44	4.46	**1.57**	0.0%
NMT (batch 64)	10.72	OOM	2	14.19	11.54	4.99	**4.04**	23.5%
			4	11.23	11.78	4.73	**3.92**	20.6%
Inception-V3 (batch 32)	26.21	**4.60**	2	25.24	22.88	11.22	**4.60**	0.0%
			4	23.41	24.52	10.65	**3.85**	19.0%

Figure and table from [Mirhoseini et al., ICML 2017]

Automatic Parallelization Methods

Search-based methods

- MCMC:
  - ➔ [Jia et al., 2018]
  - ➔ [Jia et al., 2019]
- Heuristics
  - ➔ [Fan et al., 2021]

Learning-based methods

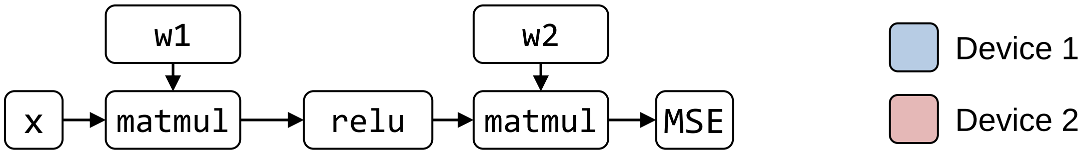
- Reinforcement Learning:
  - ➔ [Mirhoseini et al., 2017]
  - ➔ [Mirhoseini et al., 2018]
  - ➔ [Addanki, et al., 2019]
- ML-based cost model:
  - ➔ [Chen et al., 2018],
  - ➔ [Zhou et al., 2020],
  - ➔ [Zhang, 2020]
- Bayesian optimization:
  - ➔ [Sergeev et al., 2018],
  - ➔ [Peng et al., 2019]

Optimization-based methods

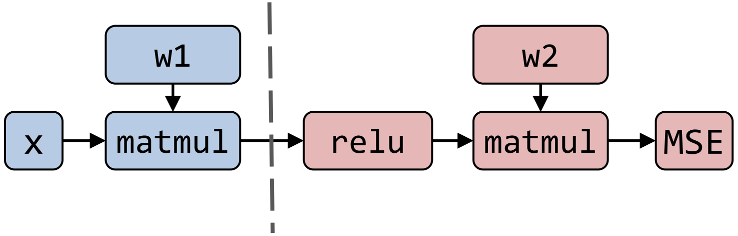
- Dynamic programming
  - ➔ [Wang, et al., 2018]
  - ➔ [Narayanan, et al., 2019]
  - ➔ [Li, et al., 2021]
  - ➔ [Narayanan, et al., 2012]
  - ➔ [Tarnawski, et al., 2020]
  - ➔ [Tarnawski, et al., 2021]
- Integer linear programming
  - ➔ [Tarnawski, et al., 2020]
- Hierarchical optimization
  - ➔ **Alpa [Zheng, et al., 2022]**

The complete list of references is available on the tutorial website

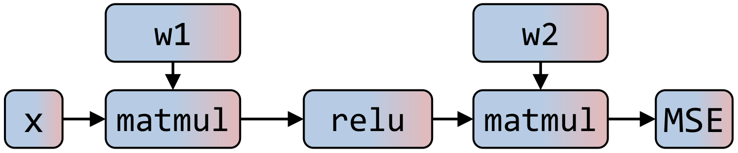
Optimization-based Method: Alpa



Inter-op parallelism



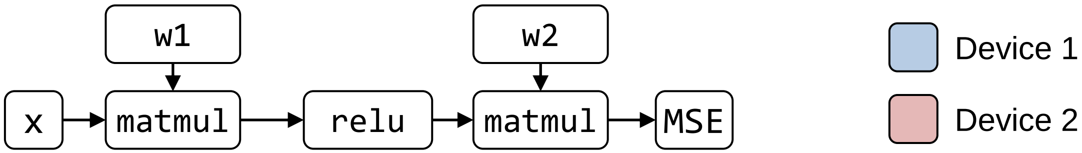
Intra-op parallelism



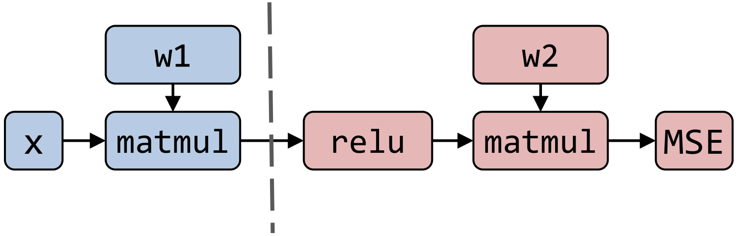
Trade-off

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

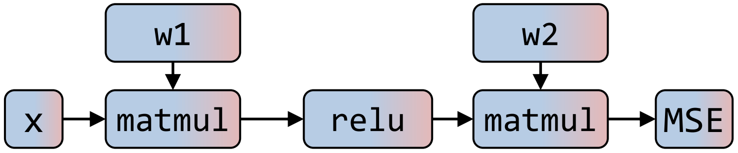
Alpa Rationale

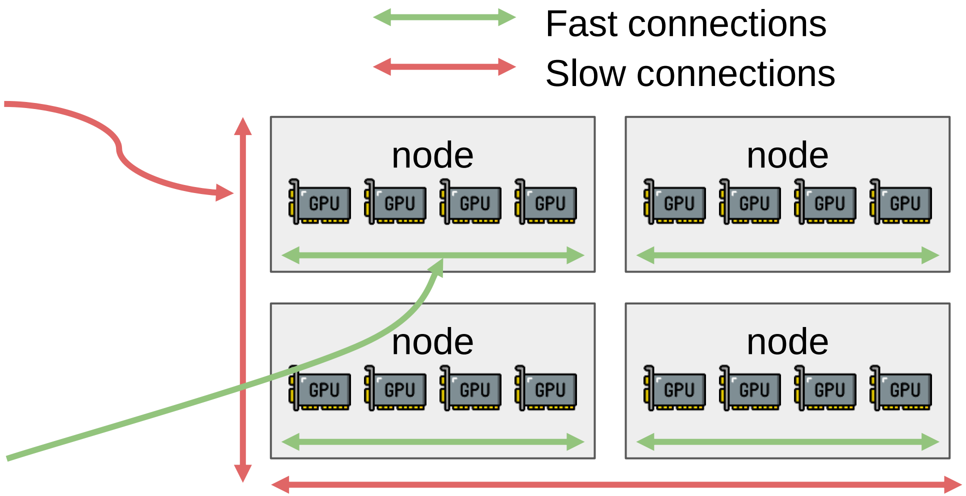


Inter-op parallelism

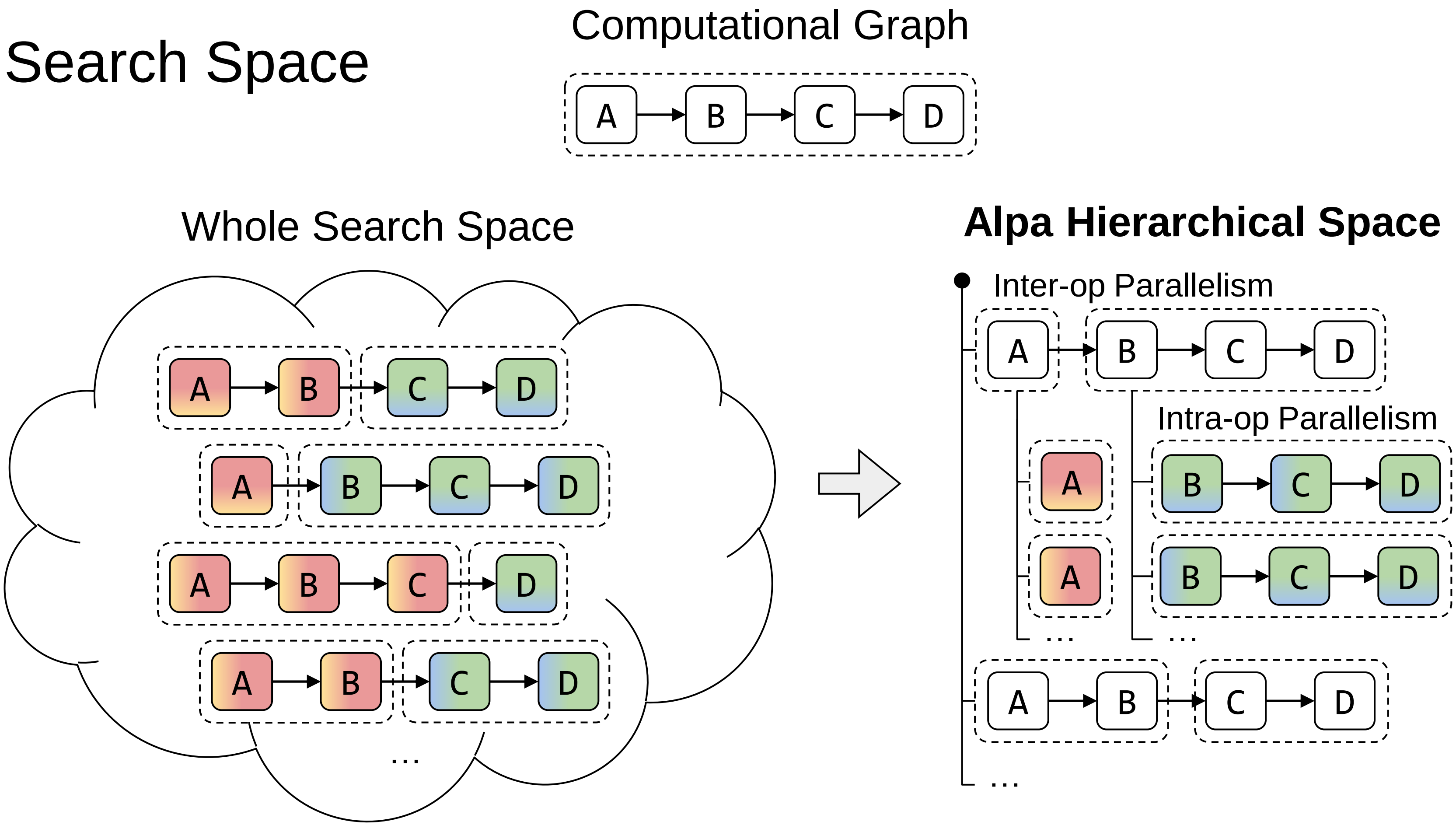


Intra-op parallelism

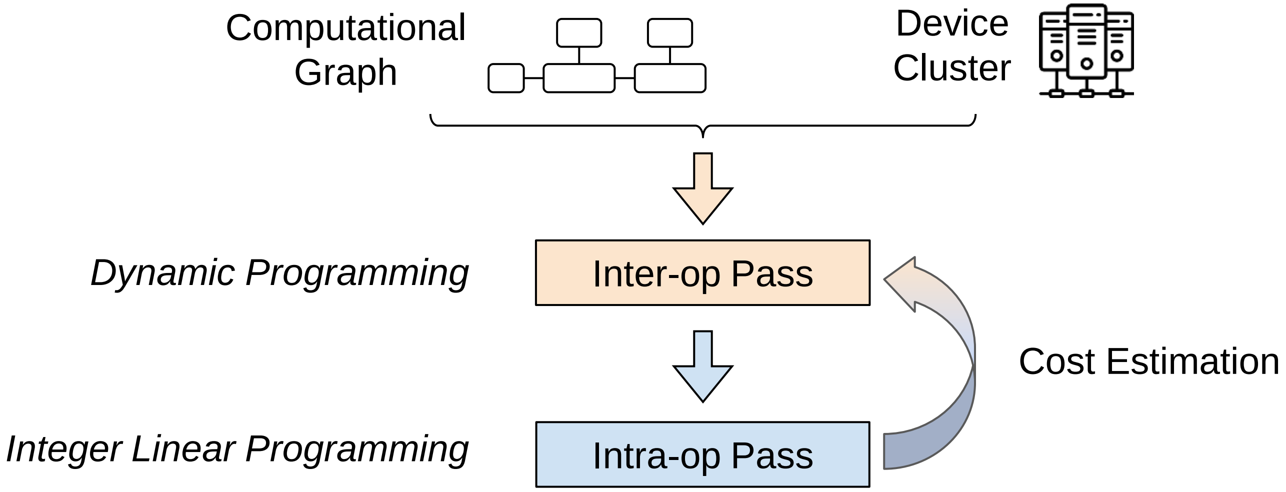




Search Space

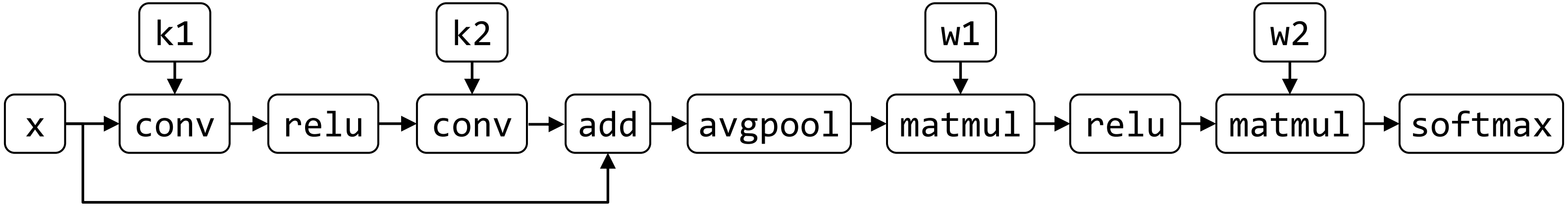


Alpa Compiler: Hierarchical Optimization

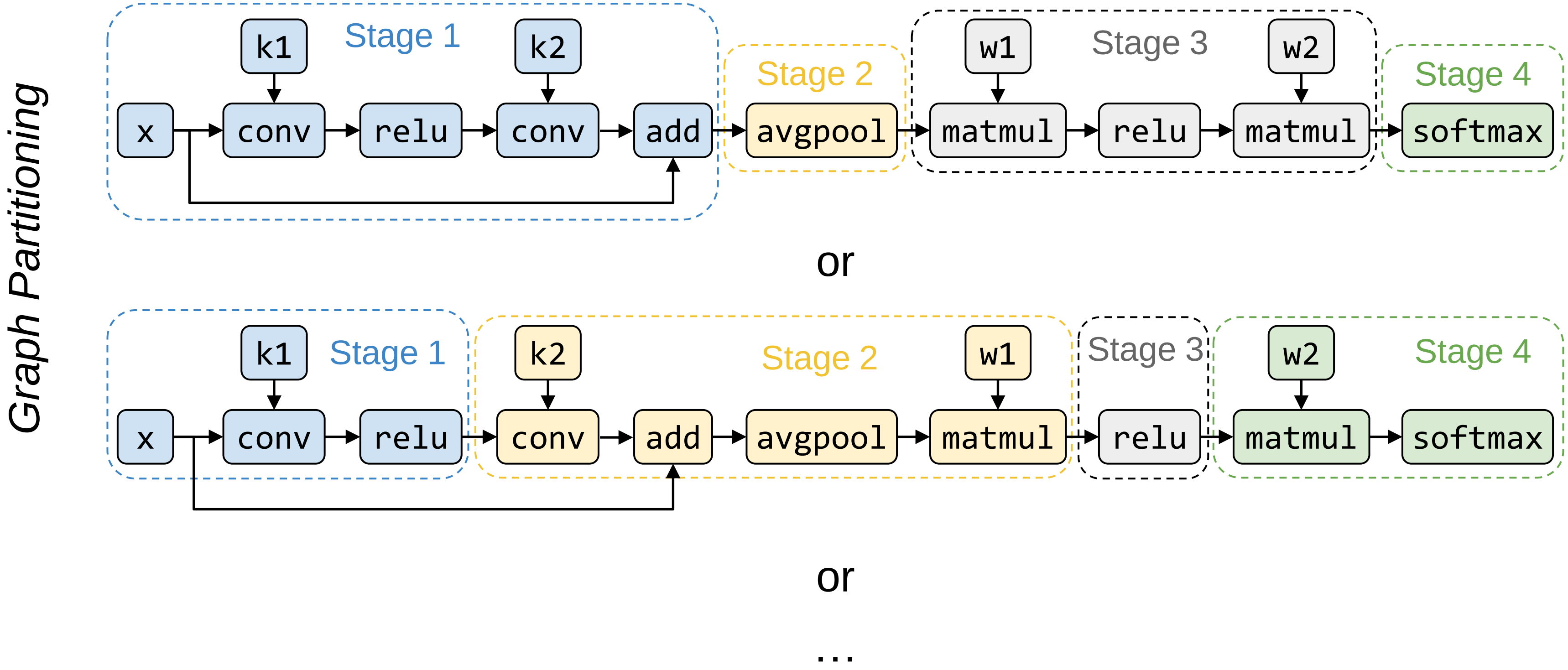


Inter-op Pass

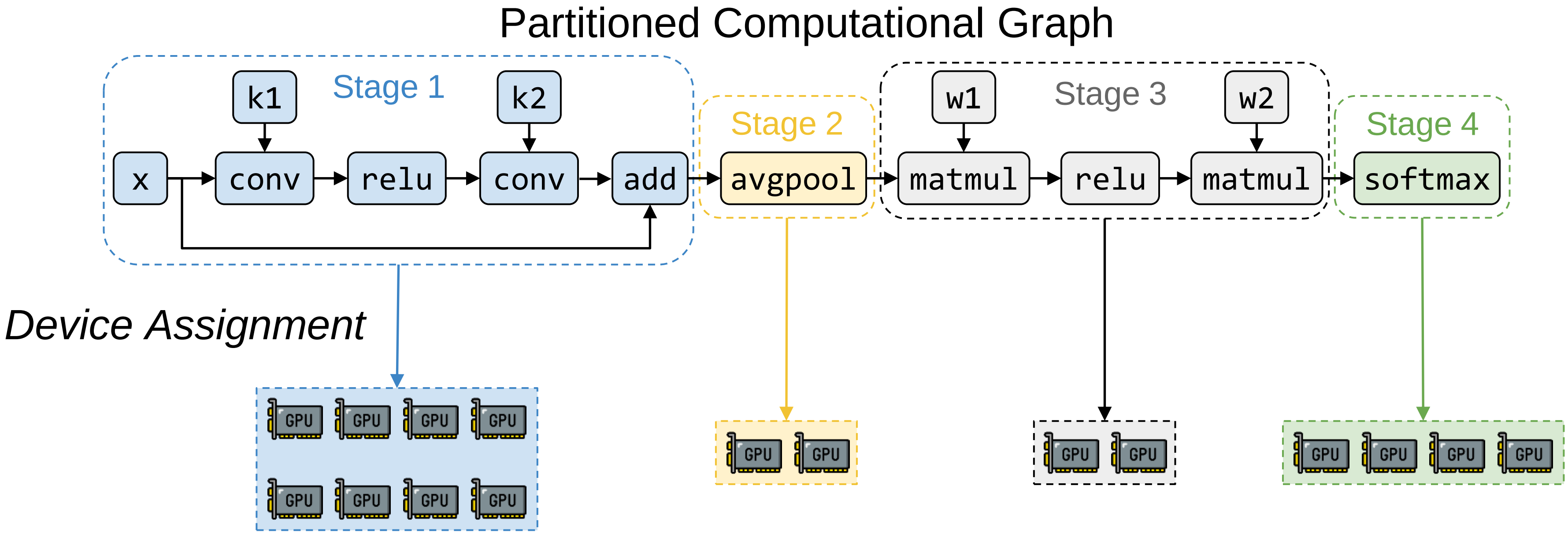
Computational Graph



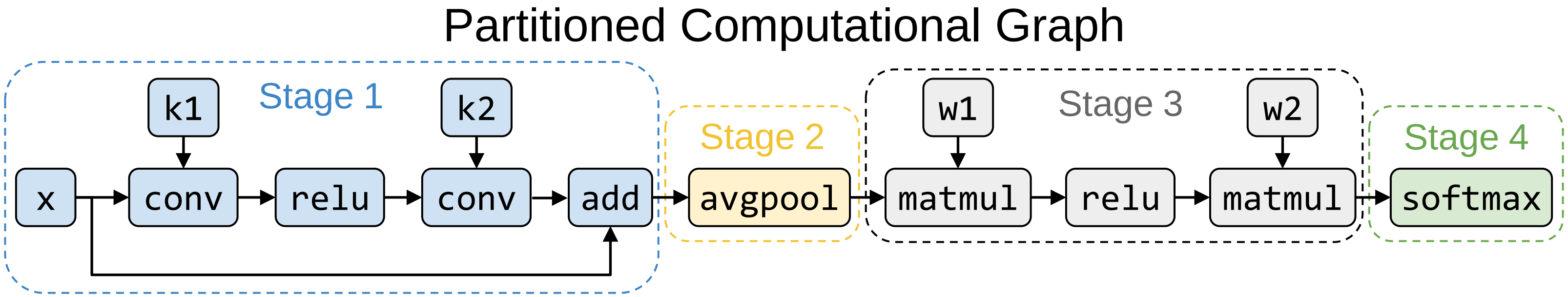
Inter-op Pass

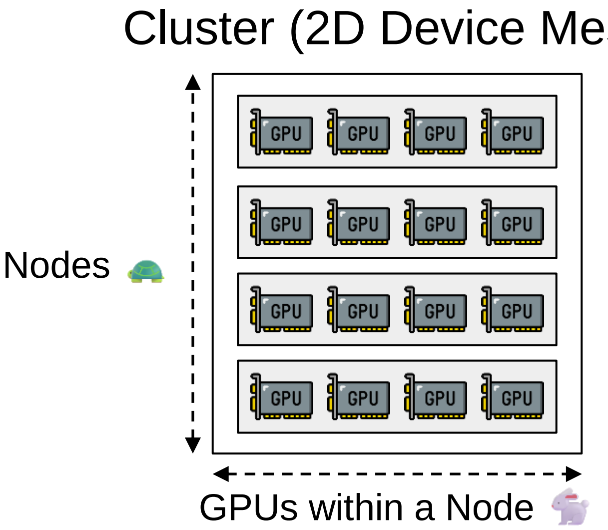


Inter-op Pass

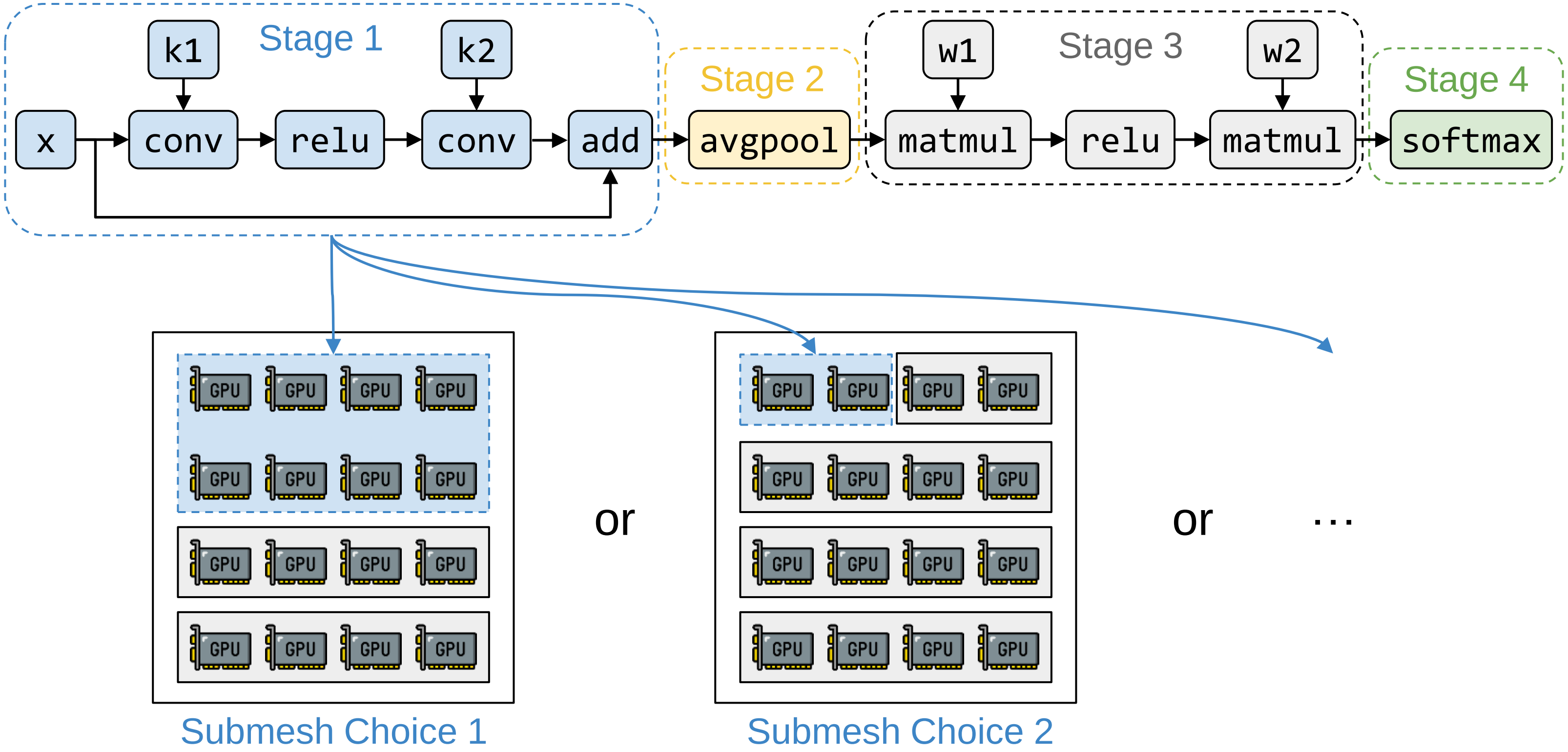


Inter-op Pass

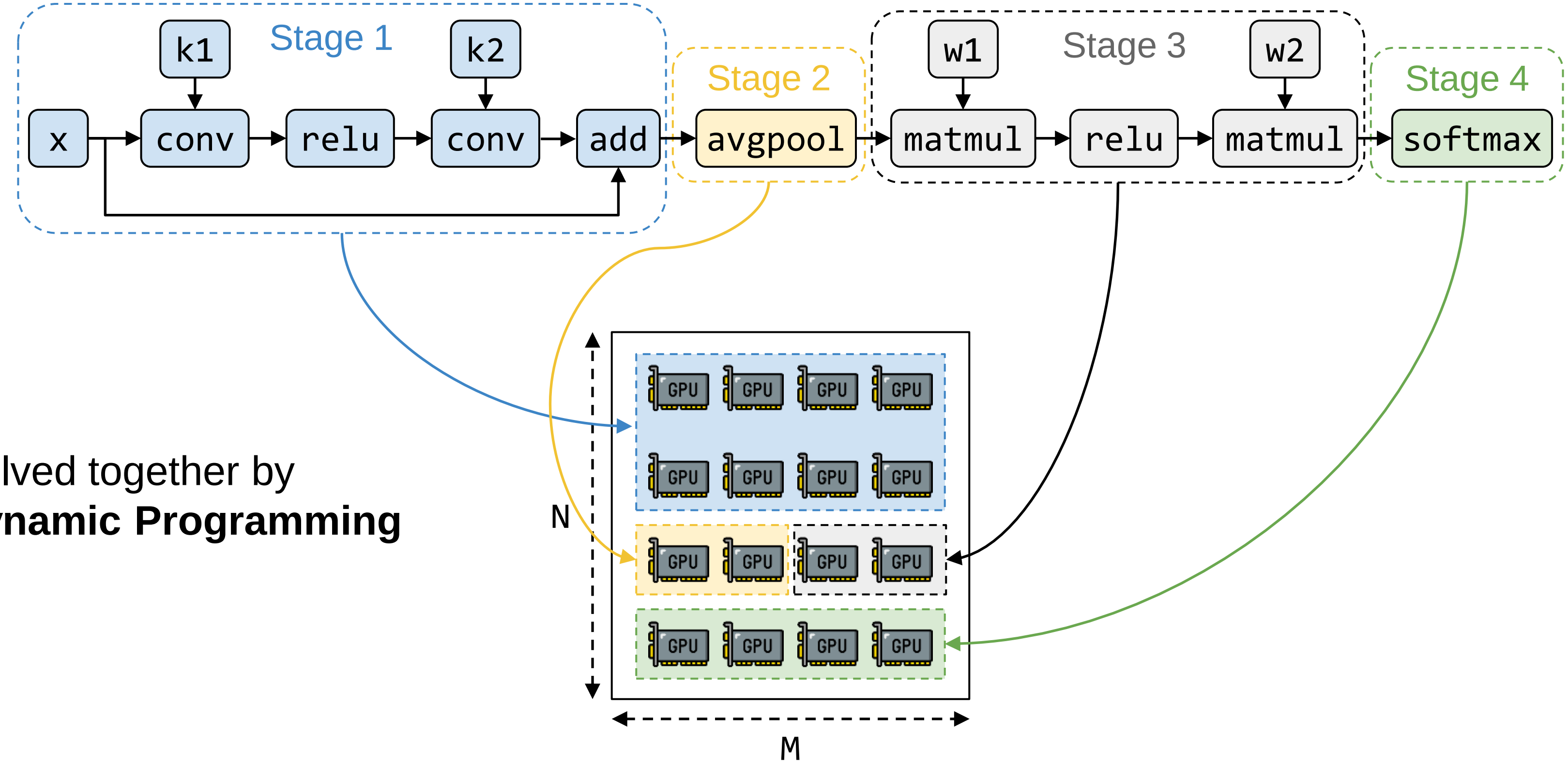




Inter-op Pass

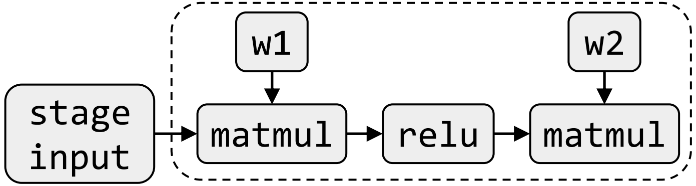


Inter-op Pass

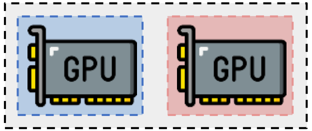


Solved together by
Dynamic Programming

Intra-op Pass

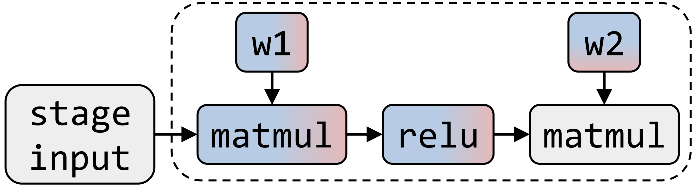


Stage



Submesh

Solved by
Integer Linear Programming

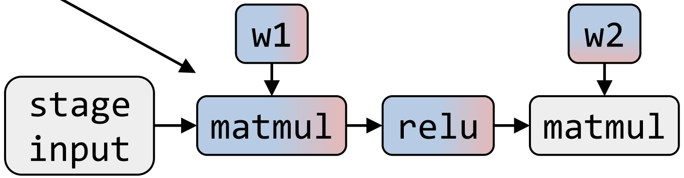


Stage with intra-operator parallelization

Intra-op Pass

Integer Linear Programming Formulation

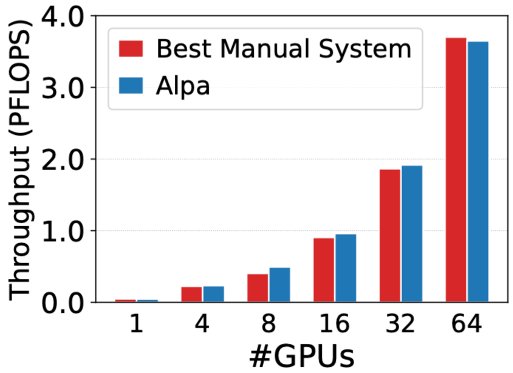
Decision vector
Parallel strategies of each operator



Minimize Computation cost + Communication cost

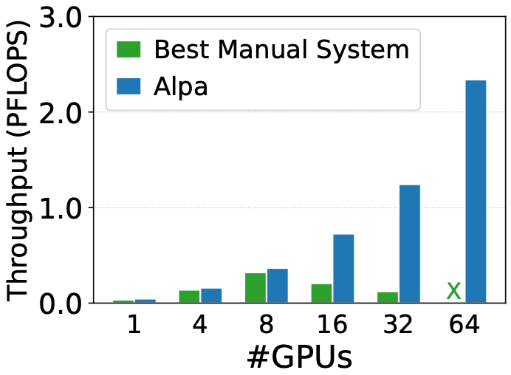
Evaluation: Comparing with Previous Works

GPT (up to 39B)



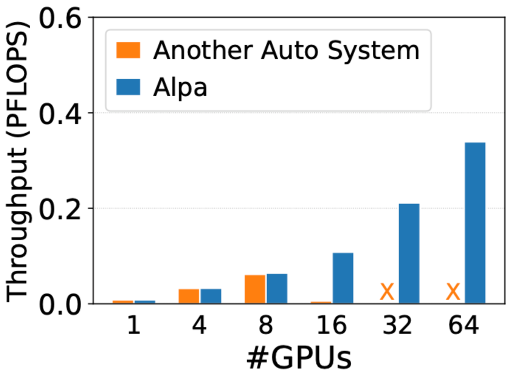
Match specialized manual systems.

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).

Automatic Parallelization Methods

Search-based methods

✅ Easy to extend the search space
✅ No training cost
❌ High inference cost
❌ Not explainable
❌ No optimality guarantee

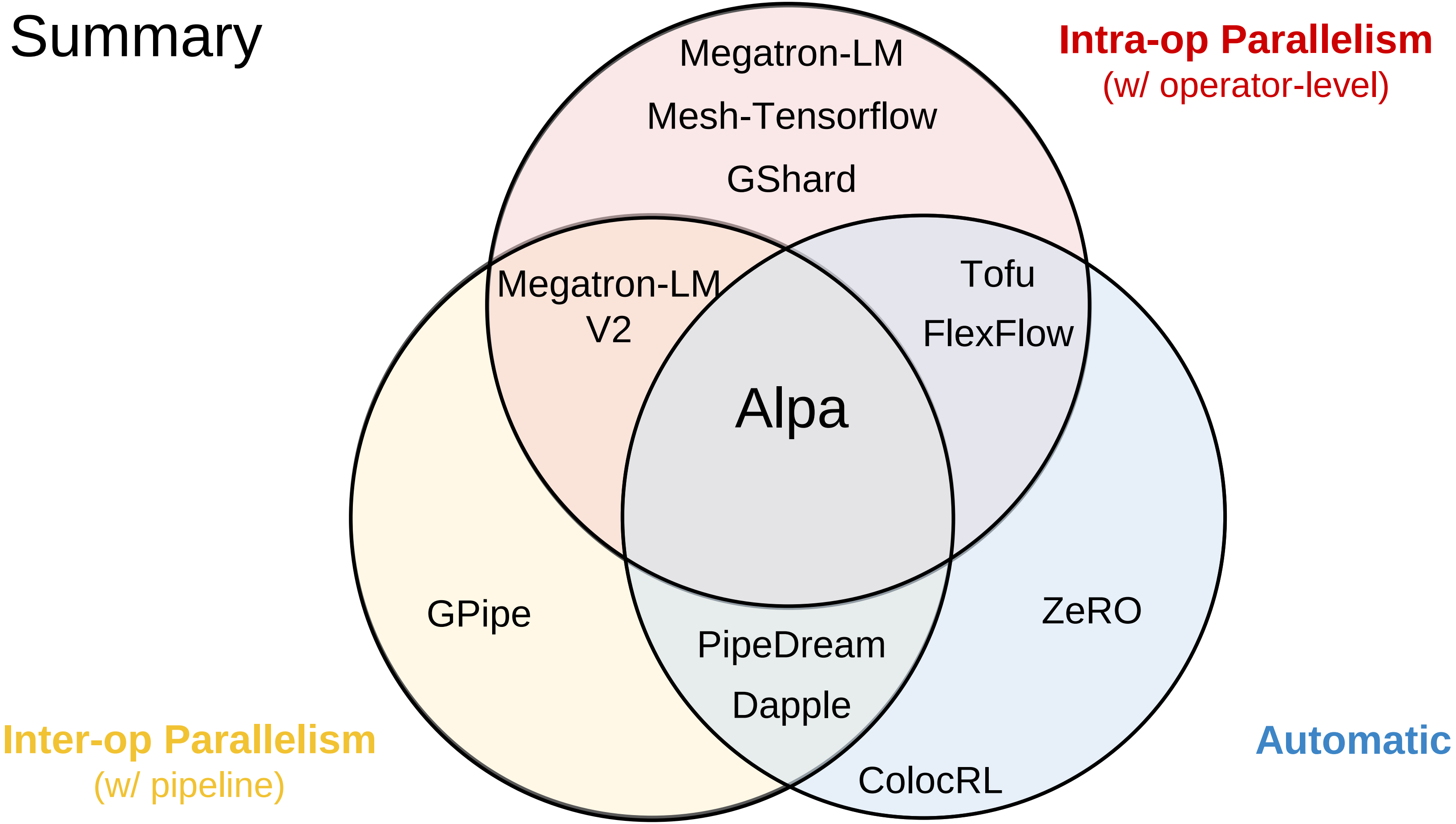
Learning-based methods

✅ Easy to extend the search space
❌ High training cost
✅ Low inference cost
❌ Not explainable
❌ No optimality guarantee

Optimization-based methods

❌ Non-trivial to extend the search space
✅ No training cost
✅ Medium inference cost
✅ Explainable
✅ Some optimality guarantee

Summary


Intra-op Parallelism
(w/ operator-level)
Megatron-LM
Mesh-Tensorflow
GShard
Megatron-LM V2
Tofu
FlexFlow
Alpa
GPipe
PipeDream
Dapple
ZeRO
ColocRL
Inter-op Parallelism
(w/ pipeline)
Automatic

Future ML Systems and Challenges

More heterogeneous models?

Better objectives?

Maximize Performance(
strategy

Any Goal

Model Configs,

Cluster Configs)

What’s the right amount of cluster resources?

Systems to support more parallelisms and optimizations

Easily scale up to 1000s of GPUs/TPUs?

Better Objectives

$\text{Maximize}_{\text{strategy}}$ Performance(Model Configs, Cluster Configs)

Maximize ***System Performance***	Maximize ***User Performance***
Maximize Throughput	Maximize Goodput
Maximize utilization	Maximize $$$ utilization