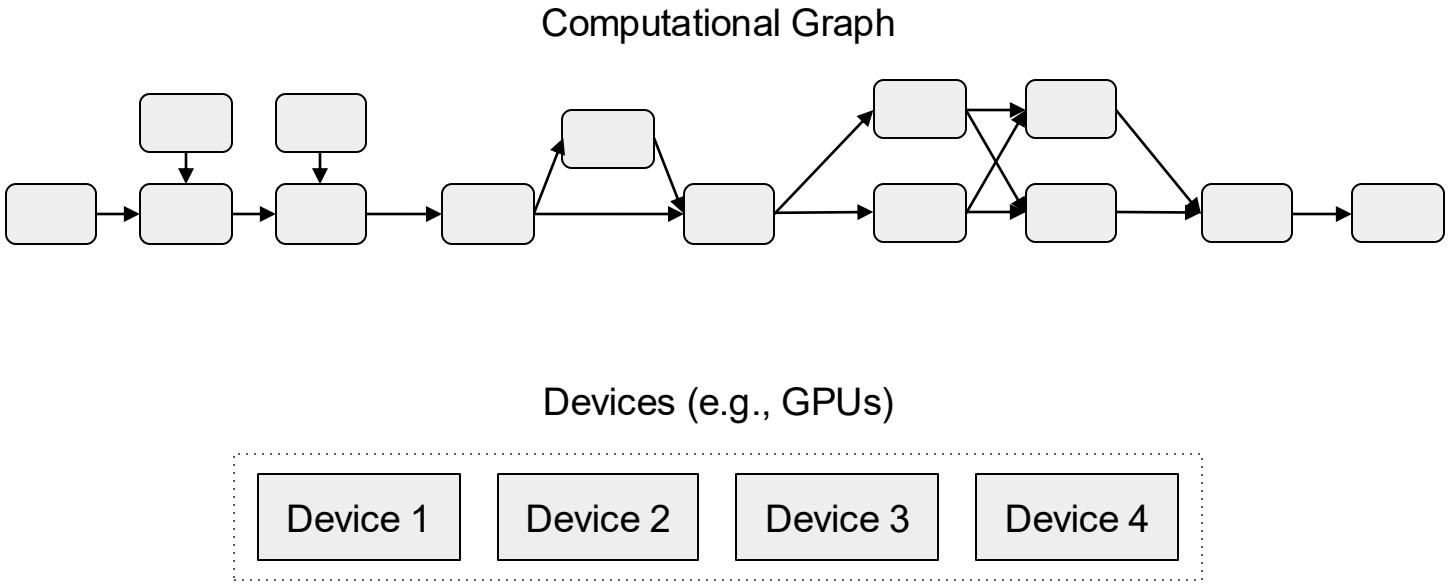
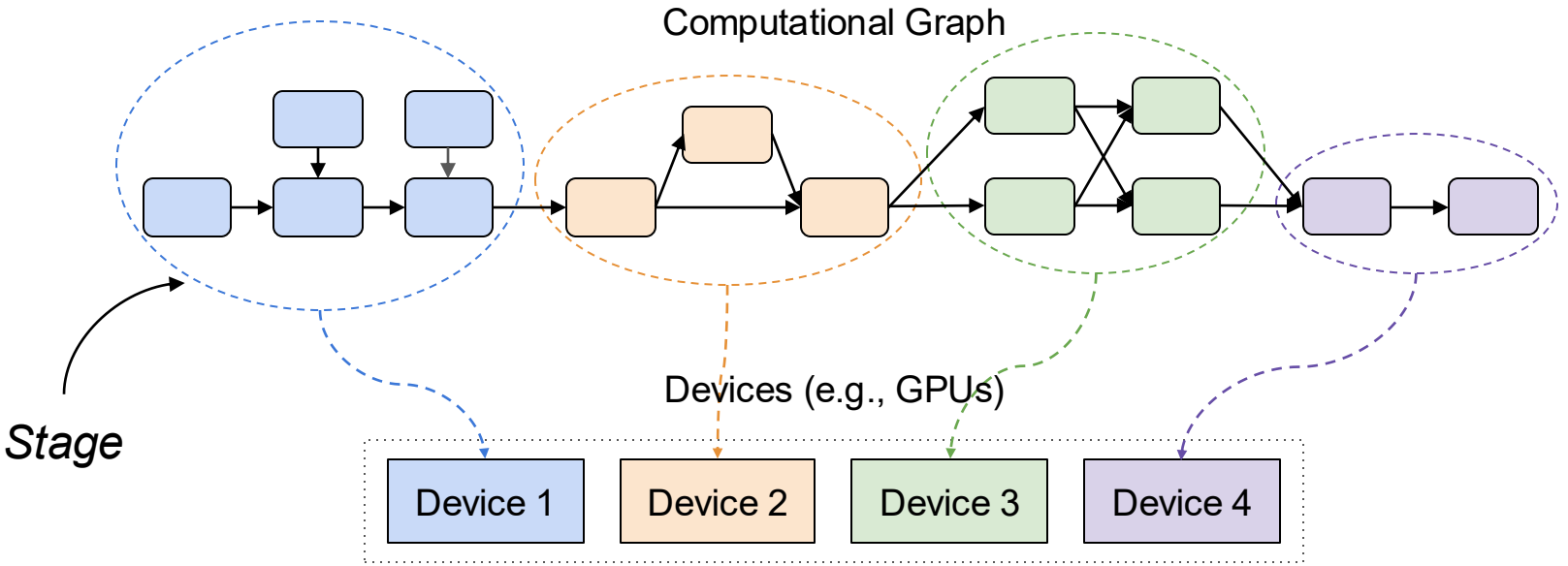
Model Parallelism

- Model parallelism
  - **Inter-op parallelism**
  - Intra-op parallelism

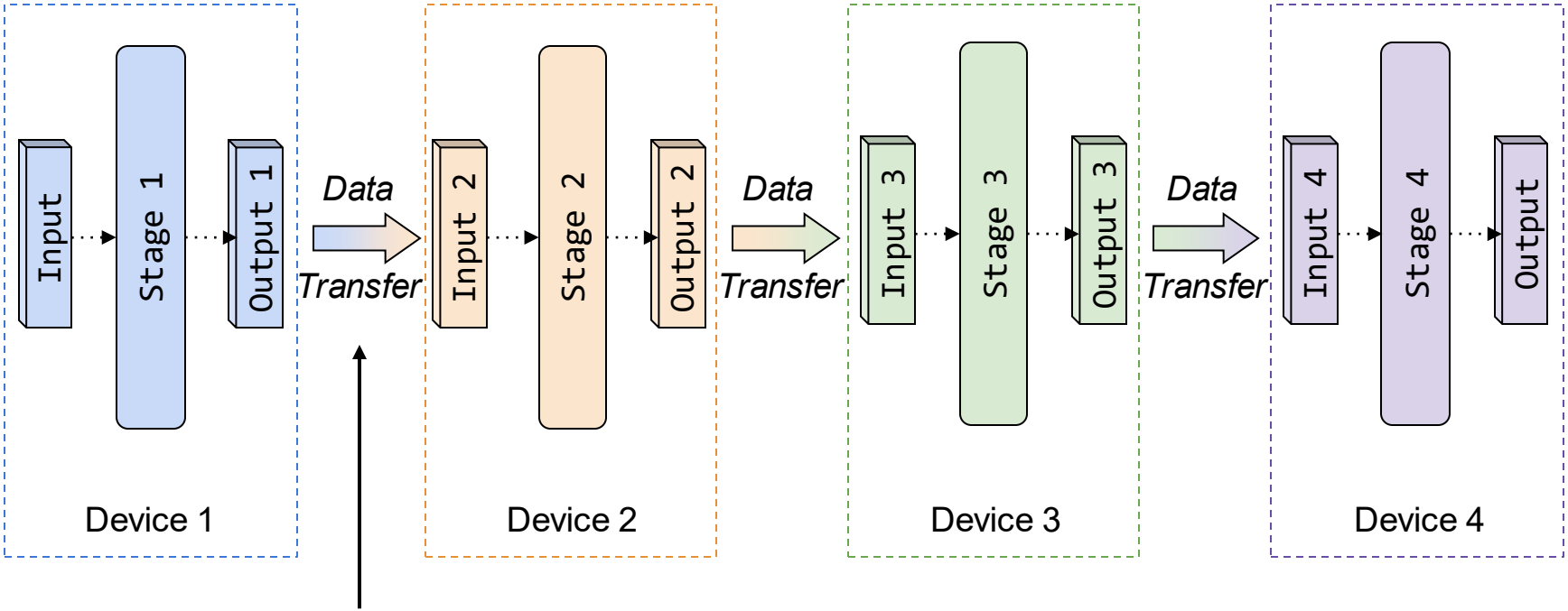
Computational Graph (Neural Networks) → Stages



Computational Graph (Neural Networks) → Stages

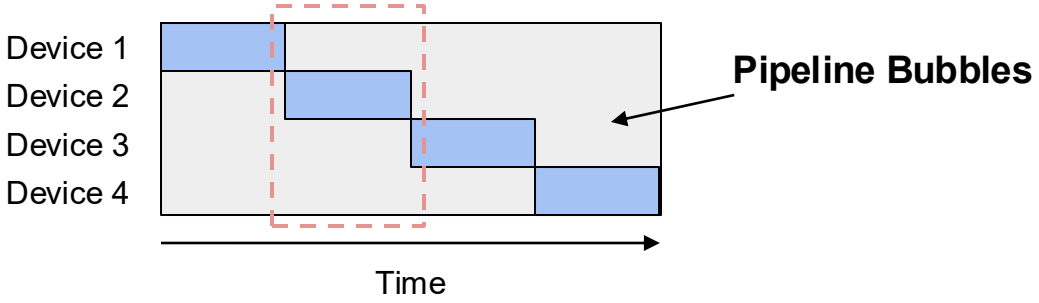


Execution & Data Movement



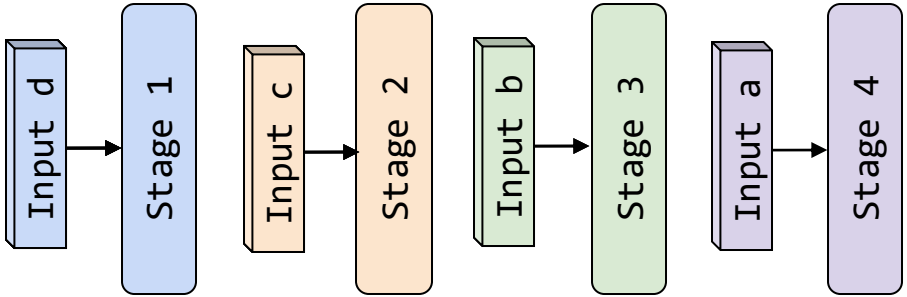
Note: The time spent on data transfer is typically **small,** since we only communicates stage outputs at stage boundaries between two stages.

Timeline: Visualization of Inter-Operator Parallelism

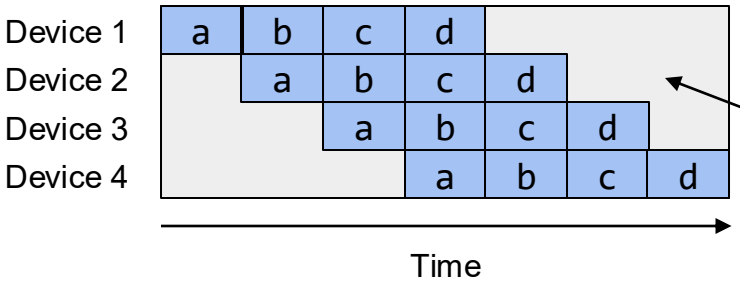


- Gray area ( ) indicates devices being idle (a.k.a. Pipeline bubbles).
- Only 1 device activated at a time.
- **Pipeline bubble percentage** = `bubble_area / total_area`
  = `(D - 1) / D`, assuming `D` devices.

Reduce Pipeline Bubbles via Pipelining Inputs

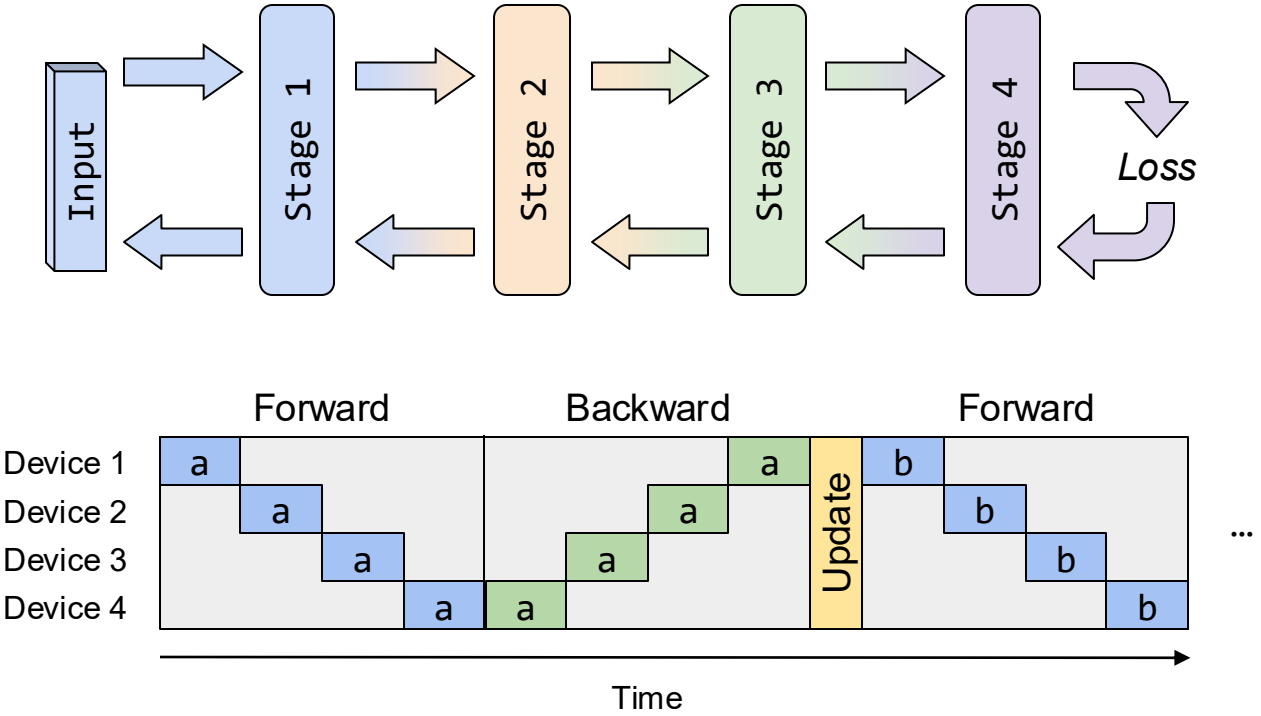


Used in inference.



Pipeline bubbles percentage
= (D - 1) / (D - 1 + N)
with D devices and N inputs.

Training: Forward & Backward Dependency

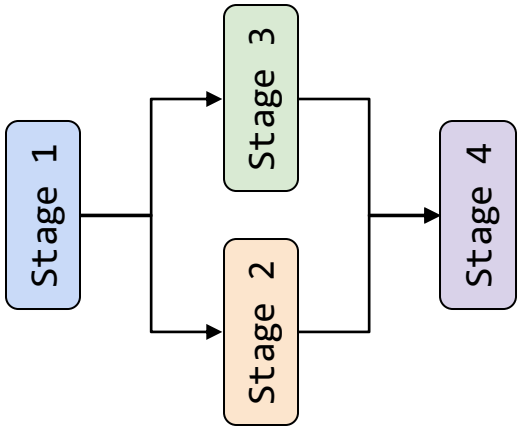


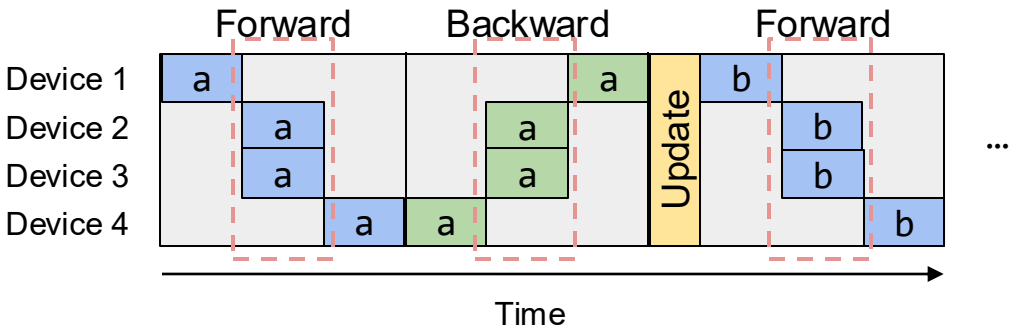
How to Reduce Pipeline Bubbles for Training?

- Device Placement
- Synchronous Pipeline Parallel Algorithms
  - GPipe
  - 1F1B
  - Interleaved 1F1B
  - TeraPipe
  - Chimera
- Asynchronous Pipeline Parallel Algorithms
  - AMPNet
  - Pipedream/Pipedream-2BW

Device Placement

Idea: Slice the branches of a neural network into multiple stages so they can be calculated concurrently.

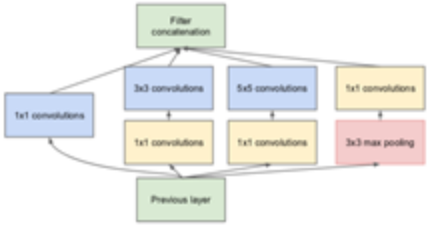




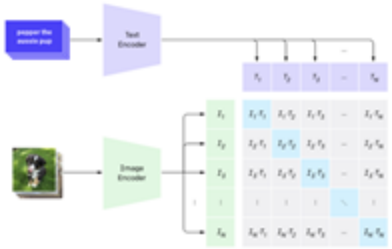
Mirhoseini, Azalia, et al. "Device placement optimization with reinforcement learning." *ICML 2017*.

Device Placement: Limitations

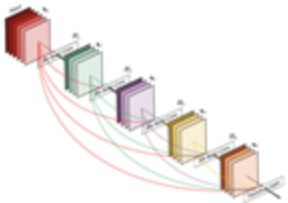
Only works for specific NNs with branches:



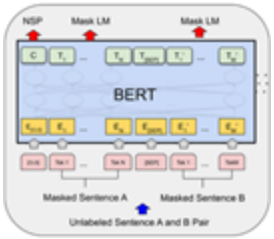
✅ Inception Module



✅ Contrastive Model

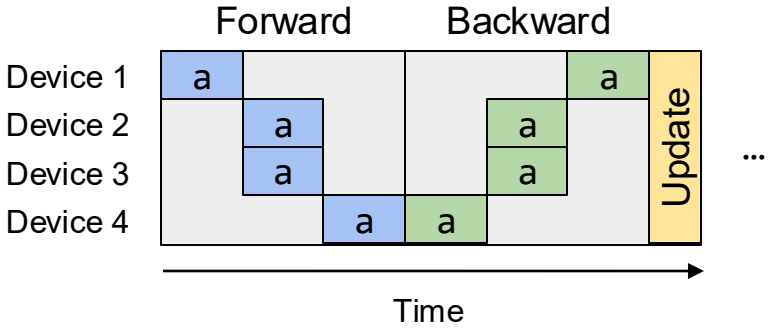


❌ Other ConvNets



❌ Transformers

Device Utilization is still low:



Note: device placement needs to be combined with the other pipeline schedules discussed later to further improve device utilization.

Synchronous Pipeline Parallel Schedule

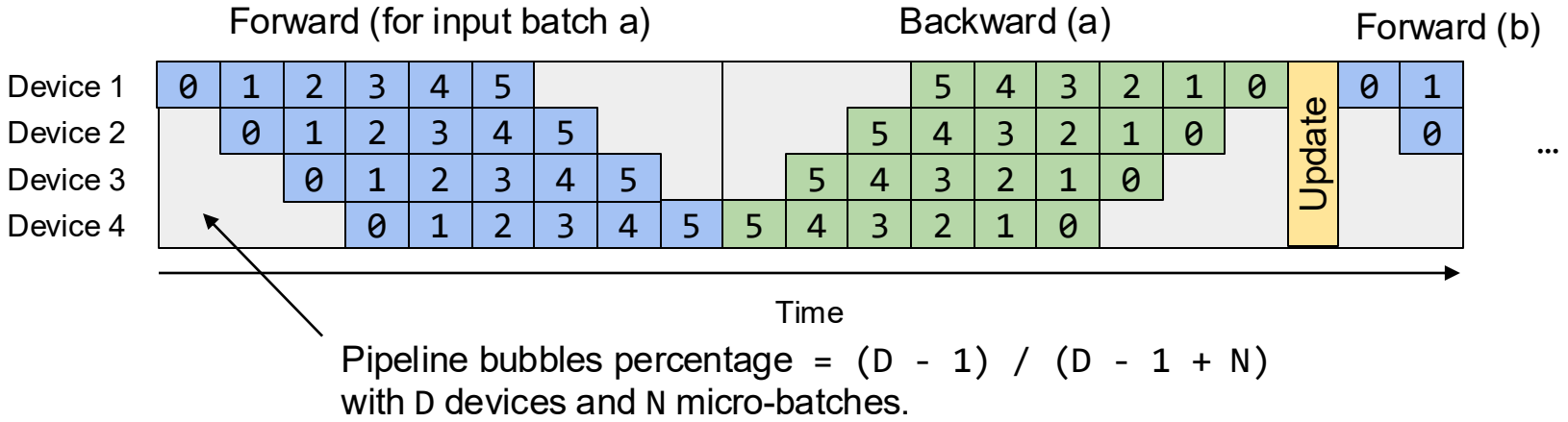
Idea: Modify pipeline schedule to improve efficiency, but keep the computation and convergence semantics exactly the same as if training with a single device.

GPipe

Idea: Partition the input batch into multiple "*micro-batches*". Pipeline the micro-batches. Accumulate the gradients of the micro-batches:

$$\nabla L_\theta(x) = \frac{1}{N} \sum_{i=1}^{N} \nabla L_\theta(x_i)$$

Example: Slice each input batch into 6 micro-batches:



Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." *NeurIPS 2019*.

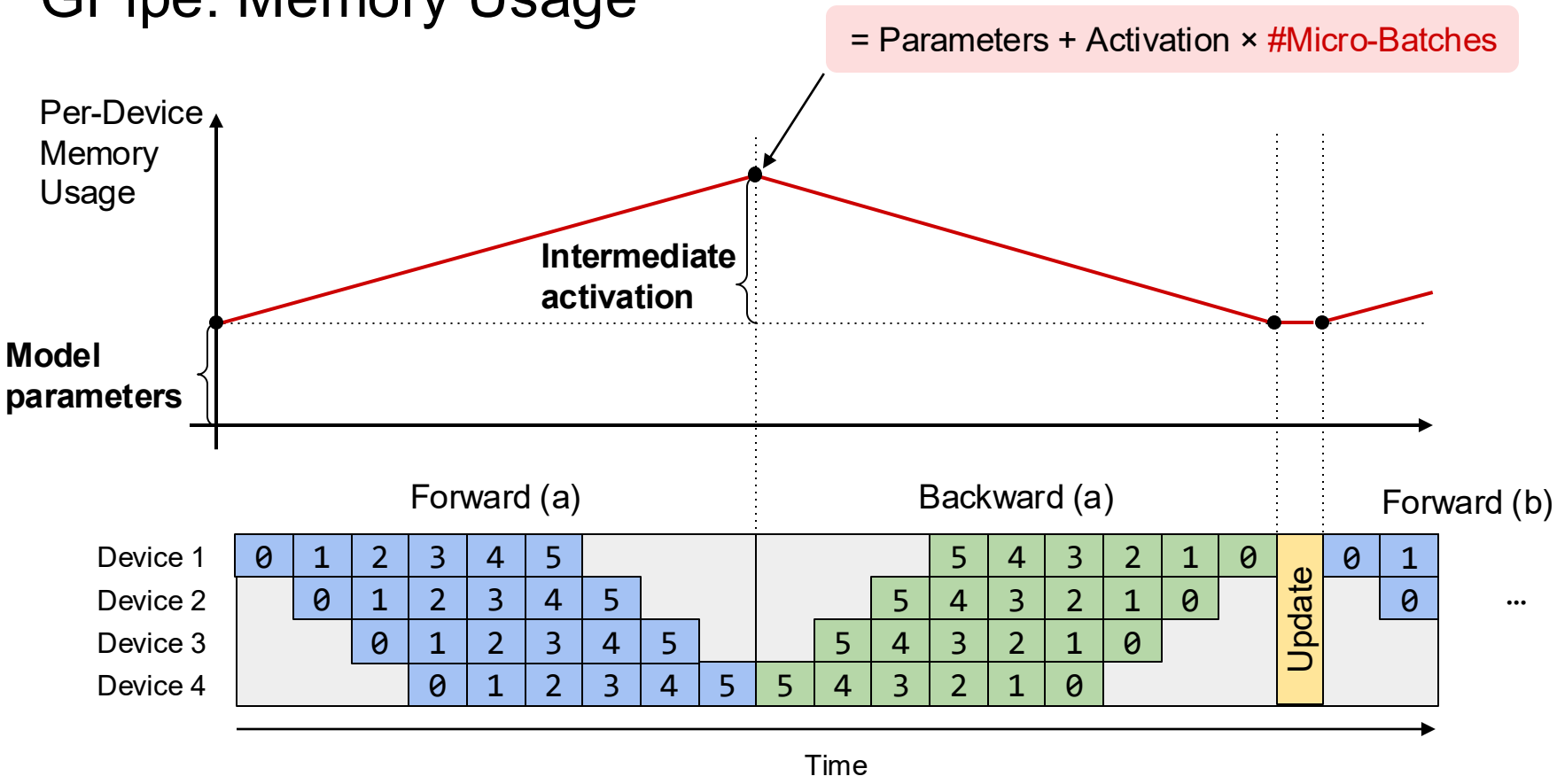
GPipe: Experimental Results

Table: Normalized training throughput using GPipe with different number of devices (stages) and different number of micro-batches M on TPUs.

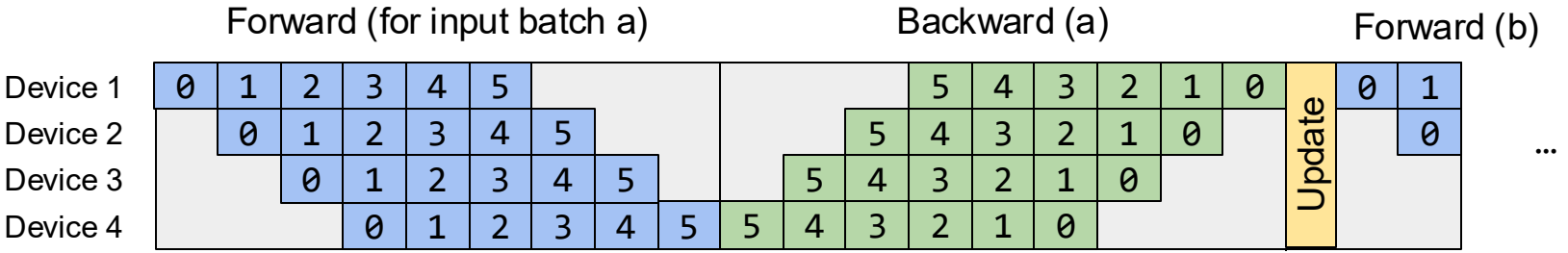
	#TPUs = 2	**#TPUs = 4**	**#TPUs = 8**
#Micro-batches = 1	1	1.07	1.3
#Micro-batches = 4	1.7	3.2	4.8
#Micro-batches = 32	1.8	3.4	6.3

Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." *NeurIPS 2019*.

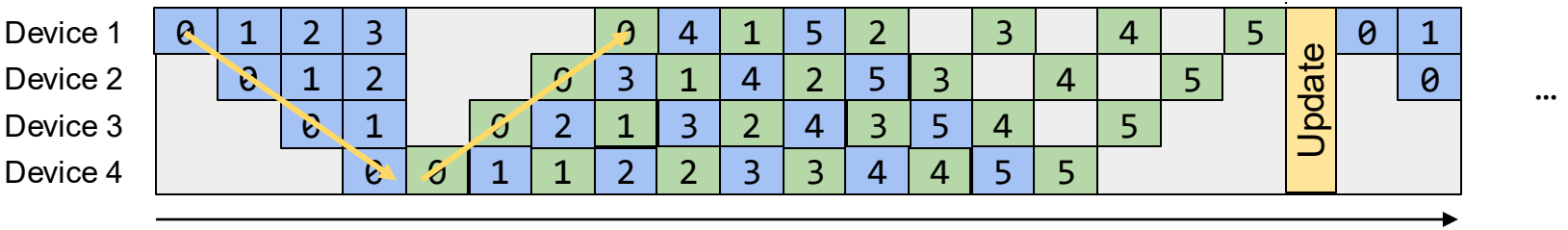
GPipe: Memory Usage



GPipe Schedule:



1F1B (1 Forward 1 Backward) Schedule:

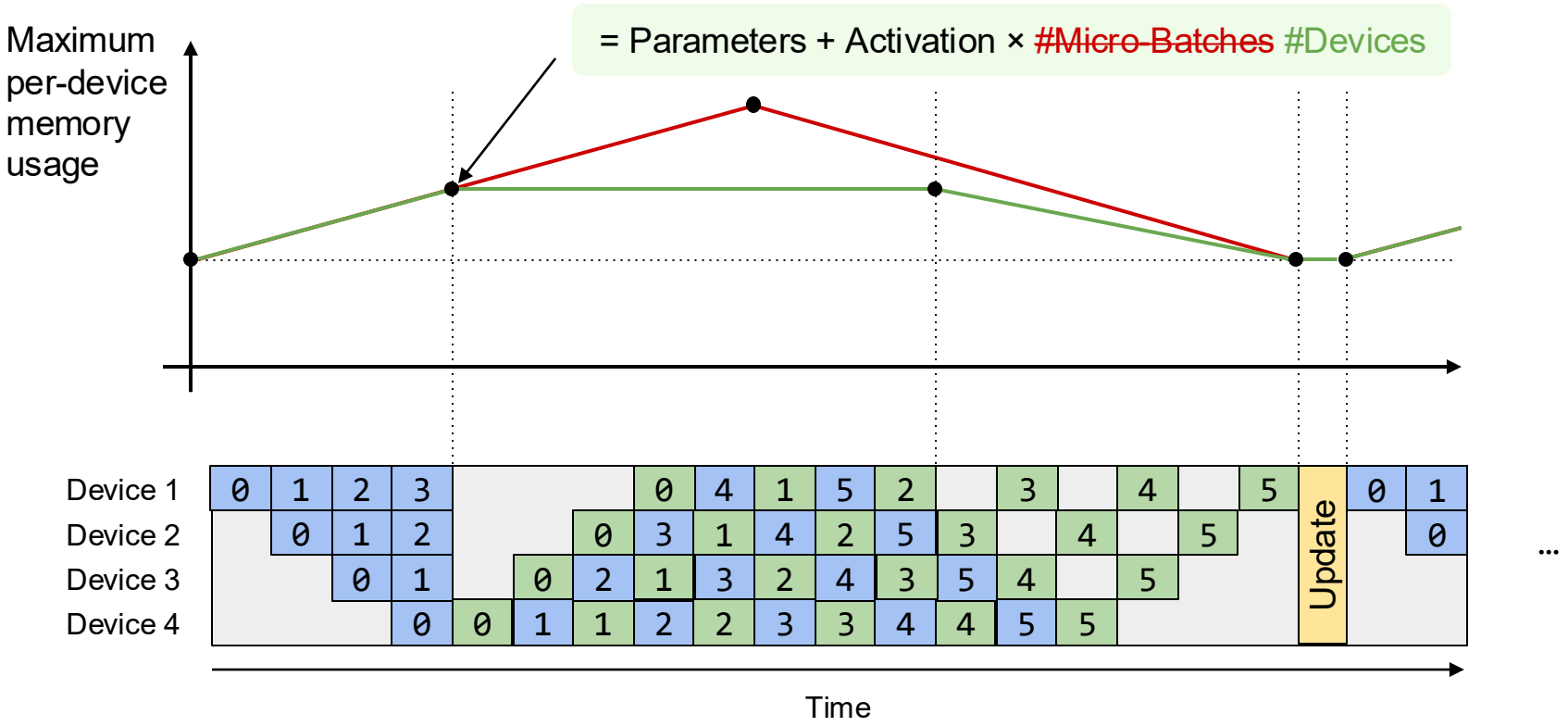


Same Latency

Perform backward as early as possible

Fan, Shiqing, et al. "DAPPLE: A pipelined data parallel approach for training large models." *PPoPP* 2021.

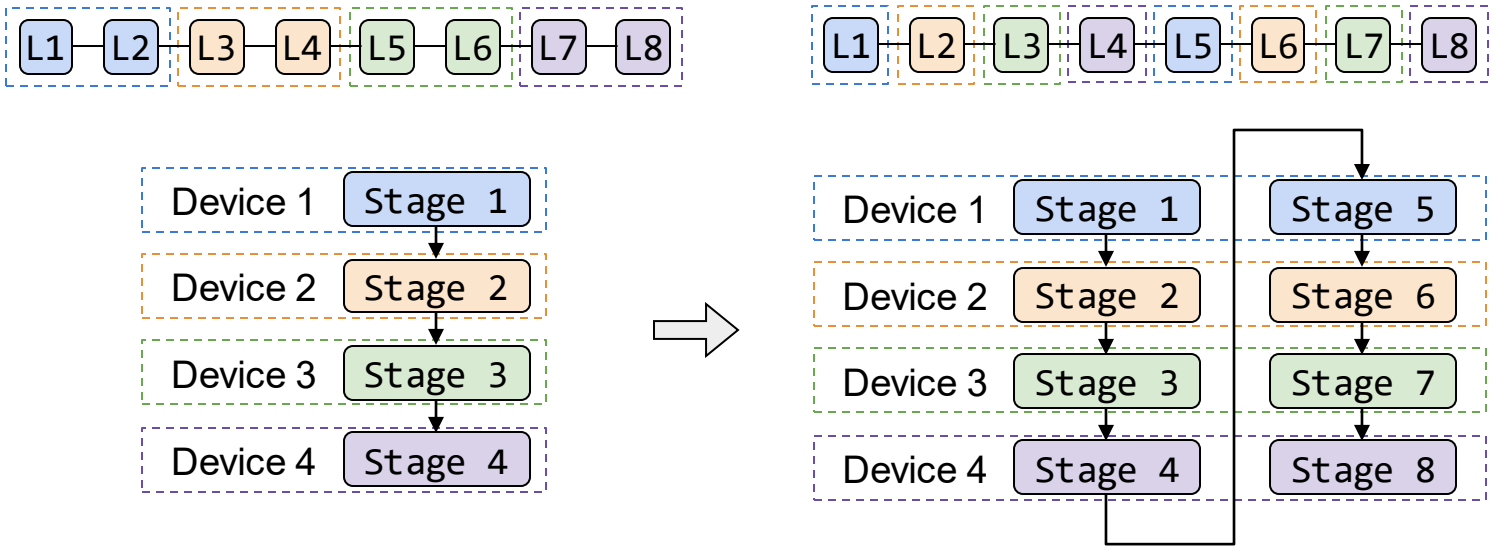
1F1B Memory Usage



Fan, Shiqing, et al. "DAPPLE: A pipelined data parallel approach for training large models." *PPoPP 2021*.

Interleaved 1F1B

Idea: Slice the neural network into more fine-grained stages and assign multiple stages to reduce pipeline bubble.

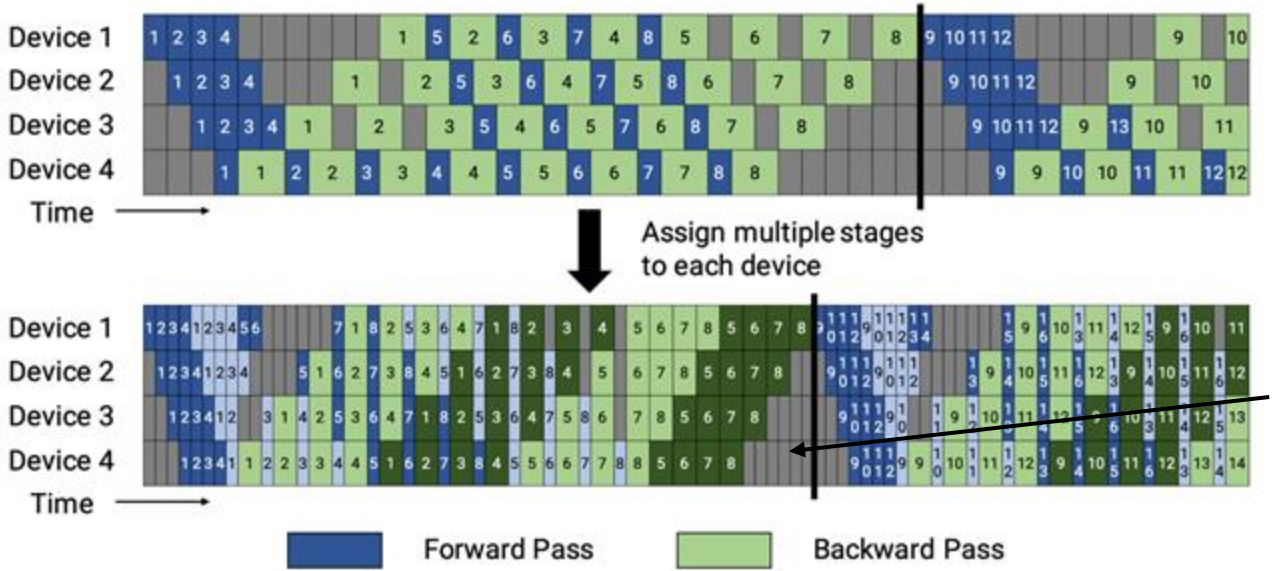


Narayanan, Deepak, et al. "Efficient large-scale language model training on gpu clusters using megatron-lm." *SC 2021*.

Interleaved 1F1B

Pro:
Higher pipeline efficiency with fewer pipeline bubbles.

Con:
More communication overhead between stages.

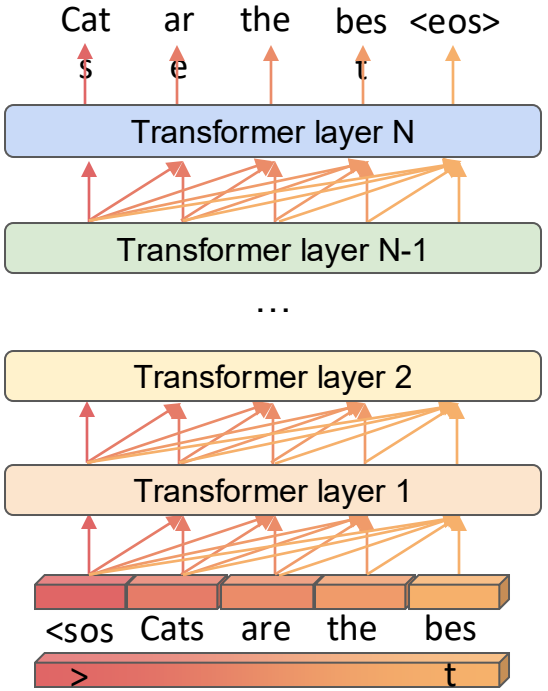


Pipeline bubbles percentage = `(D - 1) / (D - 1 + KN)` with `D` devices, `K` stages on each device, and `N` micro-batches.

TeraPipe

Idea: The computation of an input token only depends on previous tokens but not future tokens for autoregressive models.

Further reduce the bubble size by pipelining within a sequence.

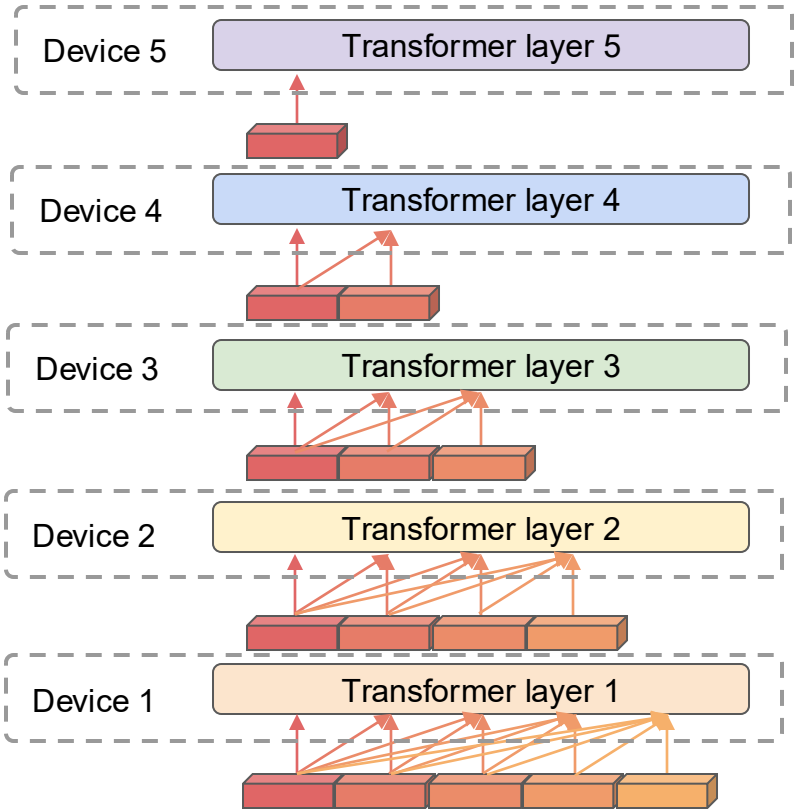


Li, Zhuohan, et al. "TeraPipe: Token-Level Pipeline Parallelism for Training Large-Scale Language Models." *ICML 2021*.

TeraPipe

Idea: The computation of an input token only depends on previous tokens but not future tokens for autoregressive models.

Further reduce the bubble size by pipelining within a sequence.

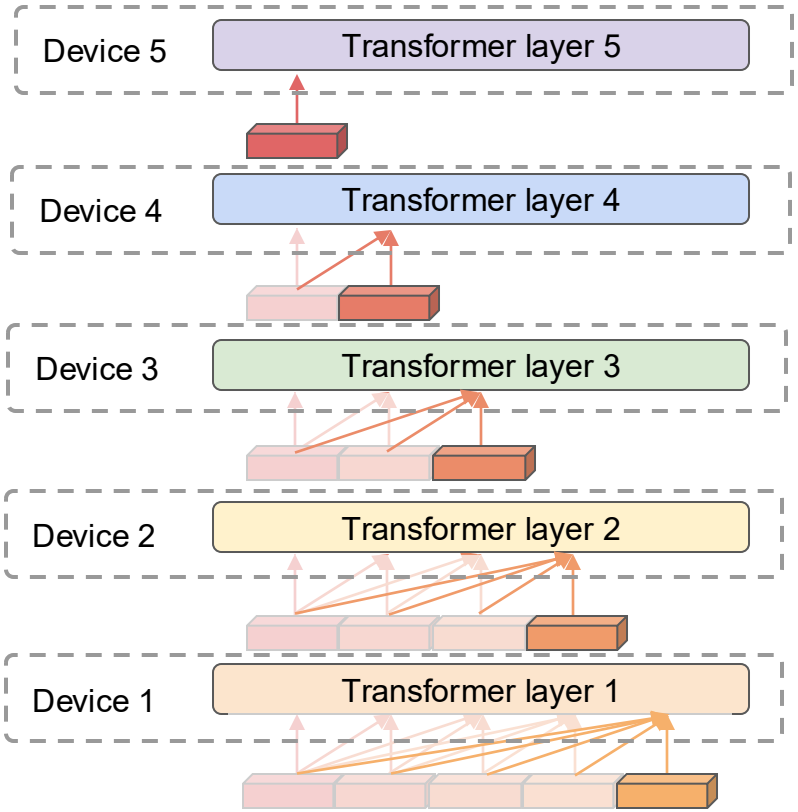


Li, Zhuohan, et al. "TeraPipe: Token-Level Pipeline Parallelism for Training Large-Scale Language Models." *ICML 2021*.

TeraPipe

Idea: The computation of an input token only depends on previous tokens but not future tokens for autoregressive models.

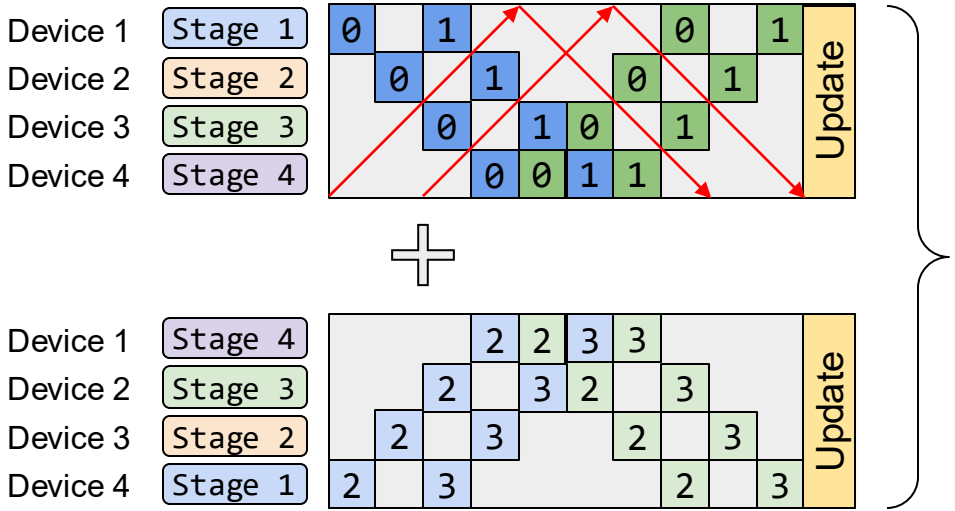
Further reduce the bubble size by pipelining within a sequence.

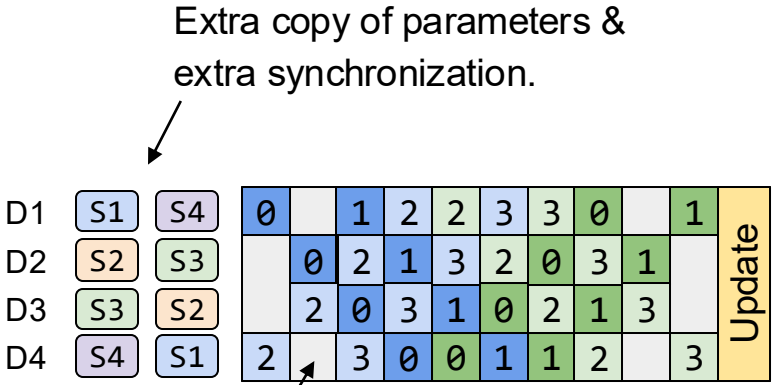


Li, Zhuohan, et al. "TeraPipe: Token-Level Pipeline Parallelism for Training Large-Scale Language Models." *ICML 2021*.

Chimera

Idea: Store bi-directional stages and combine bidirectional pipeline to further reduce pipeline bubbles.





Pipeline bubbles percentage
= (D - 2) / (D - 2 + 2N)
with D devices and N micro-batches.

Li, Shigang, and Torsten Hoefler. "Chimera: efficiently training large-scale neural networks with bidirectional pipelines." *SC 21*.

Synchronous Pipeline Schedule Summary

✅ **Pros:**

- Keep the convergence semantics. The training process is exactly the same as training the neural network on a single device.

❌ **Cons:**

- Pipeline bubbles.
- Reducing pipeline bubbles typically requires splitting inputs into smaller components, but too small input to the neural network will reduce the hardware efficiency.

Asynchronous Pipeline Schedules

Idea: Start next round of forward pass before backward pass finishes.

✅ **Pros:**

- No Pipeline bubbles.

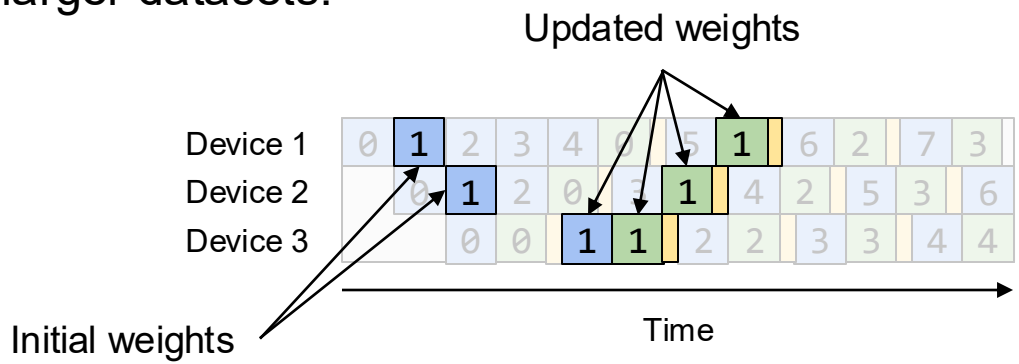
❌ **Cons:**

- Break the synchronous training semantics. Now the training will involve stalled gradient.
- Algorithms may store multiple versions of model weights for consistency.

AMPNet

Idea: Fully asynchronous. Each device performs forward pass whenever free and updates the weights after every backward pass.

Convergence: Achieve similar accuracy on small datasets (MNIST 97%), hard to generalize to larger datasets.



PipeMare: modify the optimizer to improve AMPNet convergence

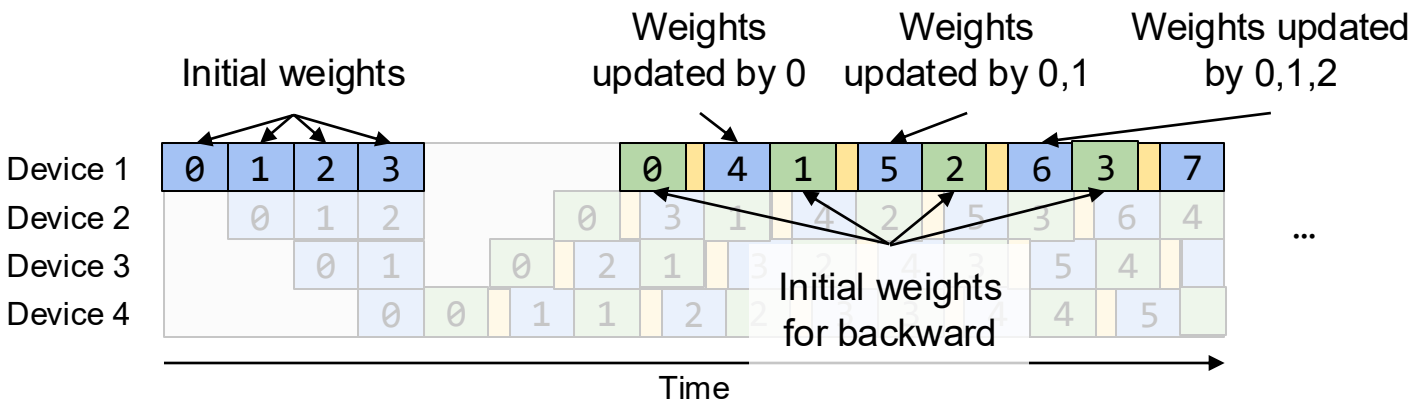
Gaunt, Alexander L., et al. "AMPNet: Asynchronous model-parallel training for dynamic neural networks." *arXiv 2017.*
Yang, Bowen, et al. "Pipemare: Asynchronous pipeline parallel dnn training." *MLSys 2021.*

Pipedream

Idea: Enforce the same version of weight for a single input batch by storing multiple weight versions.

Convergence: Similar accuracy on ImageNet with a 5x speedup compared to data parallel.

Con: No memory saving compared to single device case.

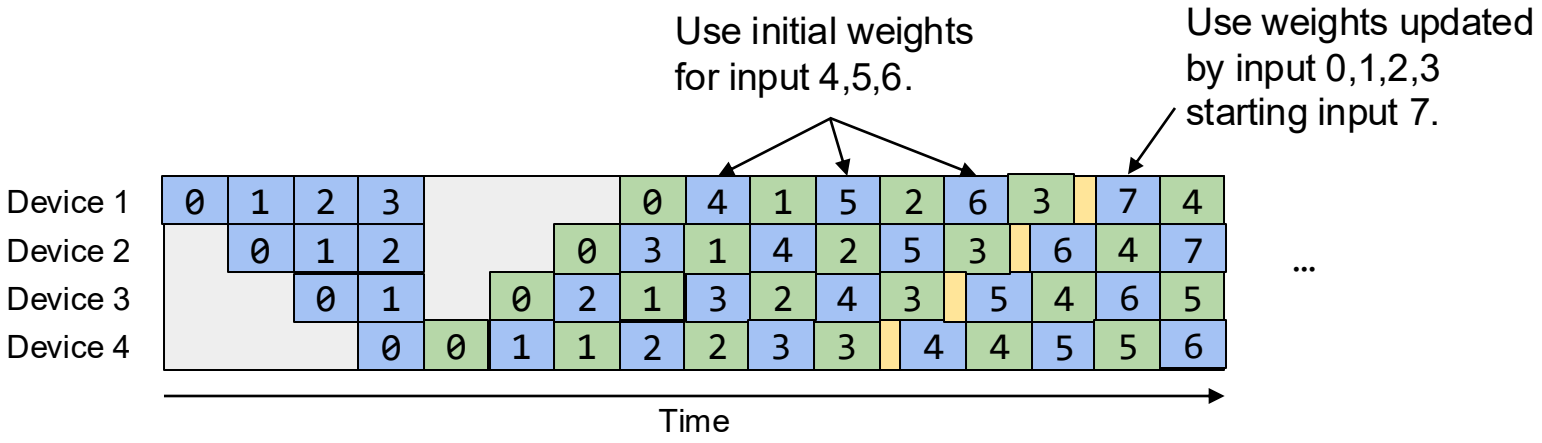


Narayanan, Deepak, et al. "PipeDream: generalized pipeline parallelism for DNN training." *SOSP 2019*.

Pipedream-2BW

Idea: Reduce Pipedream's memory usage (only store 2 copies) by updating weights less frequently. Weights always stalled by 1 update.

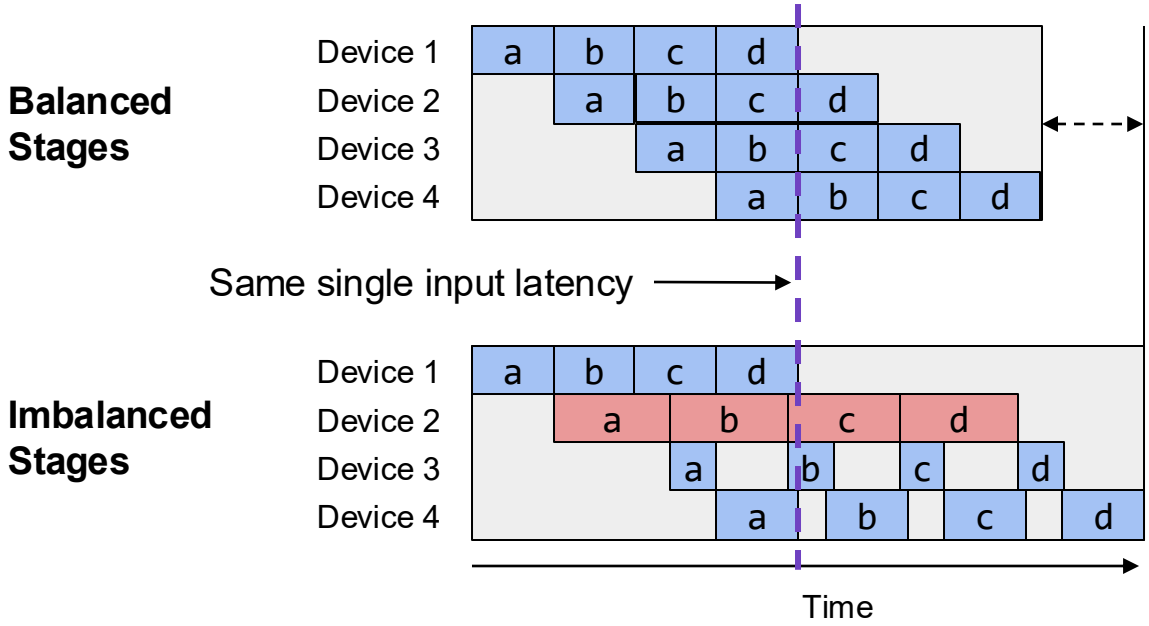
Convergence: Similar training accuracy on language models (BERT/GPT)



Narayanan, Deepak, et al. "Memory-efficient pipeline-parallel dnn training." *ICML 2021*.

Imbalanced Pipeline Stages

Pipeline schedules works best with balanced stages:



Frontier: Automatic Stage Partitioning

Goal: Minimize maximum stage latency & maximize parallelization

Reinforcement Learning Based (mainly for device placement):

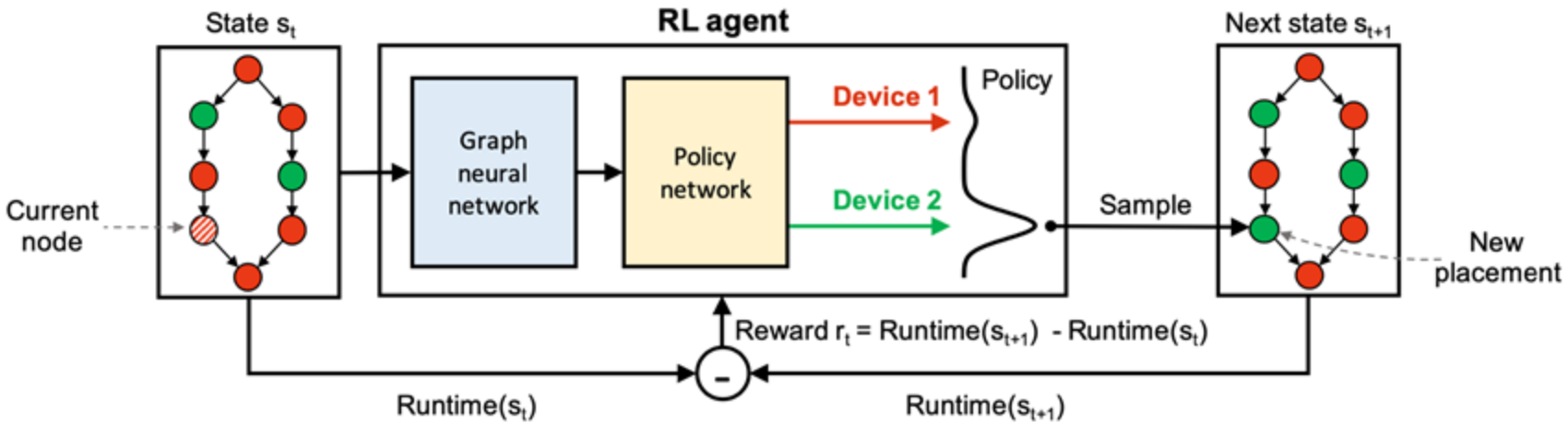
1. Mirhoseini, Azalia, et al. "Device placement optimization with reinforcement learning." *ICML 2017.*
2. Gao, Yuanxiang, et al. "Spotlight: Optimizing device placement for training deep neural networks." *ICML 2018.*
3. Mirhoseini, Azalia, et al. "A hierarchical model for device placement." *ICLR 2018.*
4. Addanki, Ravichandra, et al. "Placeto: Learning generalizable device placement algorithms for distributed machine learning." *NeurIPS 2019.*
5. Zhou, Yanqi, et al. "Gdp: Generalized device placement for dataflow graphs." *Arxiv 2019.*
6. Paliwal, Aditya, et al. "Reinforced genetic algorithm learning for optimizing computation graphs." *ICLR 2020.*
7. *…*

Optimization (Dynamic Programming/Linear Programming) Based:

1. Narayanan, Deepak, et al. "PipeDream: generalized pipeline parallelism for DNN training." *SOSP 2019.*
2. Tarnawski, Jakub M., et al. "Efficient algorithms for device placement of dnn graph operators." *NeurIPS 2020.*
3. Fan, Shiqing, et al. "DAPPLE: A pipelined data parallel approach for training large models." *PPoPP 2021.*
4. Tarnawski, Jakub M., Deepak Narayanan, and Amar Phanishayee. "Piper: Multidimensional planner for dnn parallelization." *NeurIPS 2021.*
5. Zheng, Lianmin, et al. "Alpa: Automating Inter-and Intra-Operator Parallelism for Distributed Deep Learning." *OSDI 2022.*
6. *…*

RL-Based Partitioning Algorithm

State: Device assignment plan for a computational graph.
Action: Modify the device assignment of a node.
Reward: Latency difference between the new and old placements.
Trained with **policy gradient** algorithm.



Addanki, Ravichandra, et al. "Placeto: Learning generalizable device placement algorithms for distributed machine learning." NeurIPS 2019.

Inter-operator Parallelism Summary

Idea: Assign different operators of the computational graph to different devices and executed in a pipelined fashion.

Method	General computational graph	No pipeline bubbles	Same convergence as single device
Device Placement	❌	❌	✅
Synchronous Schedule	✅	❌	✅
Asynchronous Schedule	✅	✅	❌

Stage Partitioning: Imbalance stage → More pipeline bubble

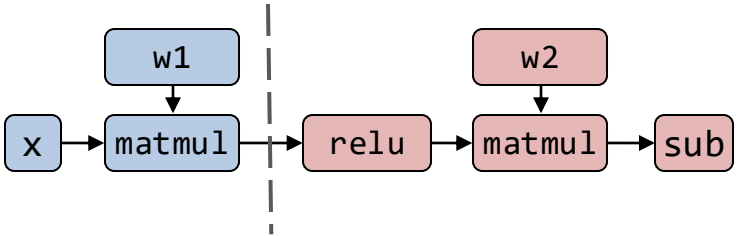
RL-Based / **Optimization-Based** Automatic Stage Partitioning

Where We Are

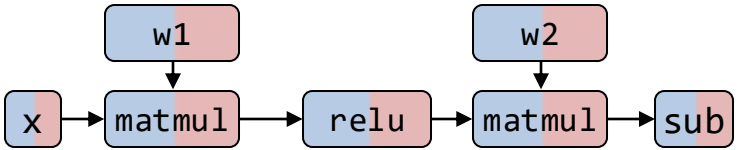
- Model parallelism
  - Inter-op parallelism
  - **Intra-op parallelism**

Recap: Intra-op and Inter-op

Strategy 1: Inter-operator Parallelism



Strategy 2: Intra-operator Parallelism



This section:

1. How to parallelize an **operator** ?
2. How to parallelize a **graph** ?

Parallelize One Operator

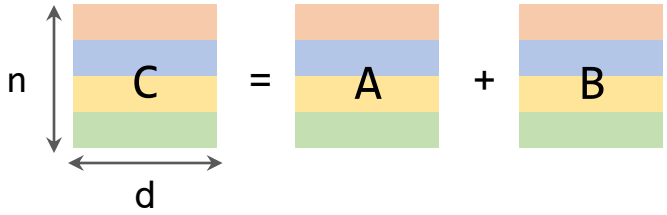
Element-wise operators

```
for n in range(0, N):
  for d in range(0, D):
    C[n,d] = A[n,d] + B[n,d]
```

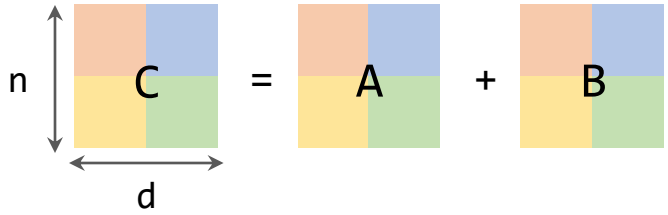
No dependency on the two for-loops.
Can arbitrarily split the for-loops on different devices.

device 1 device 2 device 3 device 4

Parallelize loop n



Parallelize both loop n and loop d



a lot of
other variants
…

Parallelize One Operator

Matrix multiplication

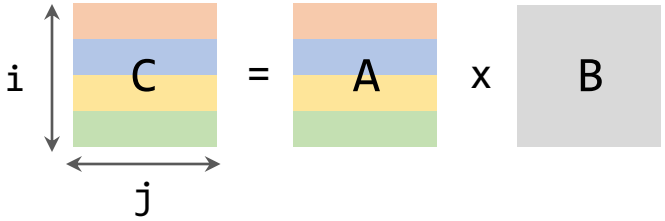
No dependency on the two spatial for-loops.
Can arbitrarily split the for-loops on different devices.

```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

Accumulation on this reduction loop.
Have to accumulate partial results if we split this for-loop

device 1 device 2 device 3 device 4 replicated

Parallelize loop i



$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} \times B$$

Parallelize One Operator

Matrix multiplication

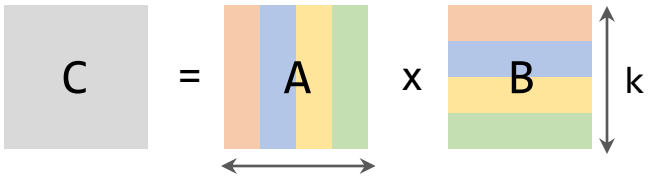
```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

No dependency on the two spatial for-loops.
Can arbitrarily split the for-loops on different devices.

Accumulation on this reduction loop.
Have to accumulate partial results if we split this for-loop

device 1 device 2 device 3 device 4 replicated

Parallelize loop k



(got by all-reduce)

$$C = [A_1\ A_2\ A_3\ A_4] \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} = A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4$$

Parallelize One Operator

Matrix multiplication

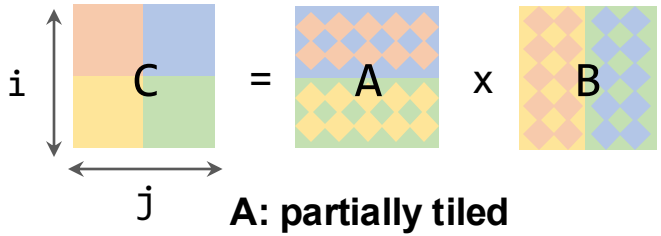
No dependency on the two spatial for-loops.
Can arbitrarily split the for-loops on different devices.

```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

Accumulation on this reduction loop.
Have to accumulate partial results if we split this for-loop

device 1 device 2 device 3 device 4

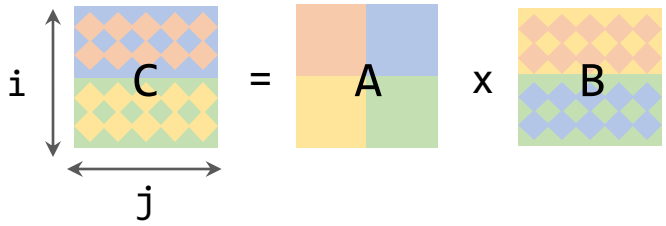
Parallelize loop i and j



A: partially tiled

Device 1 and 2 hold a replicated tile
Device 3 and 4 hold a replicated tile

Parallelize loop i and k

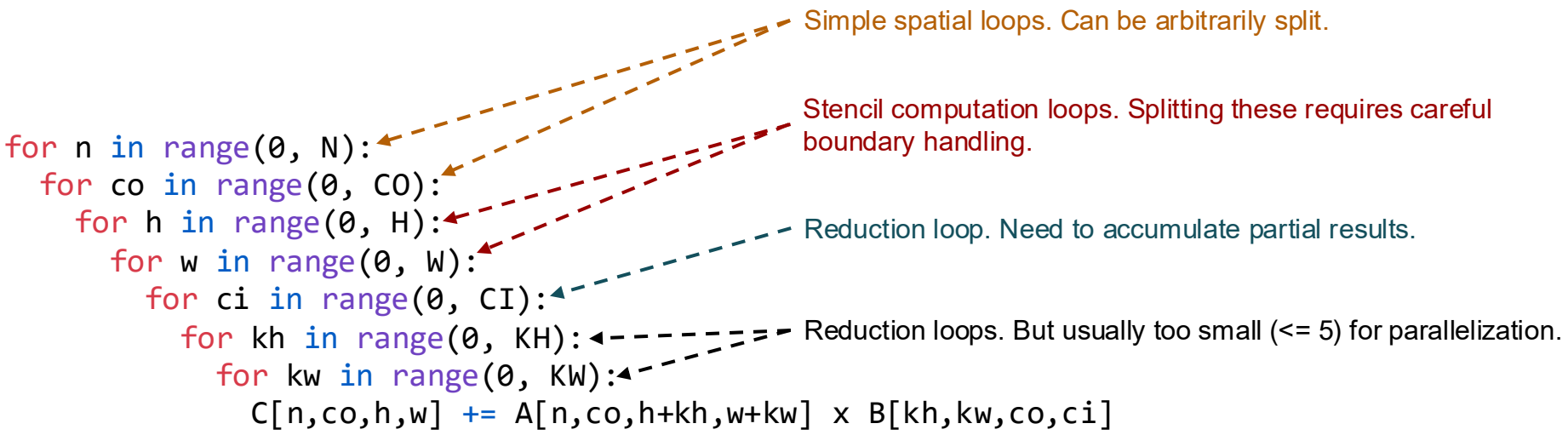


C: got by all-reduce

a lot of other variants …

Parallelize One Operator

2D Convolution



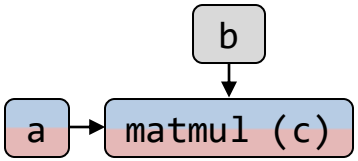
Simple case: Parallelize loop n, co, ci, then the parallelization strategies are almost the same as matmul's.

Complicated case: Parallelize loop h and w

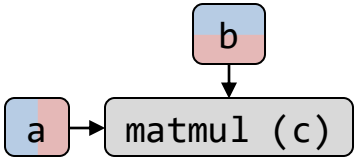
Data Parallelism as A Case of Intra-op Parallelism

Replicated Row-partitioned Column-partitioned

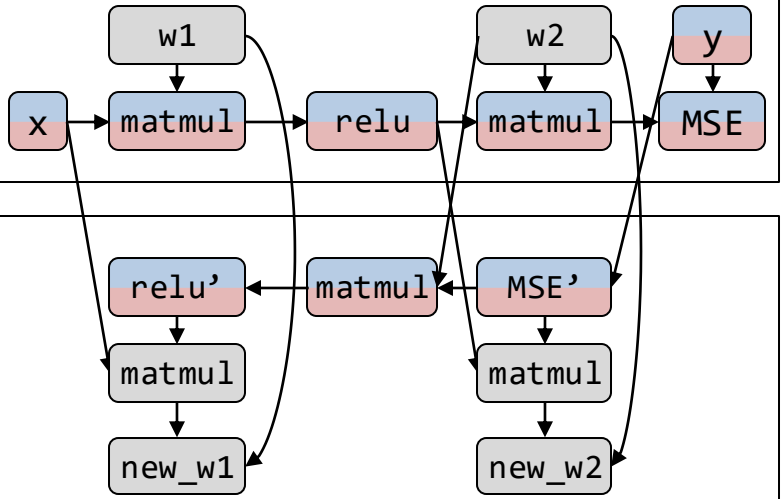
Matmul Parallelization Type 1
communication cost = 0



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



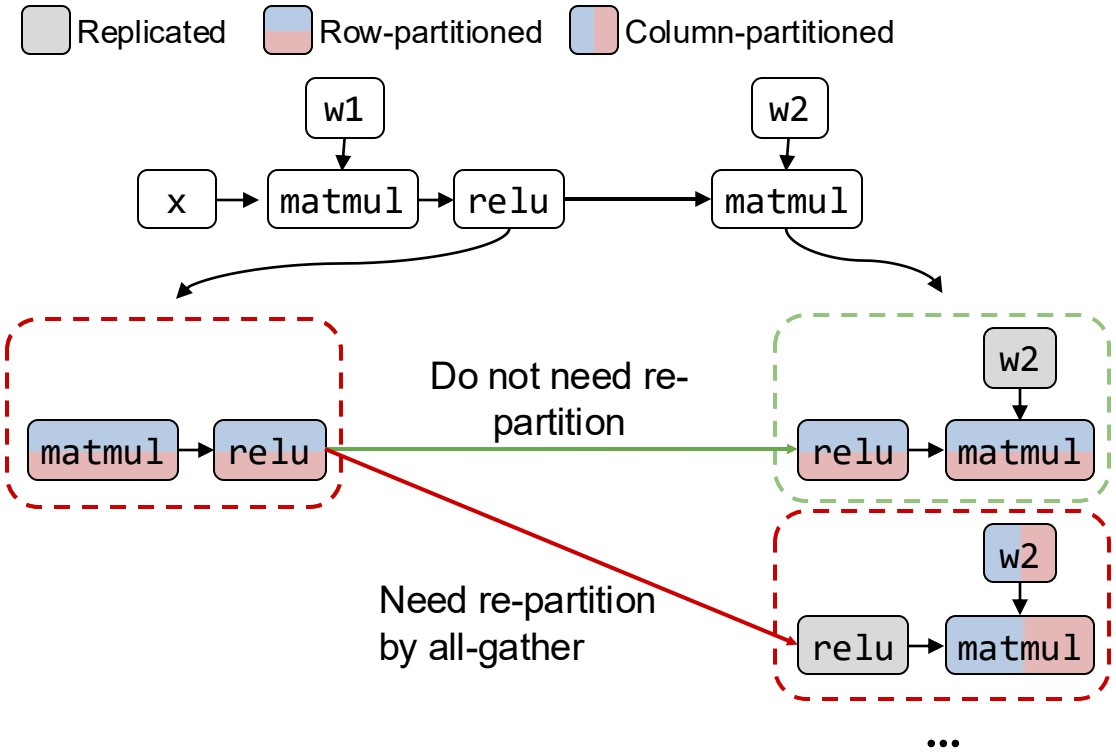
Forward Pass
Two “Type 1” matmuls: no communication



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

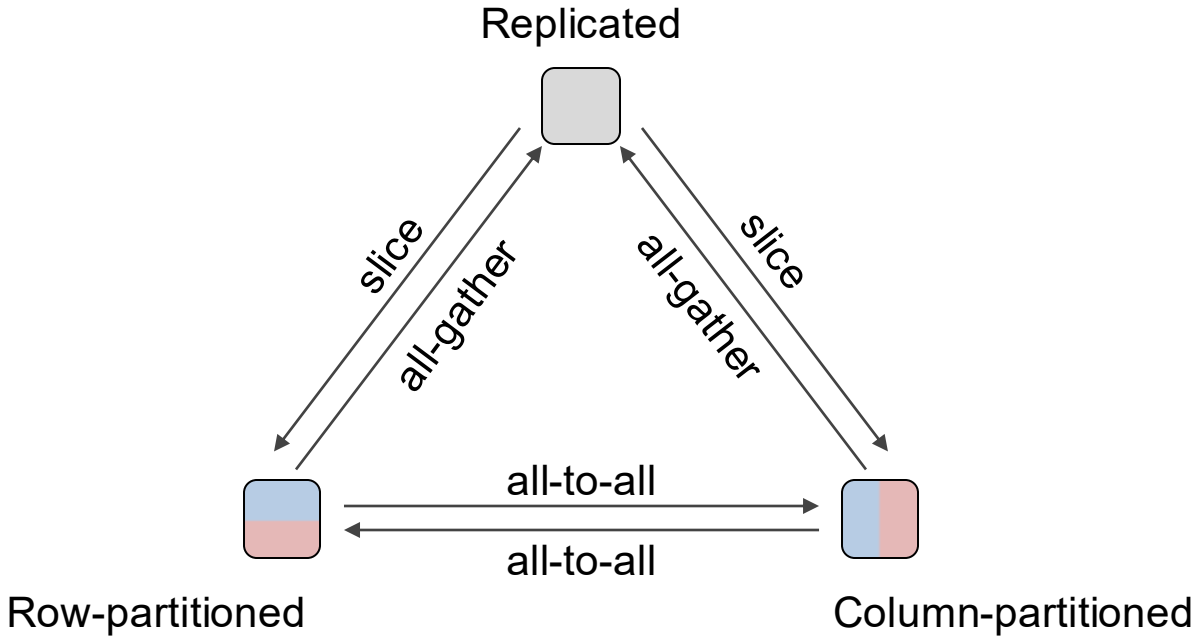
Re-partition Communication Cost

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



Re-partition Communication Cost

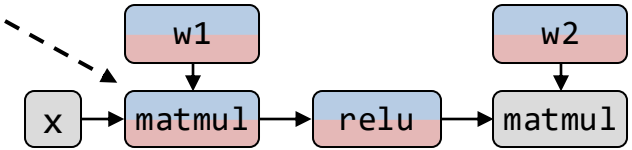
Different operators' parallelization strategies require different partition format of the same tensor



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

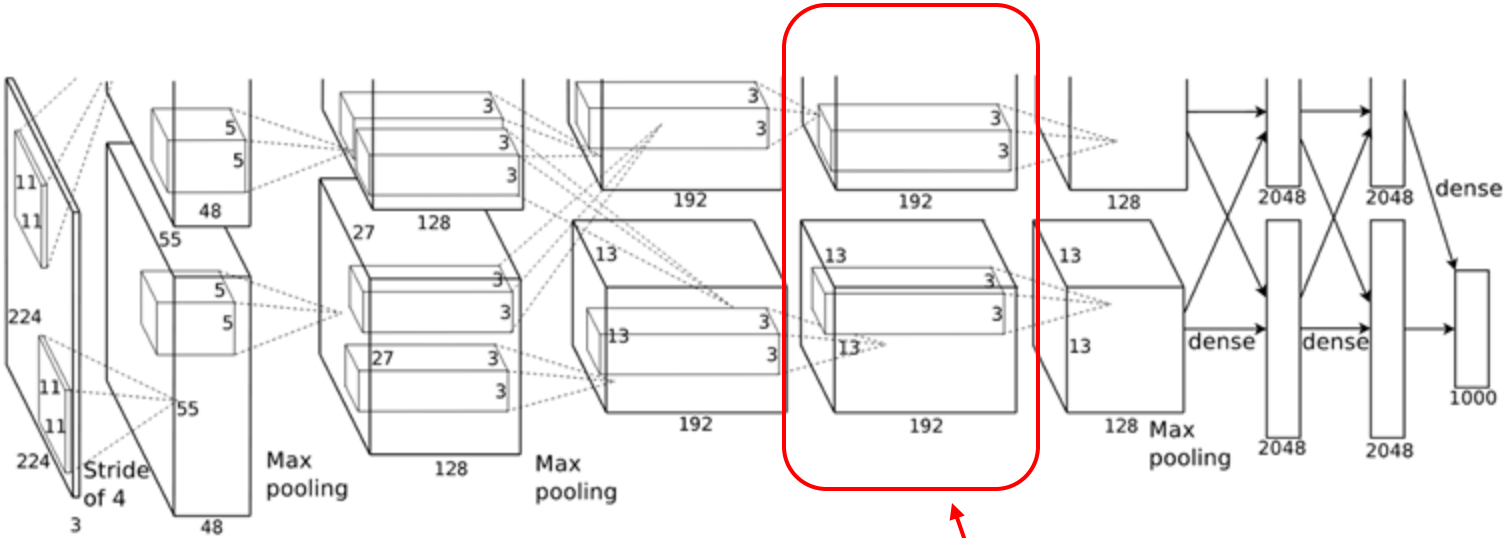
- AlexNet
- Megatron-LM
- GShard MoE

Systems for Intra-op Parallelism

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

AlexNet

Result: increase top-1 accuracy by 1.7%



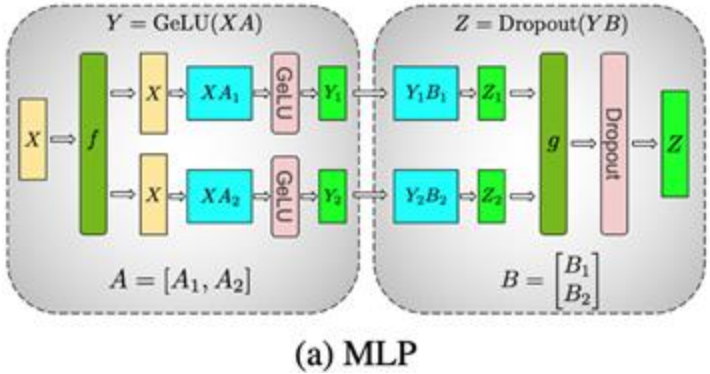
Assign a group convolution layer to 2 GPUs

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

Megaton-LM

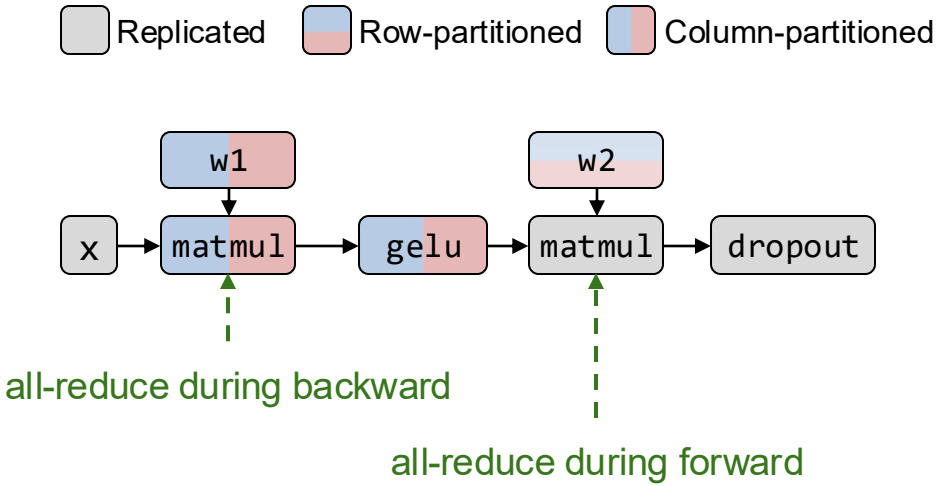
Result: a large language model with 8.3B parameters that outperforms SOTA results

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



(a) MLP

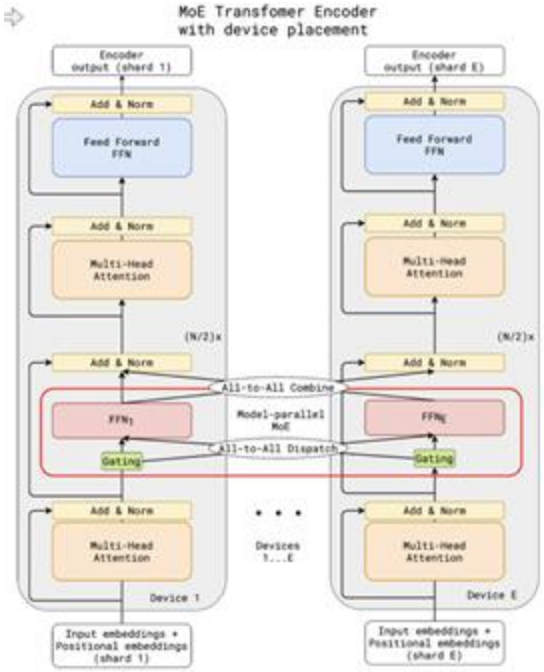
Illustrated with the notations in this tutorial



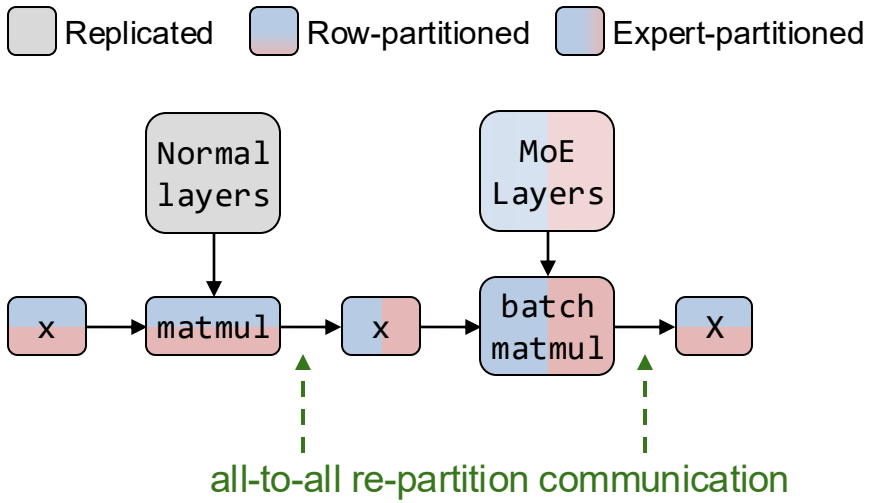
Shoeybi, Mohammad, et al. "Megatron-LM: Training multi-billion parameter language models using model parallelism."

GShard MoE

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



Illustrated with the notations in this class



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.

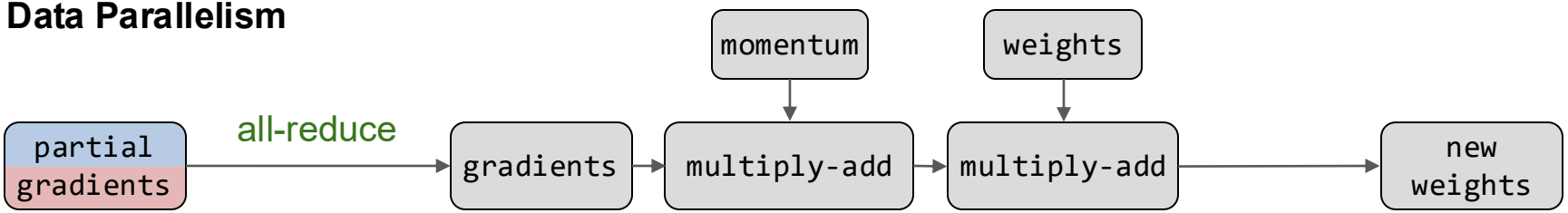
	Optimizer States (12M)	Gradients (2M)	Model Weights (2M)	Memory Cost	Communication Cost
Data Parallelism	Replicated	Replicated	Replicated	$16M$	all-reduce(2M)
ZeRO Stage 1	Partitioned	Replicated	Replicated	$4M + \frac{12M}{N}$	all-reduce(2M)
ZeRO Stage 2	Partitioned	Partitioned	Replicated	$2M + \frac{14M}{N}$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

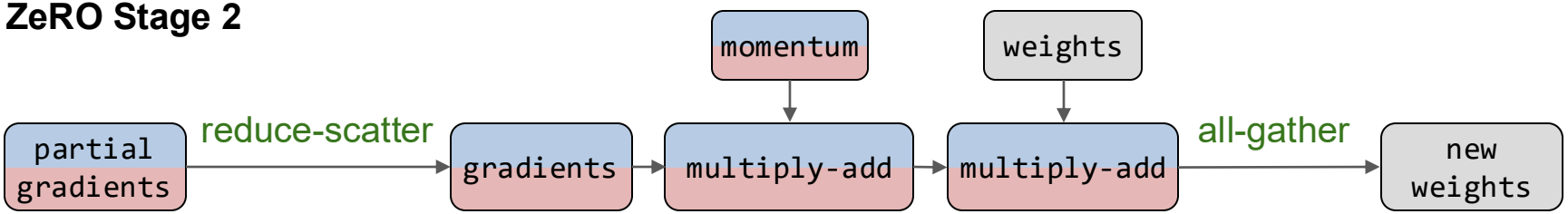
ZeRO Stage 2

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

Data Parallelism



ZeRO Stage 2

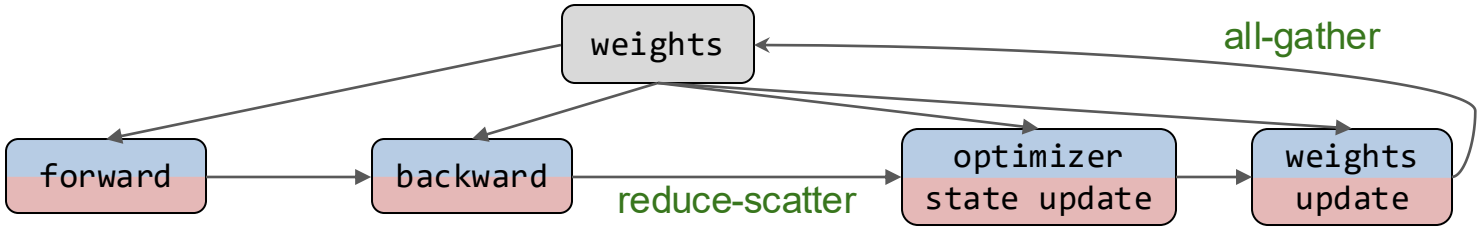


Same communication cost but save memory by partitioning more tensors

ZeRO Stage 3

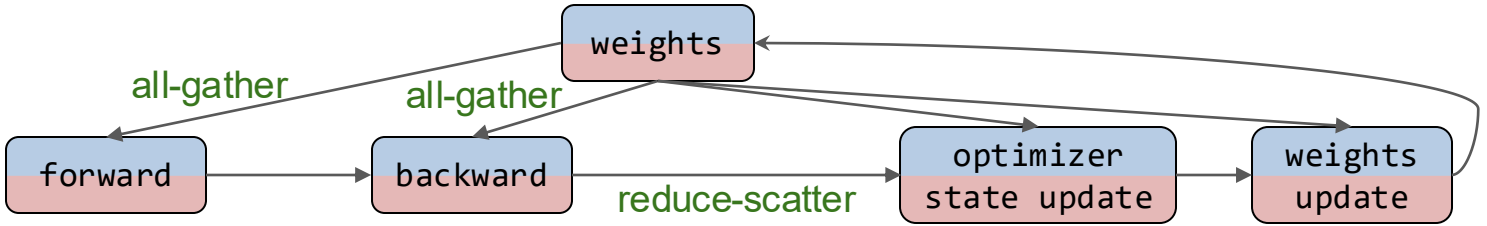
ZeRO Stage 2

communication cost
= all-reduce



ZeRO Stage 3

communication cost
= 1.5 all-reduce



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*

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])
...


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

Tensor dimension

Mesh dimension

Mapping

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

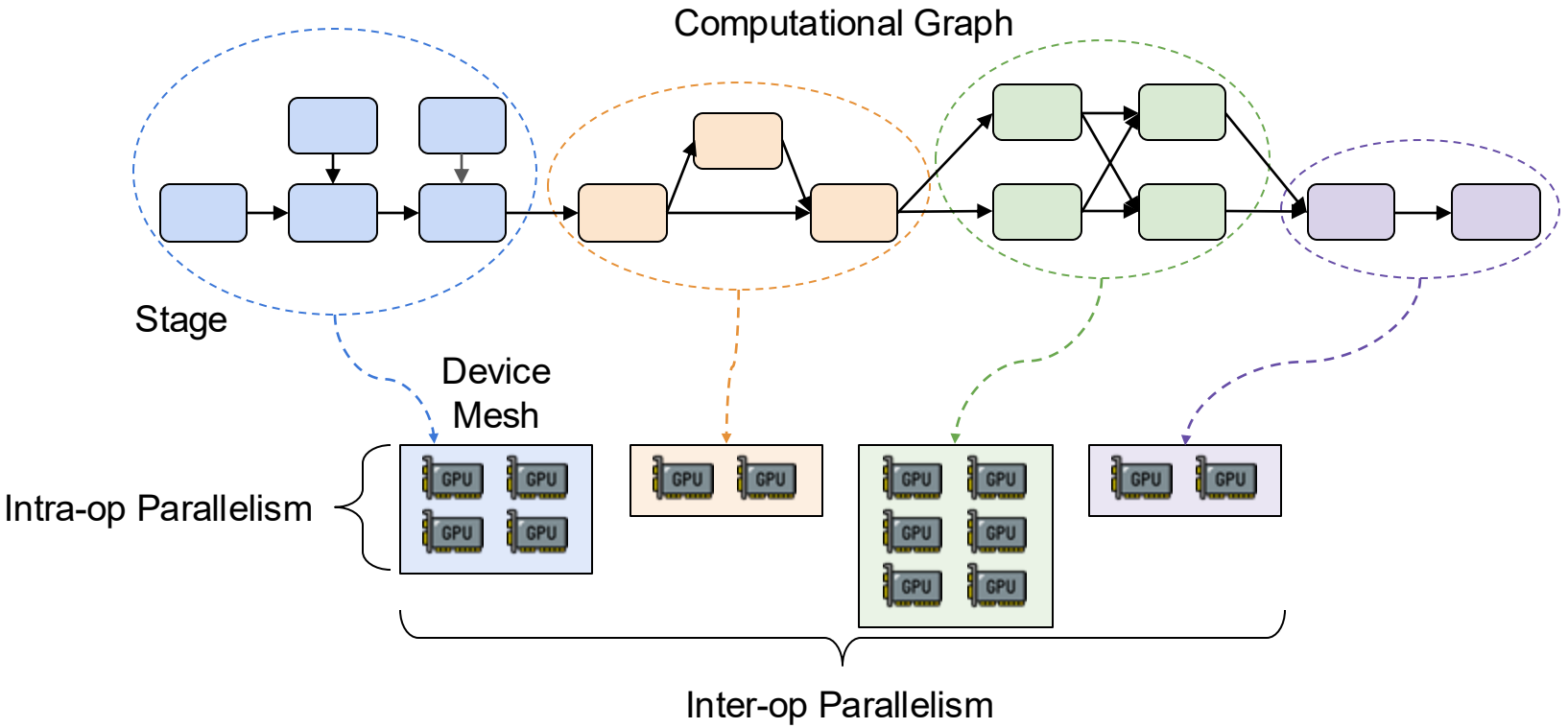
GSPMD

- 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*

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*

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.

Thanks for Enrolling DSC 204A!