Model-Parallel Deep Learning

Efficient DL, Episode ++i ‘26

Yandex
Research

British Hedgehog
Preservation Society

Dealing with large models

~~Model-Parallel Deep Learning~~

Efficient DL, Episode ++i ‘26

Yandex
Research

British Hedgehog
Preservation Society

Recap: large models

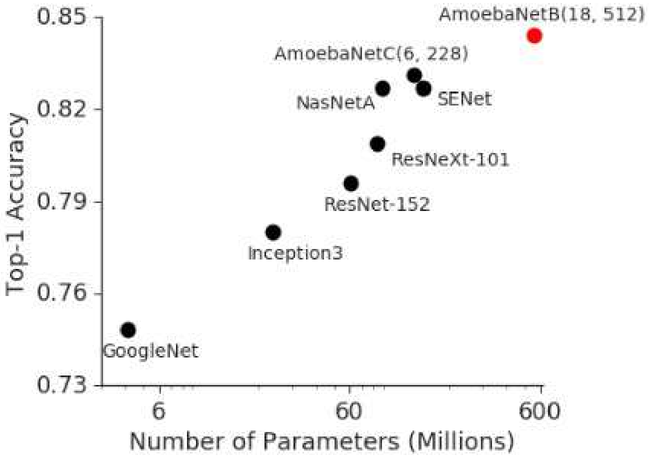
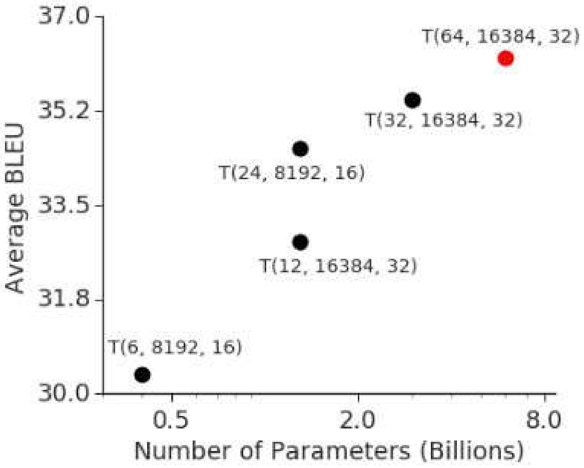


Image Classification
ImageNet

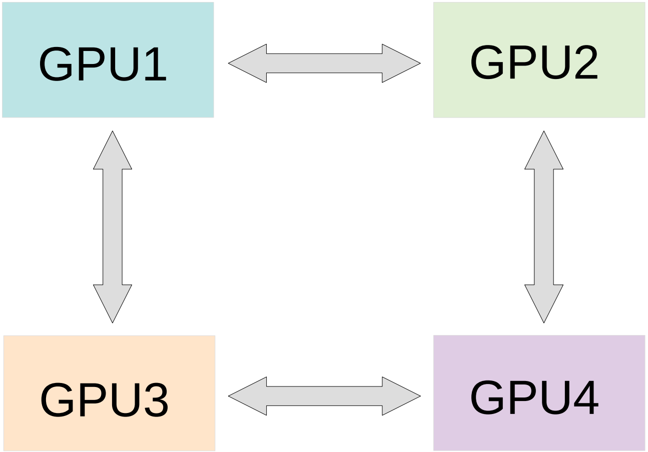


Machine Translation
average over WMT

Source: https://arxiv.org/abs/1811.06965

Recap: Ring allreduce

Bonus quest: you can only send data between **adjacent** gpus



Ring topology

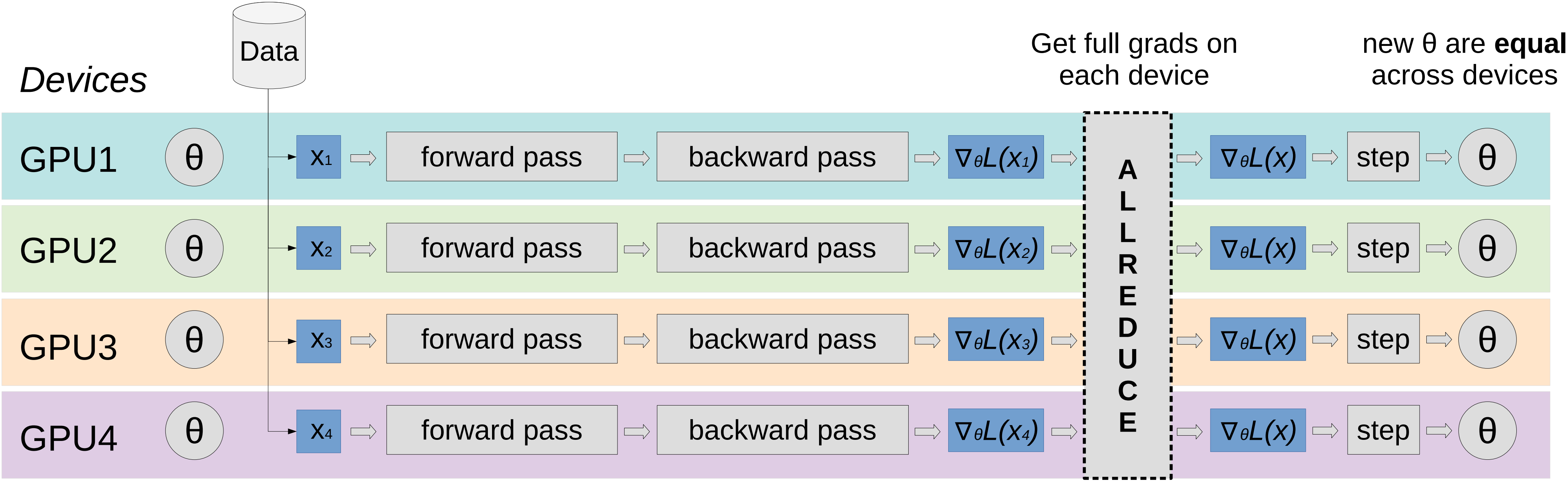
Image: graphcore ipu server

Answer & more: tinyurl.com/ring-allreduce-blog

Recap: All-Reduce SGD

arxiv.org/abs/1706.02677

Idea: get rid of the host, each gpu runs its own computation
Q: why will weights be equal after such step?



Q: What if a model is larger than GPU?

Q: What if a model is larger than GPU?

easy mode: cannot fit the right batch size

hard mode: cannot fit a single sample

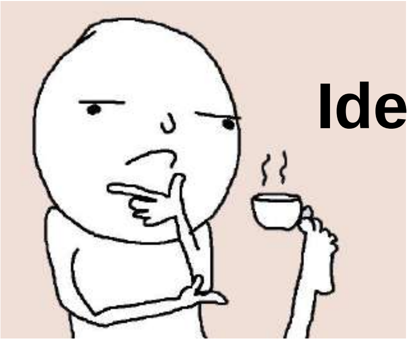
expert mode: not even parameters!

Q: What if a model is larger than GPU?

easy mode: cannot fit the right batch size

hard mode: cannot fit a single sample

expert mode: not even parameters!



Ideas?

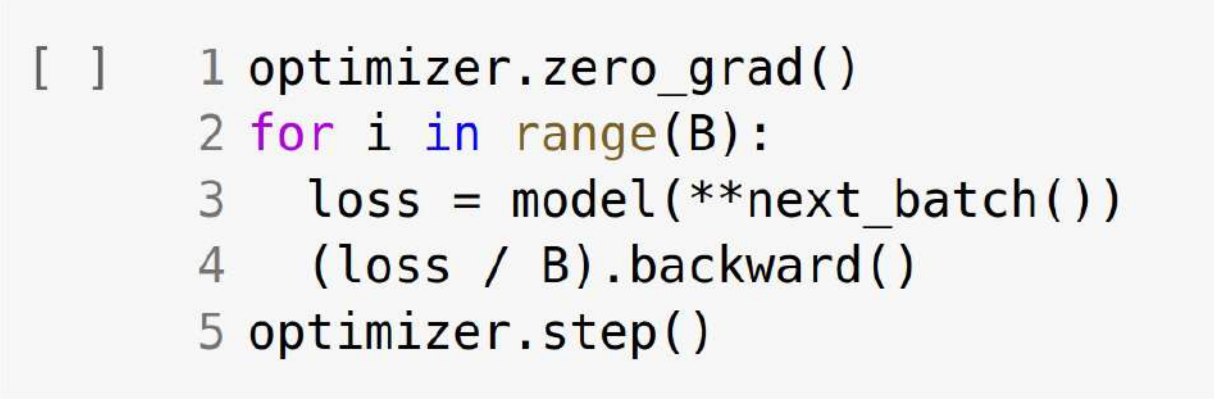
Q: What if a model is larger than GPU?

easy mode: cannot fit the right batch size

hard mode: cannot fit a single sample

expert mode: not even parameters!

Solution: accumulate grads from several training batches



```
optimizer.zero_grad()
for i in range(B):
  loss = model(**next_batch())
  (loss / B).backward()
optimizer.step()
```

Q: What if a model is larger than GPU?

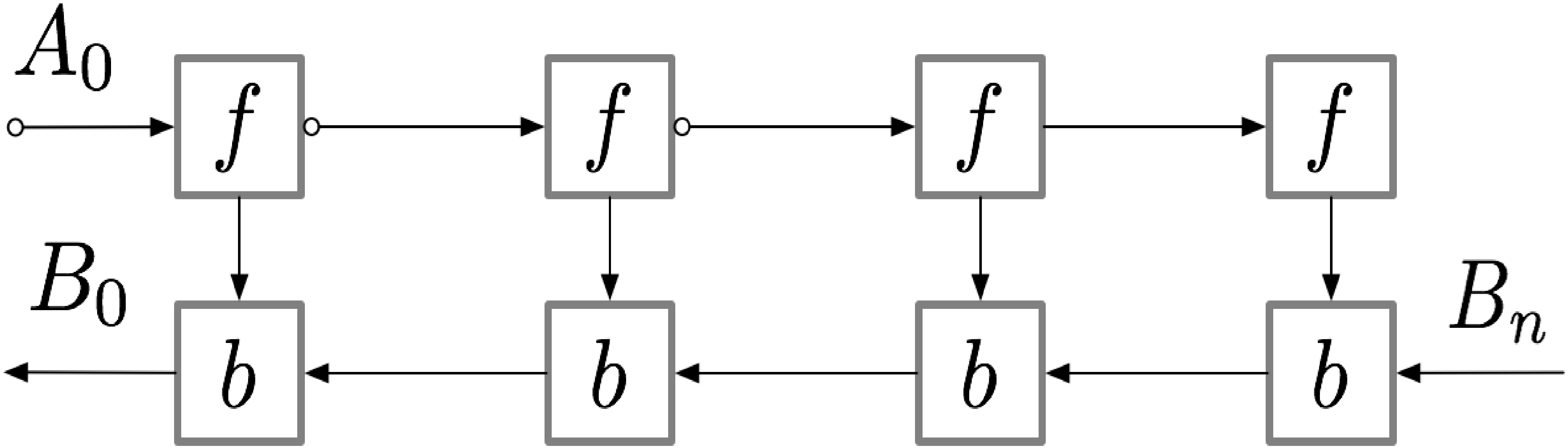
easy mode: cannot fit the right batch size

hard mode: cannot fit one training sample

expert mode: not even parameters!

Gradient checkpointing

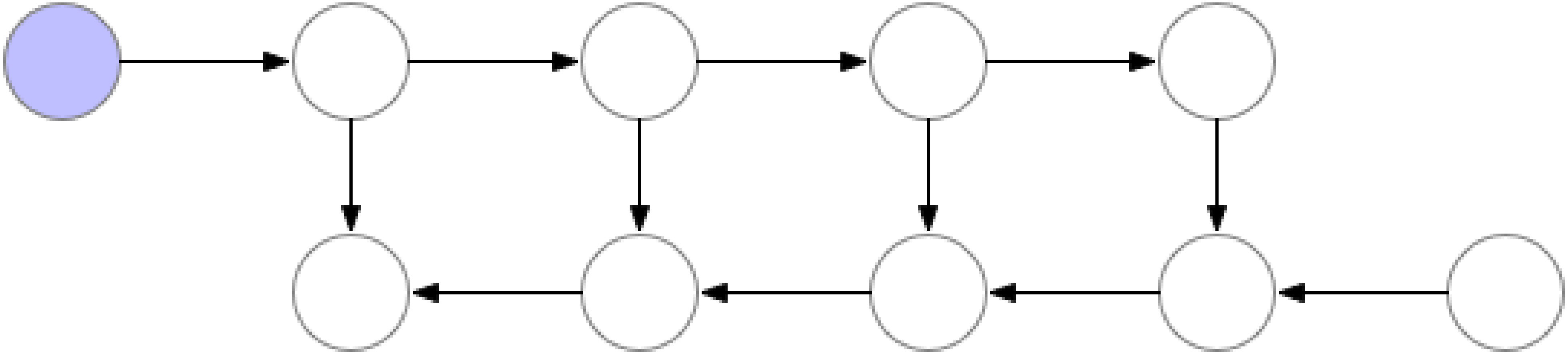
aka rematerialization



Paper (DL): arxiv.org/pdf/1604.06174.pdf
TF: github.com/cybertronai/gradient-checkpointing
Pytorch: pytorch.org/docs/stable/checkpoint.html

Gradient checkpointing

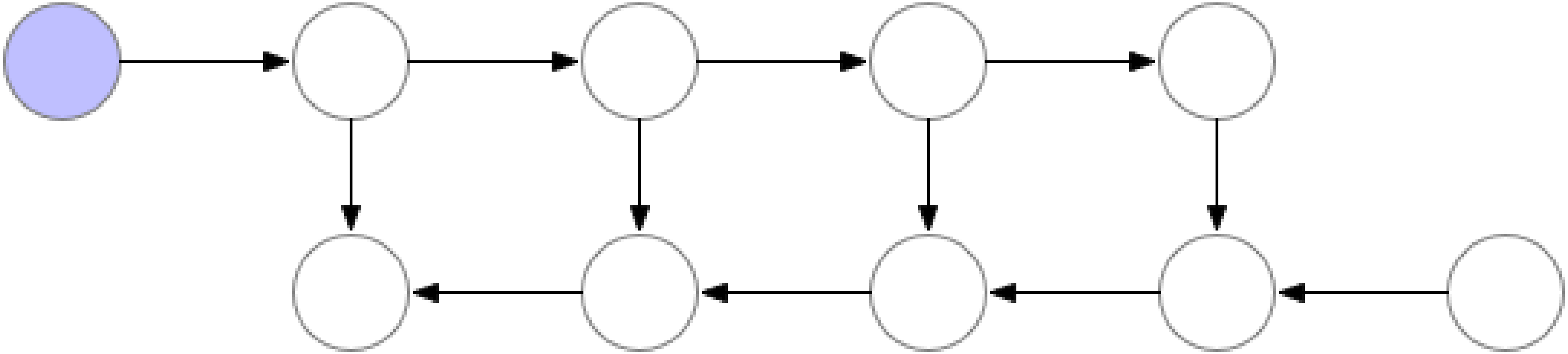
Normal backprop



Paper (DL): arxiv.org/pdf/1604.06174.pdf
TF: github.com/cybertronai/gradient-checkpointing
Pytorch: pytorch.org/docs/stable/checkpoint.html

Gradient checkpointing

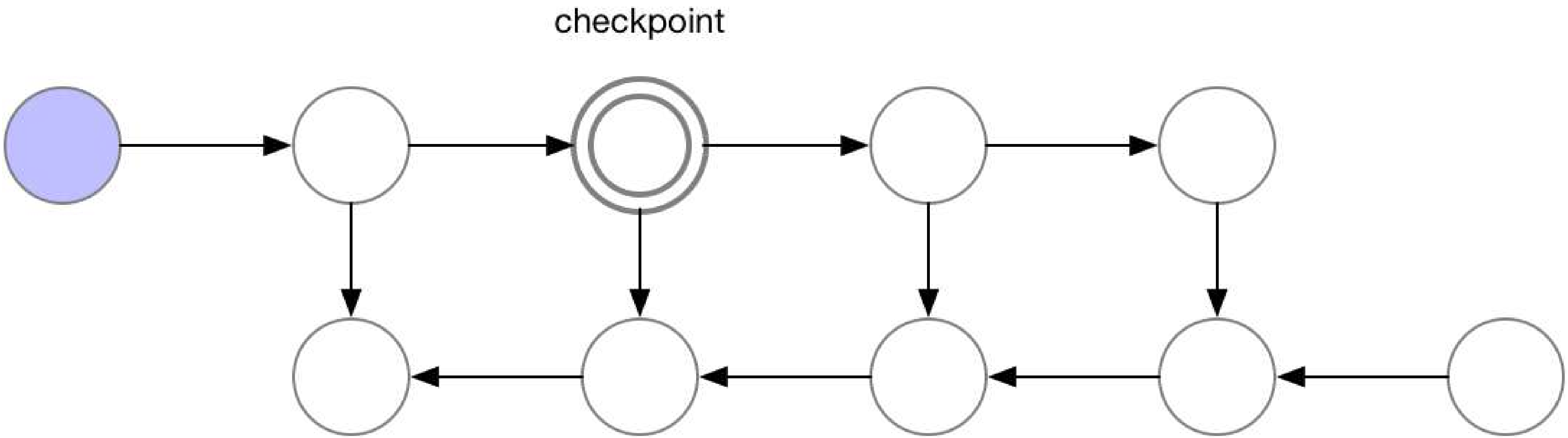
Full rematerialization



Paper (DL): arxiv.org/pdf/1604.06174.pdf
TF: github.com/cybertronai/gradient-checkpointing
Pytorch: pytorch.org/docs/stable/checkpoint.html

Gradient checkpointing

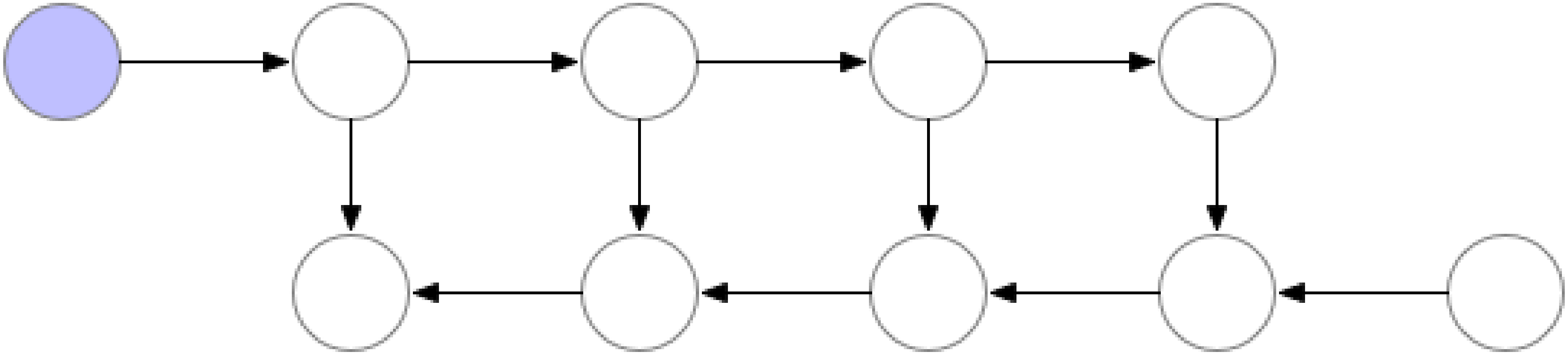
Single checkpoint



Paper (DL): arxiv.org/pdf/1604.06174.pdf
TF: github.com/cybertronai/gradient-checkpointing
Pytorch: pytorch.org/docs/stable/checkpoint.html

Gradient checkpointing

Single checkpoint



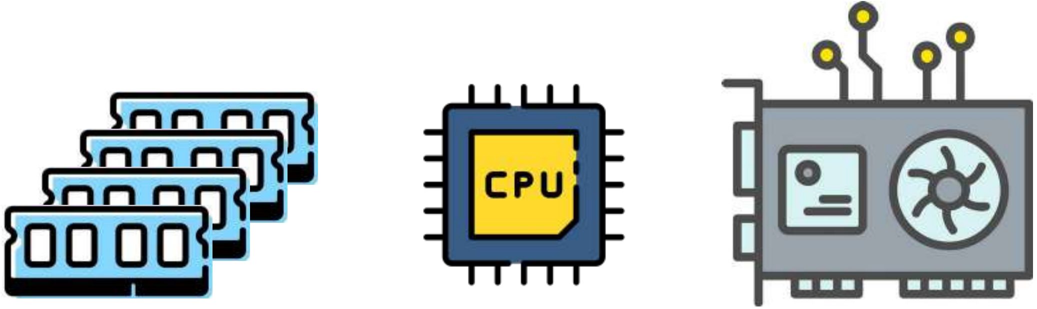
Paper (DL): arxiv.org/pdf/1604.06174.pdf
TF: github.com/cybertronai/gradient-checkpointing
Pytorch: pytorch.org/docs/stable/checkpoint.html

Q: What if a model is larger than GPU?

easy mode: cannot fit batch size 1

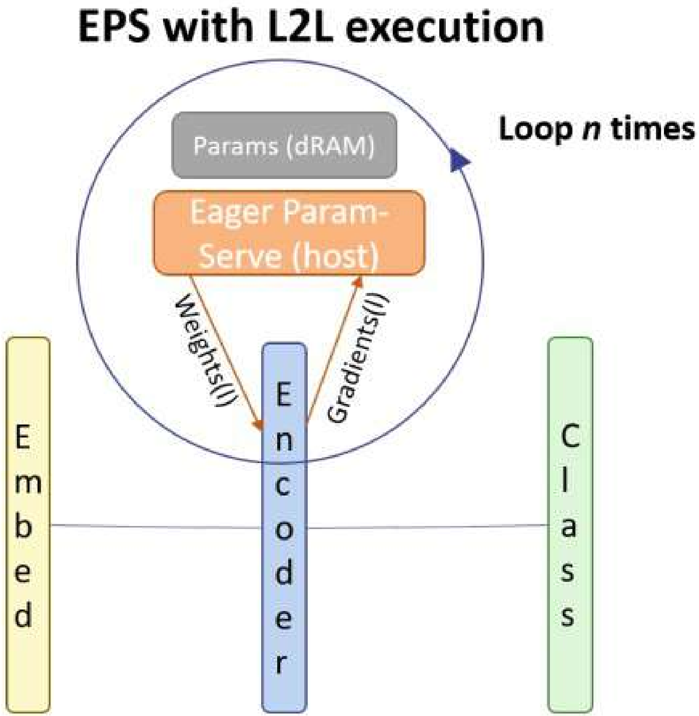
expert mode: not even parameters!

You still have one GPU… *(but not only a GPU)*



Memory offloading

L2L: https://arxiv.org/abs/2002.05645

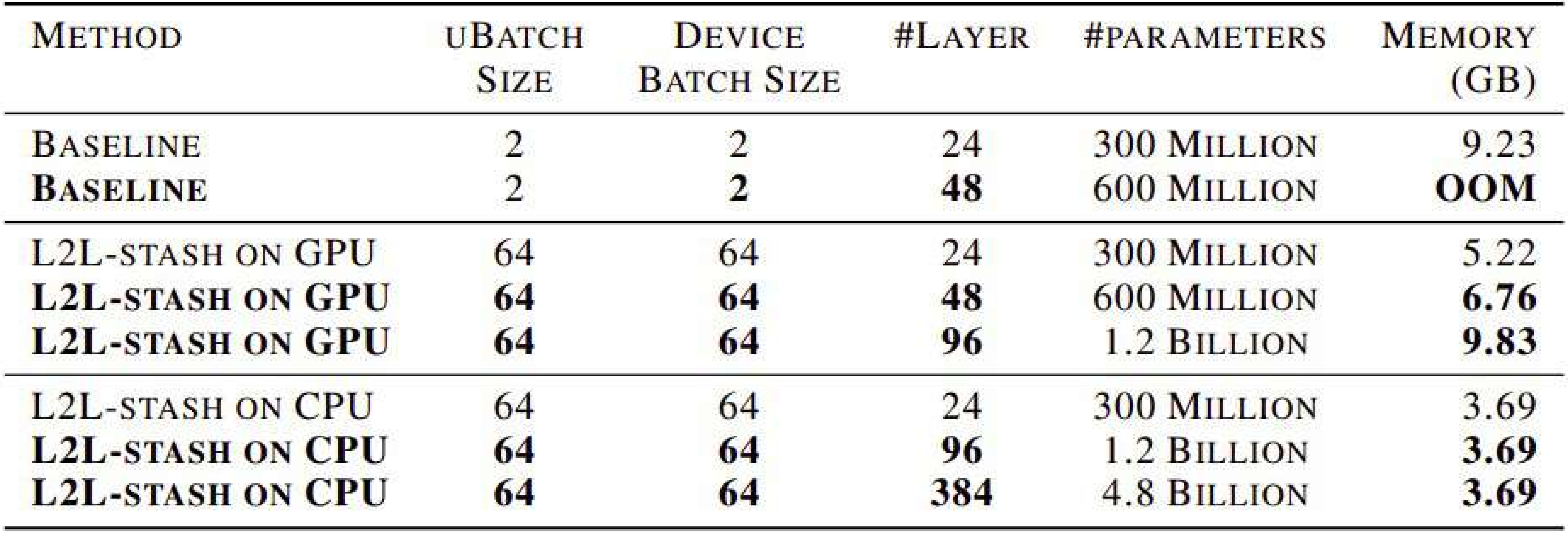


- Initialize all layers on CPU
- Move k layers at a time to GPU
- Remove layers after computation

- Fetch k+1-st layer while k-th runs
- Still 20-50% overhead

Memory offloading

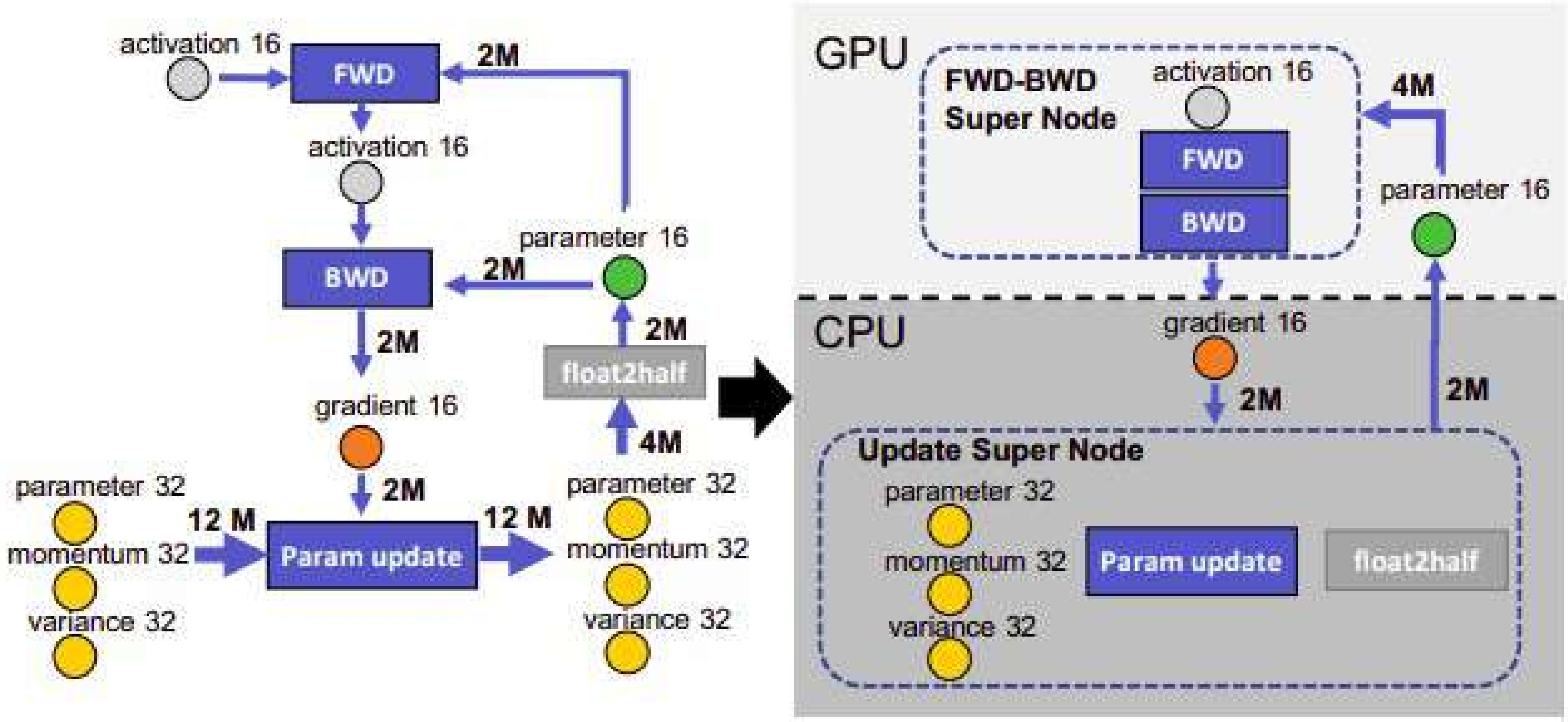
L2L: https://arxiv.org/abs/2002.05645



Method	uBatch Size	Device Batch Size	#Layer	#Parameters	Memory (GB)
Baseline	2	2	24	300 Million	9.23
Baseline	2	**2**	**48**	600 Million	**OOM**
L2L-stash on GPU	64	64	24	300 Million	5.22
L2L-stash on GPU	**64**	**64**	**48**	600 Million	**6.76**
L2L-stash on GPU	**64**	**64**	**96**	1.2 Billion	**9.83**
L2L-stash on CPU	64	64	24	300 Million	3.69
L2L-stash on CPU	**64**	**64**	**96**	1.2 Billion	**3.69**
L2L-stash on CPU	**64**	**64**	**384**	4.8 Billion	**3.69**

Memory offloading

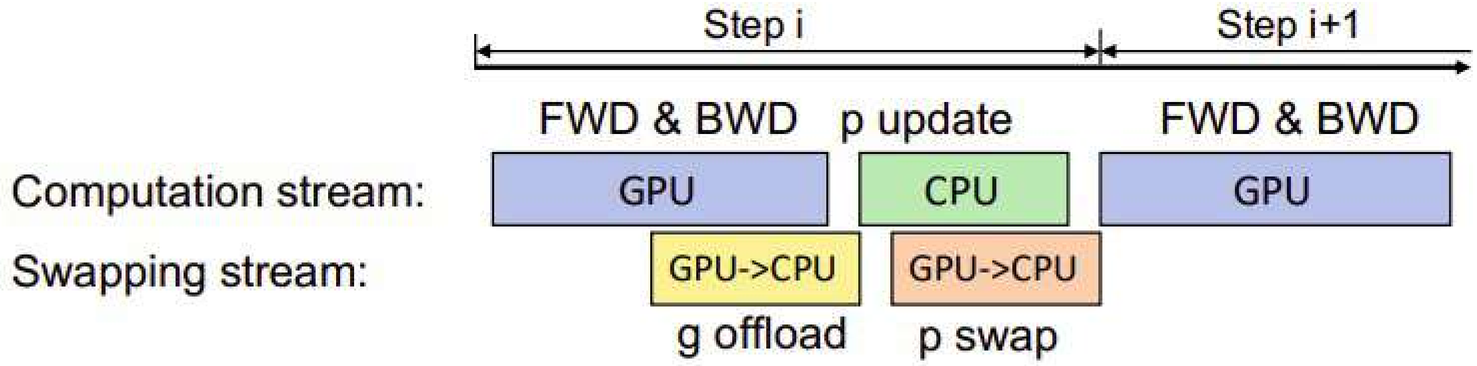
ZeRO-offload: https://arxiv.org/abs/2101.06840



Memory offloading

ZeRO-offload: https://arxiv.org/abs/2101.06840

- Offload **in parallel** with computation
- Use gradient checkpointing
- Delayed parameter update



Memory offloading

ZeRO-offload: https://arxiv.org/abs/2101.06840

- Offload in parallel with computation
- Use gradient checkpointing
- **Delayed parameter update**

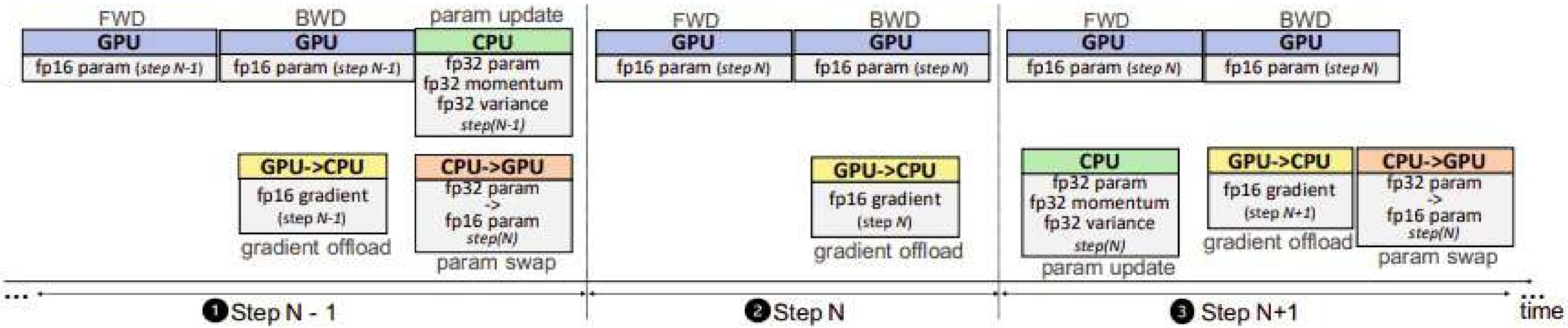
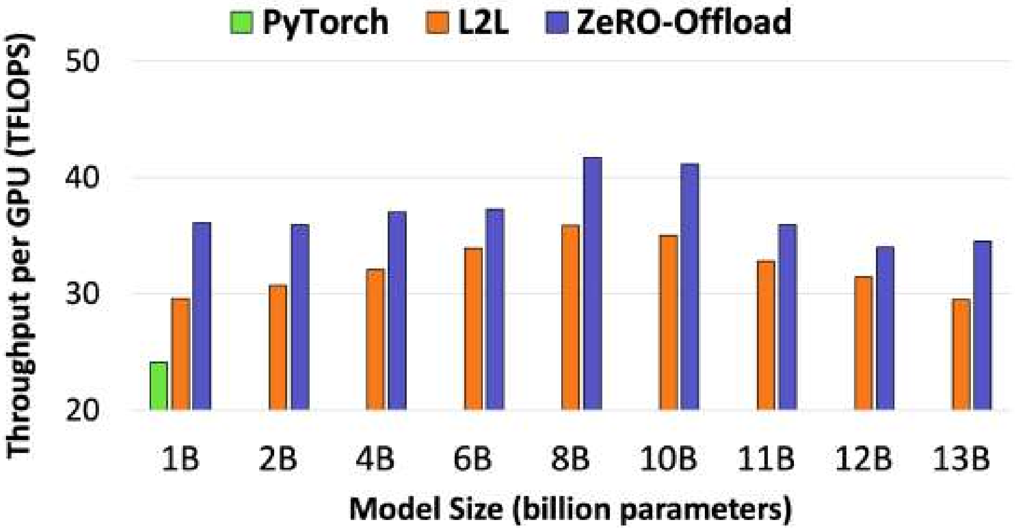


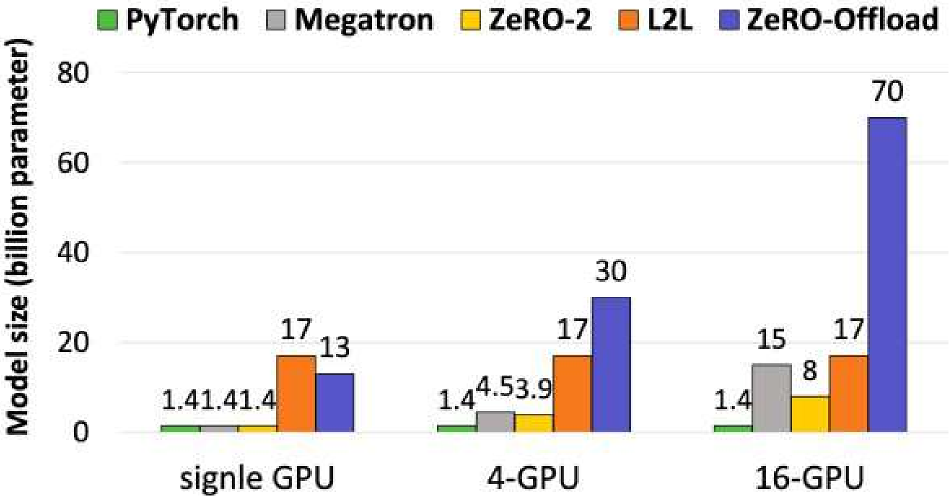
Figure 6: Delayed parameter update during the training process.

Memory offloading

ZeRO-offload: https://arxiv.org/abs/2101.06840

- Offload in parallel with computation
- Use gradient checkpointing
- Delayed parameter update





Q: What if a model is larger than GPU?

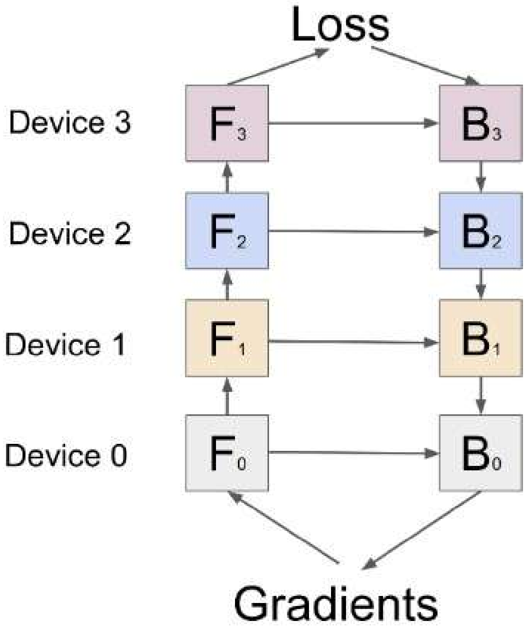
easy mode: cannot fit batch size 1

expert mode: not even parameters!

Can we do it better with multiple GPUs?

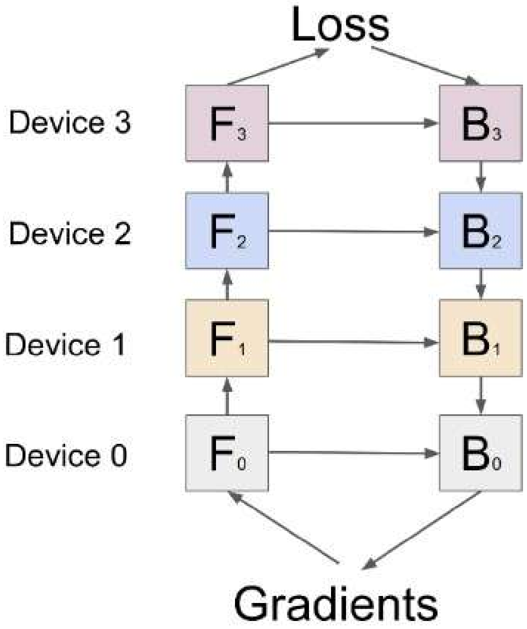
Model-parallel training

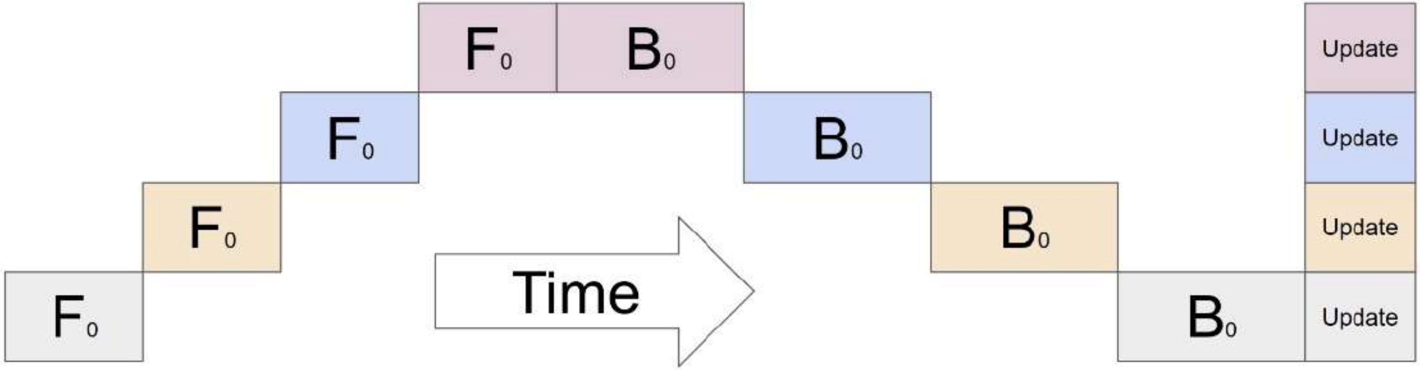
Q: What if a model is larger than GPU?



Model-parallel training

Q: What if a model is larger than GPU?





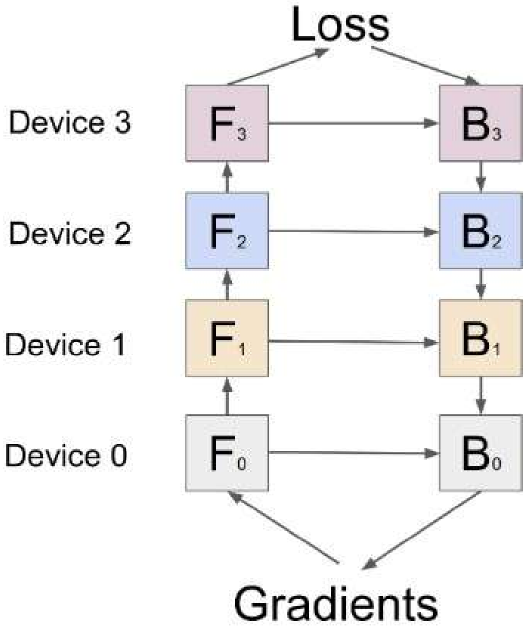
model size: O(N)
throughput: O(1)

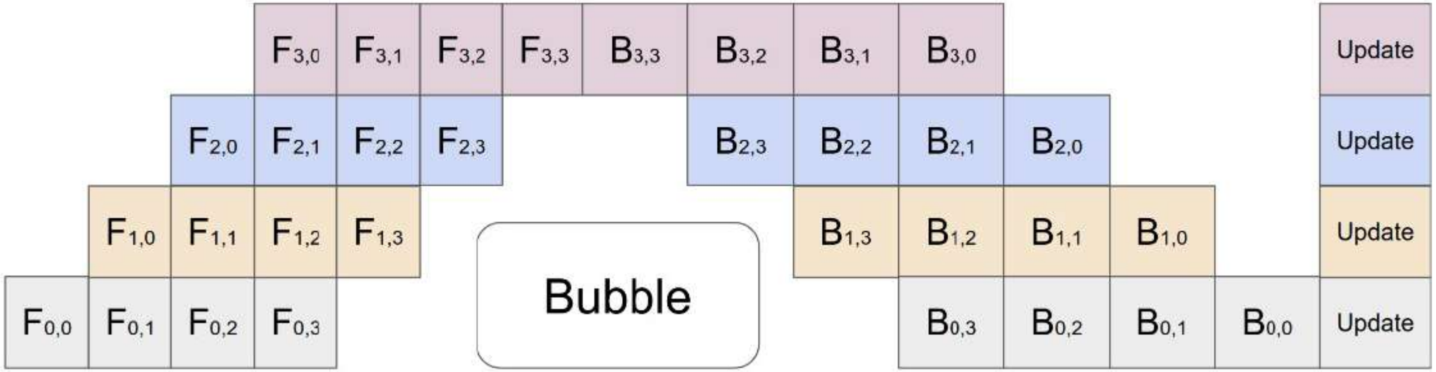
Q: Can we go faster?

Pipelining

GPipe: arxiv.org/abs/1811.06965 – good starting point, *not* the 1st paper

Idea: split data into micro-batches and form a pipeline (right)





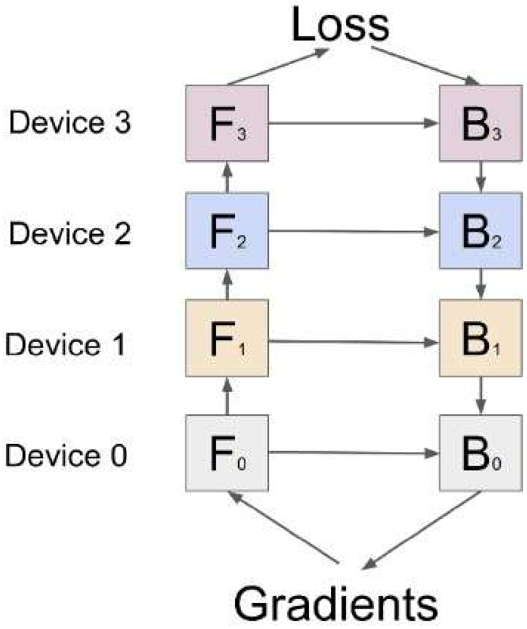
$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$ Update
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$ Update
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$ Update
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$ Update
Bubble

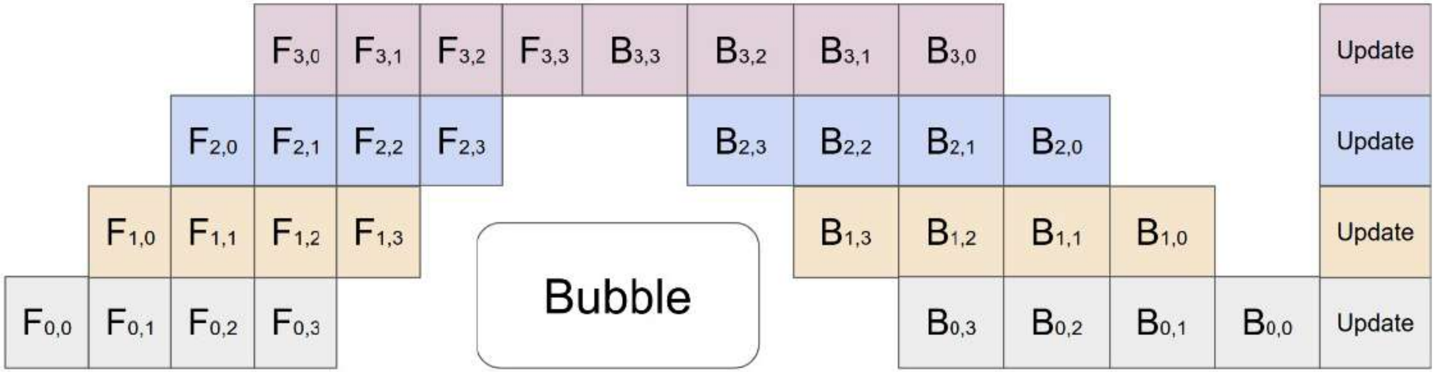
model size: O(n)
throughput: O(n) – with caveats

Pipelining

GPipe: arxiv.org/abs/1811.06965 – good starting point, *not* the 1st paper

Idea: split data into micro-batches and form a pipeline (right)




$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$ Update
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$ Update
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$ Update
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$ Update
Bubble

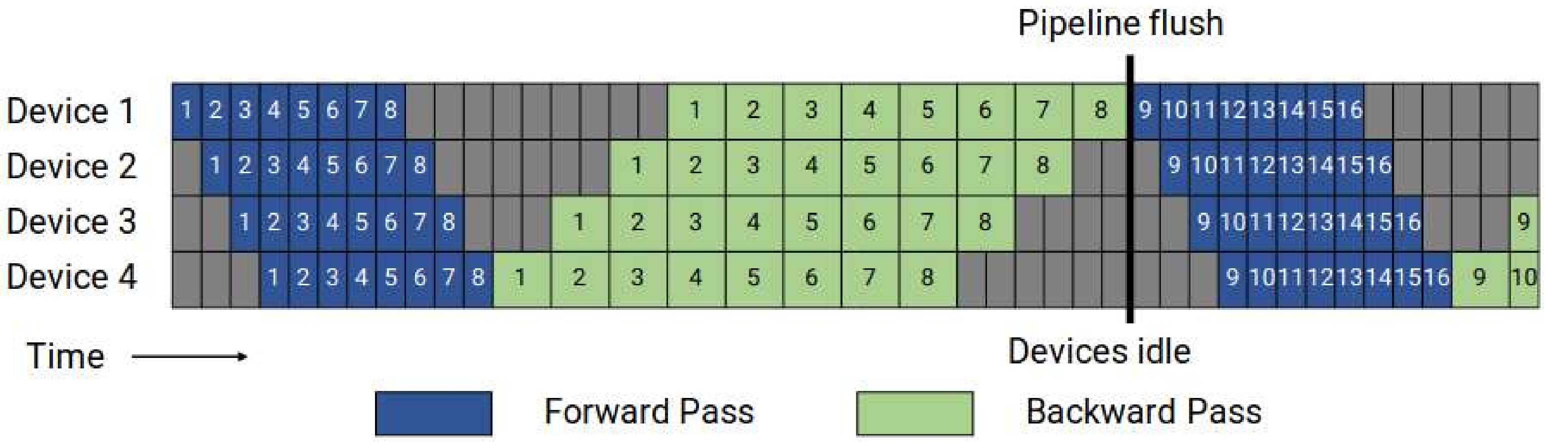
model size: O(n)
throughput: O(n) – with caveats

Q: Even faster?

Reducing the bubble

GPipe: arxiv.org/abs/1811.06965

GPipe:



… to be improved in a moment

Note: backward takes longer than forward in practice

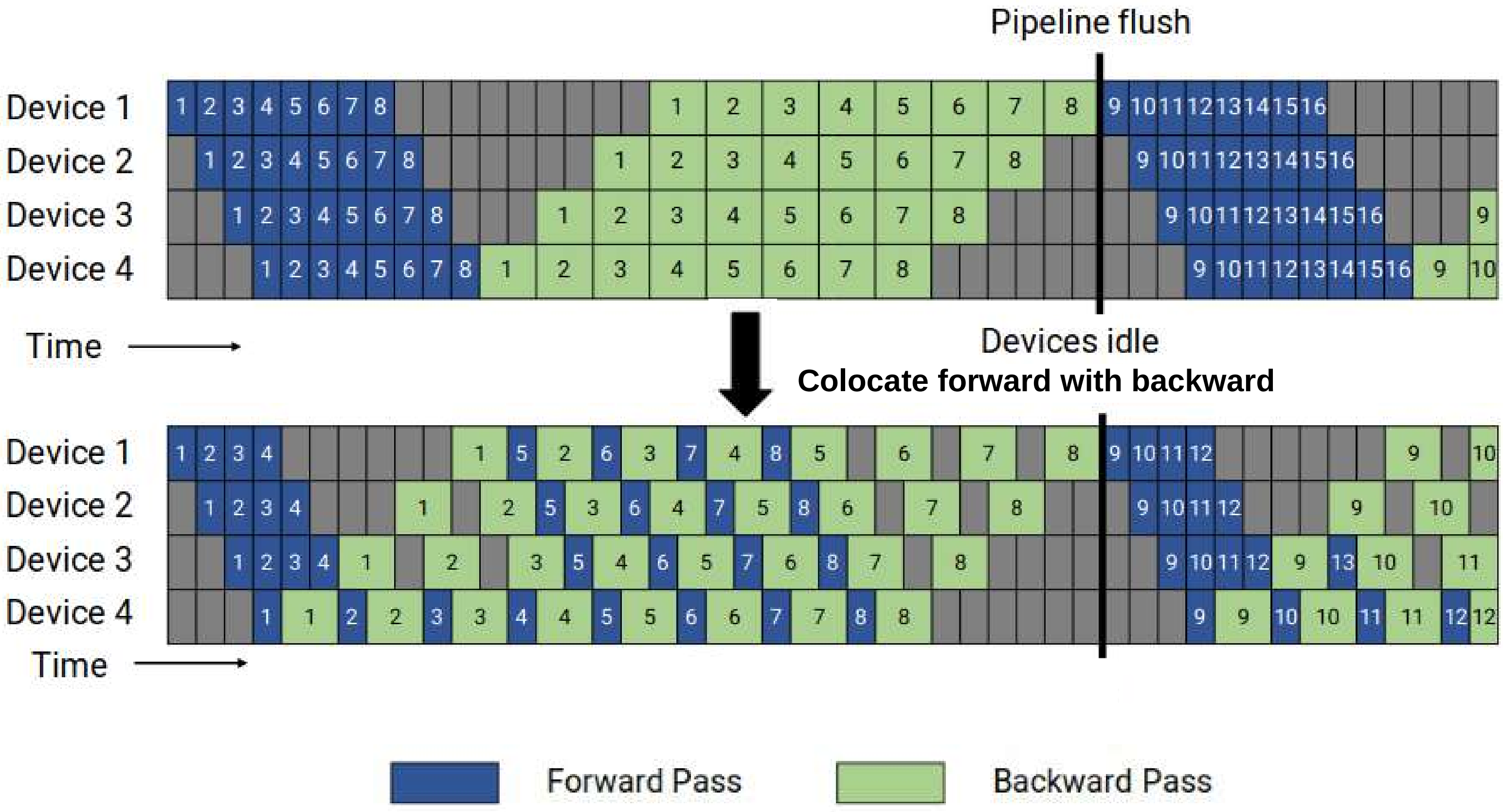
E.g. linear forward has one matmul,
backward has two matmuls (dW and dX)

Reducing the bubble

1F1B pipeline from Megatron: https://arxiv.org/abs/2104.04473

GPipe:

1F1B:



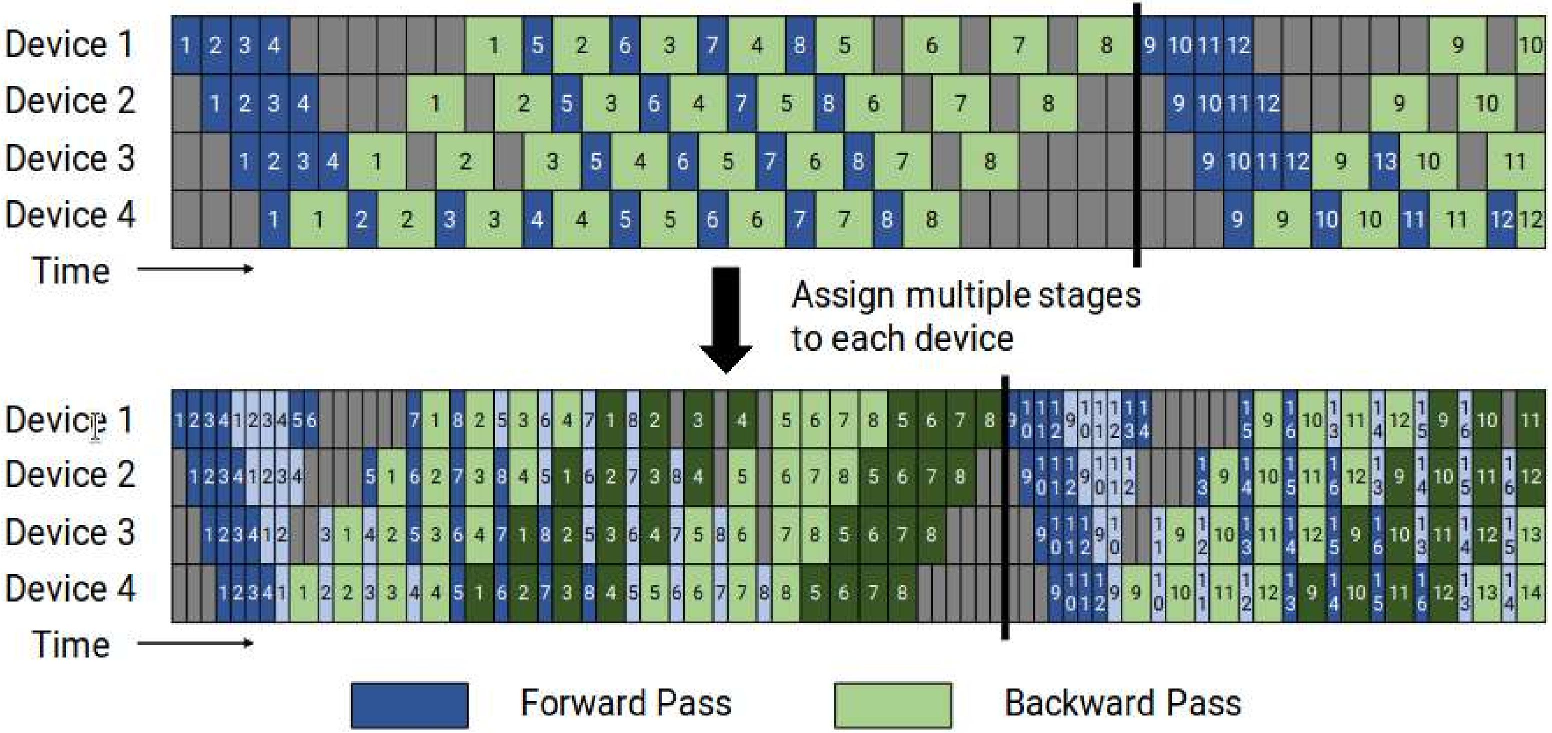
Colocate forward with backward

Reducing the bubble **(further)**

1F1B pipeline from Megatron: https://arxiv.org/abs/2104.04473

1F1B:

1F1B interleaved:



Reducing the bubble **(furtherer)**

ZB1P: "almost zero bubble" https://arxiv.org/abs/2401.10241

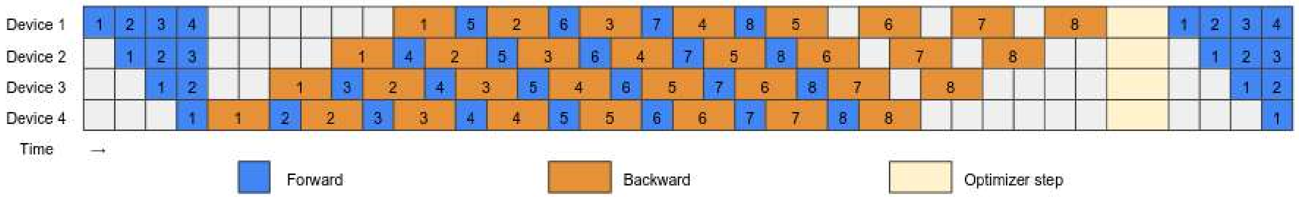


Figure 2: 1F1B pipeline schedule.

Idea: split backward into two ops:
- **w.r.t inputs** and **w.r.t. weights**

Grad w.r.t. weights doesn't block backward pass to prev stage

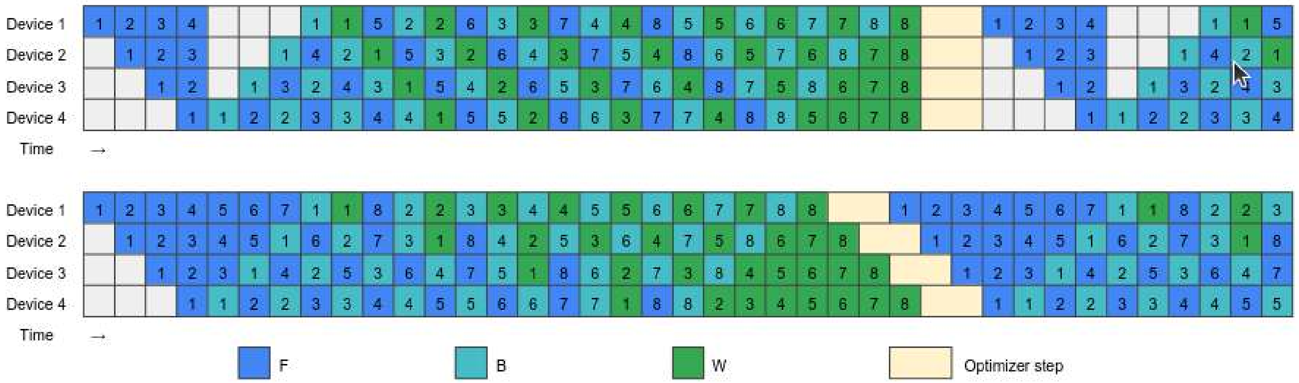


Figure 3: Handcrafted pipeline schedules, top: ZB-H1; bottom: ZB-H2

Reducing the bubble **(furtherer yet)**

Deepseek V1 schedule: https://arxiv.org/abs/2412.19437

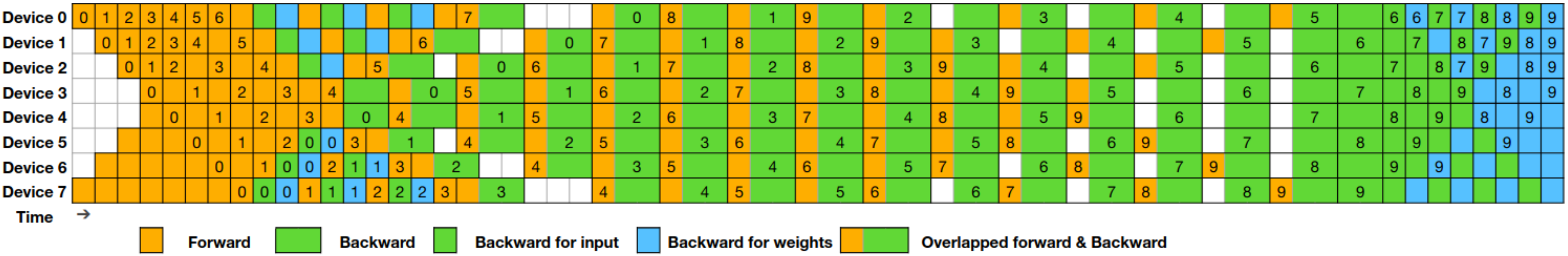


Figure 5 | Example DualPipe scheduling for 8 PP ranks and 20 micro-batches in two directions. The micro-batches in the reverse direction are symmetric to those in the forward direction, so we omit their batch ID for illustration simplicity. Two cells enclosed by a shared black border have mutually overlapped computation and communication.

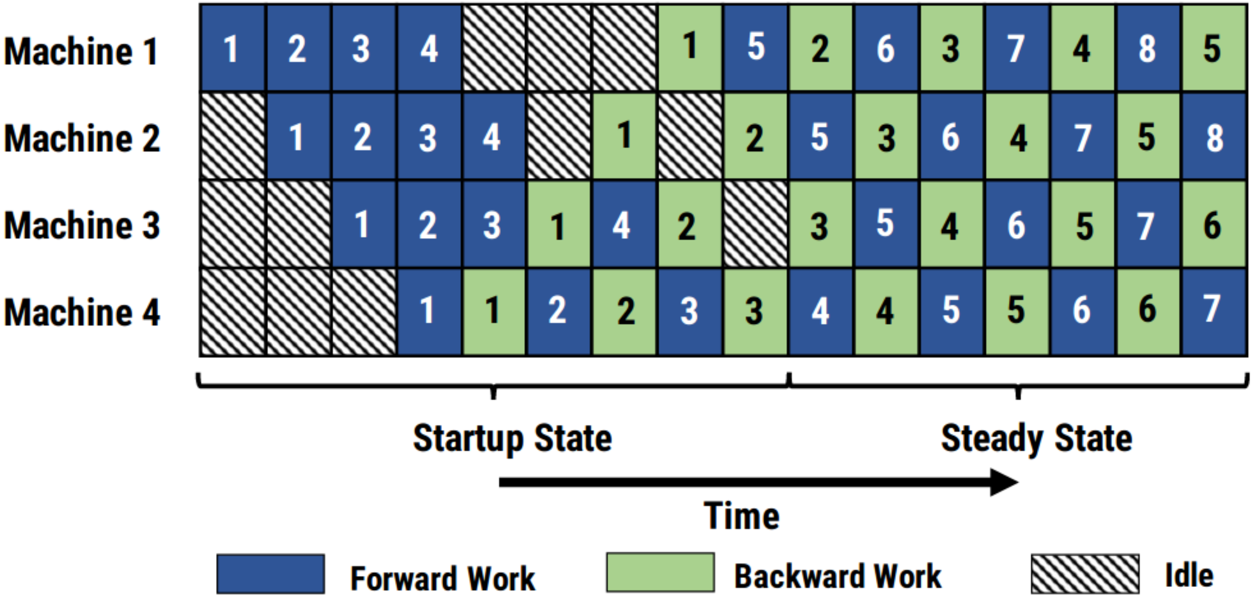
Asynchronous Pipelining

PipeDream: arxiv.org/abs/1806.03377

Idea: apply gradients with every microbatch for maximum throughput

Also neat:

- Automatically partition layers to GPUs via dynamic programming
- *Store k* past weight versions to reduce gradient staleness
- Aims at high latency



Machine 1	🟦1	🟦2	🟦3	🟦4	▨	▨	▨	🟩1	🟦5	🟩2	🟦6	🟩3	🟦7	🟩4	🟦8	🟩5
Machine 2	▨	🟦1	🟦2	🟦3	🟦4	▨	🟩1	▨	🟩2	🟦5	🟩3	🟦6	🟩4	🟦7	🟩5	🟦8
Machine 3	▨	▨	🟦1	🟦2	🟦3	🟩1	🟦4	🟩2	▨	🟩3	🟦5	🟩4	🟦6	🟩5	🟦7	🟩6
Machine 4	▨	▨	▨	🟦1	🟩1	🟦2	🟩2	🟦3	🟩3	🟦4	🟩4	🟦5	🟩5	🟦6	🟩6	🟦7

Pipelining Recap

When to use:

- model doesn’t fit on GPU; have multiple GPUs
- if model fits, but not the activations: **???**
- if model doesn’t fit, but you only have one GPU: **???**

How to use:

(just a moment...)

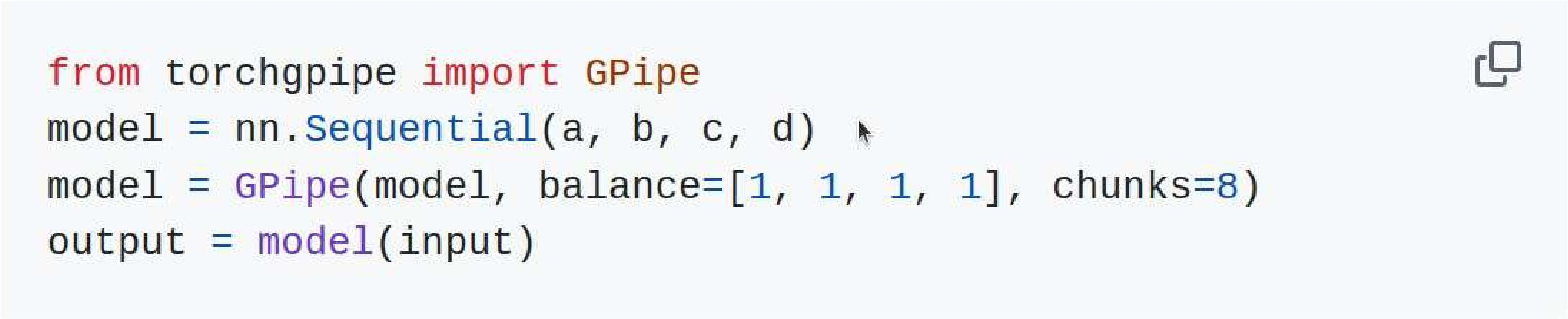
Pipelining Recap

When to use:

- model doesn't fit on GPU; have multiple GPUs
- if model fits, but not the activations: just do grad checkpointing!
- if model doesn't fit, but you only have one GPU: offloading!

How to use:

- Basic implementation (GPipe): github.com/kakaobrain/torchgpipe

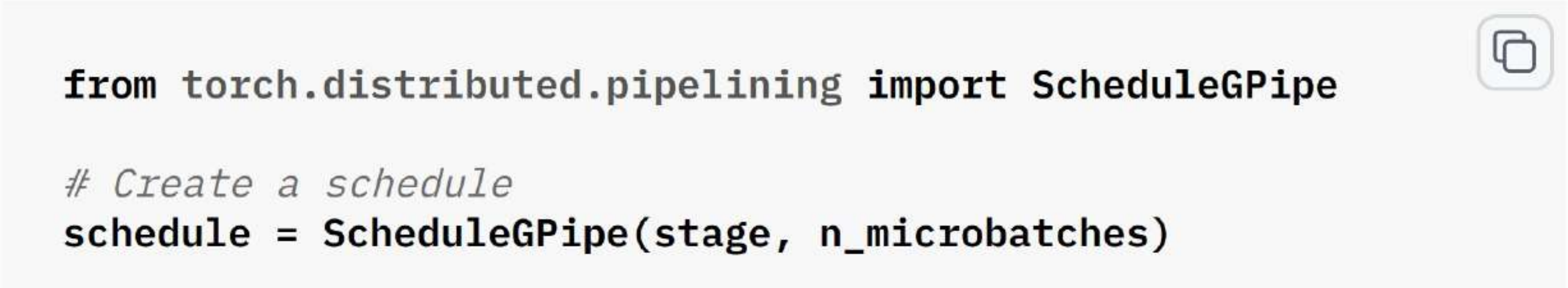


```
from torchgpipe import GPipe
model = nn.Sequential(a, b, c, d)
model = GPipe(model, balance=[1, 1, 1, 1], chunks=8)
output = model(input)
```

Pipelining Recap

When to use:
- model doesn’t fit on GPU; have multiple GPUs
- if model fits, but not the activations: just do grad checkpointing!
- if model doesn’t fit, but you only have one GPU: offloading!

How to use:
- Basic implementation (GPipe): github.com/kakaobrain/torchgpipe
- PyTorch built-in: pytorch.org/tutorials/intermediate/pipelining_tutorial.html



```
from torch.distributed.pipelining import ScheduleGPipe

# Create a schedule
schedule = ScheduleGPipe(stage, n_microbatches)
```

uses torch.distributed (torchrun) | supports GPipe, 1F1B, extendable!

Pipelining Recap

When to use:
- model doesn’t fit on GPU; have multiple GPUs
- if model fits, but not the activations: just do grad checkpointing!
- if model doesn’t fit, but you only have one GPU: offloading!

How to use:
- Basic implementation (GPipe): github.com/kakaobrain/torchgpipe
- PyTorch built-in: pytorch.org/tutorials/intermediate/pipelining_tutorial.html
- DeepSpeed: https://deepspeed.readthedocs.io/en/latest/pipeline.html

Custom pipelines in many applications
- Megatron-LM: https://github.com/NVIDIA/Megatron-LM (transformer-specific)
- Megablocks: https://github.com/databricks/megablocks (mixture-of-experts)

Pipelining Recap

When to use:

- model doesn’t fit on GPU; have multiple GPUs
- if model fits, but not the activations: just do grad checkpointing!
- if model doesn’t fit, but you only have one GPU: offloading!

How to use:

- Basic implementation (GPipe): github.com/kakaobrain/torchgpipe
- PyTorch built-in: pytorch.org/tutorials/intermediate/pipelining_tutorial.html
- DeepSpeed: https://deepspeed.readthedocs.io/en/latest/pipeline.html

Problems:

- Bubbles = wasted compute time (duh)
- What if model layers aren’t symmetric? (e.g. LLM “head”, local attn, ViT pooling)

Balancing a pipeline is a world of hurt.

[short break]
How else can we run a large model over multiple GPUs / hosts?

Tensor-parallel training

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

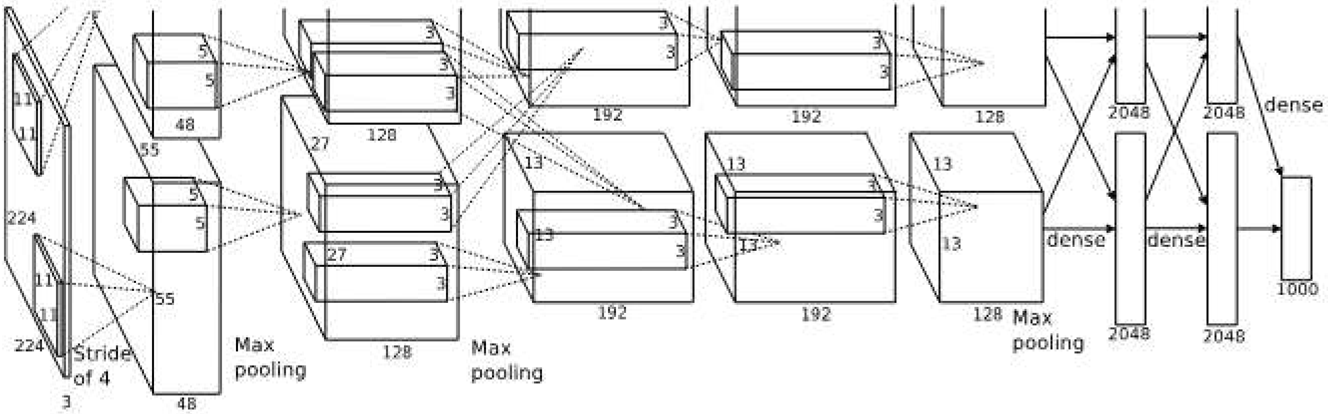
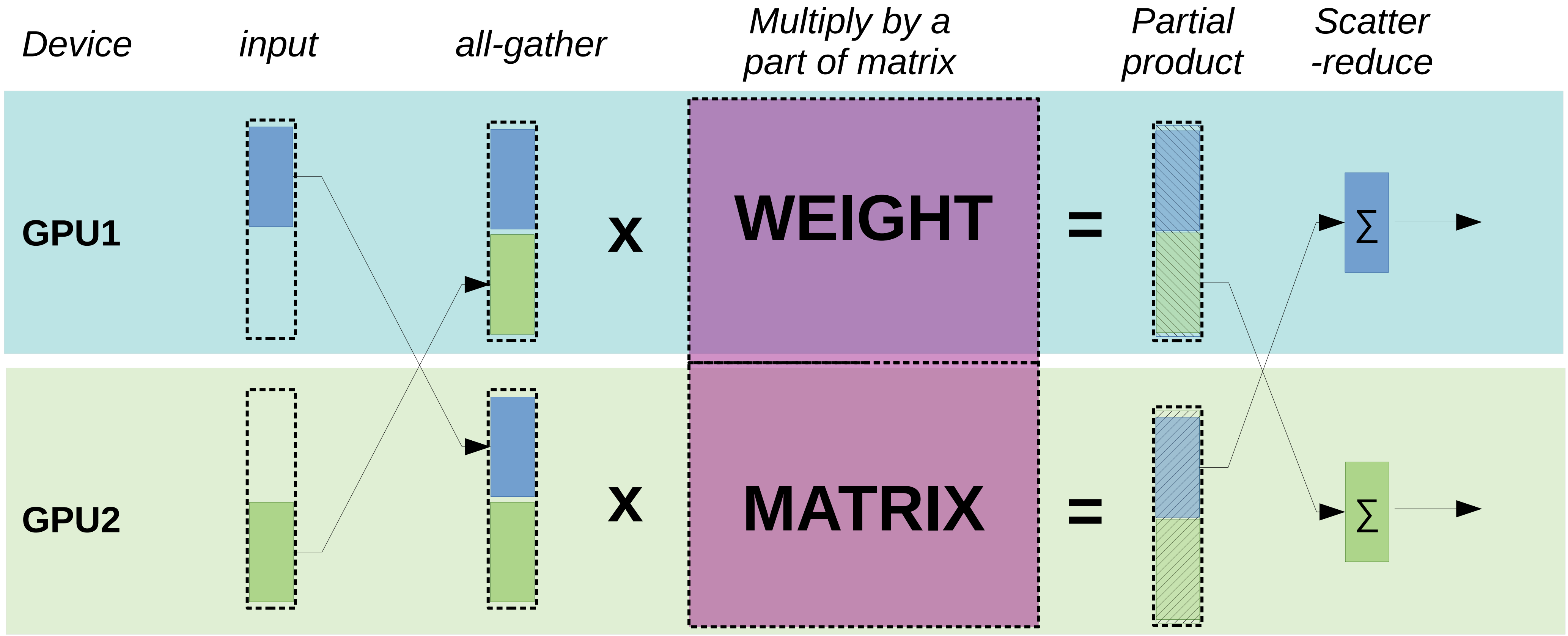


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

See also: DP + TP https://arxiv.org/abs/1404.5997

Tensor-parallel training



Device

input

all-gather

Multiply by a part of matrix

Partial product

Scatter -reduce

GPU1

GPU2

X

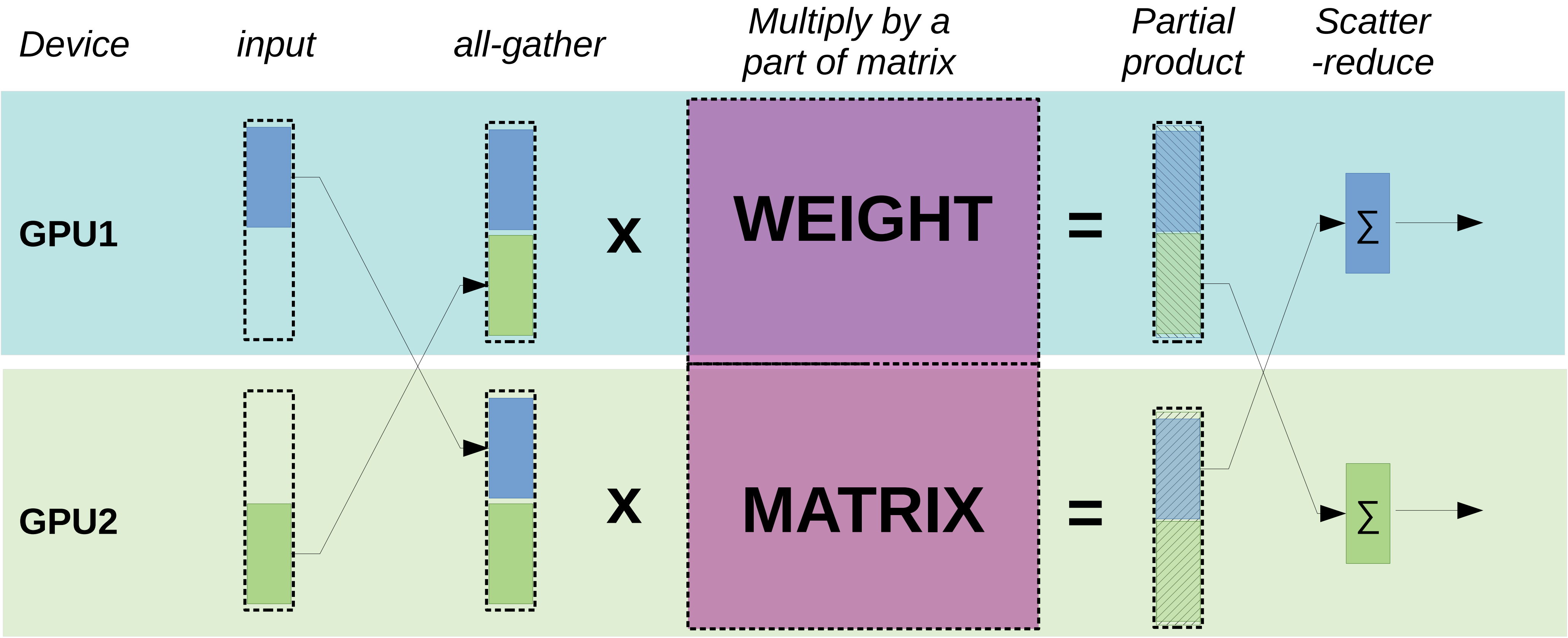
WEIGHT

MATRIX

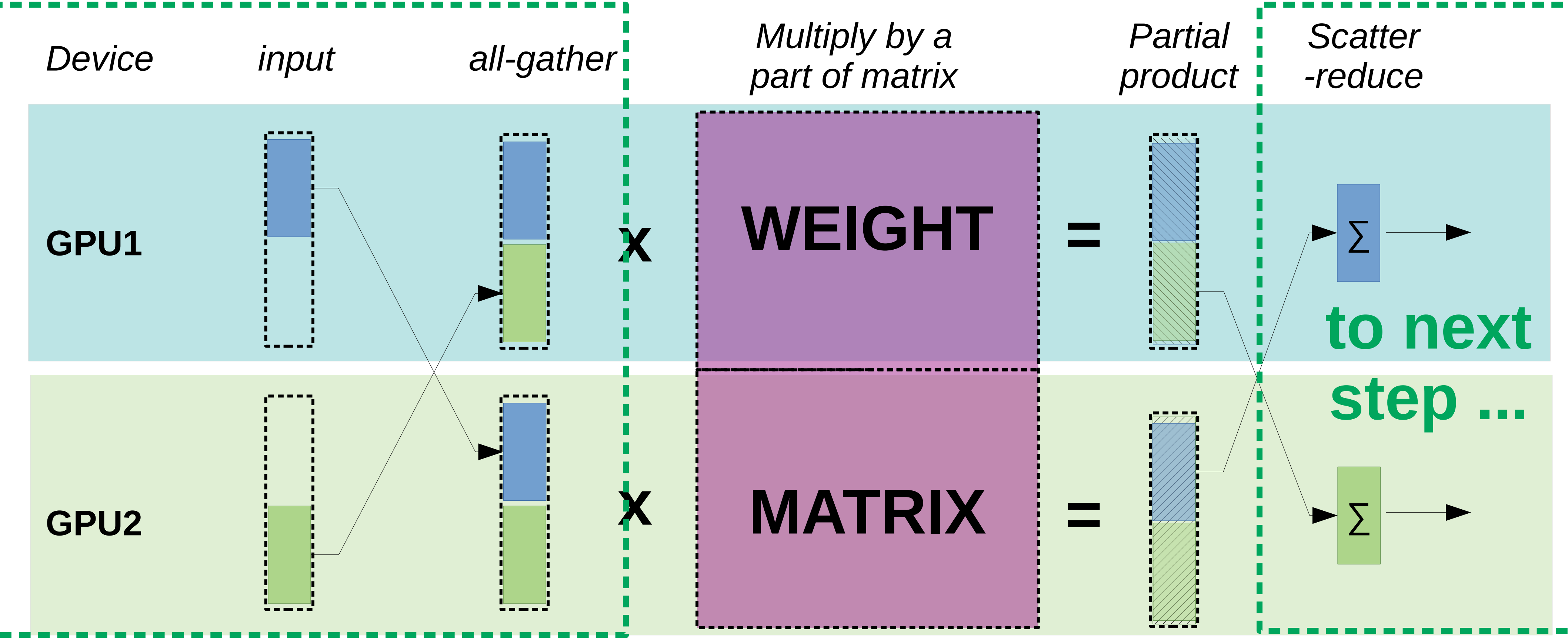
=

∑

Q: find AllReduce op here



Q: find AllReduce op here



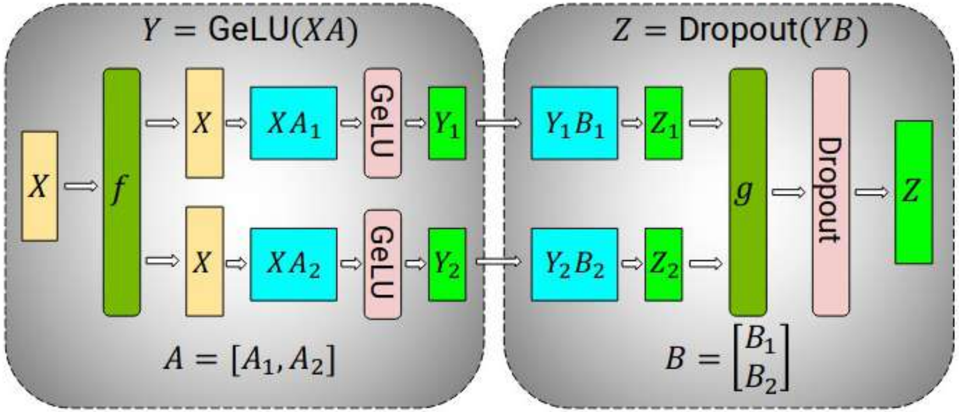
Tensor-parallel training

https://arxiv.org/pdf/2104.04473

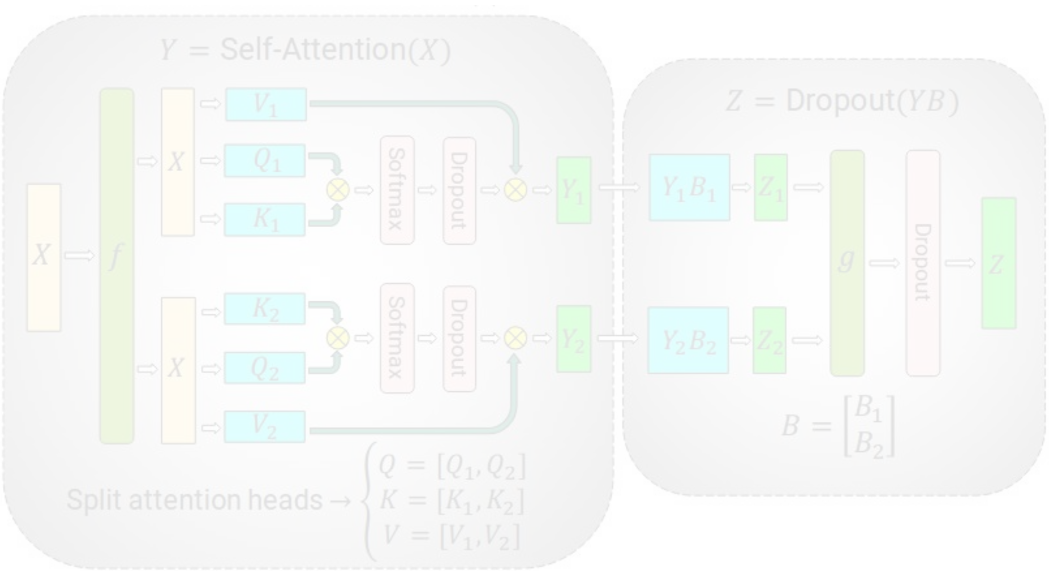
Mix and match parallelism directions to reduce synchronization

MLP: split over neurons

Attention: split over heads



(a) MLP.



(b) Self-Attention.

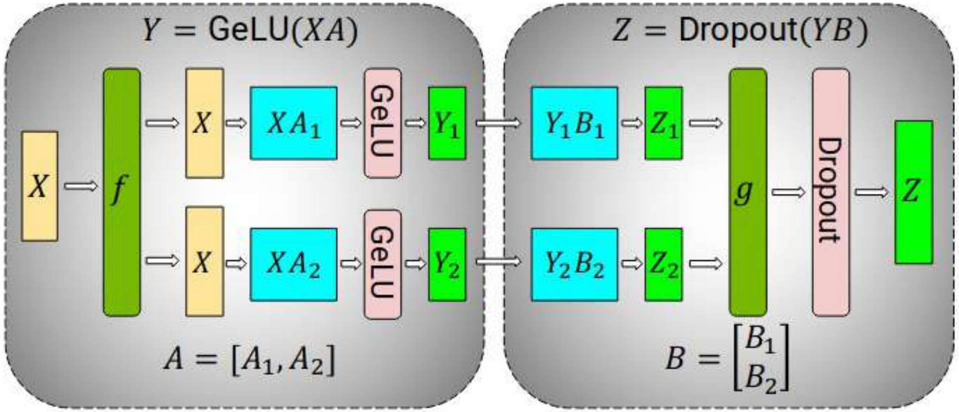
Figure 5: Blocks of transformer model partitioned with tensor model parallelism (figures borrowed from Megatron [40]). f and g are conjugate. f is the identity operator in the forward pass and all-reduce in the backward pass, while g is the reverse.

Tensor-parallel training

https://arxiv.org/pdf/2104.04473

Mix and match parallelism directions to reduce synchronization

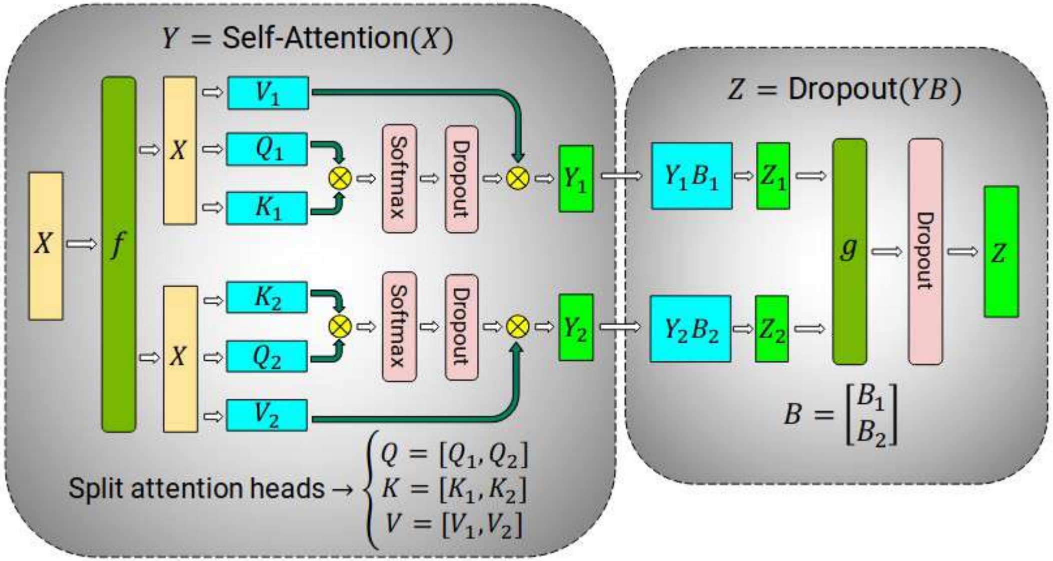
MLP: split over neurons



(a) MLP.

Figure 5: Blocks of transformer model partitioned with tensor model parallelism (figures borrowed from Megatron [40]). f and g are conjugate. f is the identity operator in the forward pass and all-reduce in the backward pass, while g is the reverse.

Attention: split over heads



(b) Self-Attention.

Sequence Parallelism

https://arxiv.org/abs/2309.14509

Avoid storing the all activations on every device

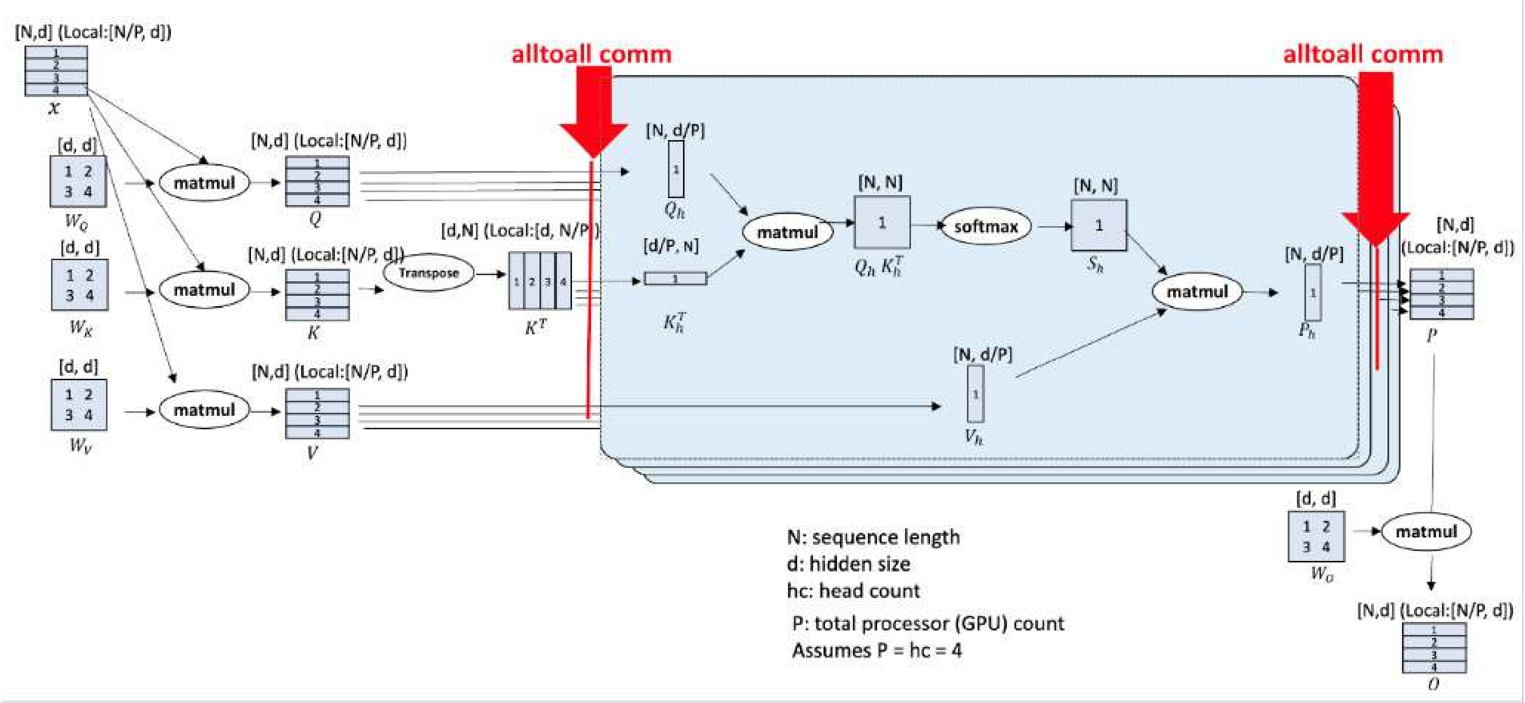


Figure 2: DeepSpeed sequence parallelism (DeepSpeed-Ulysses) design

[MOAR] Sequence Parallelism

Early mention of parallelism over sequences
https://arxiv.org/abs/2105.05720

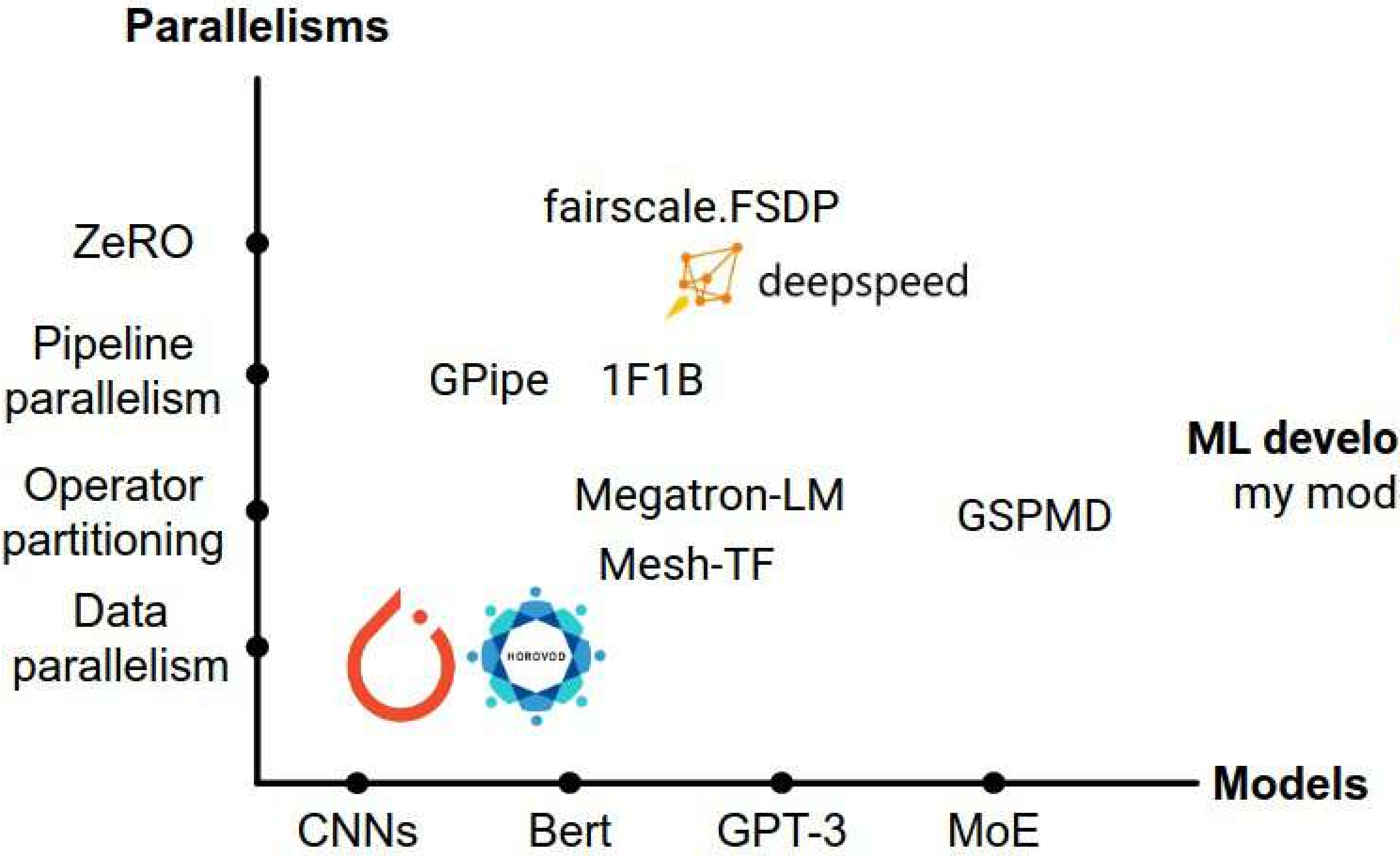
DeepSpeed Ulysses – the method from previous slide
https://arxiv.org/abs/2309.14509

Ring Attention – compute attention dot / softmax in parallel
https://arxiv.org/abs/2310.01889

FLUX – overlap computation and communication with custom kernels
https://arxiv.org/abs/2406.06858

Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model



Parallelisms

ZeRO

Pipeline parallelism

Operator partitioning

Data parallelism

fairscale.FSDP

deepspeed

GPipe

1F1B

Megatron-LM

Mesh-TF

GSPMD

HOROVOD

Models

CNNs

Bert

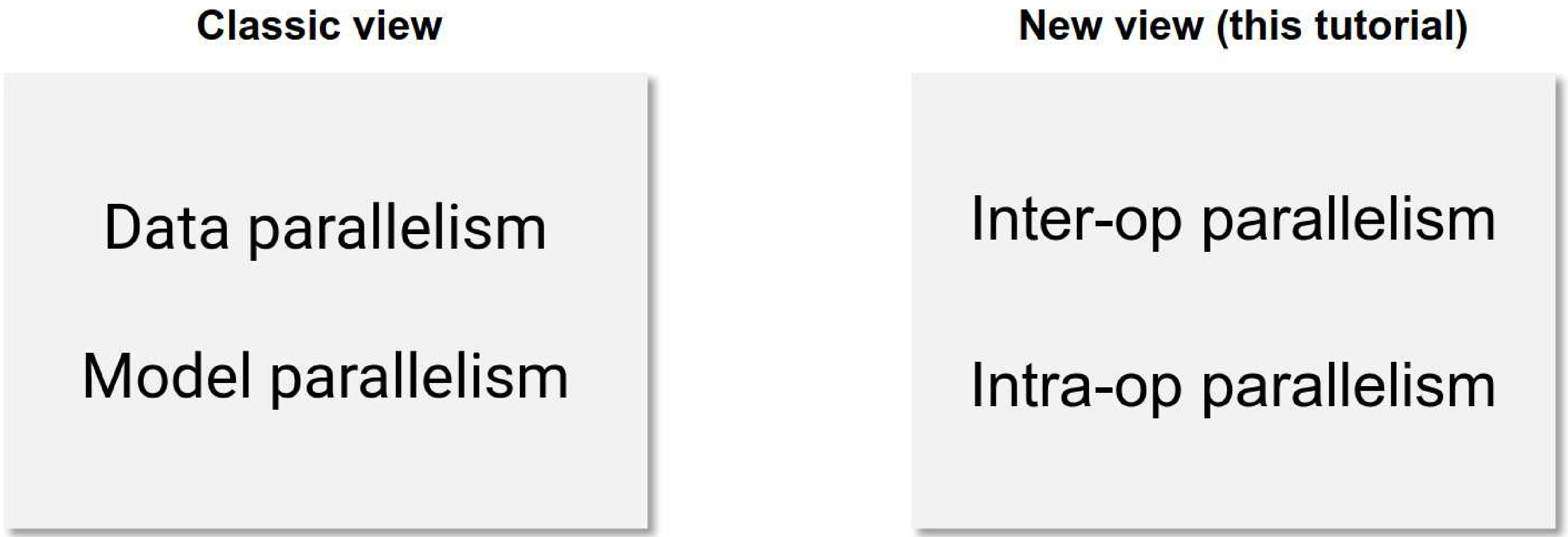
GPT-3

MoE

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

Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model



Classic view

Data parallelism

Model parallelism

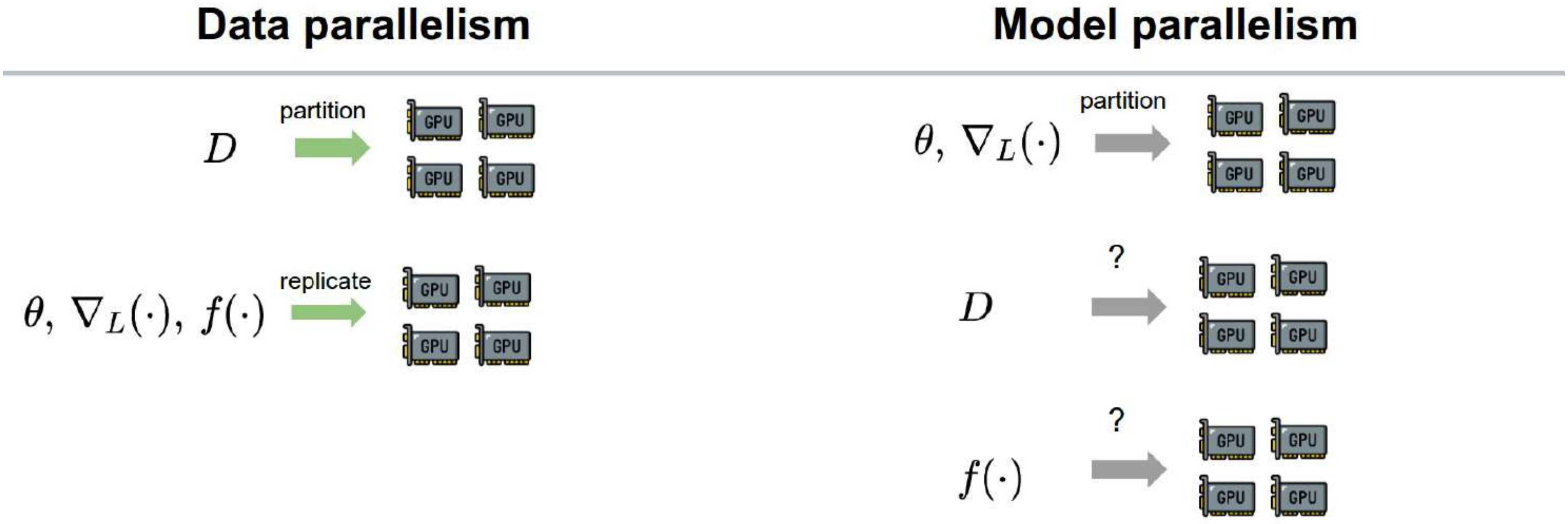
New view (this tutorial)

Inter-op parallelism

Intra-op parallelism

Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model



Data parallelism

$D \xrightarrow{\text{partition}}$ GPU GPU GPU GPU

$\theta,\ \nabla_L(\cdot),\ f(\cdot) \xrightarrow{\text{replicate}}$ GPU GPU GPU GPU

Model parallelism

$\theta,\ \nabla_L(\cdot) \xrightarrow{\text{partition}}$ GPU GPU GPU GPU

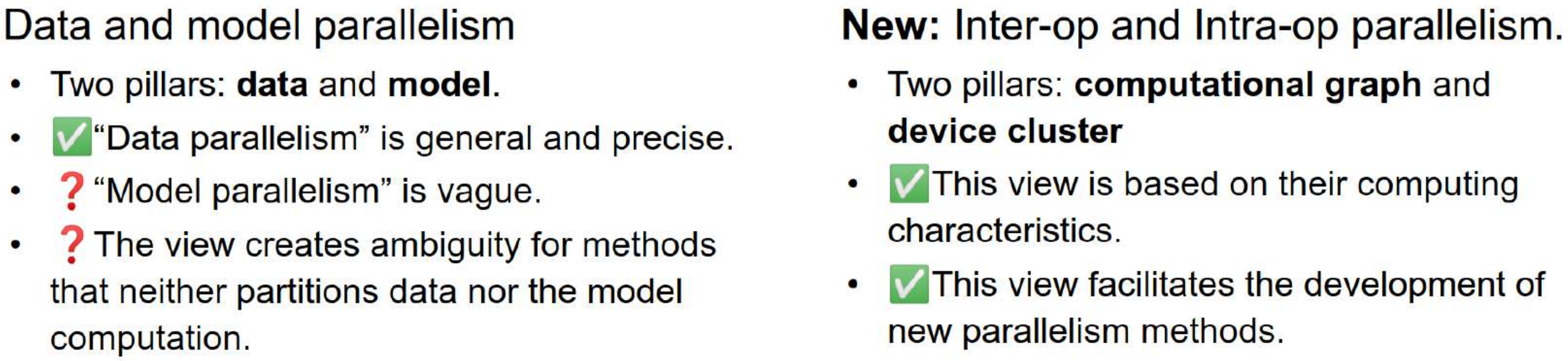
$D \xrightarrow{?}$ GPU GPU GPU GPU

$f(\cdot) \xrightarrow{?}$ GPU GPU GPU GPU

$$\underbrace{\theta^{(t+1)}}_{\text{parameter}} = \underbrace{f}_{\substack{\text{weight update}\\\text{(sgd, adam, etc.)}}}\Big(\theta^{(t)},\ \underbrace{\nabla_L}_{\substack{\text{model}\\\text{(CNN, GPT, etc.)}}}\big(\theta^{(t)},\ \underbrace{D^{(t)}}_{\text{data}}\big)\Big)$$

Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model



Data and model parallelism

- Two pillars: **data** and **model**.
- ✅“Data parallelism” is general and precise.
- ❓“Model parallelism” is vague.
- ❓The view creates ambiguity for methods that neither partitions data nor the model computation.

New: Inter-op and Intra-op parallelism.

- Two pillars: **computational graph** and **device cluster**
- ✅This view is based on their computing characteristics.
- ✅This view facilitates the development of new parallelism methods.

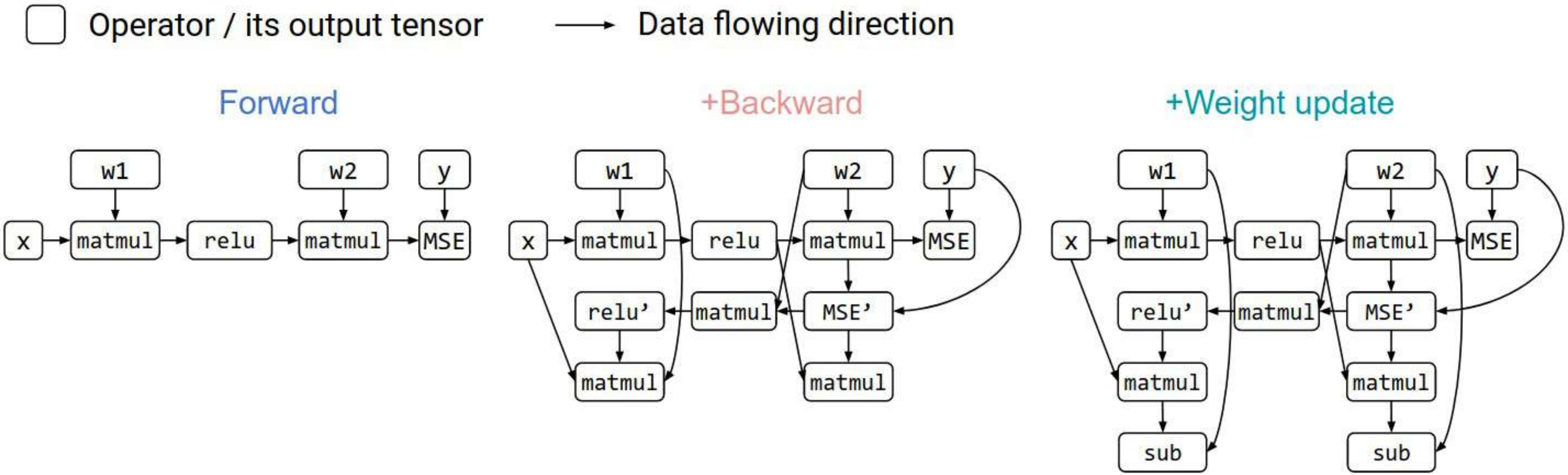
Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model

$$\theta^{(t+1)} = f\left(\theta^{(t)},\, \nabla_L\left(\theta^{(t)},\, D^{(t)}\right)\right)$$

$$L = \mathrm{MSE}(w_2 \cdot \mathrm{ReLU}(w_1 x),\, y) \quad \theta = \{w_1, w_2\},\, D = \{(x, y)\}$$

$$f(\theta, \nabla_L) = \theta - \nabla_L$$



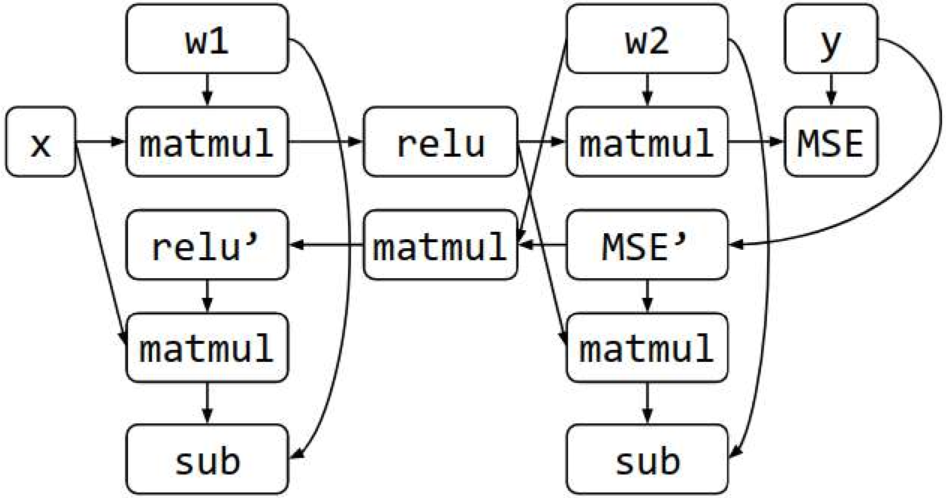
Automated parallelism

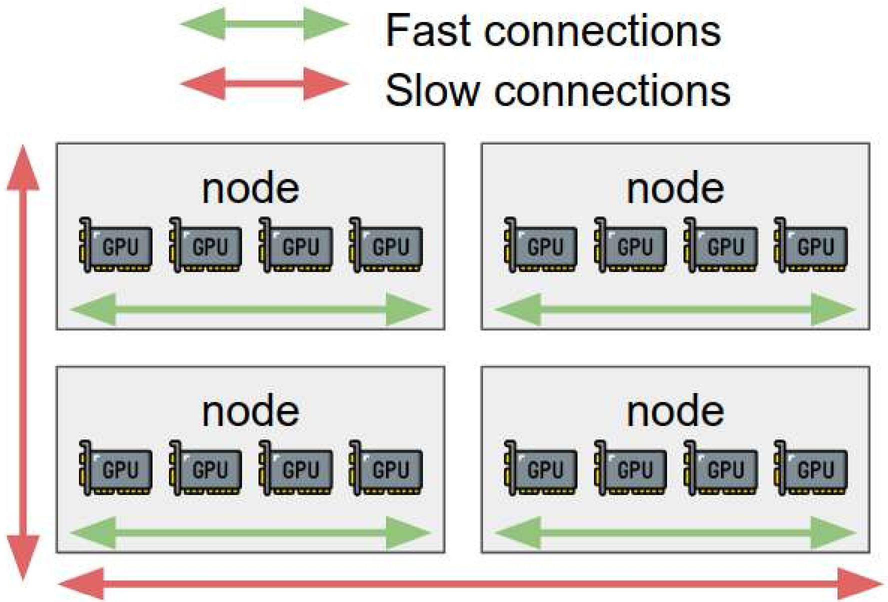
source: https://sites.google.com/view/icml-2022-big-model

Compute graph

How to partition the computational graph onto the device cluster?

Device cluster

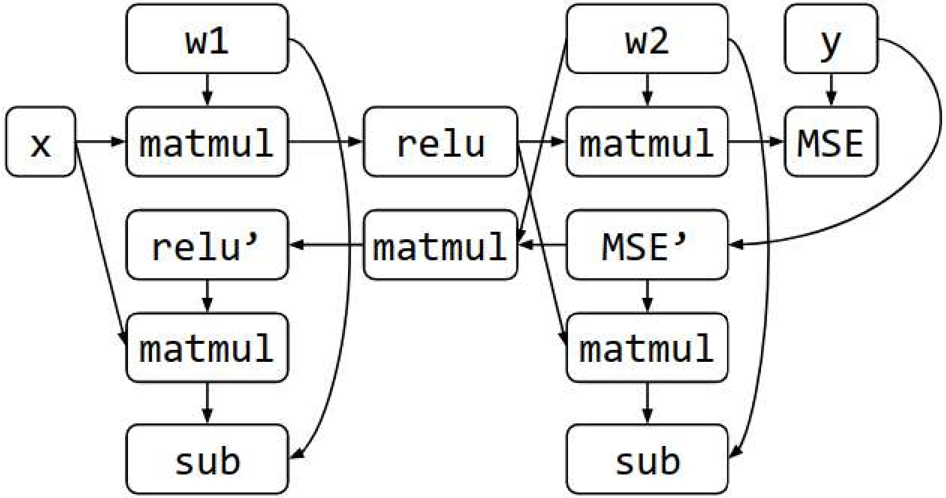


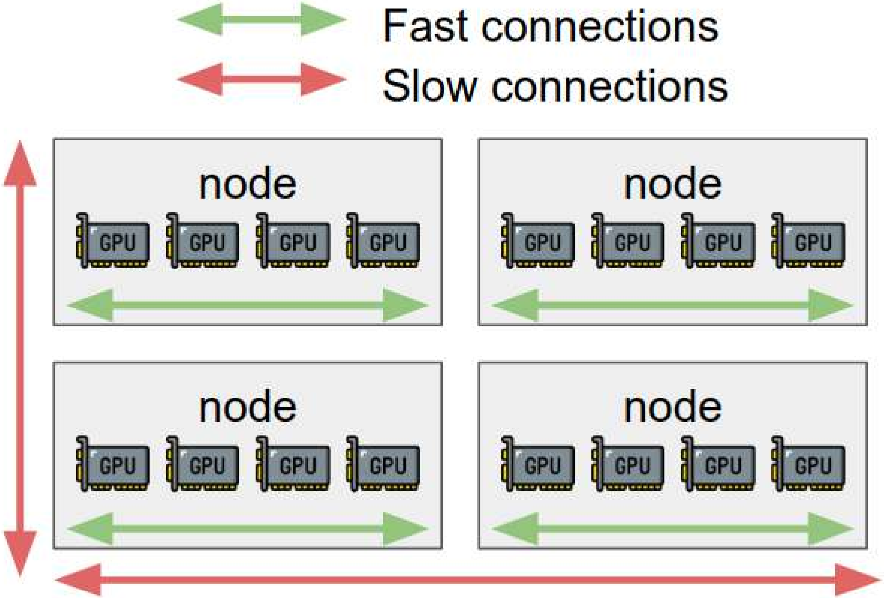


Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model

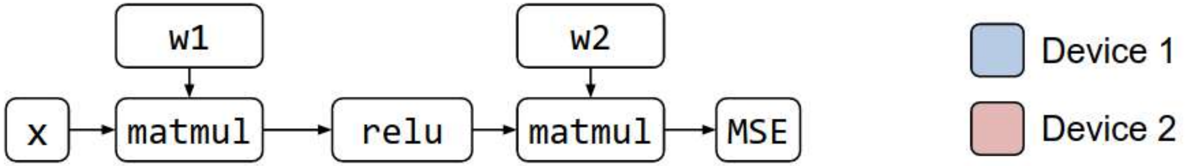
Q: How to partition the graph on the device cluster?



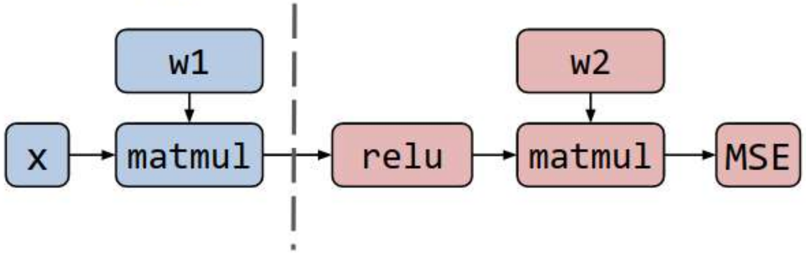


Automated parallelism

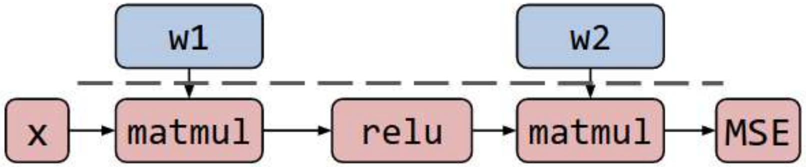
source: https://sites.google.com/view/icml-2022-big-model



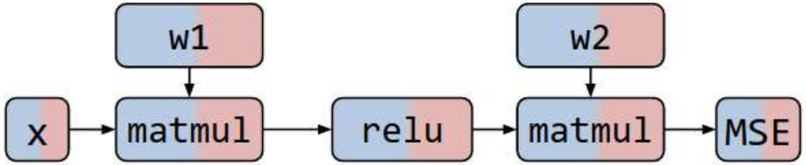
Strategy 1



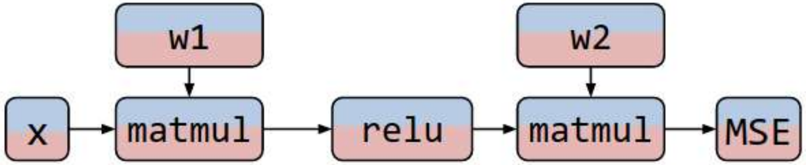
Strategy 2



Strategy 3

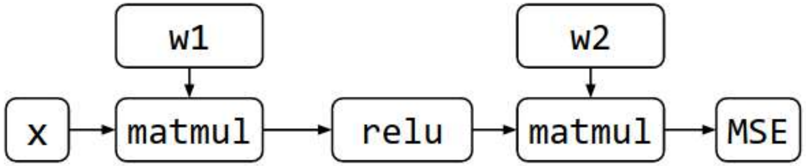


Strategy 4



Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model

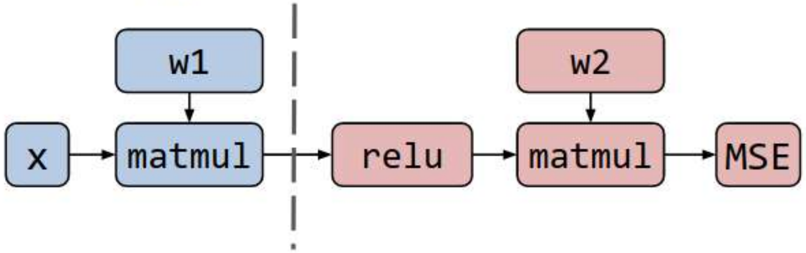


🟦 Device 1

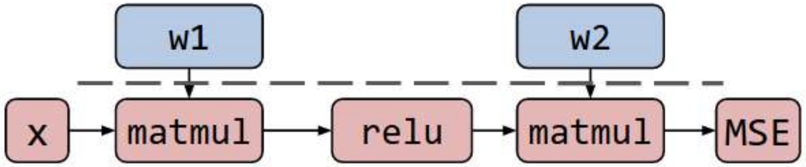
🟥 Device 2

Q: have you seen S1/2/3/4 before?

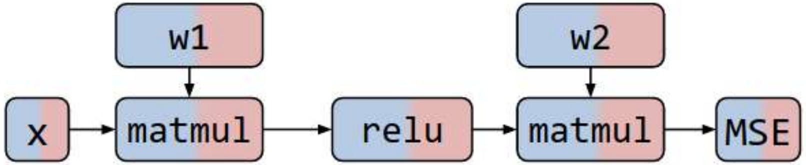
Strategy 1



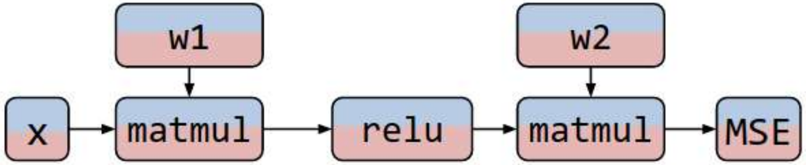
Strategy 2



Strategy 3

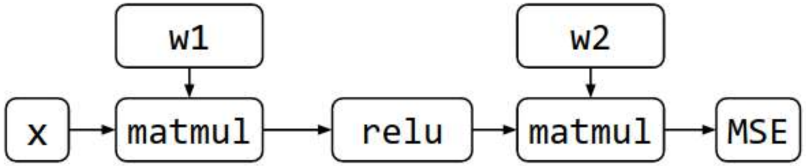


Strategy 4



Automated parallelism

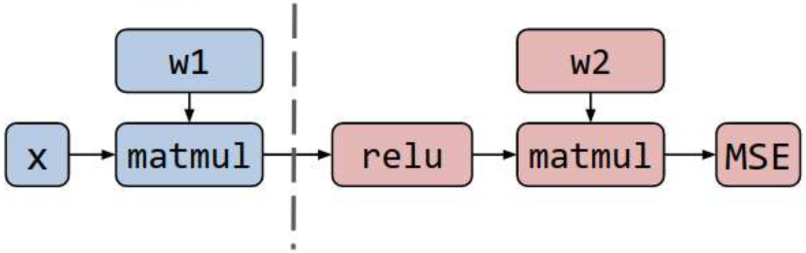
source: https://sites.google.com/view/icml-2022-big-model



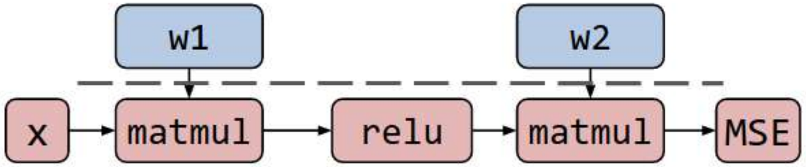
🟦 Device 1

🟥 Device 2

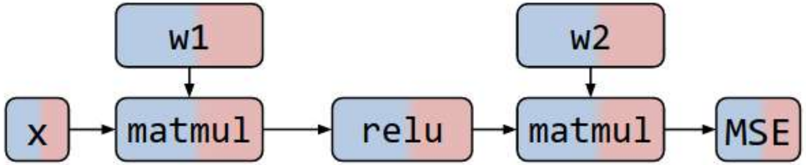
Pipeline MP



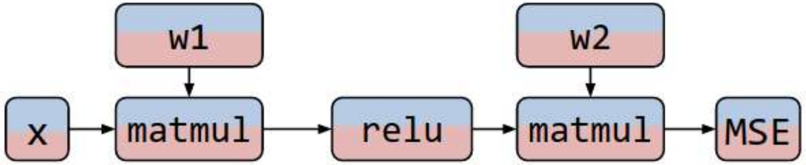
DP with offloading or PS



Tensor-parallel v1

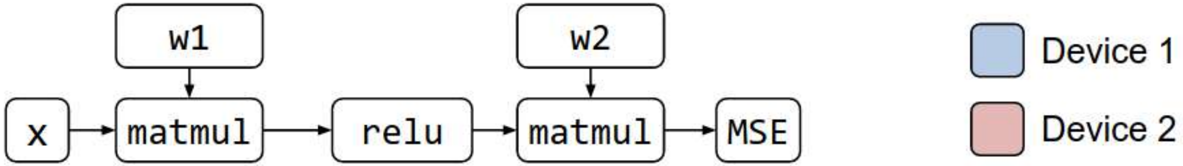


Tensor-parallel v2

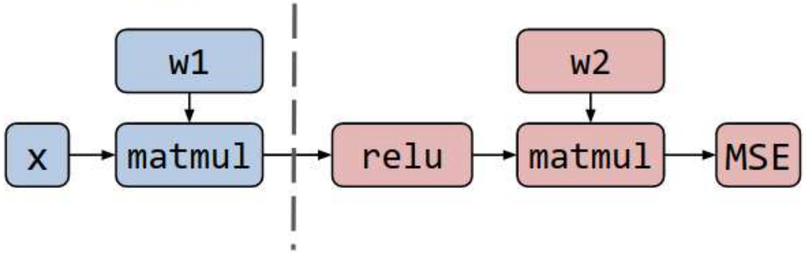


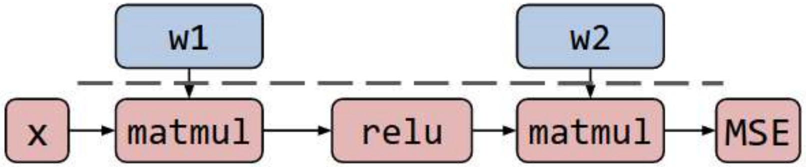
Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model

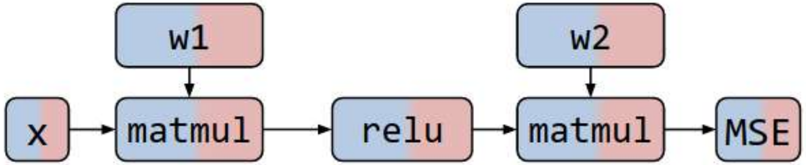


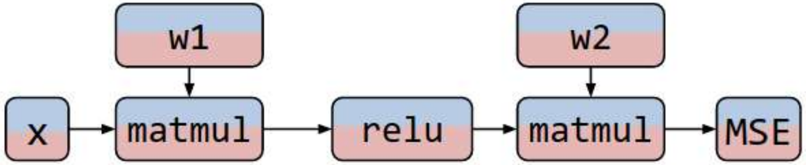
~~Pipeline MP~~ **Inter-op parallelism** ~~DP with offloading or PS~~





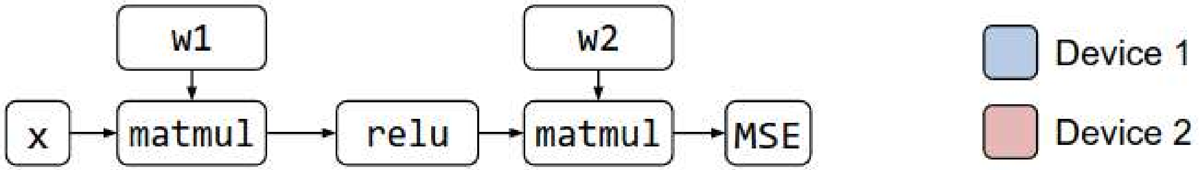
~~Tensor-parallel v1~~ **Intra-op parallelism** ~~Tensor-parallel v2~~



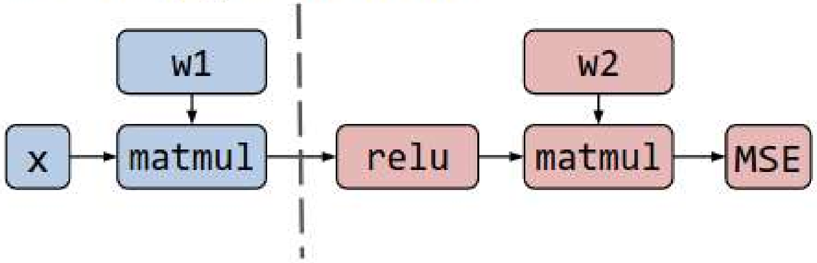


Automated parallelism

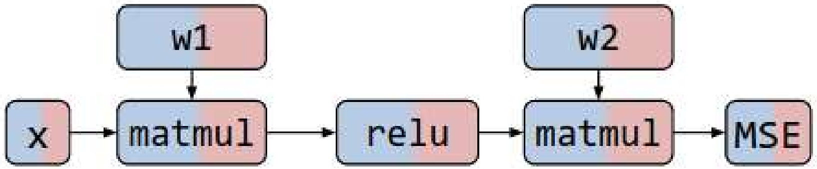
source: https://sites.google.com/view/icml-2022-big-model



Inter-op parallelism



Intra-op parallelism

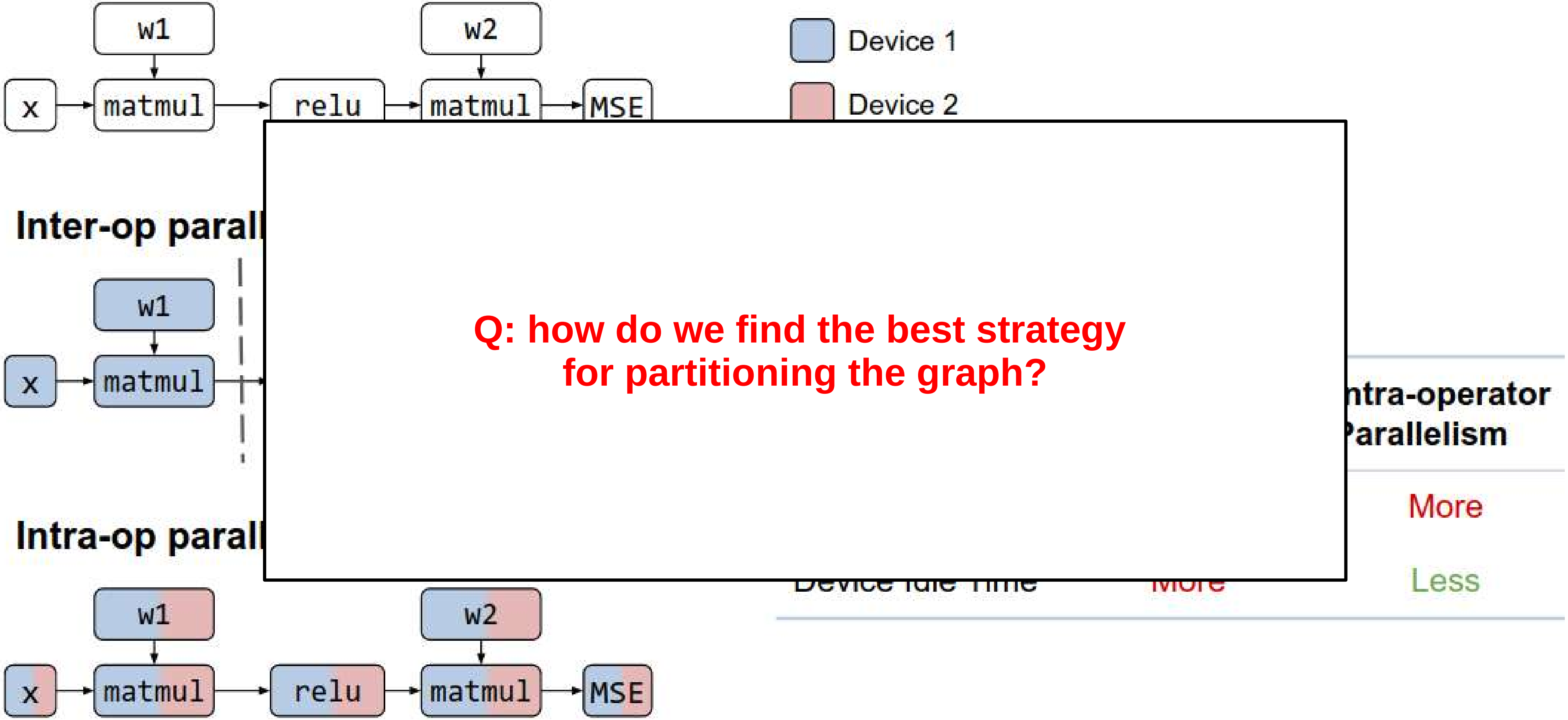


Trade-off

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

Automated parallelism

source: https://sites.google.com/view/icml-2022-big-model



Q: how do we find the best strategy for partitioning the graph?

RL-based partitioning

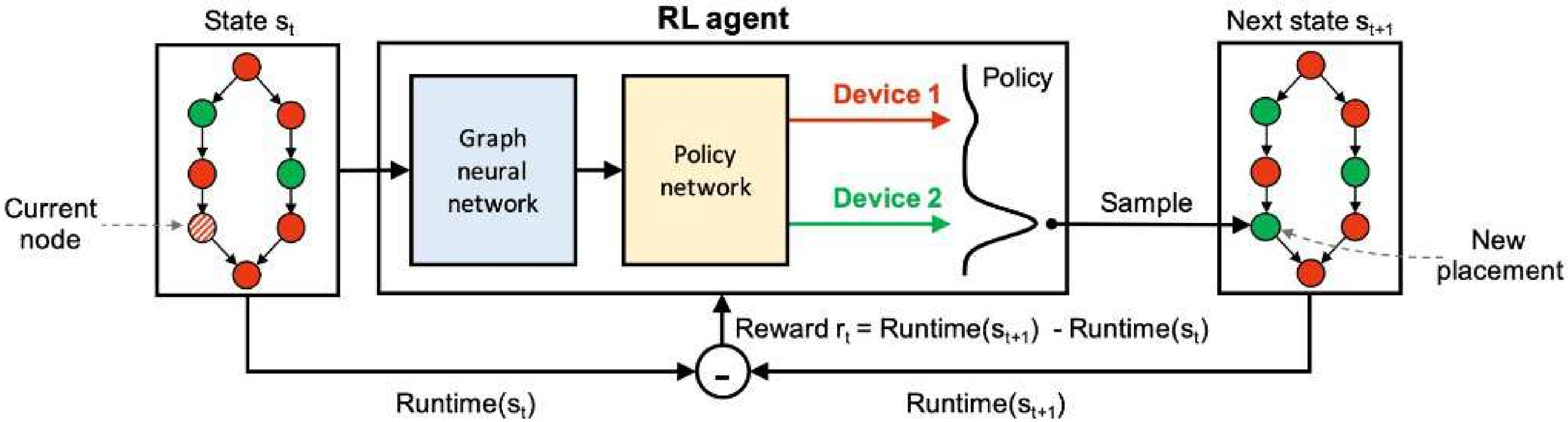
https://people.csail.mit.edu/hongzi/content/publications/placeto-neurips19.pdf

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.

Optimization-based partitioning

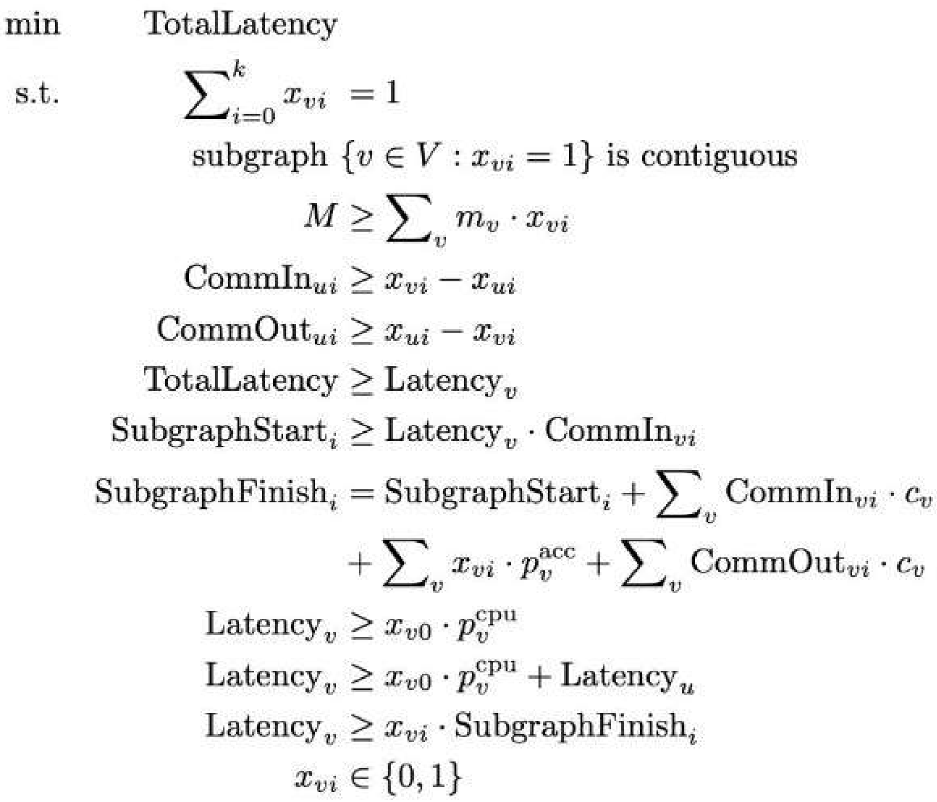
https://arxiv.org/abs/2006.16423

Integer Linear Programming:

Variable: Decision variable vector for each operator, representing device assignment.

Minimize: Maximum finishing time of all operators.

Constraint: Execution dependency & memory capacity of each device.

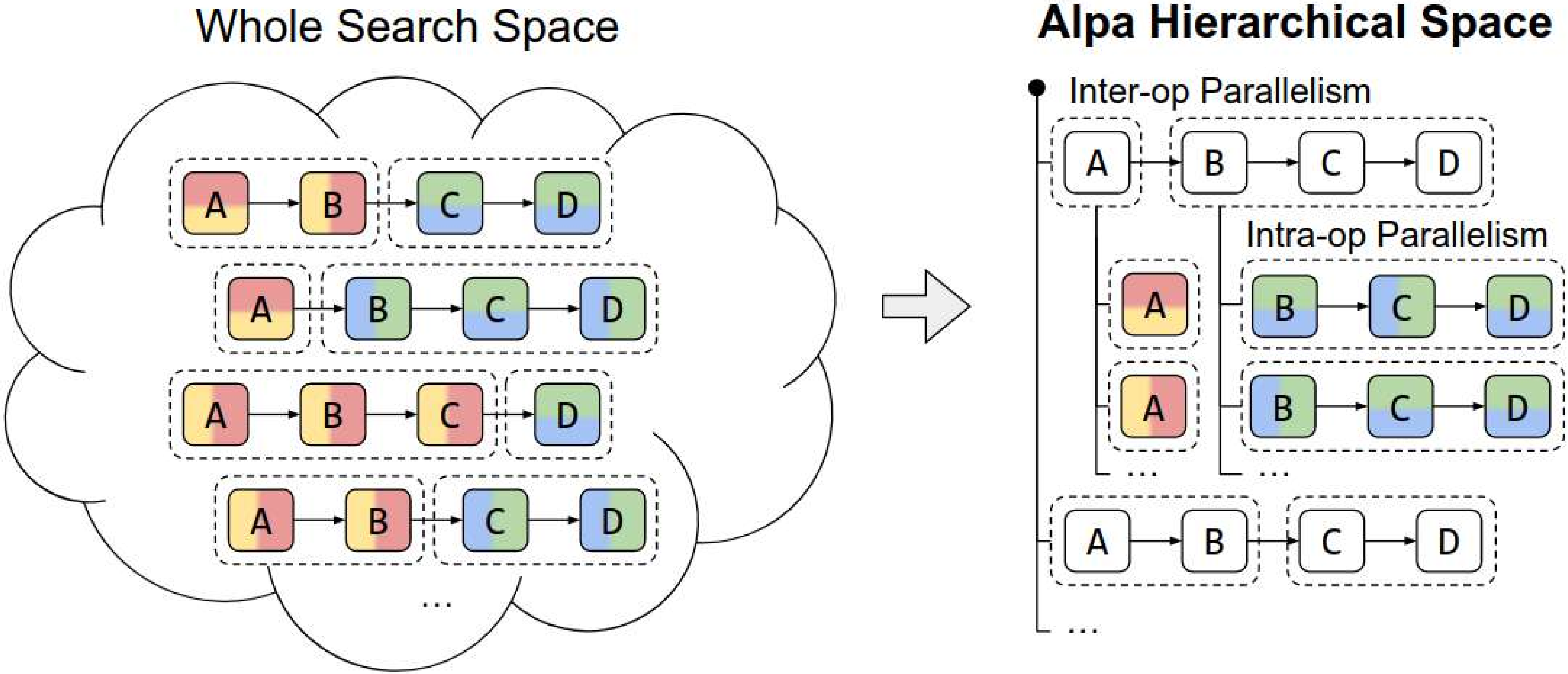


$$
\begin{aligned}
\text{min} \quad & \text{TotalLatency} \\
\text{s.t.} \quad & \textstyle\sum_{i=0}^{k} x_{vi} = 1 \\
& \text{subgraph } \{v \in V : x_{vi} = 1\} \text{ is contiguous} \\
& M \geq \textstyle\sum_v m_v \cdot x_{vi} \\
& \text{CommIn}_{ui} \geq x_{vi} - x_{ui} \\
& \text{CommOut}_{ui} \geq x_{ui} - x_{vi} \\
& \text{TotalLatency} \geq \text{Latency}_v \\
& \text{SubgraphStart}_i \geq \text{Latency}_v \cdot \text{CommIn}_{vi} \\
& \text{SubgraphFinish}_i = \text{SubgraphStart}_i + \textstyle\sum_v \text{CommIn}_{vi} \cdot c_v \\
& \qquad + \textstyle\sum_v x_{vi} \cdot p_v^{\text{acc}} + \textstyle\sum_v \text{CommOut}_{vi} \cdot c_v \\
& \text{Latency}_v \geq x_{v0} \cdot p_v^{\text{cpu}} \\
& \text{Latency}_v \geq x_{v0} \cdot p_v^{\text{cpu}} + \text{Latency}_u \\
& \text{Latency}_v \geq x_{vi} \cdot \text{SubgraphFinish}_i \\
& x_{vi} \in \{0, 1\}
\end{aligned}
$$

Tarnawski, Jakub M., et al. "Efficient algorithms for device placement of dnn graph operators." NeurIPS 2020.

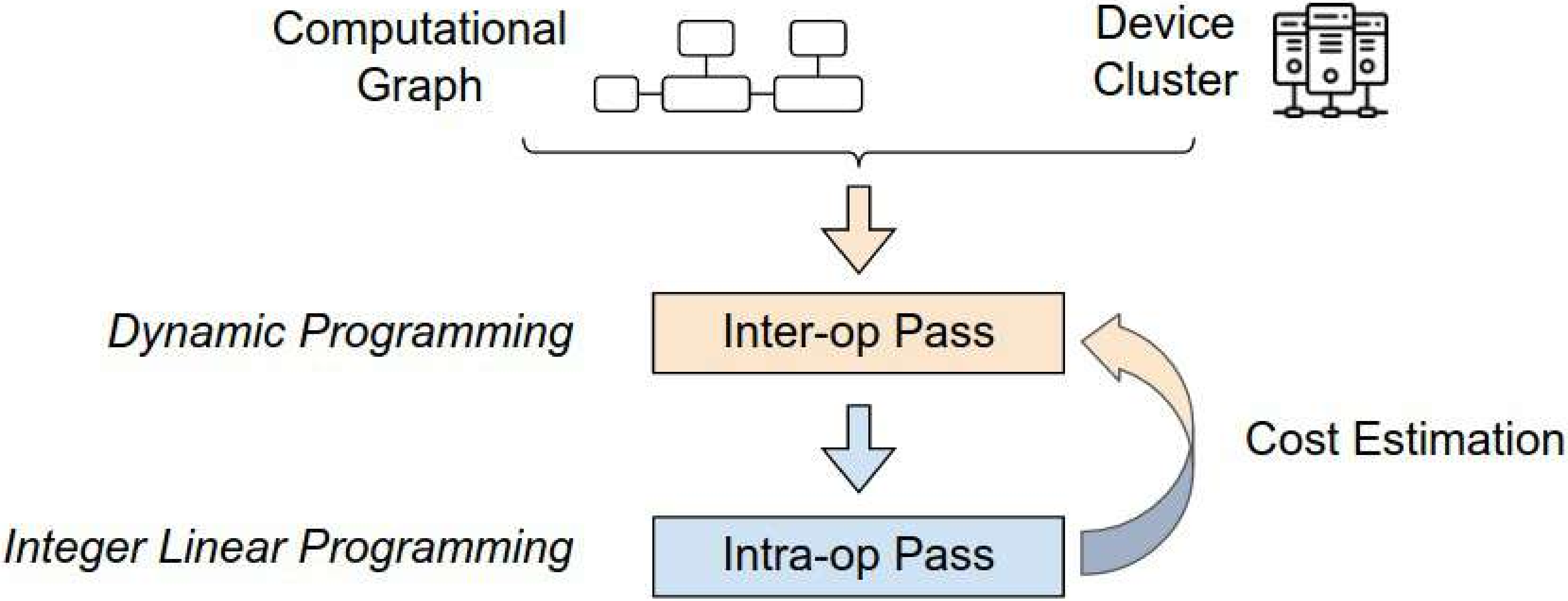
Alpa: optimization-based + reduced search space

https://arxiv.org/abs/2201.12023



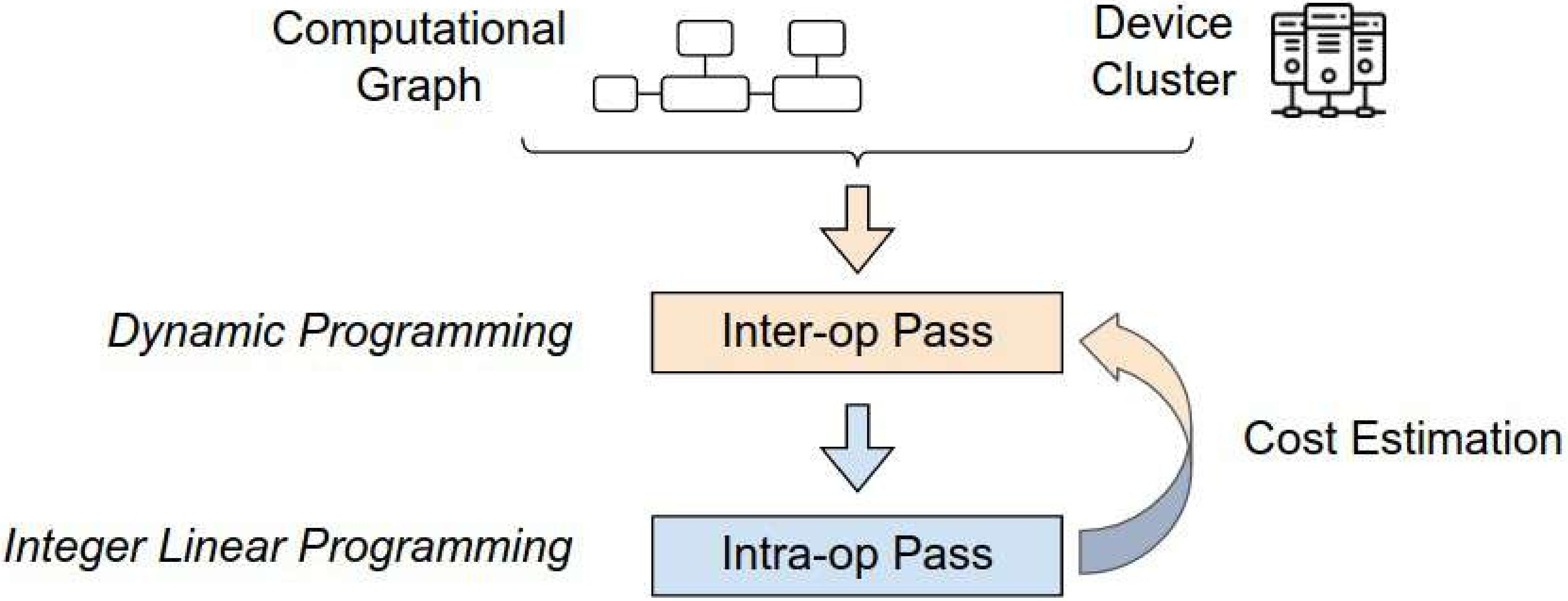
Alpa: optimization-based + reduced search space

https://arxiv.org/abs/2201.12023



Alpa: optimization-based + reduced search space

https://arxiv.org/abs/2201.12023



More details of each pass:

https://sites.google.com/view/icml-2022-big-model

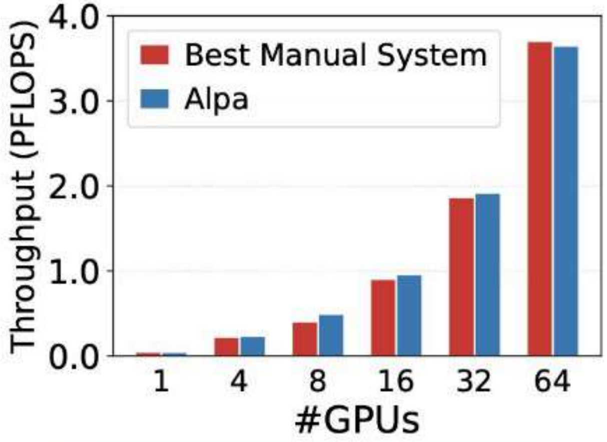
Alpa: optimization-based + reduced search space

https://arxiv.org/abs/2201.12023

Not the first algorithm for auto-parallelism…
but the first one that is usable* *(* - most of the time)*

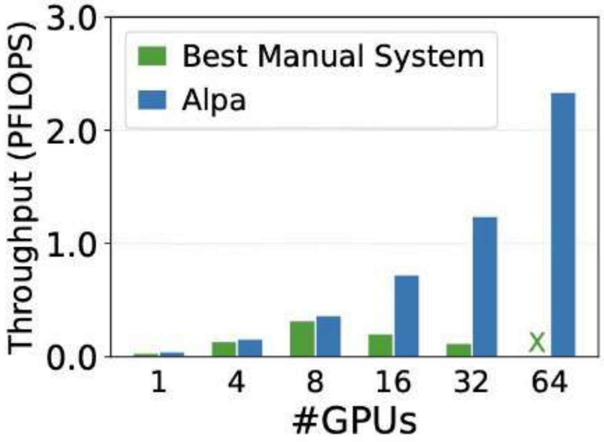
(benchmarks on AWS V100)

GPT (up to 39B)



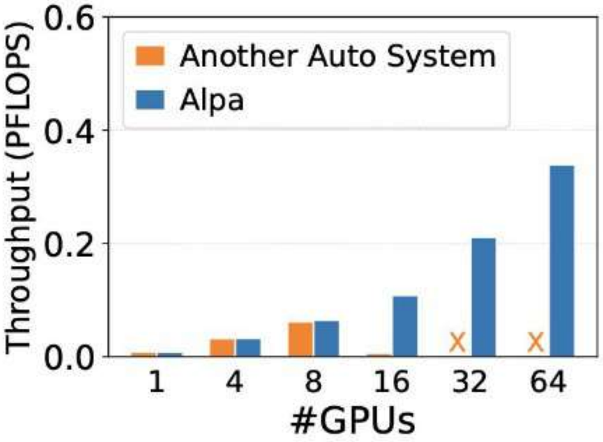
Match specialized manual systems.

GShard MoE (up to 70B)



Outperform the manual baseline by up to 8x.

Wide-ResNet (up to 13B)

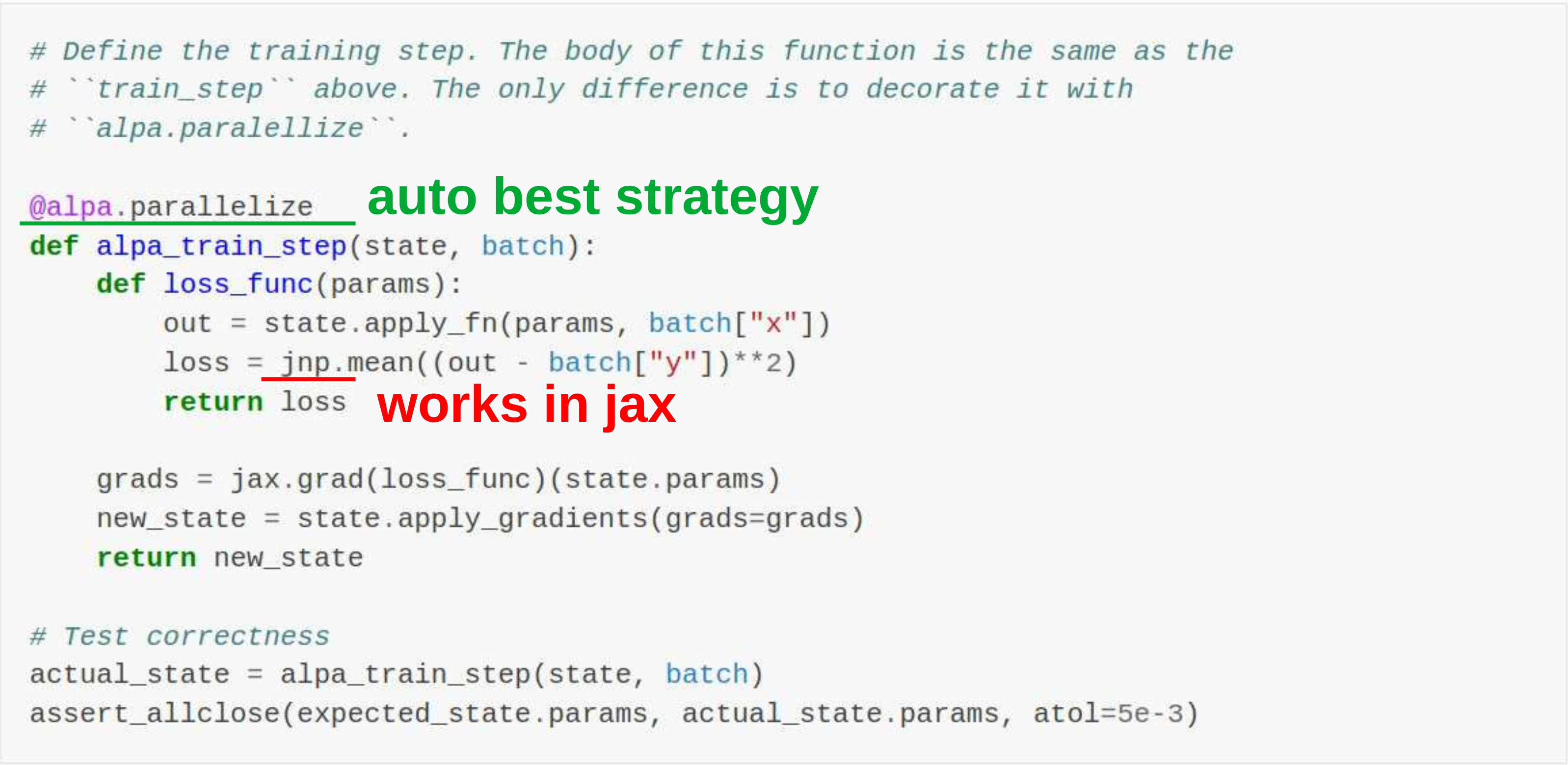


Generalize to models without manual plans.

Alpa: optimization-based + reduced search space

https://arxiv.org/abs/2201.12023

Not the first algorithm for auto-parallelism…
but the first one that is usable* *(* - most of the time)*



```
# Define the training step. The body of this function is the same as the
# ``train_step`` above. The only difference is to decorate it with
# ``alpa.paralellize``.

@alpa.parallelize
def alpa_train_step(state, batch):
    def loss_func(params):
        out = state.apply_fn(params, batch["x"])
        loss = jnp.mean((out - batch["y"])**2)
        return loss

    grads = jax.grad(loss_func)(state.params)
    new_state = state.apply_gradients(grads=grads)
    return new_state

# Test correctness
actual_state = alpa_train_step(state, batch)
assert_allclose(expected_state.params, actual_state.params, atol=5e-3)
```

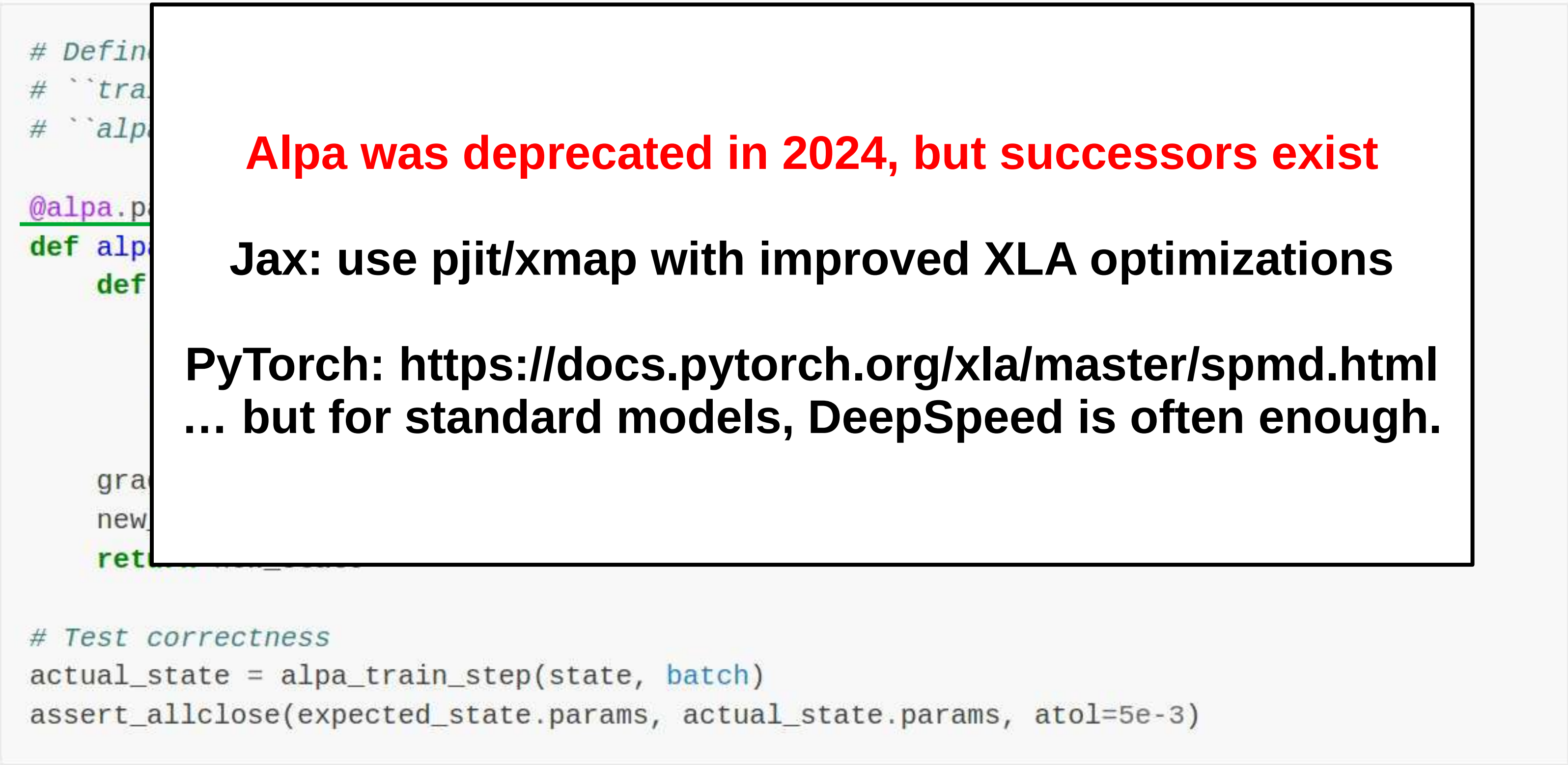
auto best strategy

works in jax

Alpa: optimization-based + reduced search space

https://arxiv.org/abs/2201.12023

Not the first algorithm for auto-parallelism…
but the first one that is usable* *(* - most of the time)*



```
# Defin
# ``tra
# ``alp

@alpa.p
def alp
    def




    gra
    new
    ret

# Test correctness
actual_state = alpa_train_step(state, batch)
assert_allclose(expected_state.params, actual_state.params, atol=5e-3)
```

Alpa was deprecated in 2024, but successors exist

Jax: use pjit/xmap with improved XLA optimizations

PyTorch: https://docs.pytorch.org/xla/master/spmd.html
… but for standard models, DeepSpeed is often enough.

</part 2>

+ model larger than GPU
+ faster for small
***** typical size: 2-8 gpus
- model partitioning is tricky
  tensor parallelism is easier, but requires ultra low latency
- latency is critical, go buy nvlink
  except for PipeDream
- *often combined with gradient checkpointing*

Tutorials:

- Simple pipelining in PyTorch – tinyurl.com/pytorch-pipelining
- Distributed model-parallel with torch RPC - https://tinyurl.com/torch-rpc
- Minimalistic tensor parallelism pip install tensor_parallel

</part 2>

+ model larger than GPU
+ faster for small
***** typical size: 2-8 gpus
- model partitioning is tricky
tensor parallelism is easier, but requires ultra low latency
- latency is critical, go buy nvlink
except for PipeDream
- *often combined with gradient checkpointing*

Tutorials:

- Simple pipelining in PyTorch – tinyurl.com/pytorch-pipelining
- Distributed model-parallel with torch RPC - https://tinyurl.com/torch-rpc
- Automatic tensor parallelism pip install tensor_parallel

Q: what if you have 1024 GPUs, but the model fits on 8?

</part 2>

+ model larger than GPU
+ faster for small
***** typical size: 2-8 gpus
- model partitioning is tricky
tensor parallelism is easier, but requires ultra low latency
- latency is critical, go buy nvlink
except for PipeDream
- *often combined with gradient checkpointing*

Tutorials:

- Simple pipelining in PyTorch – tinyurl.com/pytorch-pipelining
- Distributed model-parallel with torch RPC - https://tinyurl.com/torch-rpc
- Automatic tensor parallelism pip install tensor_parallel

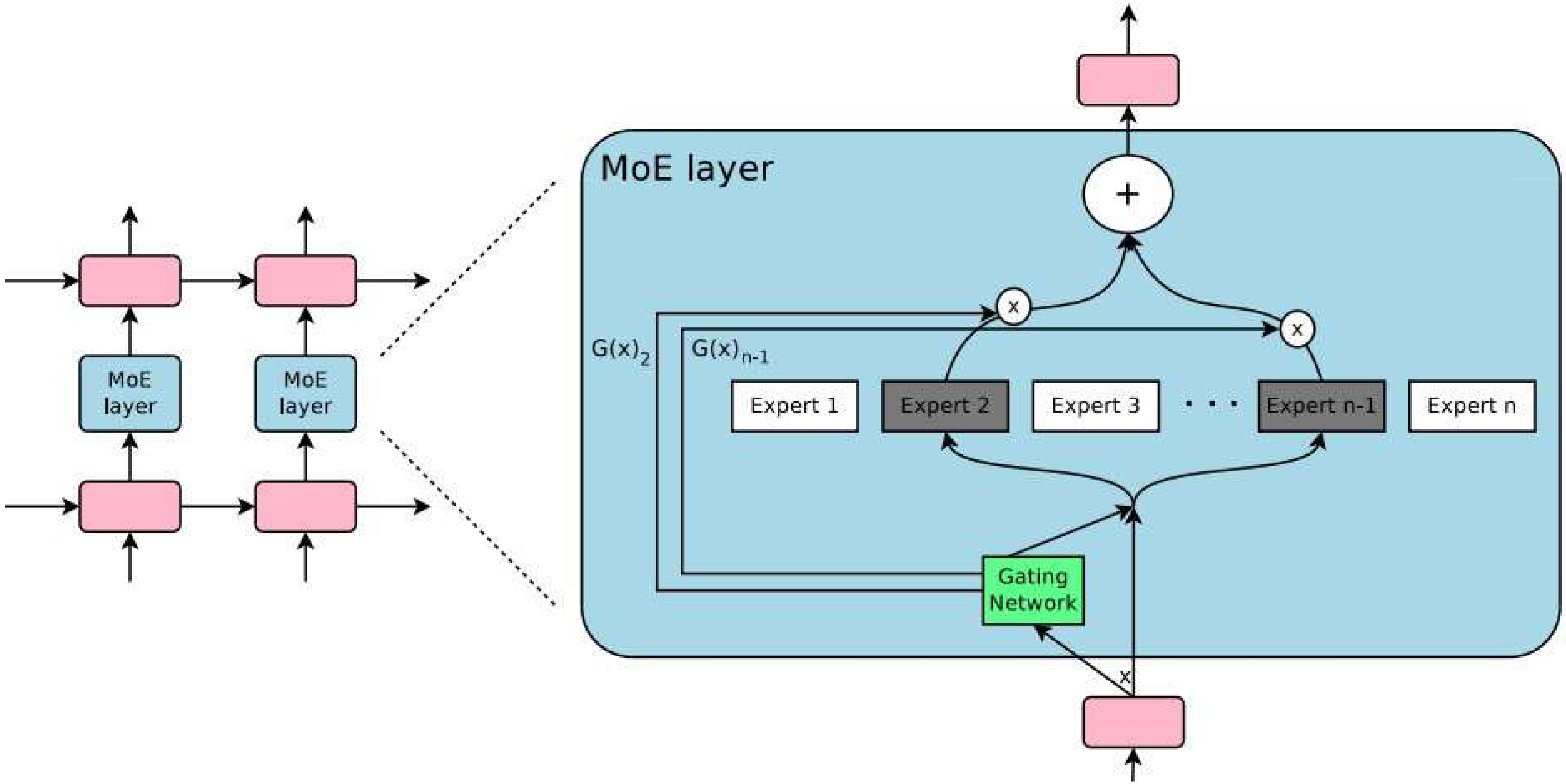
Large-scale training: combine model- and data-parallel

So far we’ve been trying to partition for existing models…

Perhaps there are models that are easier to partition?

Expert Parallelism

Sparsely gated MoE: https://arxiv.org/pdf/1701.06538.pdf



MoE Variant: Switch Transformer

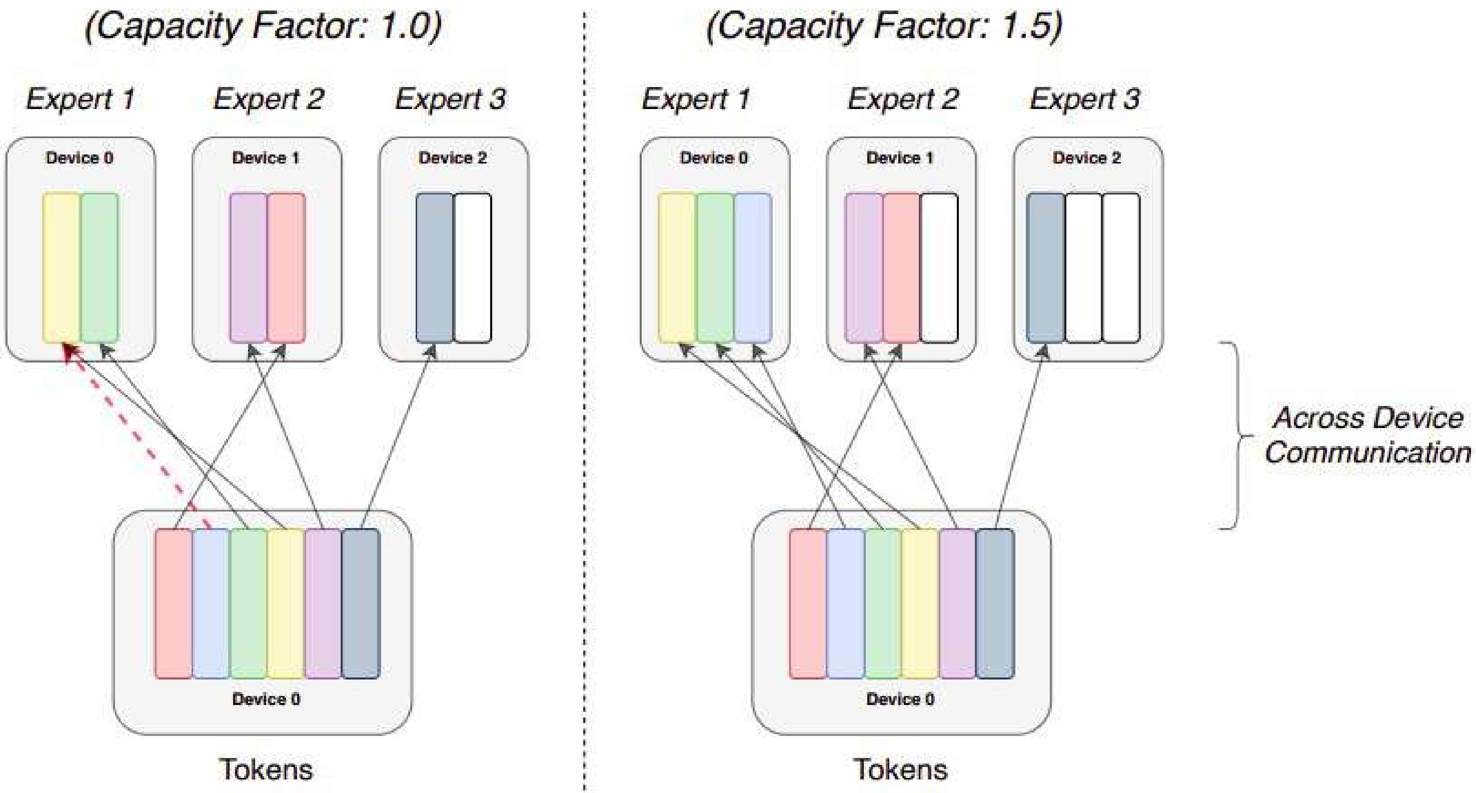
Switch: https://arxiv.org/pdf/2101.03961.pdf

Terminology

- **Experts:** Split across devices, each having their own unique parameters. Perform standard feed-forward computation.

- **Expert Capacity:** Batch size of each expert. Calculated as
- (tokens_per_batch / num_experts) * capacity_factor

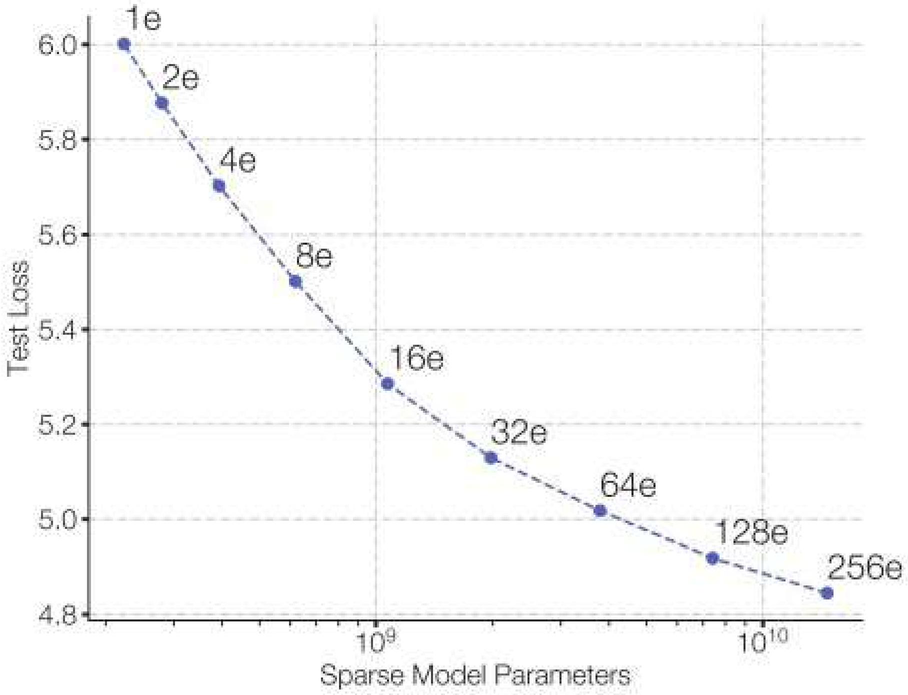
- **Capacity Factor:** Used when calculating expert capacity. Expert capacity allows more buffer to help mitigate token overflow during routing.

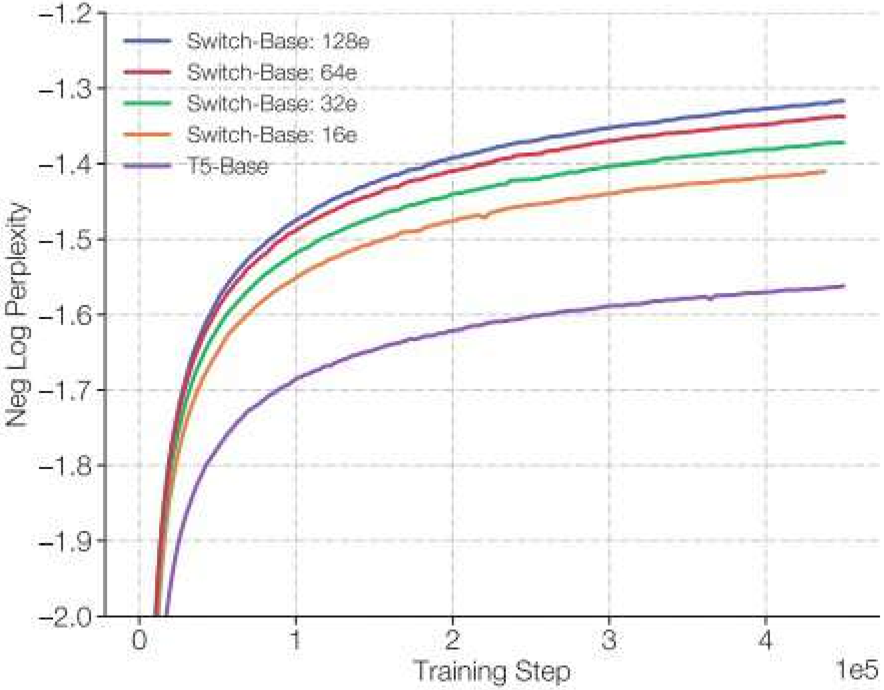


MoE Variant: Switch Transformer

Switch: https://arxiv.org/pdf/2101.03961.pdf

MLM pre-training objective [BERT-like]

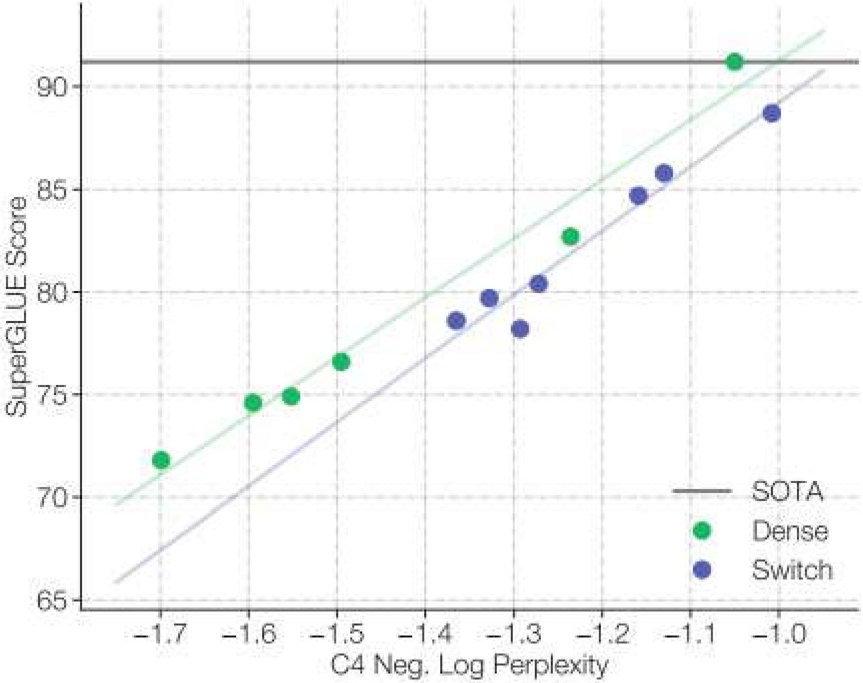


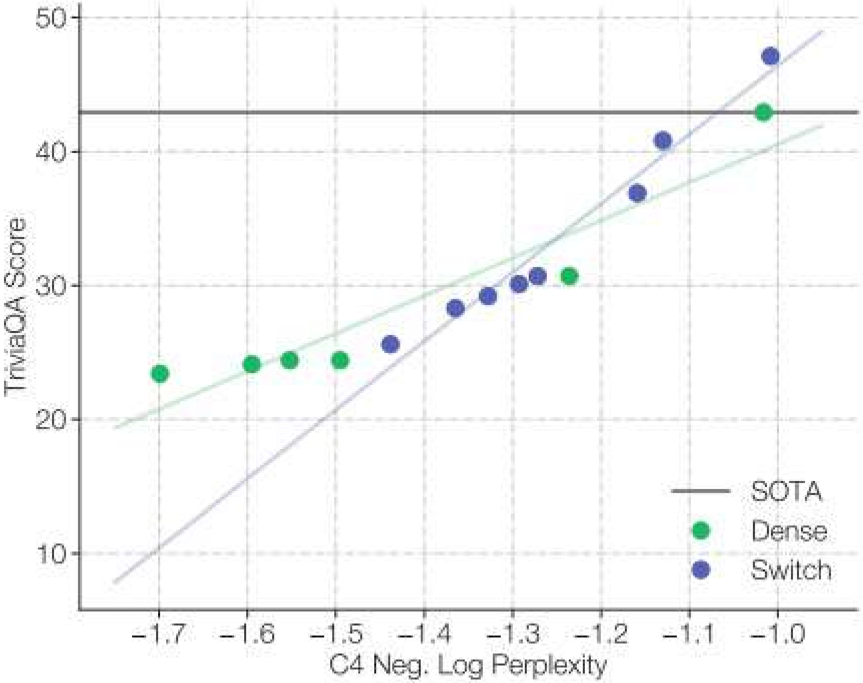


MoE Variant: Switch Transformer

Switch: https://arxiv.org/pdf/2101.03961.pdf

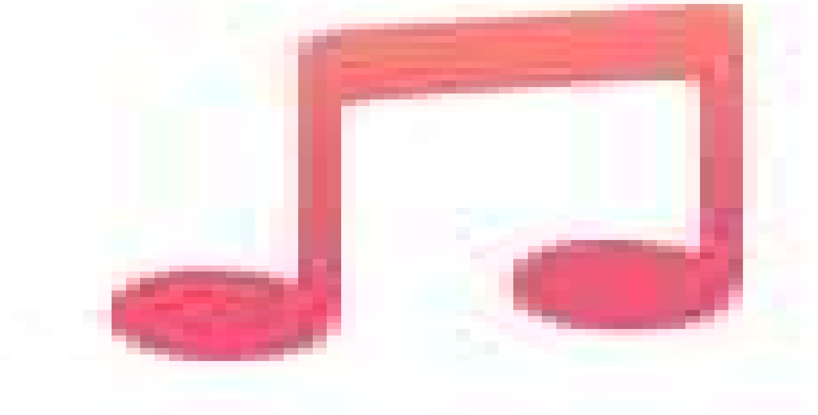
Pre-training vs downstream quality





Alternative: FSDP

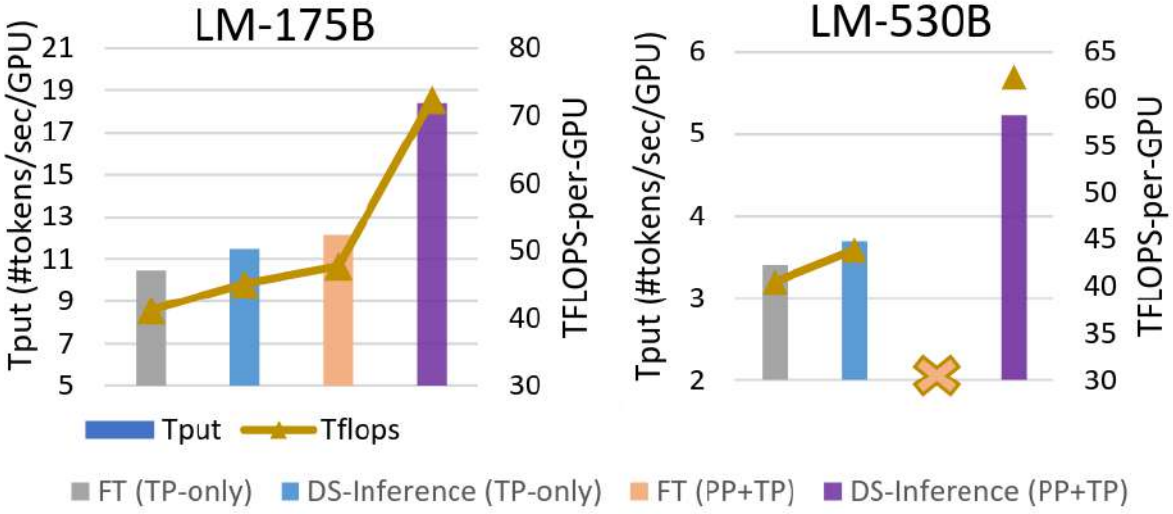
Source: microsoft



DeepSpeed Inference

Paper: https://arxiv.org/abs/2207.00032

- Same techniques, but for inference
- Offloading, tensor- & pipeline-parallel
- … and a ton of hacks



</ZeRO>

Multi-GPU strategies:
* Pipeline model-parallel – allocate layers on different GPUs
* Sharded data-parallel – split optimizer state and/or parameters

Single GPU strategies:
* Small model – gradient checkpointing & virtual batch
* Large model – optimizer state sharding (keep parameters on GPU)

Implementations:

- DeepSpeed– sharded DP, offload, tensor parallelism, active development
  - Offload – https://www.deepspeed.ai/news/2021/03/07/zero3-offload.html

- FSDP – most of DeepSpeed features with native PyTorch API
- Model-specific implementations– https://github.com/NVIDIA/Megatron-LM

If we have time…

(if not, skip)

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **???**

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **grad accumulation**

16GB model and optimizer, 16GB activations (batch 1) - **???**

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **grad accumulation**

16GB model and optimizer, 16GB activations (batch 1) → **grad checkpointing**

32GB model and optimizer, 1GB activations → **???**

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **grad accumulation**

16GB model and optimizer, 16GB activations (batch 1) → **grad checkpointing**

32GB model and optimizer, 1GB activations → **it depends…**

DDP + offloading | FSDP (ZeRO) | Pipeline-parallel | Tensor-parallel

?

When is this the best option?

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **grad accumulation**

16GB model and optimizer, 16GB activations (batch 1) → **grad checkpointing**

32GB model and optimizer, 1GB activations → **it depends…**

DDP + offloading | FSDP (ZeRO) | Pipeline-parallel | Tensor-parallel

e.g. if too few GPUs for other methods

?

When is this the best option?

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **grad accumulation**

16GB model and optimizer, 16GB activations (batch 1) → **grad checkpointing**

32GB model and optimizer, 1GB activations → **it depends…**

DDP + offloading | FSDP (ZeRO) | Pipeline-parallel | Tensor-parallel

DDP + offloading	FSDP (ZeRO)	Pipeline-parallel
e.g. if too few GPUs for other methods	*no custom model code, best for large batches*	**?**

When is this the best option?

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **grad accumulation**

16GB model and optimizer, 16GB activations (batch 1) → **grad checkpointing**

32GB model and optimizer, 1GB activations → **it depends…**

DDP + offloading	**FSDP (ZeRO)**	**Pipeline-parallel**	**Tensor-parallel**
e.g. if too few GPUs for other methods	*no custom model code, best for large batches*	*communication-efficient sequential model*	**?**

When is this the best option?

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **grad accumulation**

16GB model and optimizer, 16GB activations (batch 1) → **grad checkpointing**

32GB model and optimizer, 1GB activations → **it depends…**

DDP + offloading	**FSDP (ZeRO)**	**Pipeline-parallel**	**Tensor-parallel**
e.g. if too few GPUs for other methods	*no custom model code, best for large batches*	*communication-efficient sequential model*	*minimal latency non-symmetric model*

Mix and match: TP within one server, minimal PP between servers, DDP between groups
Parallel code: manual (e.g. Megatron-LM) vs automated (alpa, FSDP, tensor_parallel)
Unconventional hardware: hivemind, petals, varuna, etc

</lecture>

Example configuration:
Several GPU w/ 24GB memory | 128GB system memory | 16GBps interconnect

16GB model and optimizer, 128GB activations (batch 32) → **grad accumulation**

16GB model and optimizer, 16GB activations (batch 1) → **grad checkpointing**

32GB model and optimizer, 1GB activations → **it depends…**

If the model does not fit, you can also quantize it into submission!
(more on model compression in a future lecture)

That's all Folks!