Announcement

HW3 will be out today. Get started ASAP!

- There will be additional TA office hours hold by the creators of this homework: Amanda and Emmy
- It is due Nov 30th, two weeks from now, excluding Thanksgiving holiday

Final Project Presentation will be a Conference Poster like session at GHC 7107 Atrium

- More instructions on the course website

Scaling Up LLM Pretraining: Parallel Training

Chenyan Xiong

11-667

Outline

Parallel Training

- Data Parallelism
- Pipeline Parallelism
- Tensor Parallelism
- Combination of Parallelism
- ZeRO Optimizer

Parallel Training: Overview

As scale grows, training with one GPU is not enough

- There are many ways to improve efficiency on single-GPU training
  - Checkpointing: moving part of the operations to CPU memory
  - Quantizing different part of the optimization to reduce GPU memory cost
- Eventually more FLOPs are needed

Different setups of parallel training:

- When model training can fit into single-GPU

→Data parallelism

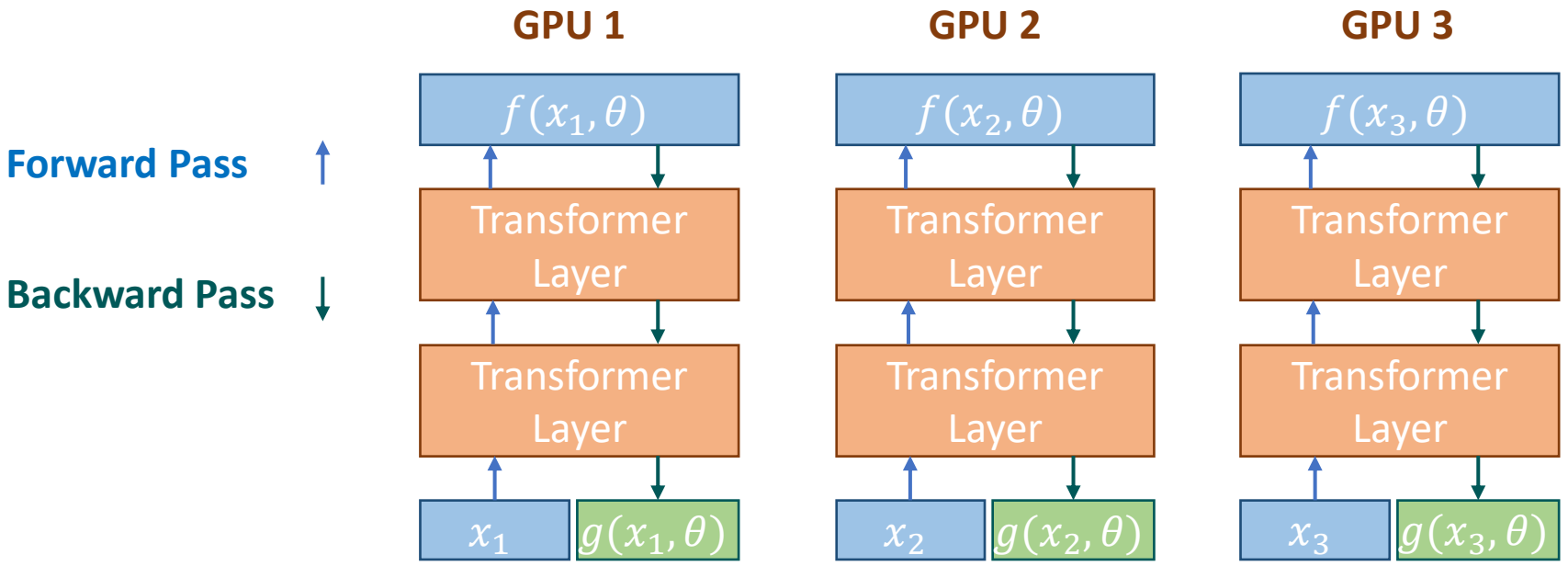
- When model training cannot fit into single-GPU

→ Model parallelism: pipeline or tensor

Parallel Training: Data Parallelism

Split training data batch into different GPUs

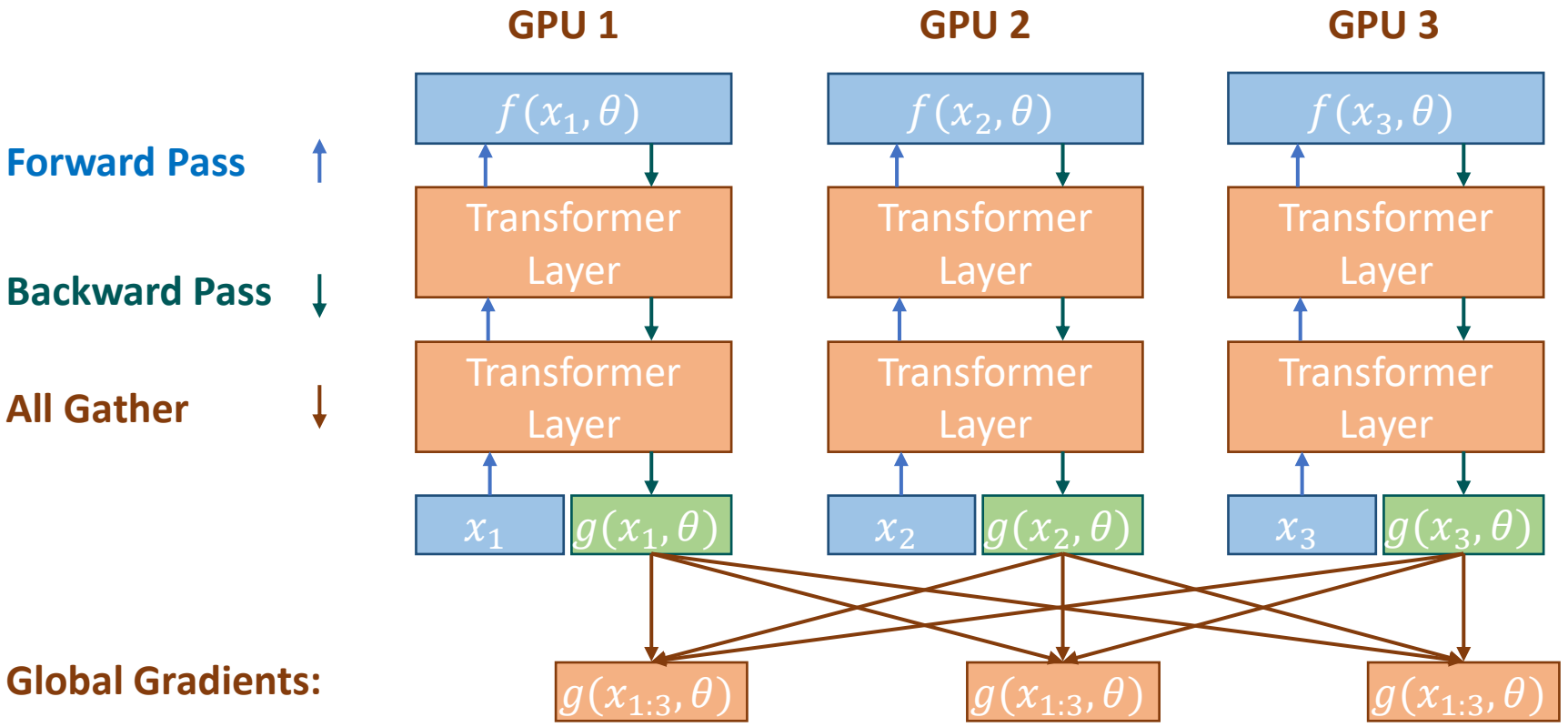
- Each GPU maintains its own copy of model and optimizer
- Each GPU gets a different local data batch, calculates its gradients



Parallel Training: Data Parallelism

Split training data batch into different GPUs

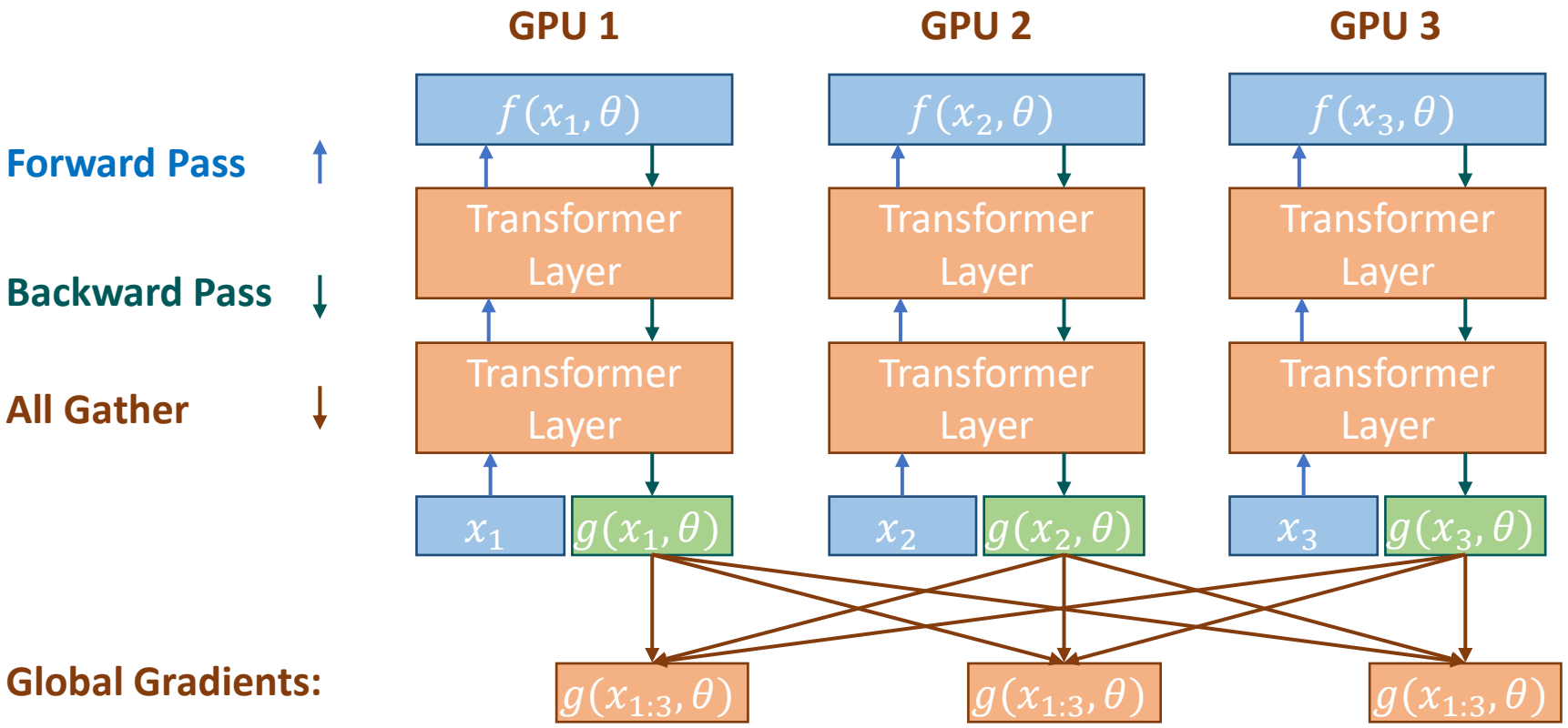
- Each GPU maintains its own copy of model and optimizer
- Each GPU gets a different local data batch, calculates its gradients
- Gather local gradients together to each GPU for global updates



Parallel Training: Data Parallelism

Split training data batch into different GPUs

- Each GPU maintains its own copy of model and optimizer
- Each GPU gets a different local data batch, calculates its gradients
- Gather local gradients together to each GPU for global updates



Communication:

- The full gradient tensor between every pair of GPUs, at each training batch.
- Not an issue between GPUs in the same machine or machines with infinity band
- Will need work around without fast cross-GPU connection

Parallel Training: Model Parallelism

LLM size grew quickly and passed the limit of single GPU memory

	Cost of 10B Model	Function to parameter count (Ψ)
Parameter Bytes	20GB	2Ψ
Gradient Bytes	20GB	2Ψ
Optimizer State: 1st Order Momentum	20GB	2Ψ
Optimizer State: 2nd Order Momentum	20GB	2Ψ
Total Per Model Instance	**80GB**	8Ψ

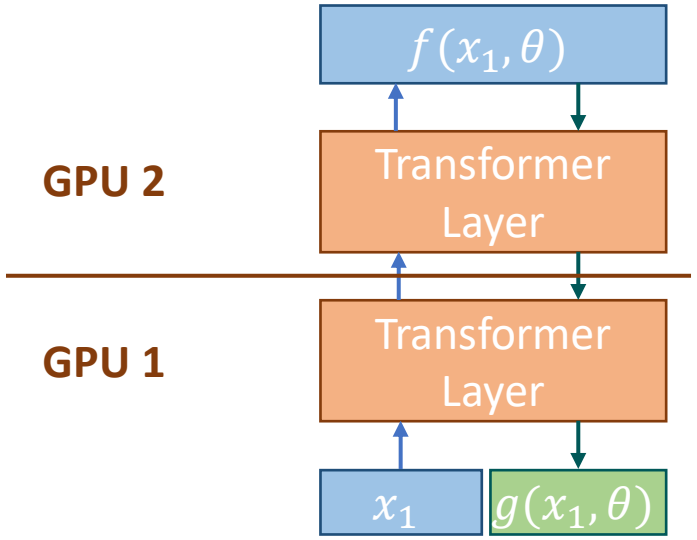
Table 1: Memory Consumption of Training Solely with **BF16** (Ideal case) of a model sized Ψ

Solution: Split network parameters (thus their gradients and corresponding optimizer states) to different GPUs

Parallel Training: Model Parallelism

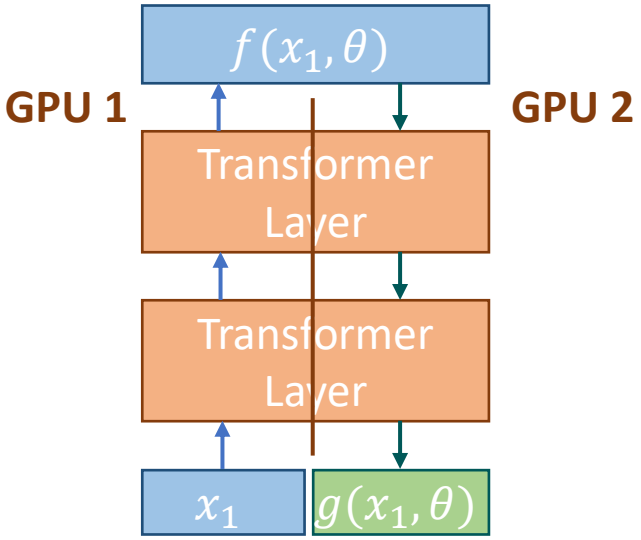
Two ways of splitting network parameters

Pipeline Parallelism



Split by Layers

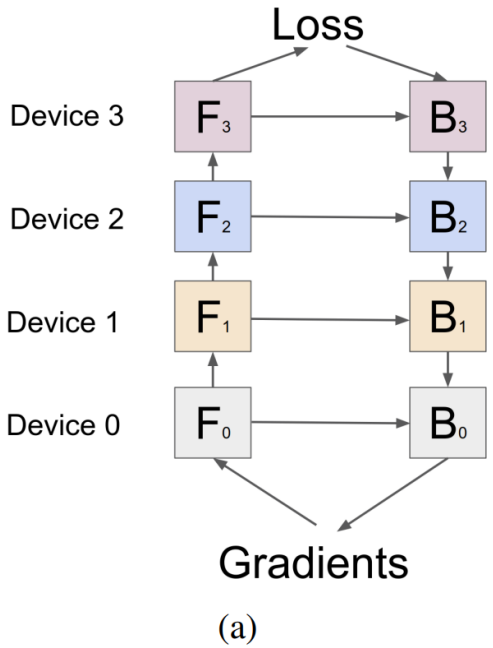
Tensor Parallelism



Split Tensors

Parallel Training: Pipeline Parallelism

Split network by layers, aligning devices by layer order to a pipeline, and pass data through devices [7]



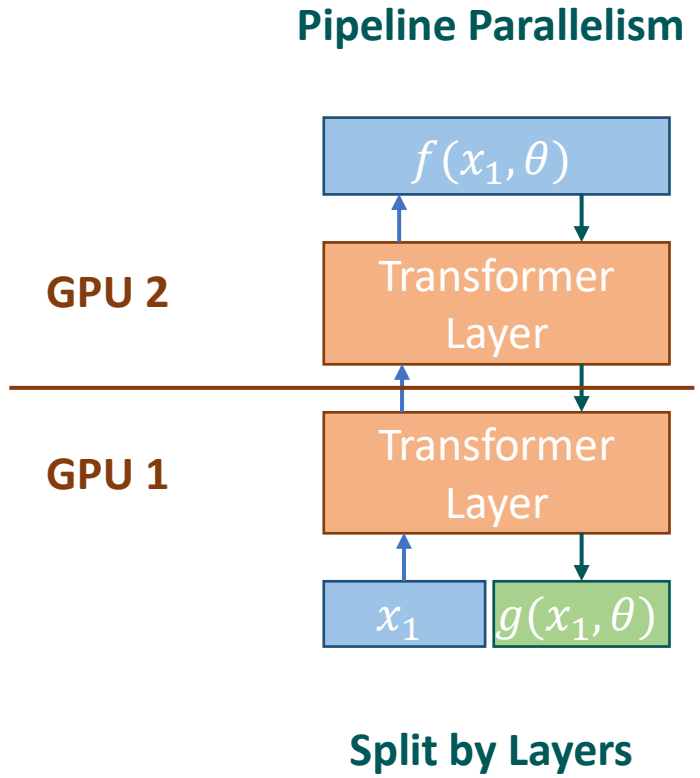


Figure 7: Illustration of Pipeline Parallelism [7]

Parallel Training: Pipeline Parallelism

Split network by layers, aligning devices by layer order to a pipeline, and pass data through devices [7]

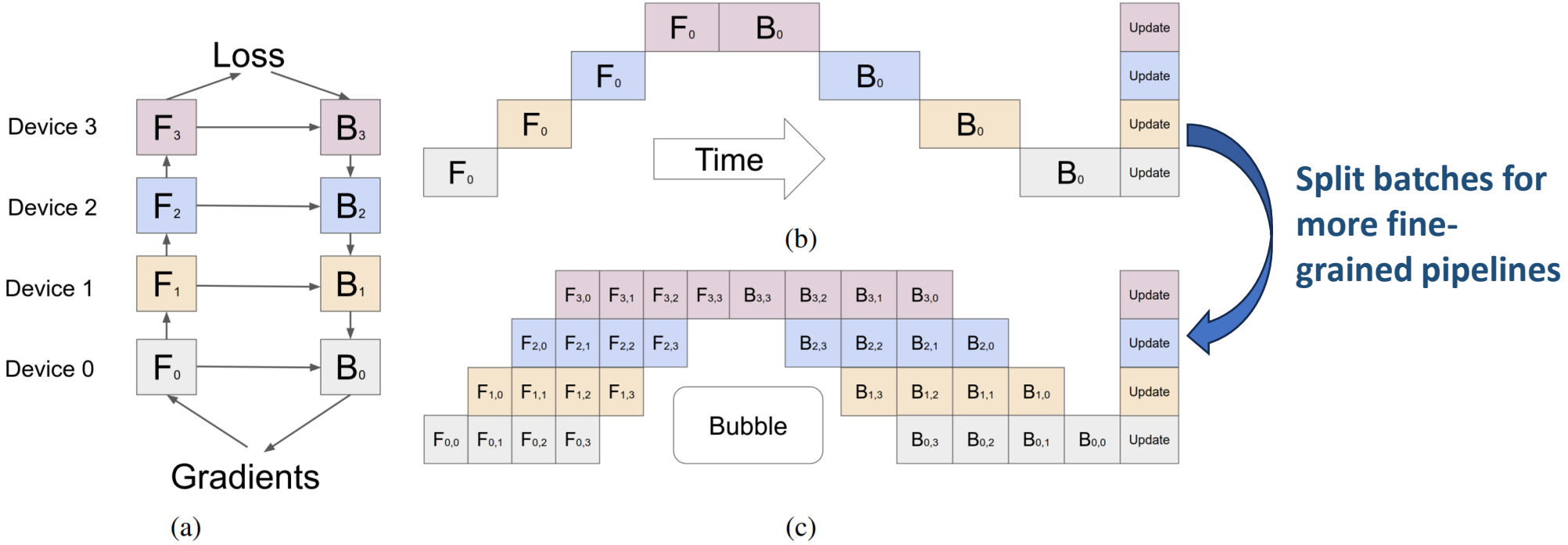


Figure 7: Illustration of Pipeline Parallelism [7]

Parallel Training: Pipeline Parallelism

Split network by layers, aligning devices by layer order to a pipeline, and pass data through devices [7]

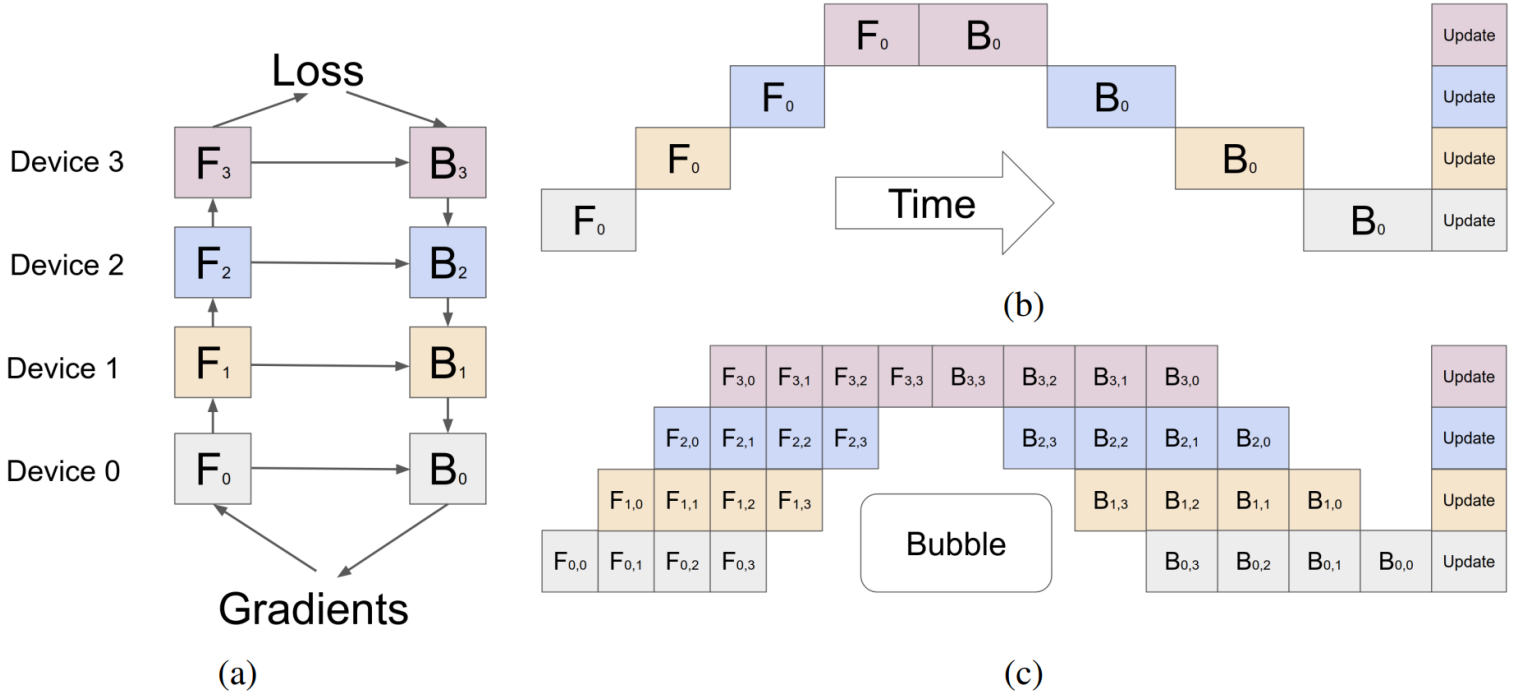


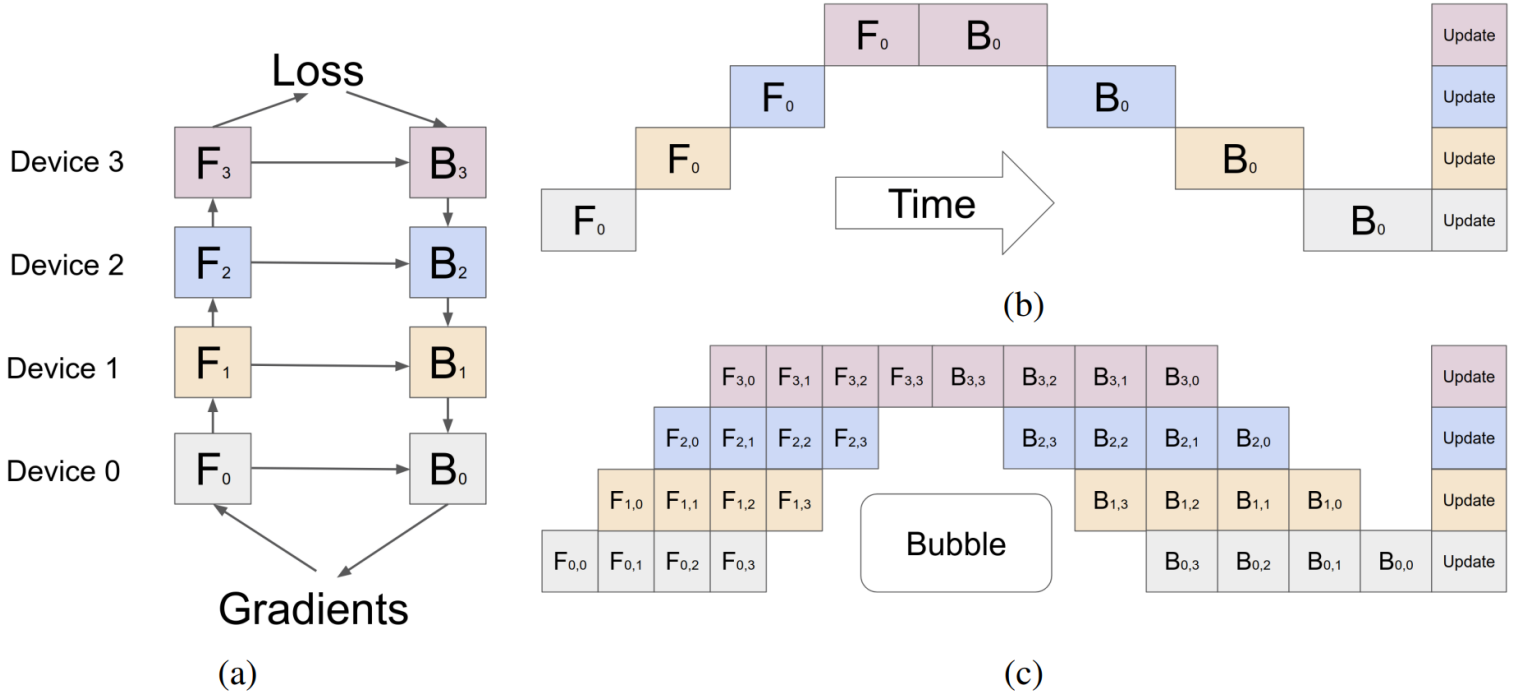
Figure 7: Illustration of Pipeline Parallelism [7]

Communication:

- Activations between nearby devices in forward pass
- Partial gradients between nearby devices in backward

Parallel Training: Pipeline Parallelism

Split network by layers, aligning devices by layer order to a pipeline, and pass data through devices [7]



Communication:

- Activations between nearby devices in forward pass
- Partial gradients between nearby devices in backward

Figure 7: Illustration of Pipeline Parallelism [7]

Pros: Conceptually simple and not coupled with network architectures. All networks have multiple layers.

Cons: Waste of compute in the Bubble. Bubble gets bigger with more devices and bigger batches.

Outline

Parallel Training

- Data Parallelism
- Pipeline Parallelism
- **Tensor Parallelism**
- Combination of Parallelism
- ZeRO Optimizer

Parallel Training: Tensor Parallelism

Split the parameter tensors of network layers into different devices for parallel matrix operations

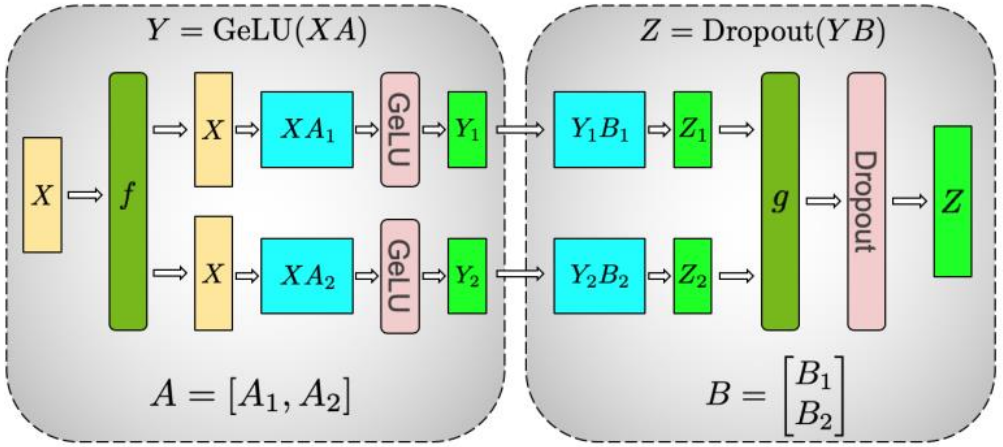


Figure 8: Tensor Parallelism of MLP blocks and Self-attention Blocks [8]

Parallel Training: Tensor Parallelism

Split the parameter tensors of network layers into different devices for parallel matrix operations

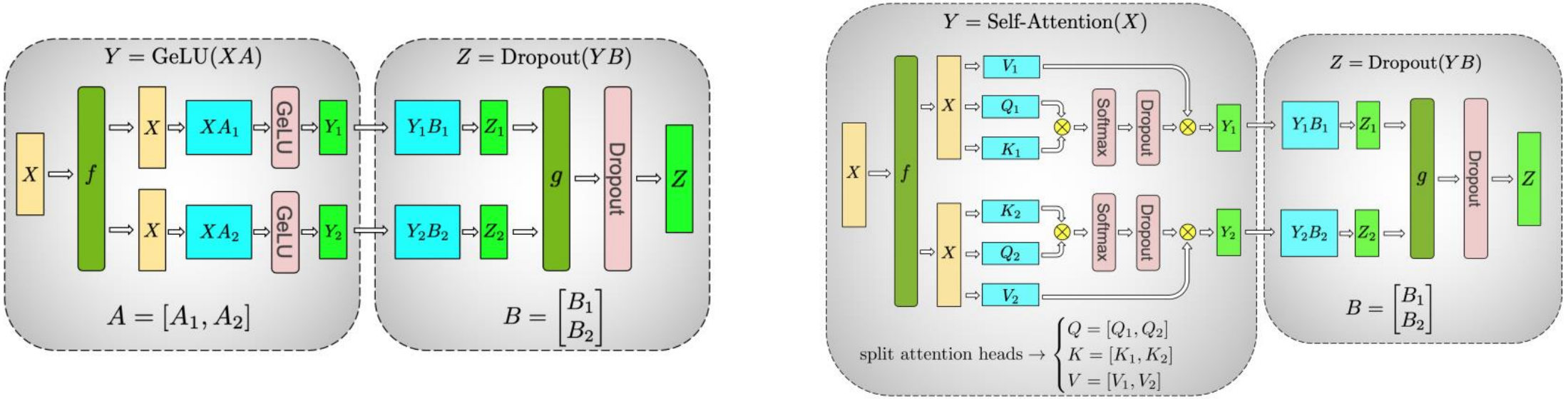


Figure 8: Tensor Parallelism of MLP blocks and Self-attention Blocks [8]

Parallel Training: Tensor Parallelism

Split the parameter tensors of network layers into different devices for parallel matrix operations

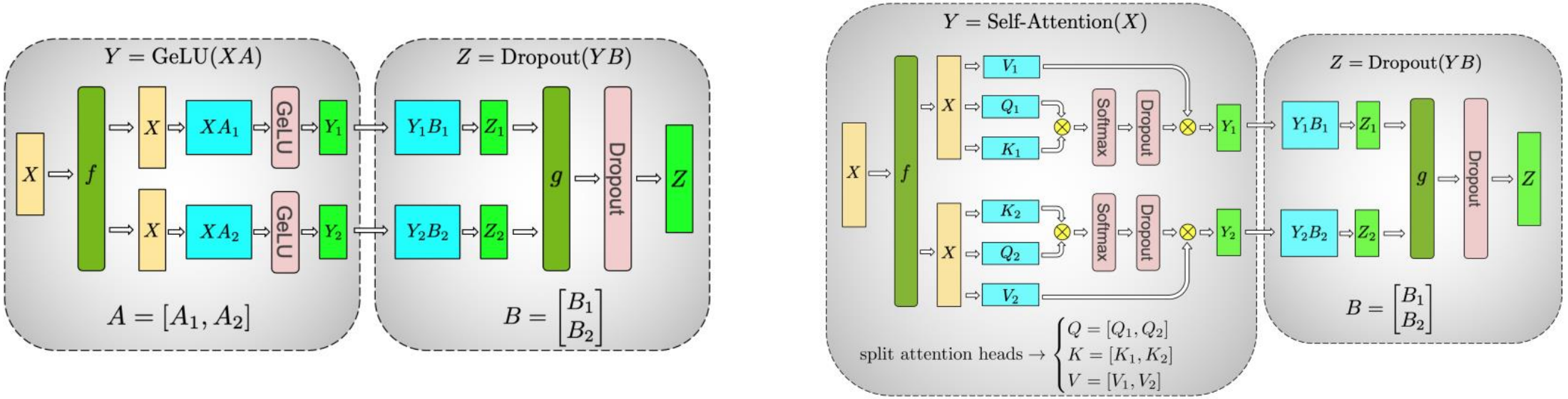


Figure 8: Tensor Parallelism of MLP blocks and Self-attention Blocks [8]

Pros: No bubble

Cons: Different blocks are better split differently, lots of customizations

Parallel Training: Tensor Parallelism

Split the parameter tensors of network layers into different devices for parallel matrix operations

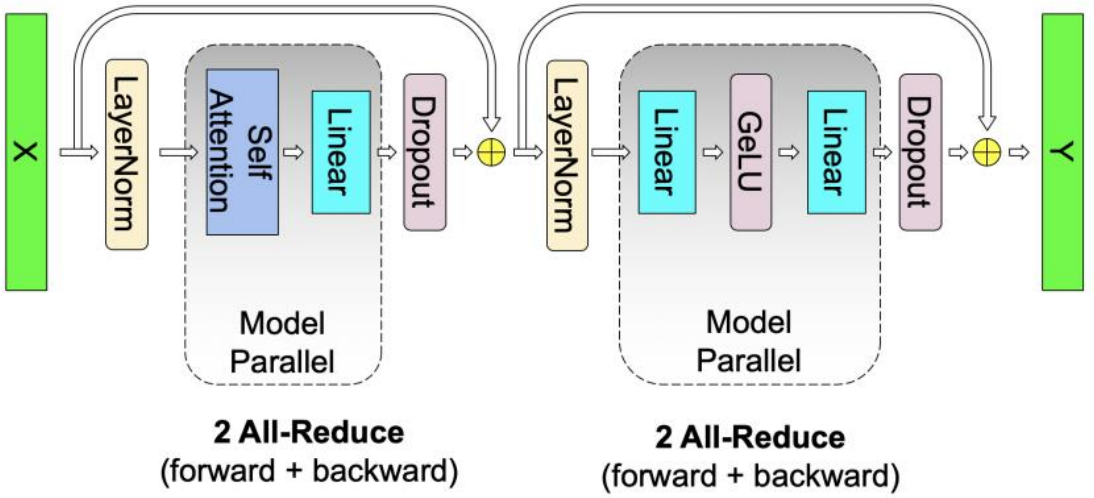


Figure 9: Communication of Tensor Papalism for a Transformer Layer [8]

Communication:

- All-gather of partial activations and gradients for each split tensor

Parallel Training: Combining Different Parallelism

Often data parallelism and model parallelism are used together.

- No need not to use data parallelism

Pipeline Parallelism and Tensor Parallelism can also be used together.

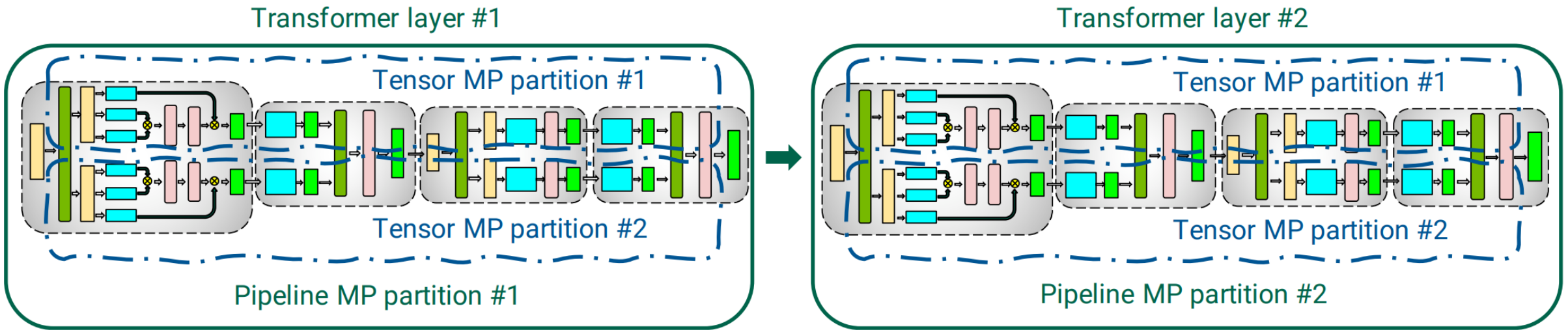


Figure 10: Combination of Tensor Parallelism and Pipeline Parallelism [9]

Outline

Parallel Training

- Data Parallelism
- Pipeline Parallelism
- Tensor Parallelism
- Combination of Combination
- **ZeRO Optimizer**

ZeRO: Redundancy in Data Parallelism

Majority of GPU memory consumption is on the optimization side: gradients and optimizer momentums

	Cost of 10B Model	Function to parameter count (Ψ)
Parameter Bytes	20GB	2Ψ
Gradient Bytes	20GB	2Ψ
Optimizer State: 1st Order Momentum	20GB	2Ψ
Optimizer State: 2nd Order Momentum	20GB	2Ψ
Total Per Model Instance	**80GB**	8Ψ

Table 1: Memory Consumption of Training Solely with BF16 (Ideal case) of a model sized Ψ

ZeRO: Reduce Memory Redundancy

ZeRO Optimizer: Split GPU memory consumption into multiple GPUs during data parallelism

Stage 1: Split Optimizer States

Stage 2: +Split Gradients

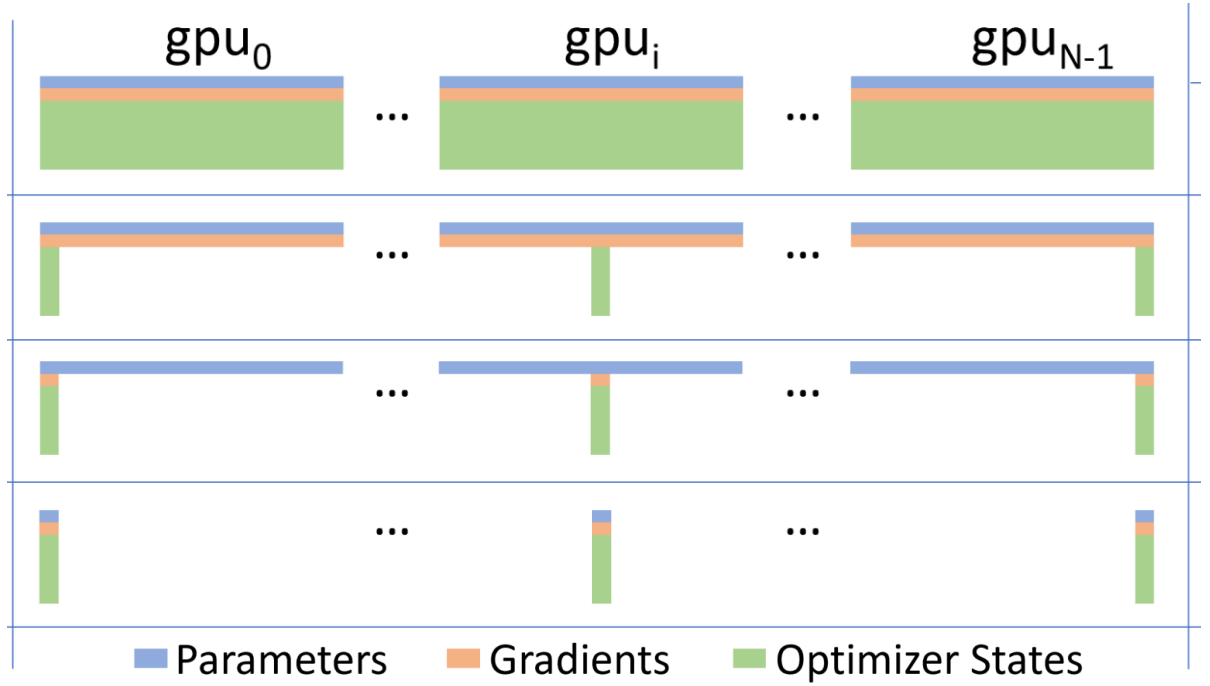
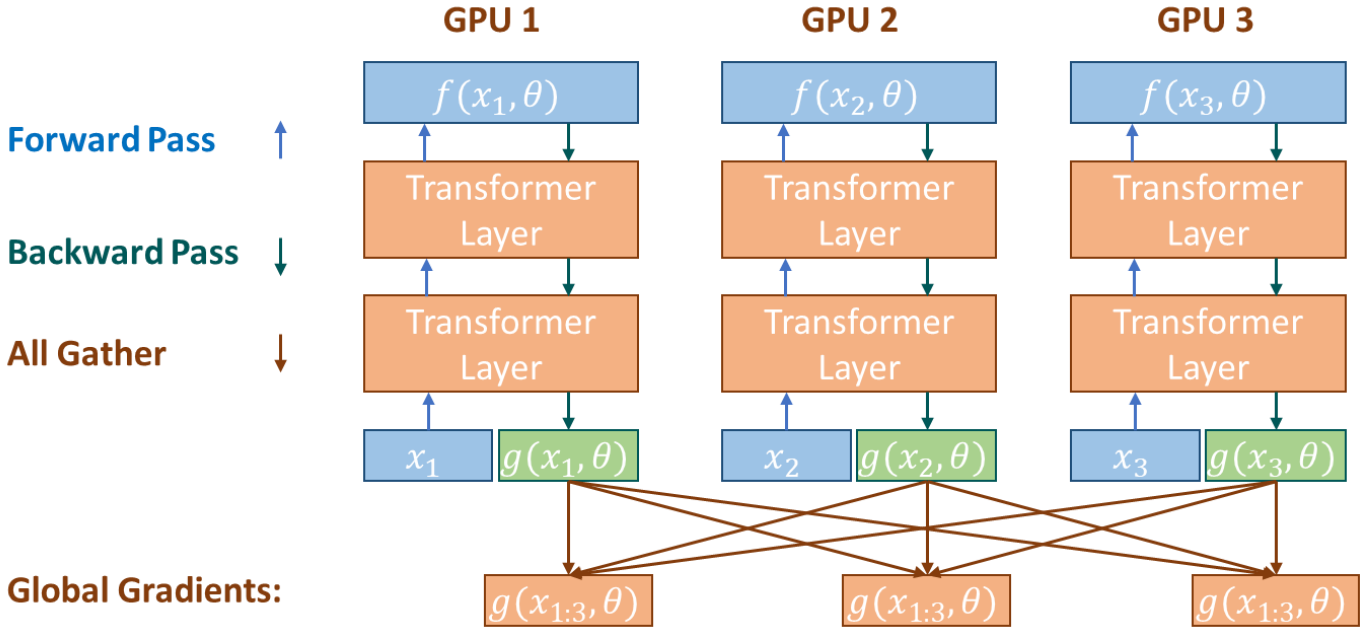


Figure 11: ZeRO Optimizer Stages [10]

ZeRO: Redundancy in Data Parallelism

ZeRO Stage 1 and 2: reducing memory redundancy

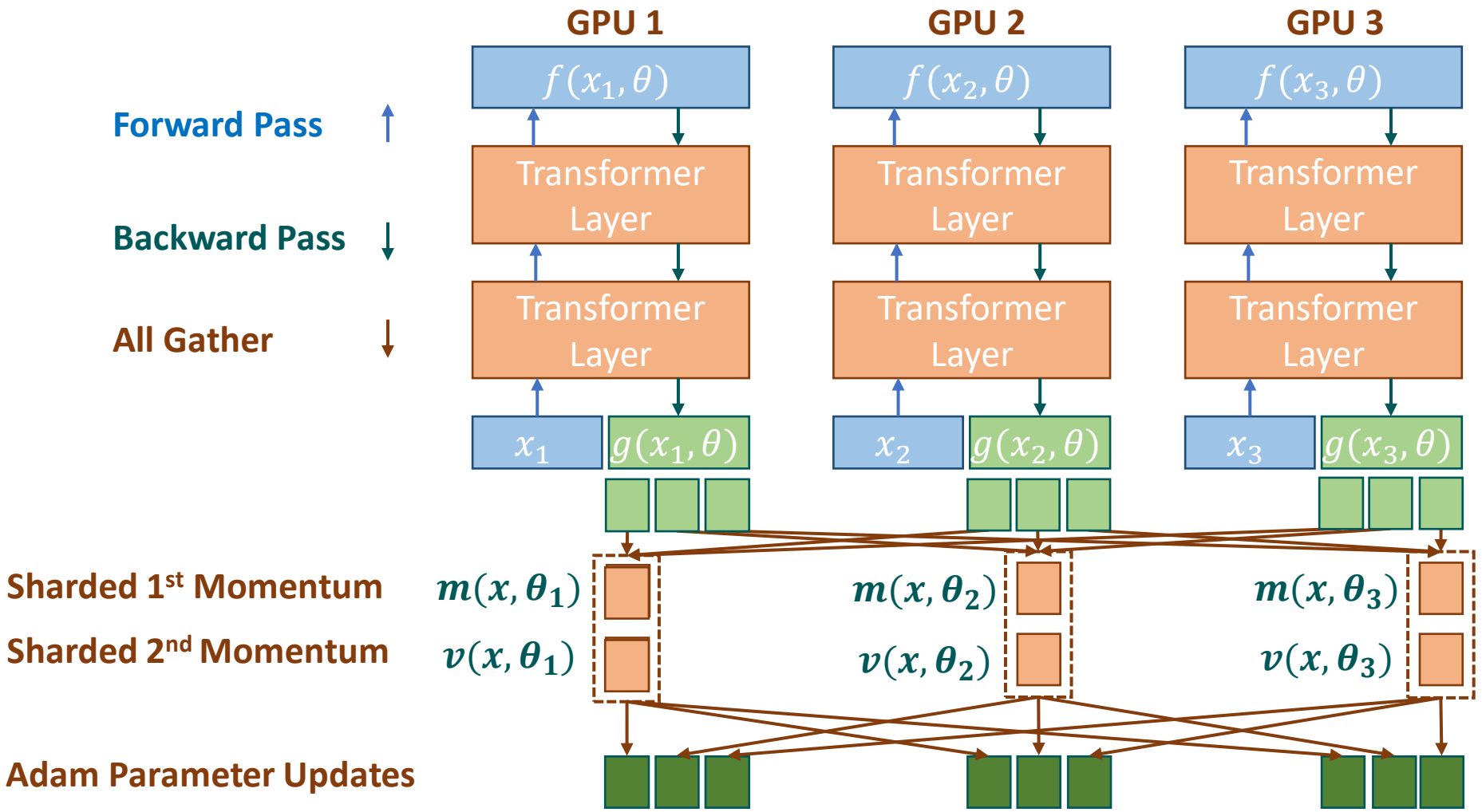


Observation:

- In data parallelism, each device only has access to local gradient
- All gather operation required on all gradients anyway

ZeRO: Redundancy in Data Parallelism

An example way to implement ZeRO Stage 1



ZeRO: Reduce Memory Redundancy

ZeRO Optimizer: Split GPU memory consumption into multiple GPUs during data parallelism

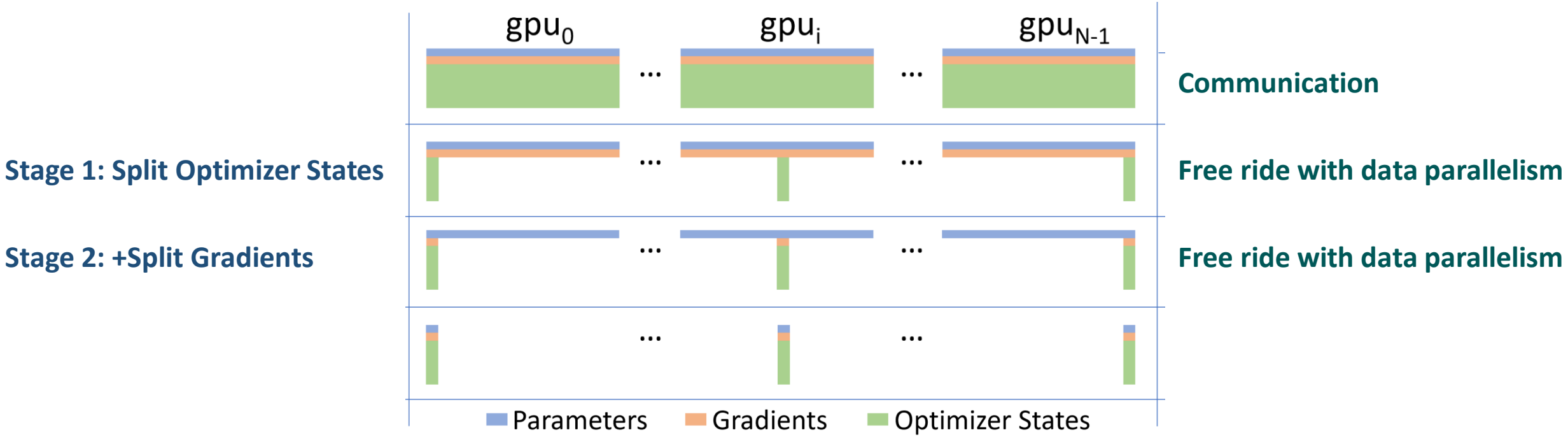


Figure 11: ZeRO Optimizer Stages [10]

ZeRO: Reduce Memory Redundancy

ZeRO Optimizer: Split GPU memory consumption into multiple GPUs during data parallelism

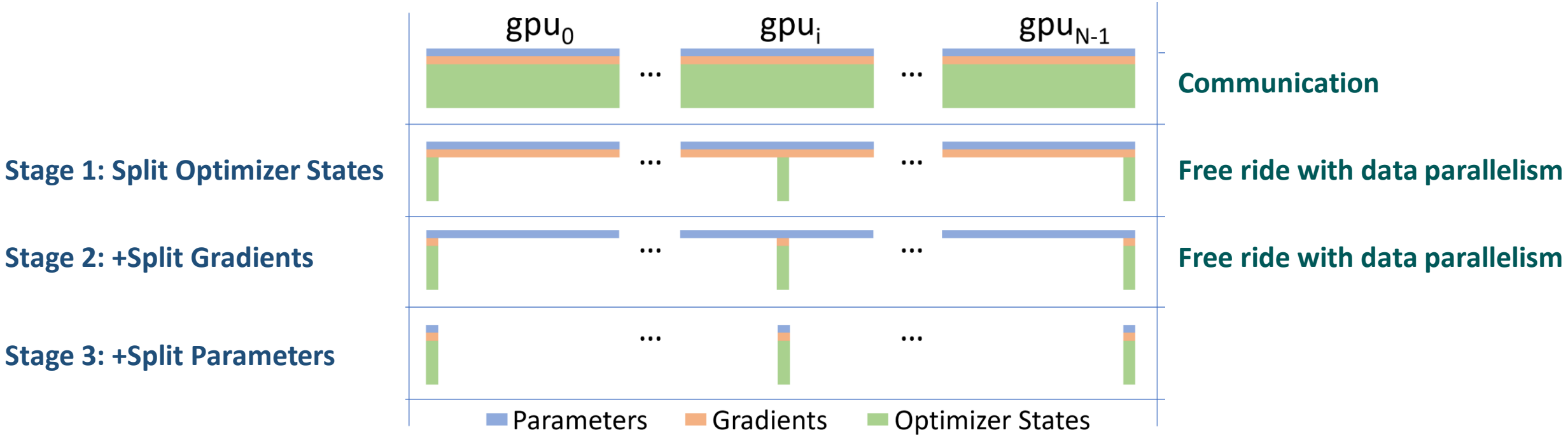
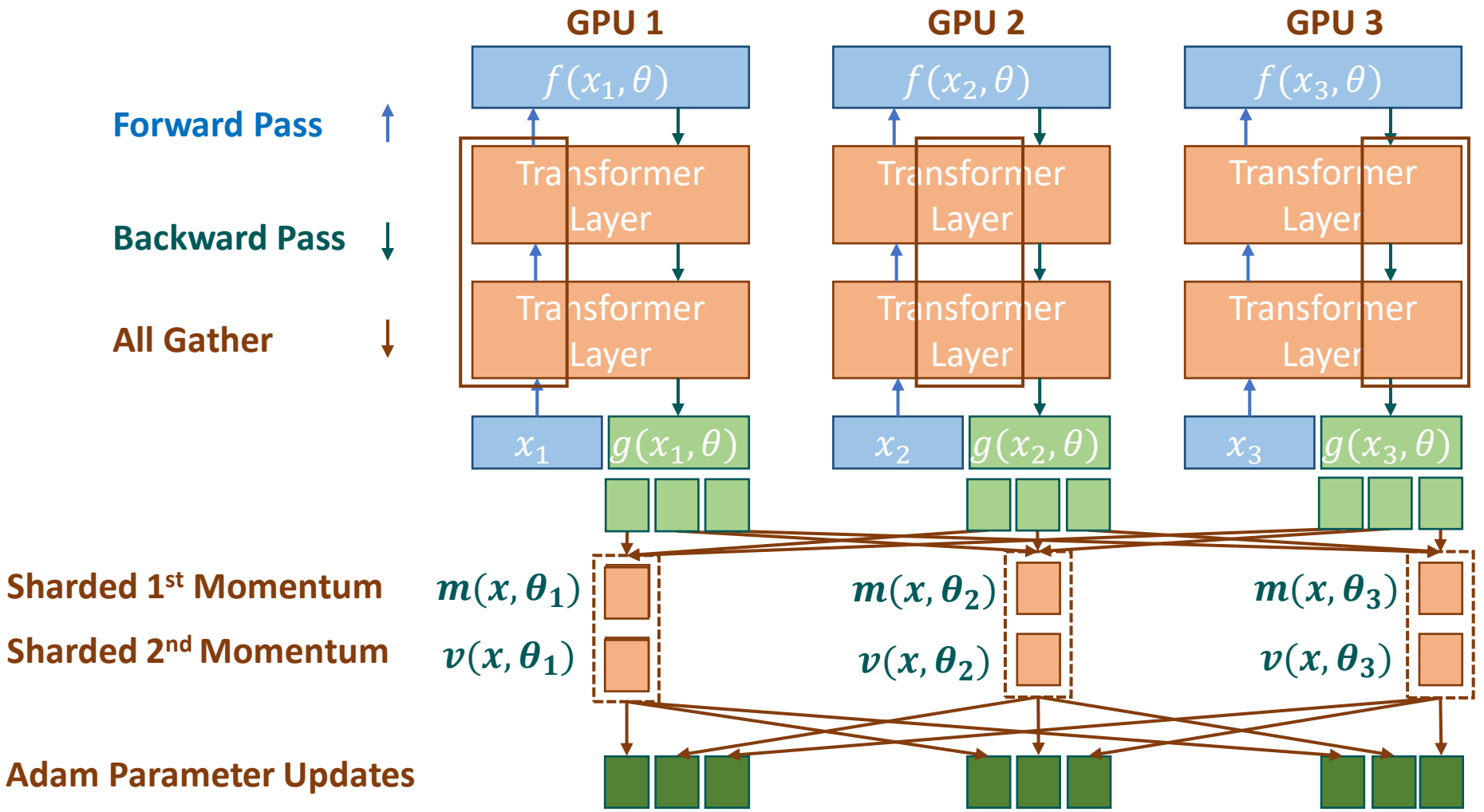


Figure 11: ZeRO Optimizer Stages [10]

ZeRO: Redundancy in Data Parallelism

Sharding parameters and passing them when needed



ZeRO: Reduce Memory Redundancy

ZeRO Optimizer: Split GPU memory consumption into multiple GPUs during data parallelism

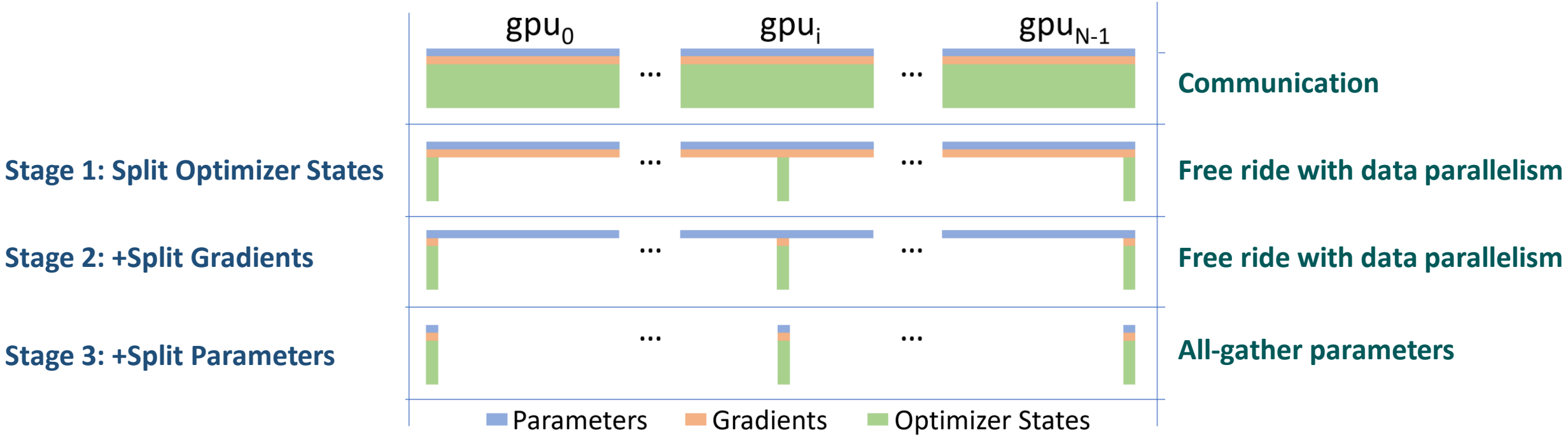


Figure 11: ZeRO Optimizer Stages [10]

Pros: Stage 1 and 2 free ride with data parallelism with huge GPU memory savings

Cons: Open-source support not as good

Notes: Stage 3 is different with tensor parallelism. It passes parameters when needed but still performs computations of the full layer/network in one GPU. It is data parallelism with GPU memory sharding

A peek into real large scale pretraining workflow

Lots of first hand information released through the FAIR OPT pretraining run:

https://github.com/facebookresearch/metaseq/tree/main/projects/OPT/chronicles

Background

A group of researchers and engineers are tasked with the goal of pretraining a large-scale model like GPT-3

- 1024 A100 80GBs to use. Yes!

Constraints:

- Task given at around Beginning of Nov 2021
- Goal is to pretrain a 175 Billion scale model by end of the year
  - Which at minimum require 33 days on 1K A100s
- With no previous experience on large scale pretraining at all

Challenge #1: Many Research Work Don’t Scale

Hope: We started with high hopes that all our research improvements at Small will give us a better GPT

We began this experiment lineage with the following settings:

- Batch-size of 2M
- FP32 Adam
- 8x Tensor Parallelism
- New data from Experiment 29
- LPE with sinusoidal initialization
- Normformer

Challenge #1: Many Research Work Don’t Scale

Reality: Short timeline, Big money on the line, Nothing too fancy

By this point, even though we made it past 1k updates without grad norms exploding or nans everywhere, it appeared as if training was going to stagnate (see the pink line above of the training ppl from update 4750 to ~6700 with experiment 11.10). We decided to follow through with our "plan B" that we set for ourselves on October 18 before starting any of these runs, where we would abort from these configurations (derived from the lineage of all the previous dense experiments conducted with the Fairseq codebase) and adopt as much of the Megatron/OpenAI GPT-3 settings as possible.

- We chose this path due to the fact that we need 33 days to fully train at this scale with 1024 80GB A100s, and time was running out before EOY hit. We also needed to buffer in time to evaluate this model on downstream tasks before EOY as well.
- We could keep going down the path of tweaking the experiment 11.xx lineage, but we have no pre-existing knowledge we would be able to make consistent progress in that time.
- Megatron/OpenAI GPT-3 settings are consistent with each other and have both been supposedly used to successfully train a 175B model (and above).

From all the things we changed in 11.xx, the main set of changes that were left to bridge the gap with Megatron/OpenAI settings were:

- Overall weight initialization
- Removing Normformer, removing embedding scaling, and changing LPE initialization to standard embedding init and not sinusoidal init

Challenge #2: Hardware Failures

GPU machines are not very reliable. With 1024 A100s, it is guaranteed to have bad nodes.

Update on 175B Training Run: 27% through

Written by: Susan Zhang, Stephen Roller, Naman Goyal, Sam Shleifer, Myle Ott

Posted on: December 3, 2021

It's been really rough for the team since the November 17th update. Since then, we've had 40+ restarts in the 175B experiment for a variety of hardware, infrastructure, or experimental stability issues.

The vast majority of restarts have been due to hardware failures and the lack of ability to provision a sufficient number of "buffer" nodes to replace a bad node with once it goes down with ECC errors. Replacement through the cloud interface can take hours for a single machine, and we started finding that more often than not we would end up getting the same bad machine again. Nodes would also come up with NCCL/IB issues, or the same ECC errors, forcing us to start instrumenting a slew of automated testing and infrastructure tooling ourselves. Some of these include:

- Replacing nodes through a script
- Adding GPU burn-in testing to detect memory errors
- Automating IB testing
- Monitoring train.log

https://github.com/facebookresearch/metaseq/blob/main/projects/OPT/chronicles/27_percent_update.md

Challenge #2: Hardware Failures

Solution? Hopefully better tooling in the future, but right now:

All in all, working around infrastructure issues has dominated the last two weeks of the team's time, given that these hardware issues can take the experiment down for hours at any time of the day. While we were fully aware that these issues would come up during training at this scale, given the time crunch of shipping a trained 175B model by end of H2 2021, we had no choice but to launch and see how far we would get without this additional tooling. Thanksgiving break was a painful reminder that automation on this front is necessary when training at this scale, and that cloud infrastructure instability is something to always prepare for, given lack of control over the underlying hardware.

Challenge #2: Hardware Failures

Forming an on-call group to watch OPT training

We are also happily inviting people to join our on-call. We have established runbooks and some tooling for dealing with the most common issues. Those who join this on-call will get firsthand experience at training a model at this scale, which will be valuable for all future large-scale efforts. However, each hour the experiment stagnates or goes down costs us $$, so the goal of the on-call is to minimize this downtime.

Challenge #3: Optimization Stability

Lots of optimization stability issues:

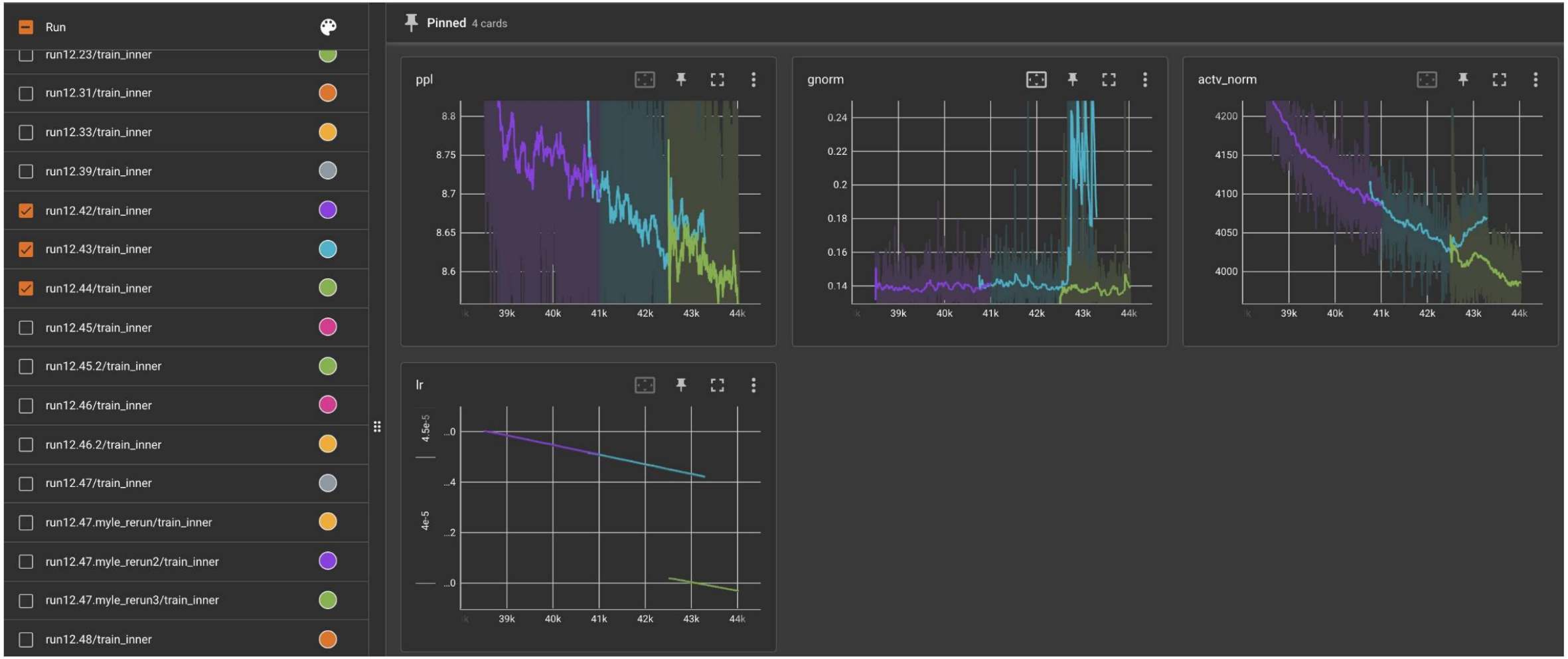
- Loss explodes, gradients overflows/underflows, training stagers…

Since the sleepless night of Thanksgiving break, this past week has been filled with gradient overflow errors / loss scale hitting the minimum threshold (which was put in place to avoid underflowing) which also causes training to grind to a halt. We restarted from previous checkpoints a couple of times, and found that occasionally training would get a bit further (~100 or more steps) before hitting the same issue again. At this point, we tried something a bit more drastic by testing out "SGD"-like settings for Adam (by setting beta1 to 0, and epsilon to 1), only to realize that reinitializing the Adam optimizer from a checkpoint also reloaded the previously saved betas. We tried switching to true vanilla SGD then, which required implementing an FP16-friendly version immediately, only to find that our learning rate might have been set too low for SGD to actually make progress.

As of this morning, we have made the decision to continue with Adam but instead try lowering the LR, which seems to have a surprising effect on reducing gradient and activation norms and allowing perplexity to continue dropping steadily. We had chosen initially to start with a much higher LR than GPT-3 175B, given ablations at smaller scale showing that GPT-3 LR settings were too low when trained in the fairseq codebase. However, now that we are later in training, it seems like the LR may not be decaying fast enough to keep training stable.

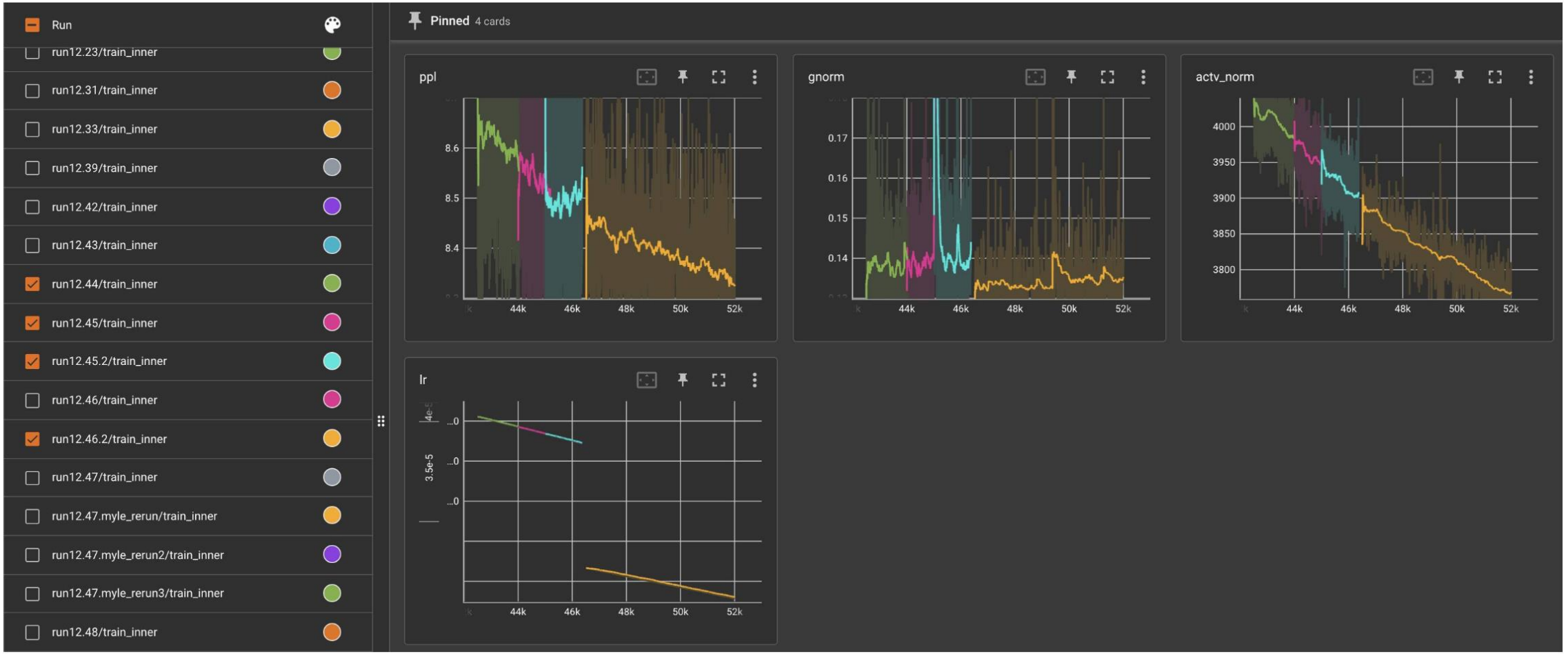
Challenge #3: Optimization Stability

Infrastructure issues aside, there have also been a few close calls with training stability. On run 12.43, we noticed grad norm and activation norm starting to spike/drift (light blue curve):



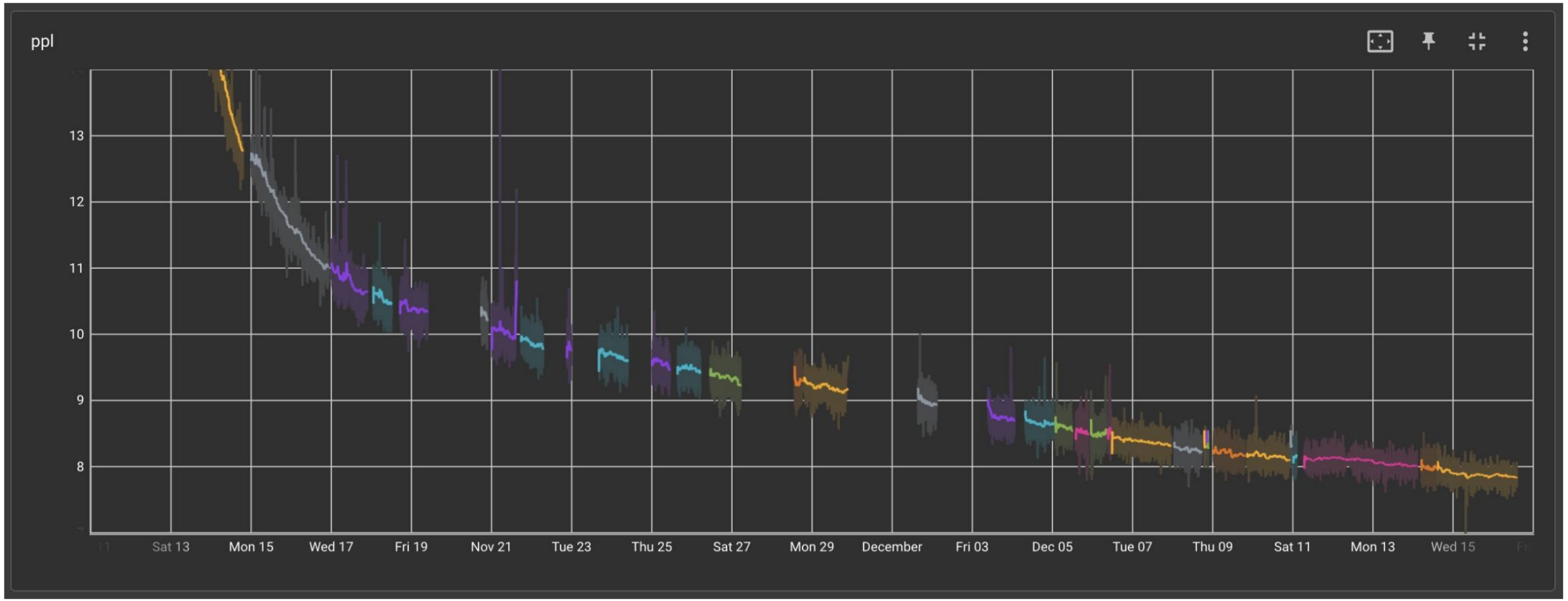
Challenge #3: Optimization Stability

In response, we lowered our learning rate by 10%, which was sufficient for stabilizing our grad norm and activation norm. This happened again on run 12.45.2 (below), where our training perplexity started to diverge (light blue) after a few large grad norm spikes. We lowered our learning rate at this point to 2/3 of what OpenAI 175B GPT-3 used, and managed to continue training.



Challenge #3: Optimization Stability

We managed to hit our top three record long runs of the experiment these past two weeks, lasting 1.5, 2.8, and 2 days each! If we were to look at only the runs that have contributed to pushing training further and plot training perplexity against wall clock time, we get the following:



The Importance of Scaling Law

Essential to determine what to do at large scale using observations at smaller scale

- DeepMind just released details last week on their 280B Gopher model (GPT-style) that was trained a year ago (for reference, OpenAI released GPT-3 details on May 2020).
  - What goes unmentioned in all of these large-scale efforts is the amount of compute needed to run all of the experiments that help inform decisions about *how/what* to scale. This will be something we need account for in the future as well, if we want to continue pushing the limits of these large-scale models. In other words, allocating just enough compute budget to train a large-scale model won't be enough to guarantee a better model.

Final Remarks from OPT

As of yesterday, at 12:46pm PST on January 6, our 175B model finally completed its training run on 300B tokens. This required ~4.30E+23 FLOPs of compute, or roughly ~33 days of continuous training on 1024 80GB A100s (assuming no hardware issues, no numerical instabilities, etc.).

To frame this:

- The 175B GPT-3 model trained by OpenAI required 14.8 days of compute on 10,000 V100s, and consumed 3.14+23 FLOPs. The code to do so is not open-sourced.
- This was not a benchmarking exercise. The model was trained to "completion" with a corpus of 180B tokens. We did not have time to curate a larger dataset before training started, given a tight deadline to deliver by the end of H2 2021.
- Scaling to 1024 A100s to handle a real workload of this size is highly nontrivial. We will discuss infrastructure pain-points below.
- Ensuring training converges at this scale is also highly nontrivial without sufficient ablations at "medium" scale. Results obtained from training at "small" scale (< ~13B params) also do not necessarily hold when "scaled-up". We will cover these learnings in a note to be released in the upcoming weeks.

Other Notable Literatures in Scaling Up

Different configurations of layer normalization: pre layernorm, post layernorm and their combination

- Xiong et al. “On Layer Normalization in the Transformer Architecture”. ICML 2020
- Zhang and Sennrich. “Root Mean Square Layer Normalization”. NeurIPS 2019

Position embeddings for longer contexts and expressiveness

- Su et al. “Roformer: Enhanced transformer with rotary position embedding.” arXiv 2021

Stability improvement from adaptive initialization

- Liu et al. “Understanding the Difficulty of Training Transformers”. EMNLP 2020

Quiz: What can we do to reduce communication overhead if only slow network connection is available in between GPUs?