OAK RIDGE National Laboratory

AI For Science At Scale – Part III
(Training LLMs with Hundreds of Billions of Parameters)

Sajal Dash

Research Scientist, HPC and AI

Analytics and AI Methods at Scale (AAIMS), NCCS

dashs@ornl.gov

ORNL is managed by UT-Battelle LLC for the US Department of Energy

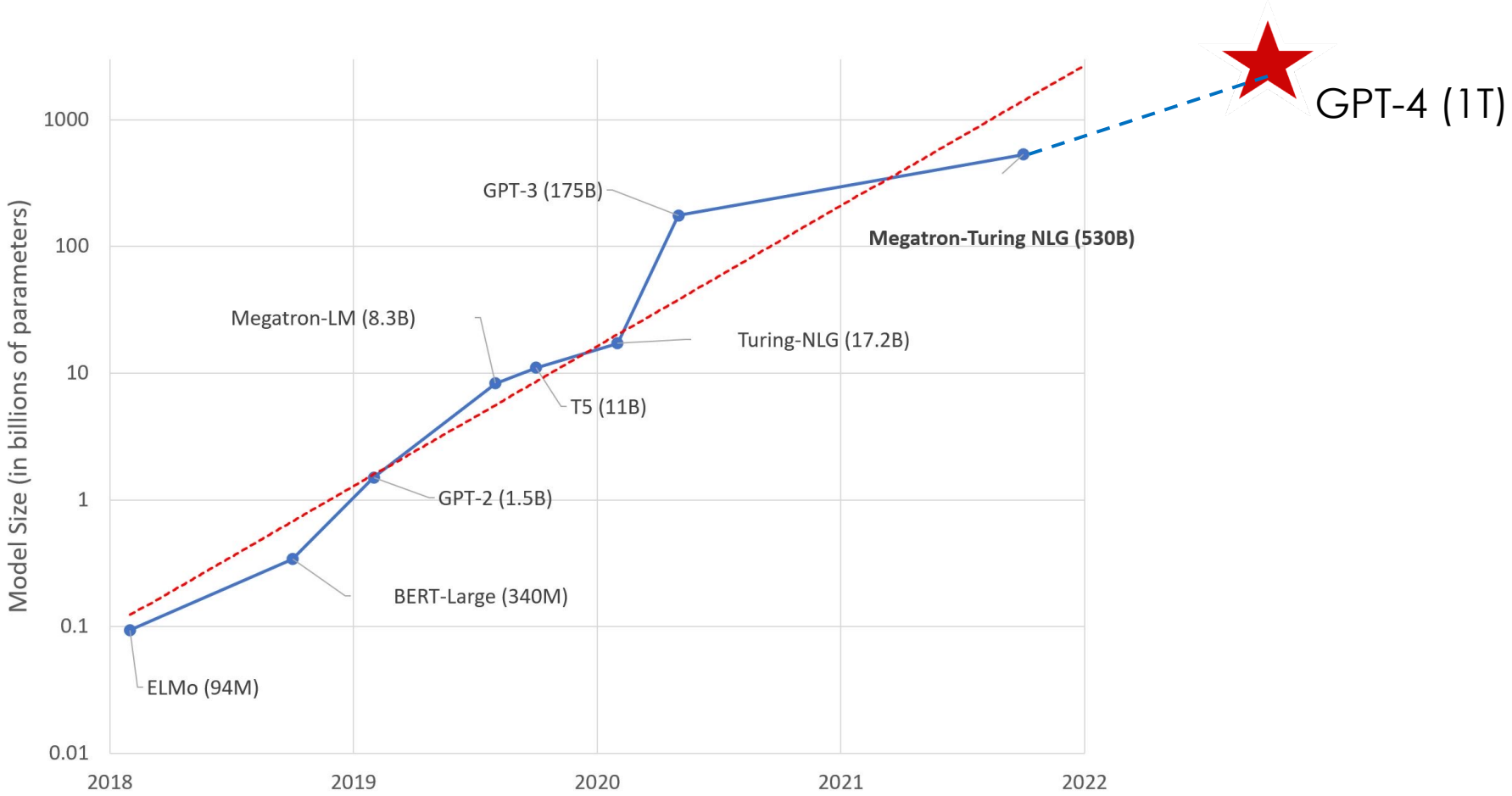
U.S. DEPARTMENT OF ENERGY

A Story where Goliath Wins

2017: Attention is all you need
[Google]

2020: Language Models are Few-Shot Learners
[OpenAI]

Model Size Exploded with Time



What Do You Need to Train a Trillion Parameter Model?

Compute:
120*P^2 ~ 120 Million ExaFlops

Memory:
20P ~ 20 Terabytes

Modern GPUs:
A100 – 80GB Memory, 312Teraflops

- How to distribute the training workload across hundreds of MI250X GPUs?
- How to take advantage of the existing software developed targeting NVIDIA gpus?

Outline

- Distributed training of LLMs
- Preparing a training software stack for Frontier
- Finding best distributed training strategies
- Training up to a Trillion parameter model on Frontier (**for a few iterations**)
- Training LLMs on OLCF resources

Transformer Model Architecture

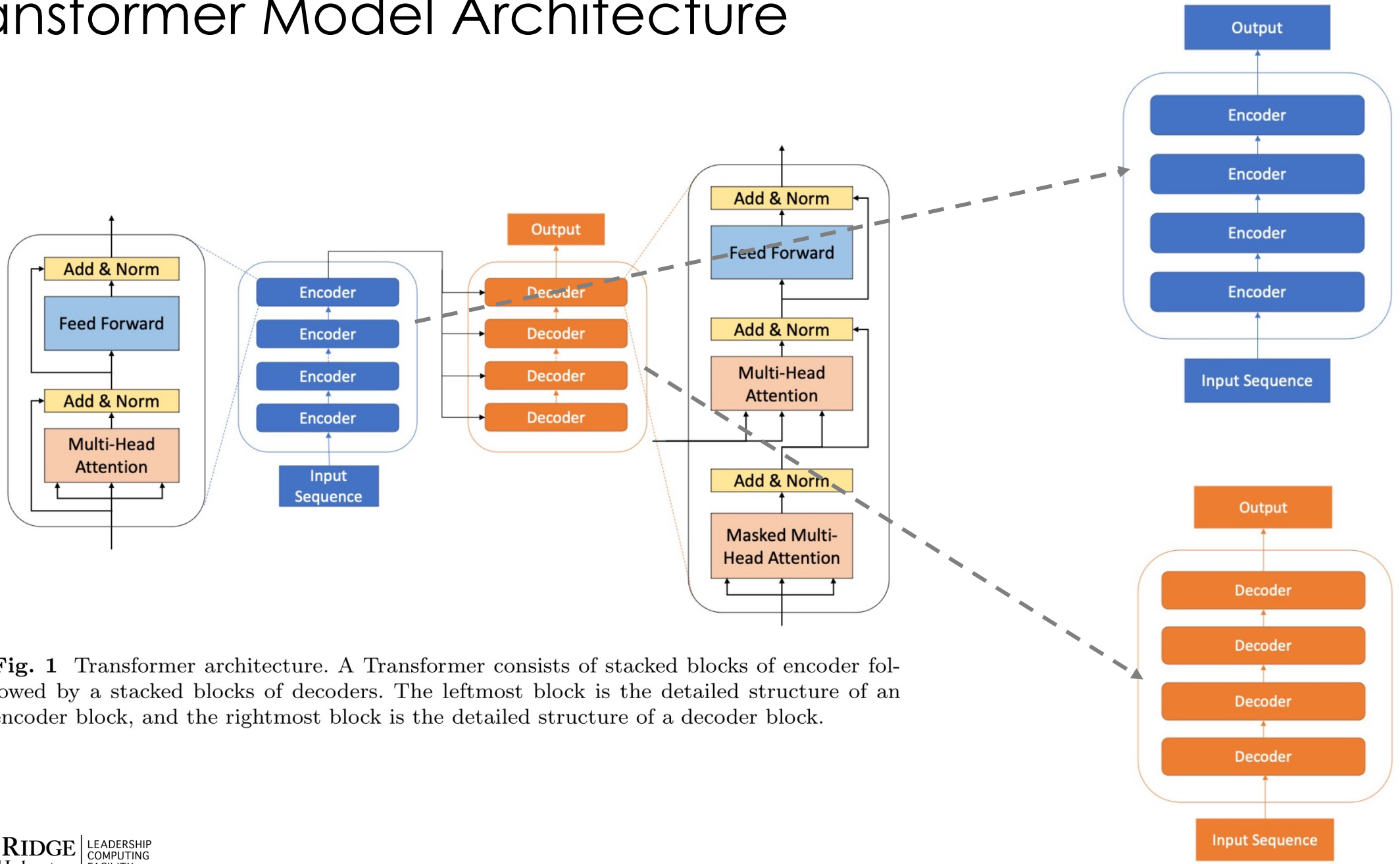
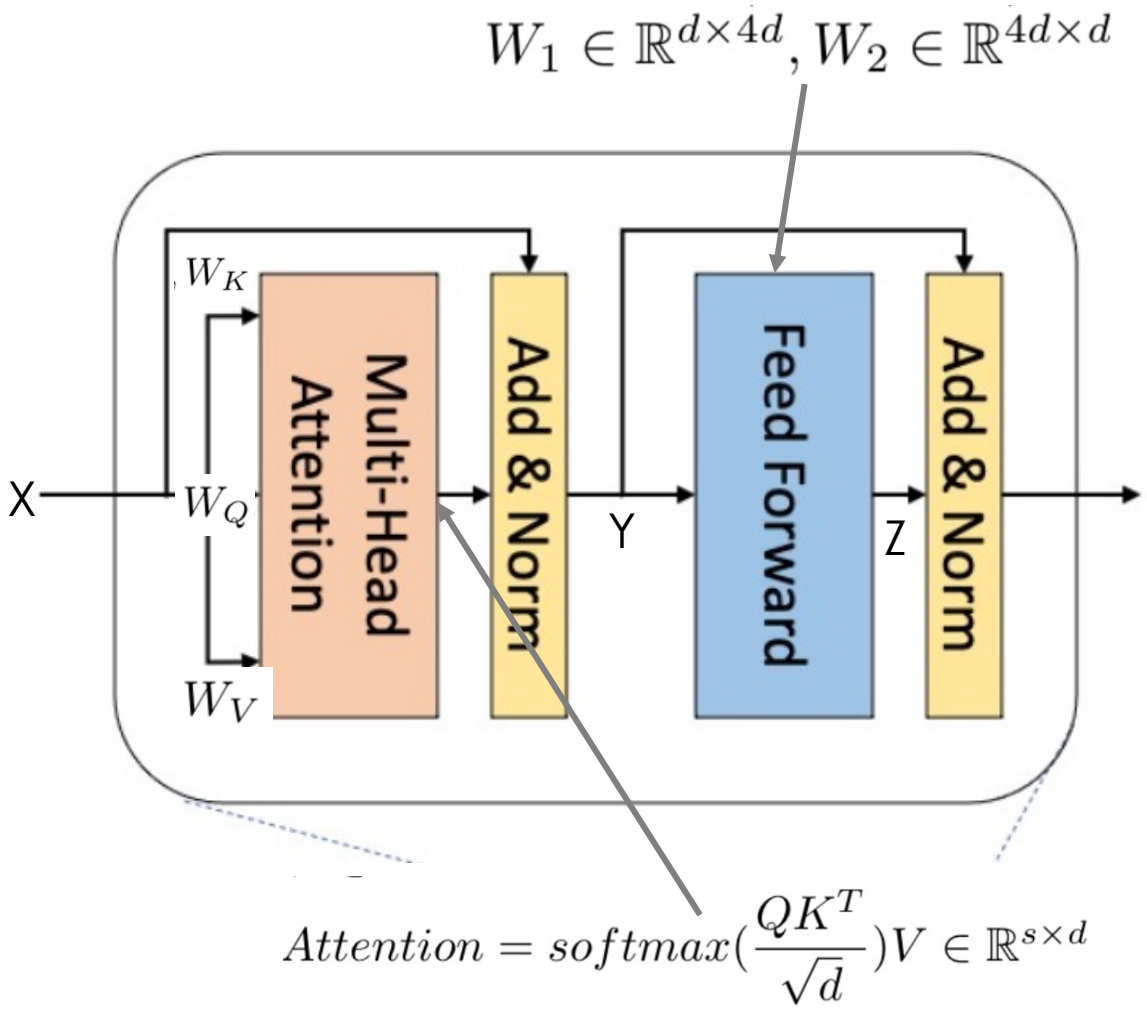


Fig. 1 Transformer architecture. A Transformer consists of stacked blocks of encoder followed by a stacked blocks of decoders. The leftmost block is the detailed structure of an encoder block, and the rightmost block is the detailed structure of a decoder block.

Forward Pass

Weights to update: W_K , W_Q, W_V, W_1, W_2



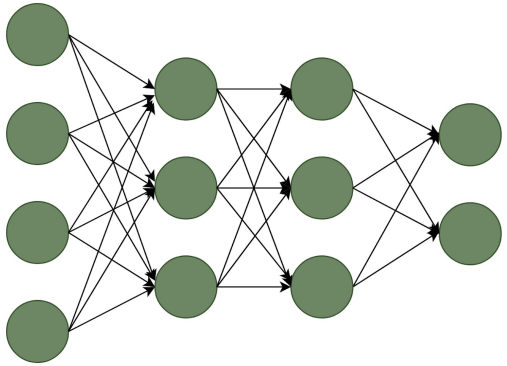
Let, $X \in \mathbb{R}^{s\times d}, W_K, W_Q, W_V \in \mathbb{R}^{d\times d}$.
Then, $K = XW_K, Q = XW_Q, V = XW_V$,
$K, Q, V \in \mathbb{R}^{s\times d}$.

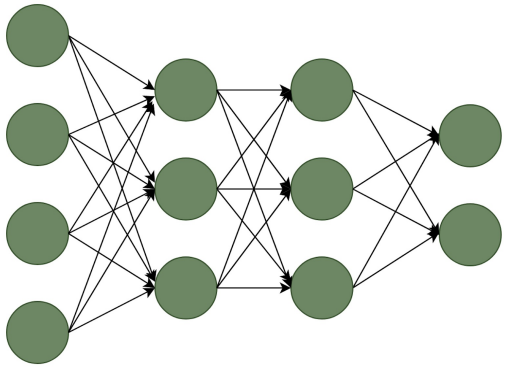
And, $Attention = softmax(\frac{QK^T}{\sqrt{d}})V \in \mathbb{R}^{s\times d}$

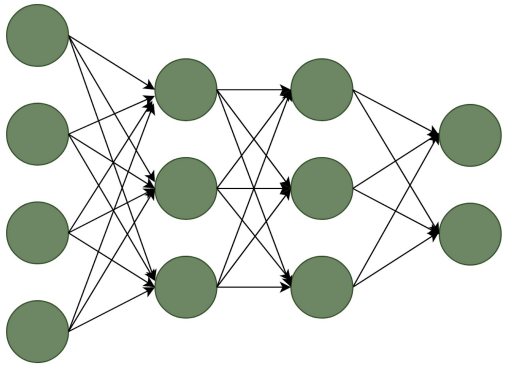
...

$Y = func(Attention) \in \mathbb{R}^{s\times d}$
$W_1 \in \mathbb{R}^{d\times 4d}, W_2 \in \mathbb{R}^{4d\times d}$
$Z = GeLU(YW_1) \times W_2 \in \mathbb{R}^{s\times d}$

Training DL Models with Large Data



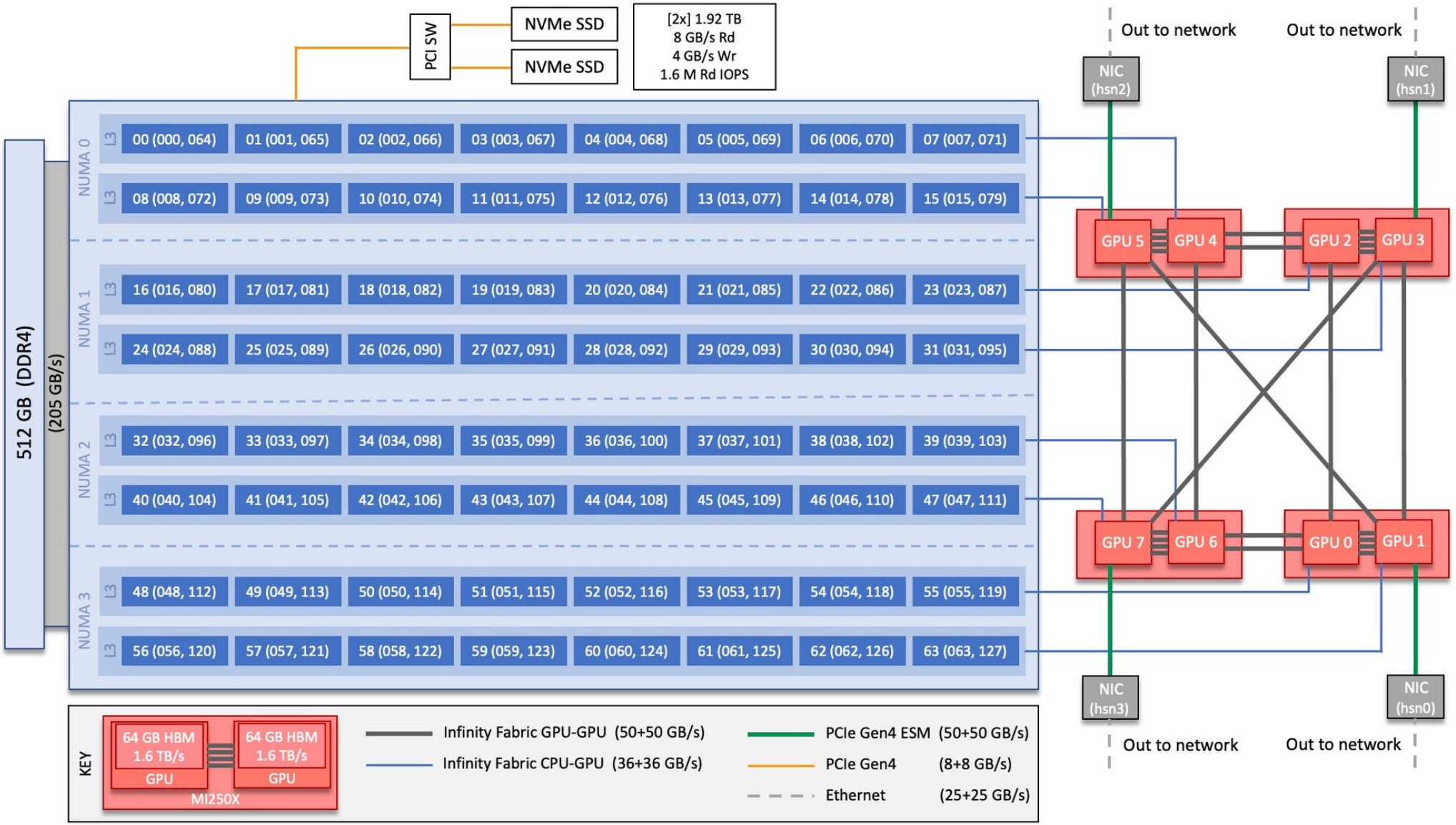




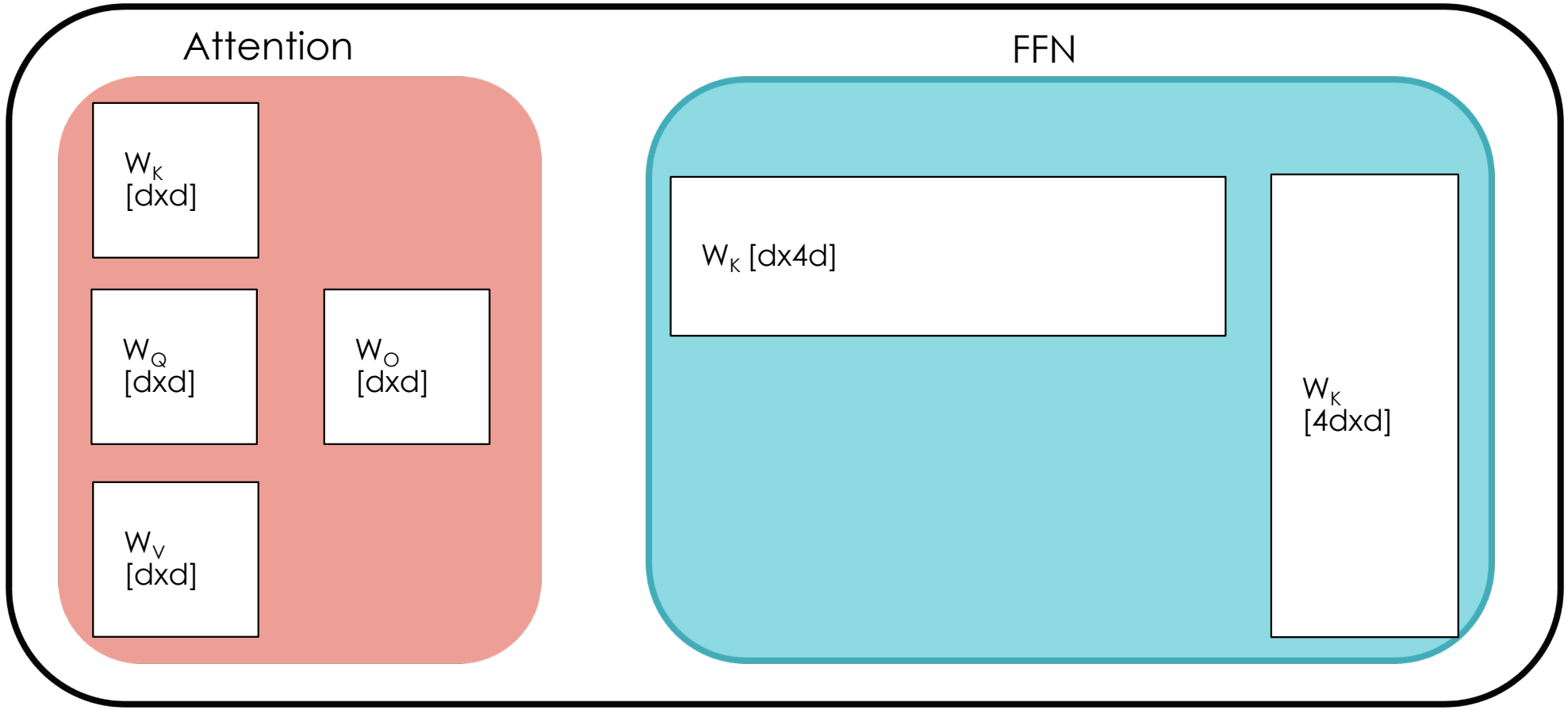
Batch 1.1
Batch 1.2
Batch 1.3

[Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour]

Frontier Node Architecture



Inside a Transformer Layer



4d^2

4d^2 + 4d^2 = 8d^2

Number of parameters
One layer: 12d^2
L layers: 12Ld^2

How to Grow a Transformer Model?

- Number of parameters 12Ld^2 can be think of a volume
  - With L (#Layers), the model grows linearly
  - With d (hidden dimension), model grows quadratically

Model	#Layers	Hidden size	#Attention heads
1.4B	24	2114	24
22B	48	6144	48
175B	96	12288	96
1T	128	25600	128

Memory Requirement During Training an LLM

Model Weights: Number of parameters: 12Ld^2

- 4 bytes * number of parameters for fp32 training
- 6 bytes * number of paramet
  memory)

Optimizer States: Adam
(mea

- 8 bytes * number of paramet
- 2 bytes * number of paramet
- 4 bytes * number of paramet

Values	Memory Requirement		
	22B Model	175B Model	1T Model
Parameters (6x)	132 GB	1050 GB	6 TB
Gradients (4x)	88 GB	700 GB	4 TB
Optimizer States (8x)	176 GB	1.4 TB	8 TB
Total Memory (20x*)	440 GB	3.5 TB	20 TB

Gradients Same as number of parameters

- 4 bytes * number of parameters for either fp32 or mixed precision training (gradients are always kept in fp32)

Forward Activations Batch-size x output-nodes?

- size depends on many factors, the key ones being sequence length, hidden size and batch size.

Model Parallelism: Why and How

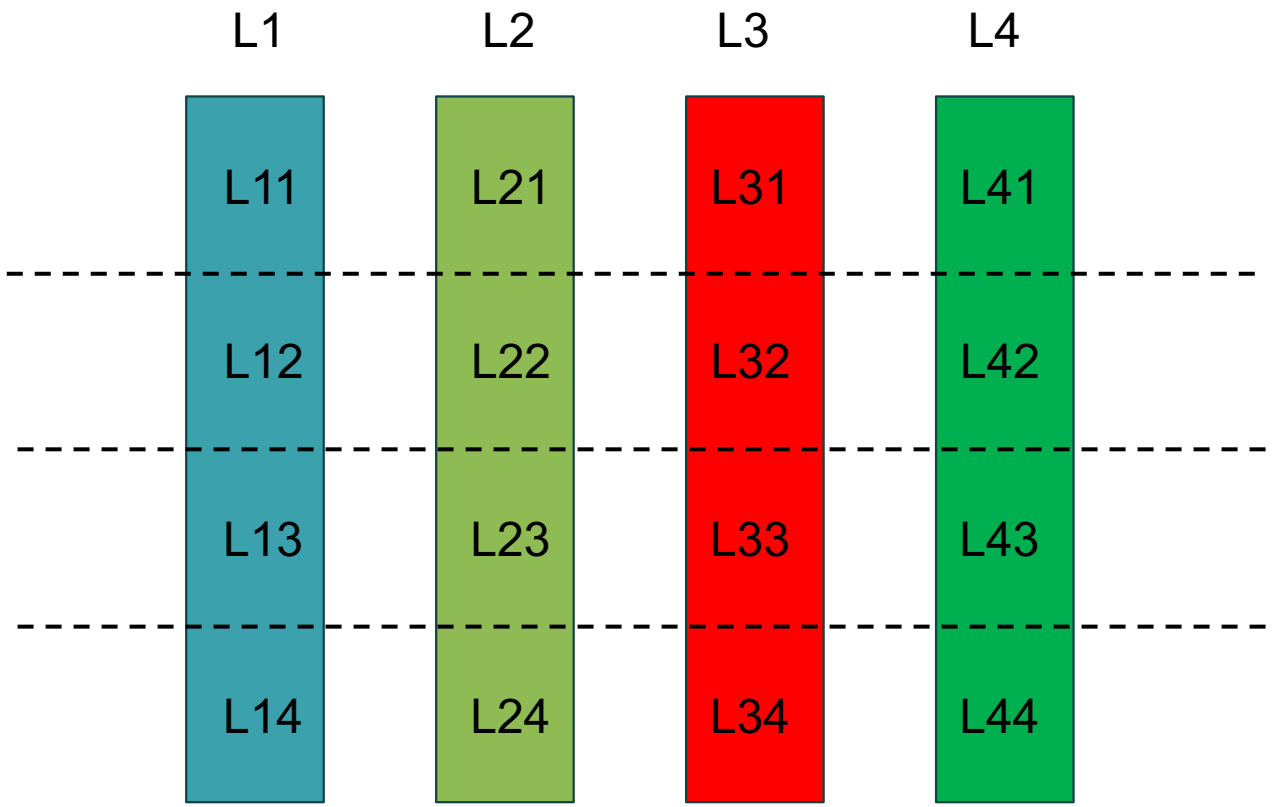
- Models (or Memory needed for their training) are too big to fit in a single GPU
- So, we need to break the model into pieces
- What's broken needs to be rebuilt from the pieces
- It's like Kintsugi, but with a more practical concern such as price of gold (comm. Latency)

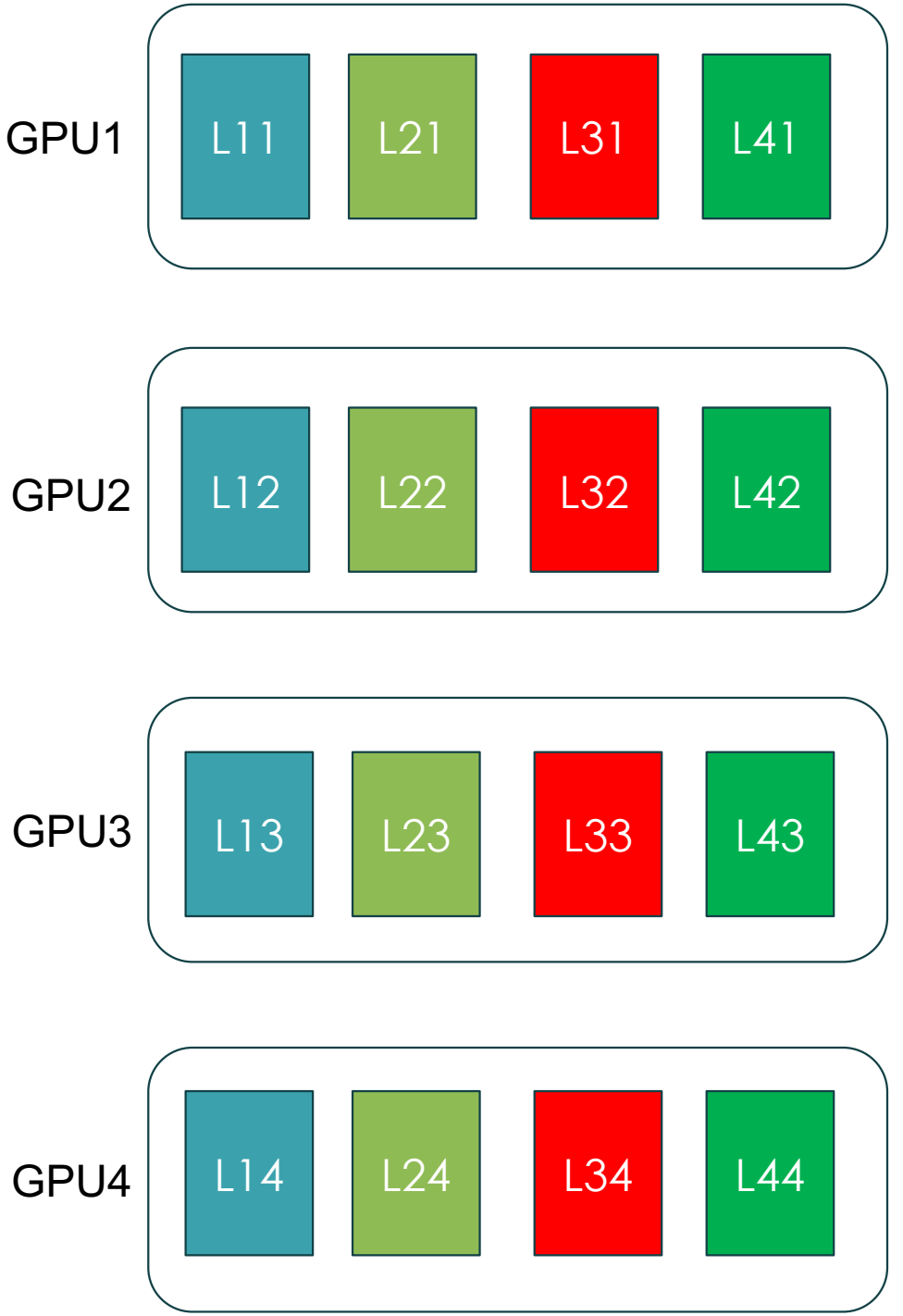
Kintsugi

Tensor Parallelism

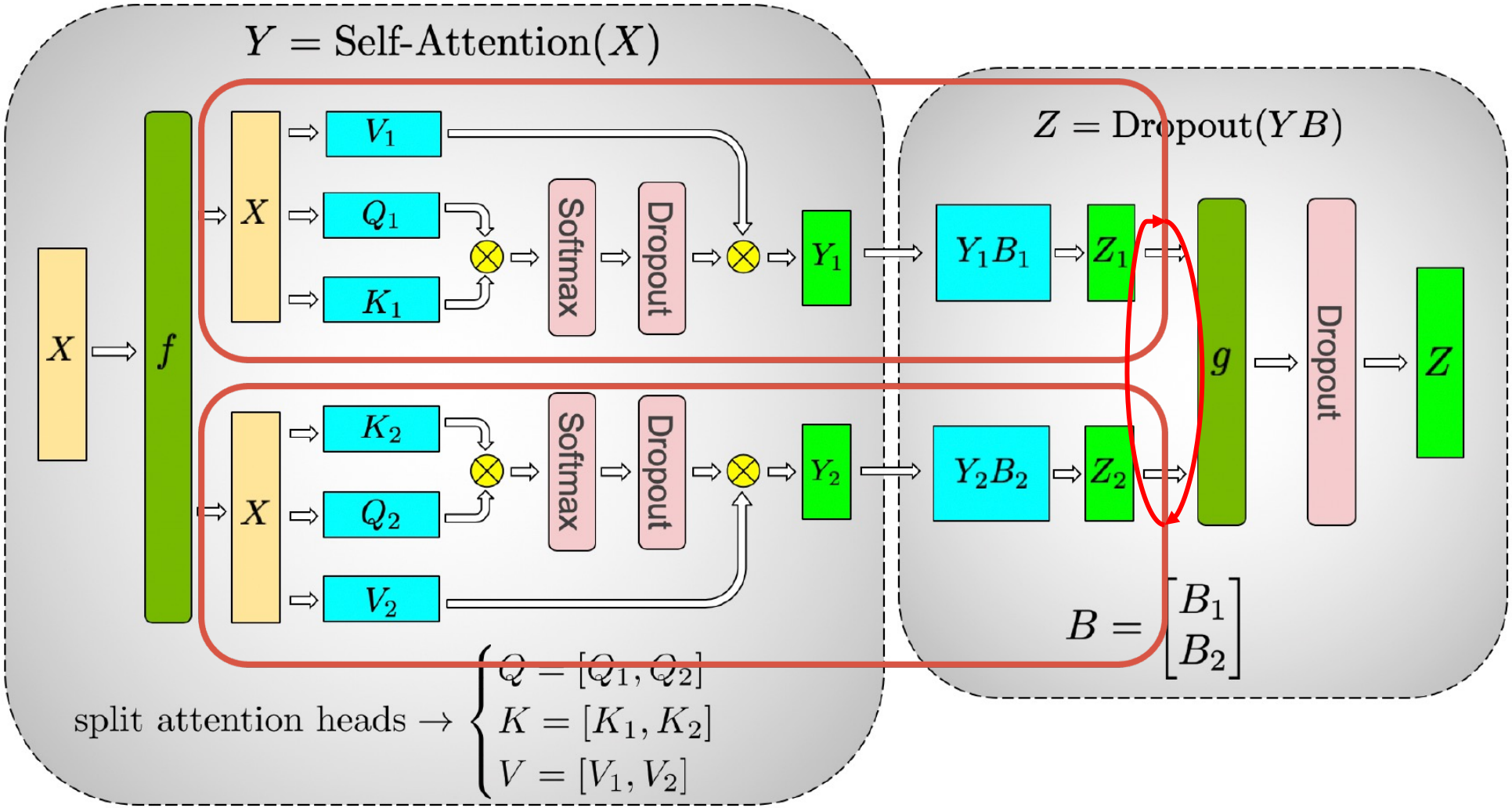
- Model is too large to fit in a GPU's memory
- We slice the model tensors along a suitable dimension (row or column), and the GPU memory is large enough to fit one slice.
- Unlike sharded data parallelism, this is not data parallelism, the same data gets evaluated by different part of the same layer, and the output gets combined.

Tensor Parallelism (TP=4)

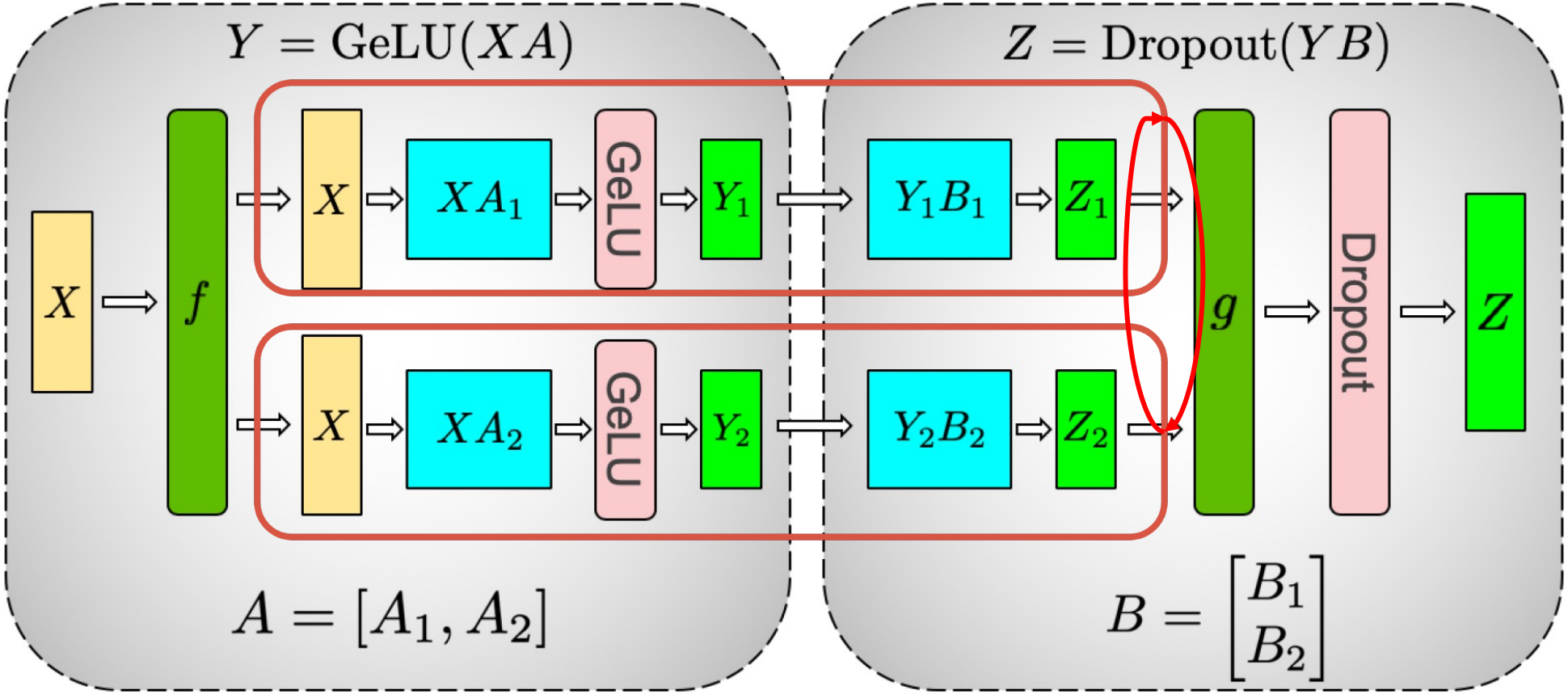




Self-Attention



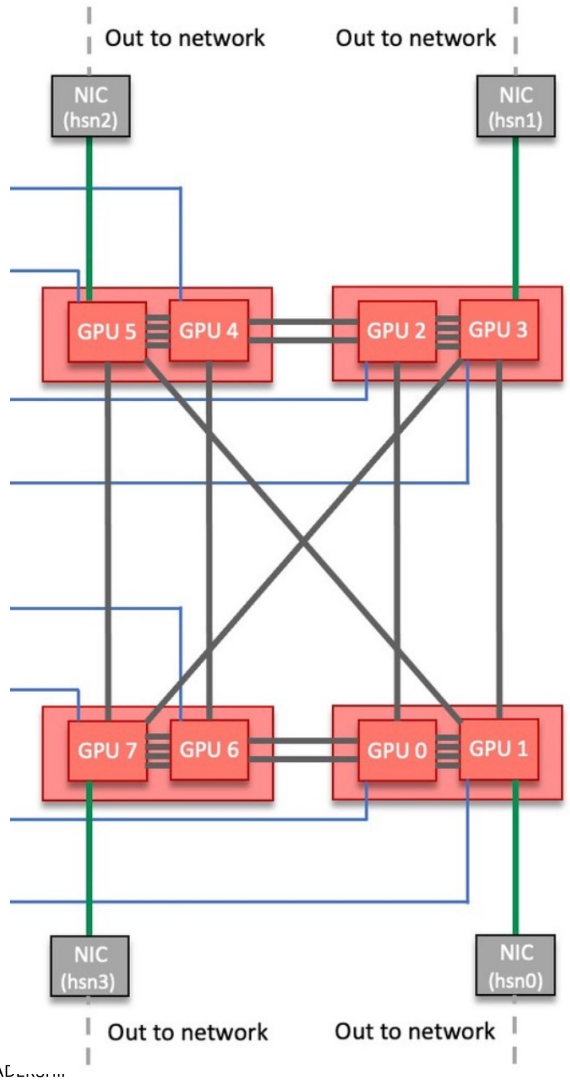
Feed Forward Network (FFN)

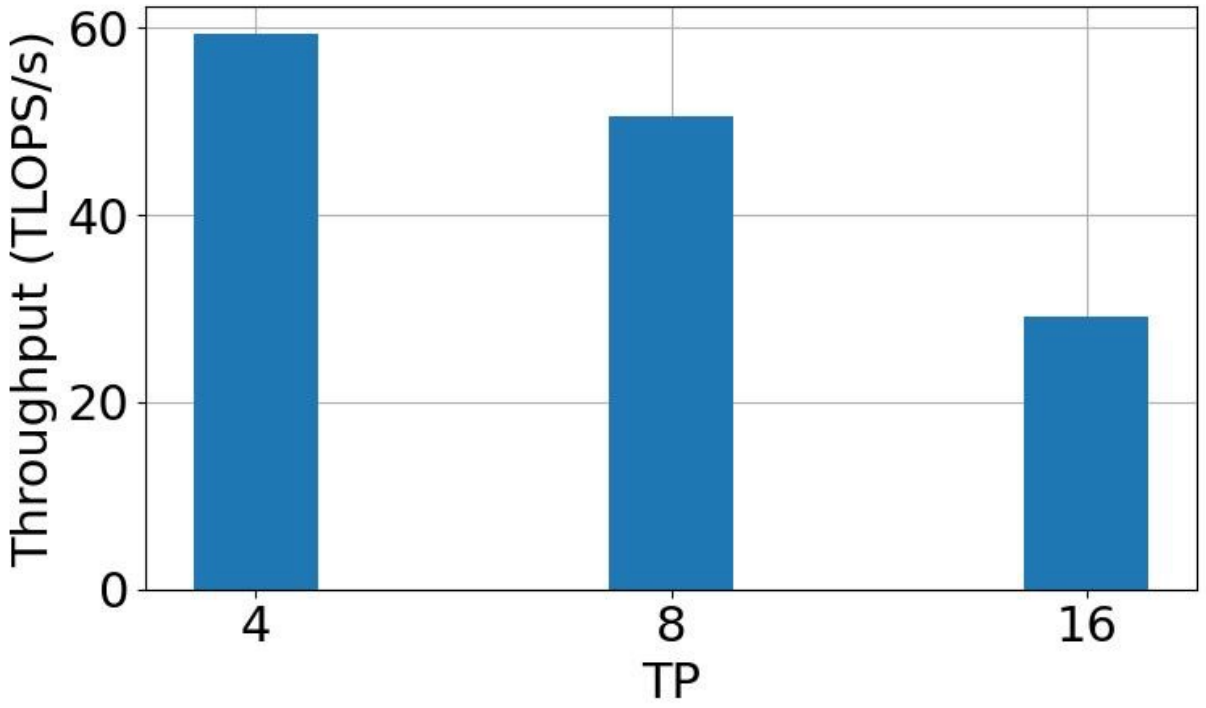


Limitations of Tensor Parallelism

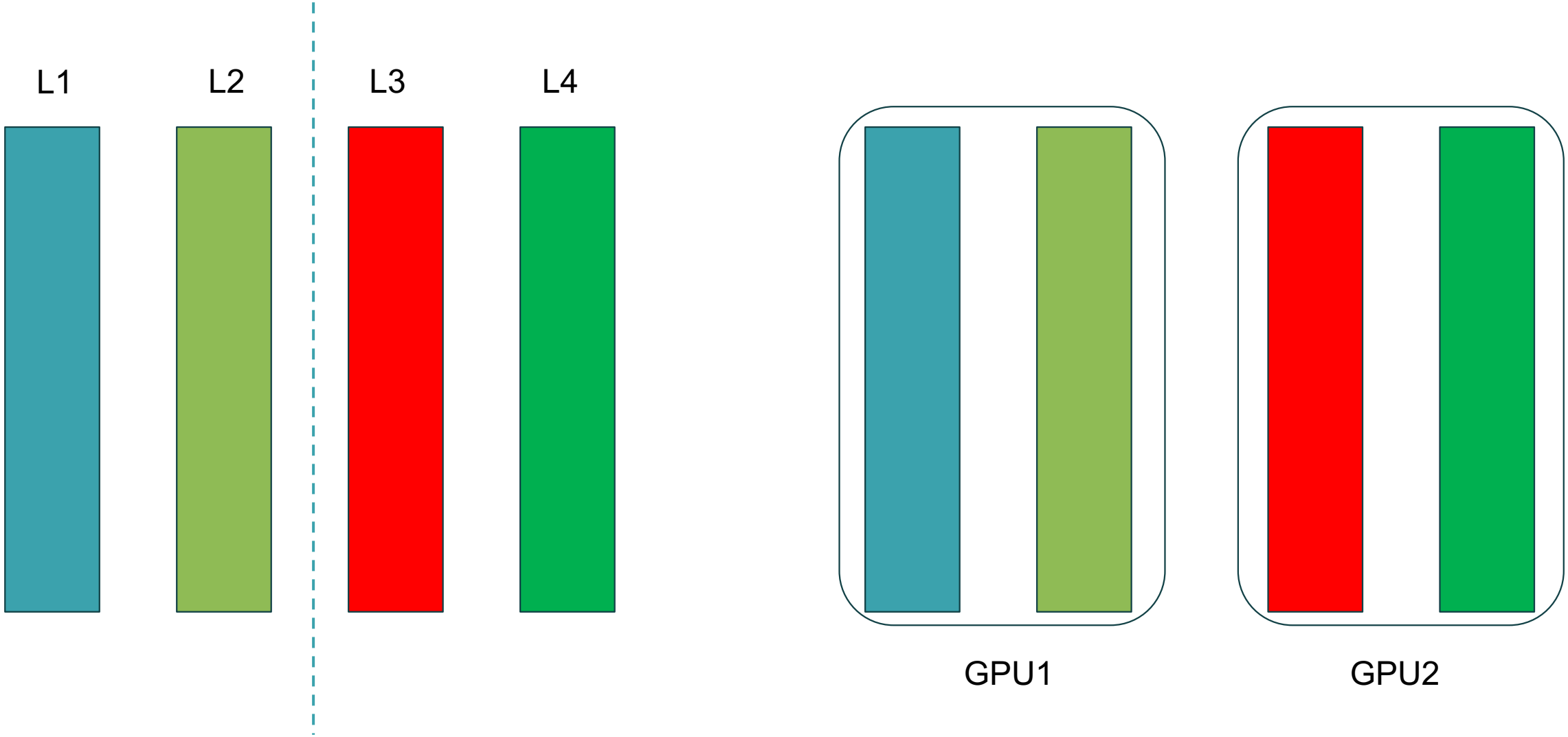
- Requires frequent AllReduce communication after every layer
- Intermediate outputs get AllReduced
- Tensor Parallel (TP) size is limited by the number of GPUs in a node (6 for Summit, 8 for Frontier)
- For TP > 6/8 the communication requires crossing node boundary through 25+25GB/s ethernet cable

Lessons Learnt From Tensor Parallelism

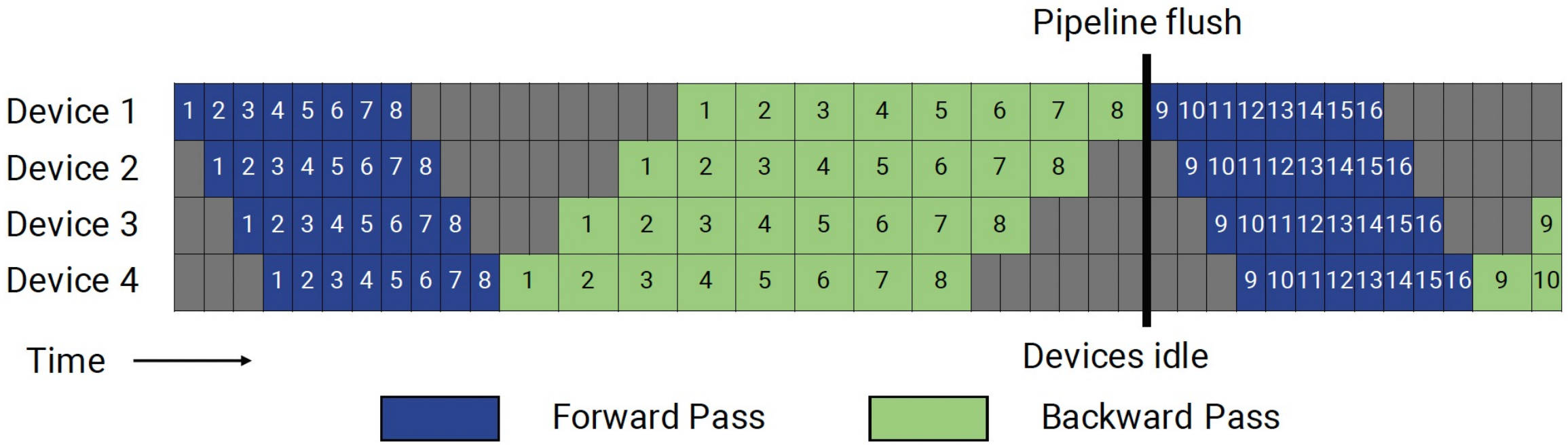




Pipeline Parallelism (PP = 2)

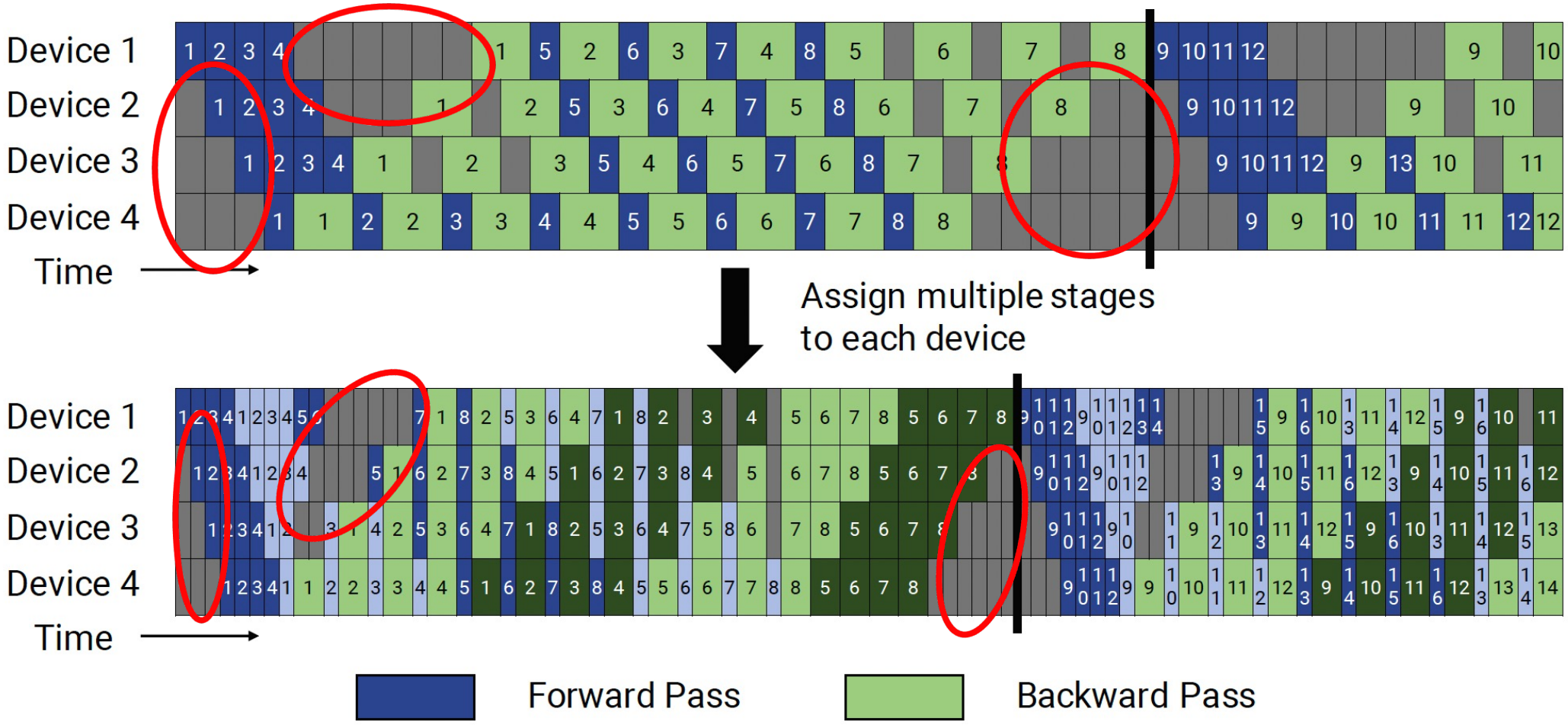


Pipeline Parallelism (Gpipe)



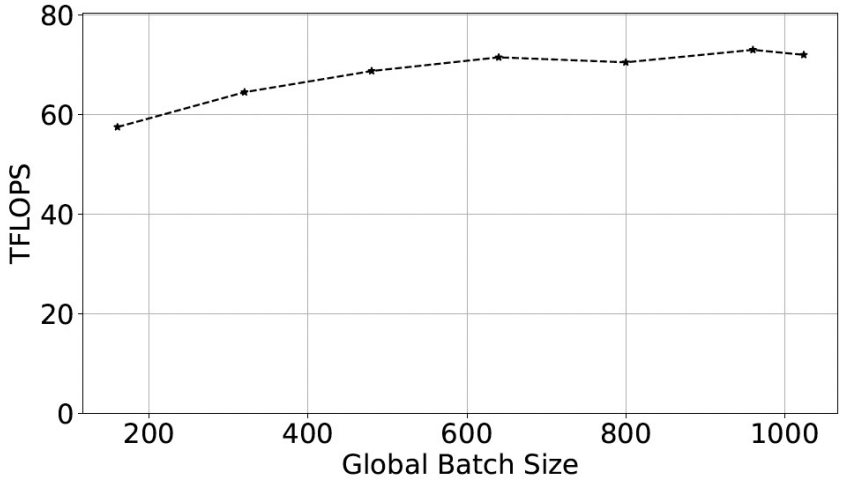
Pipeline Parallelism

Bubble size ~ (#Pipeline stages) / (#Microbatches)

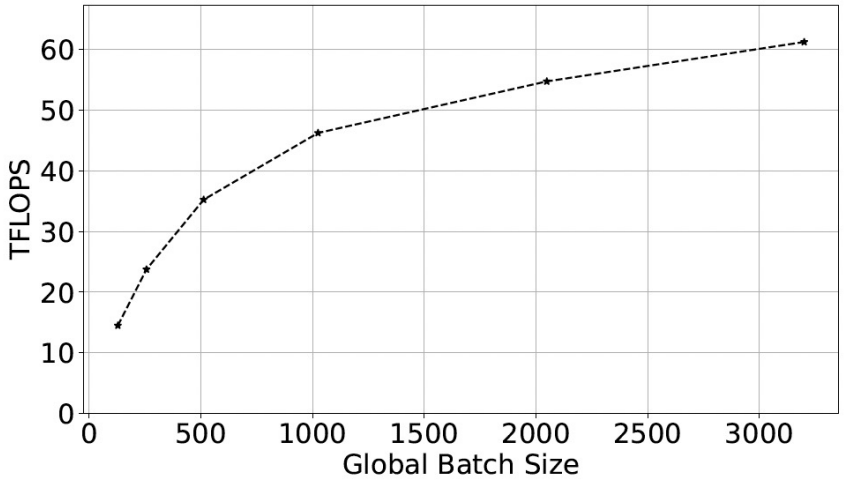


Pipeline Bubble vs #Microbatches

- Increasing the #Microbatches will reduce the bubble



(a) Throughput vs. global batch-size for 22B model.

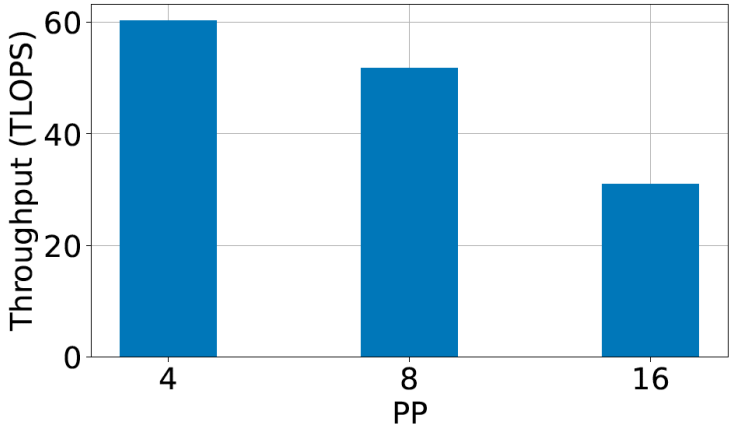


(b) Throughput vs global batch-size for 1T model.

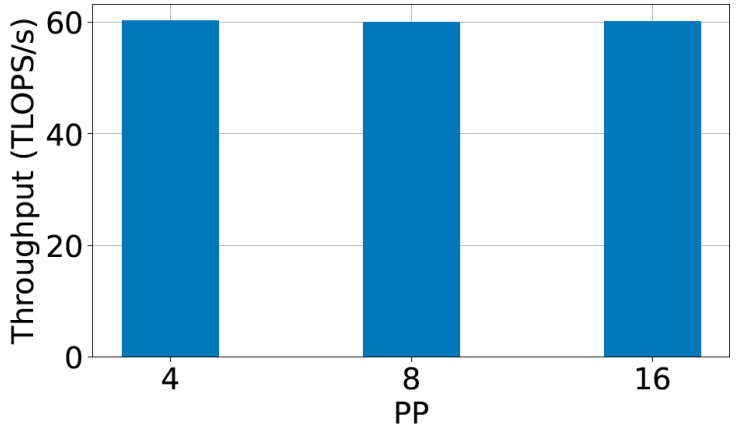
- But that will result in large global batch size, hurting the convergence

Bubble vs #pipeline-stages

- Reducing the #pipeline-stages reduces bubble



(a) Throughput vs. PP while keeping global batch size fixed at 128.

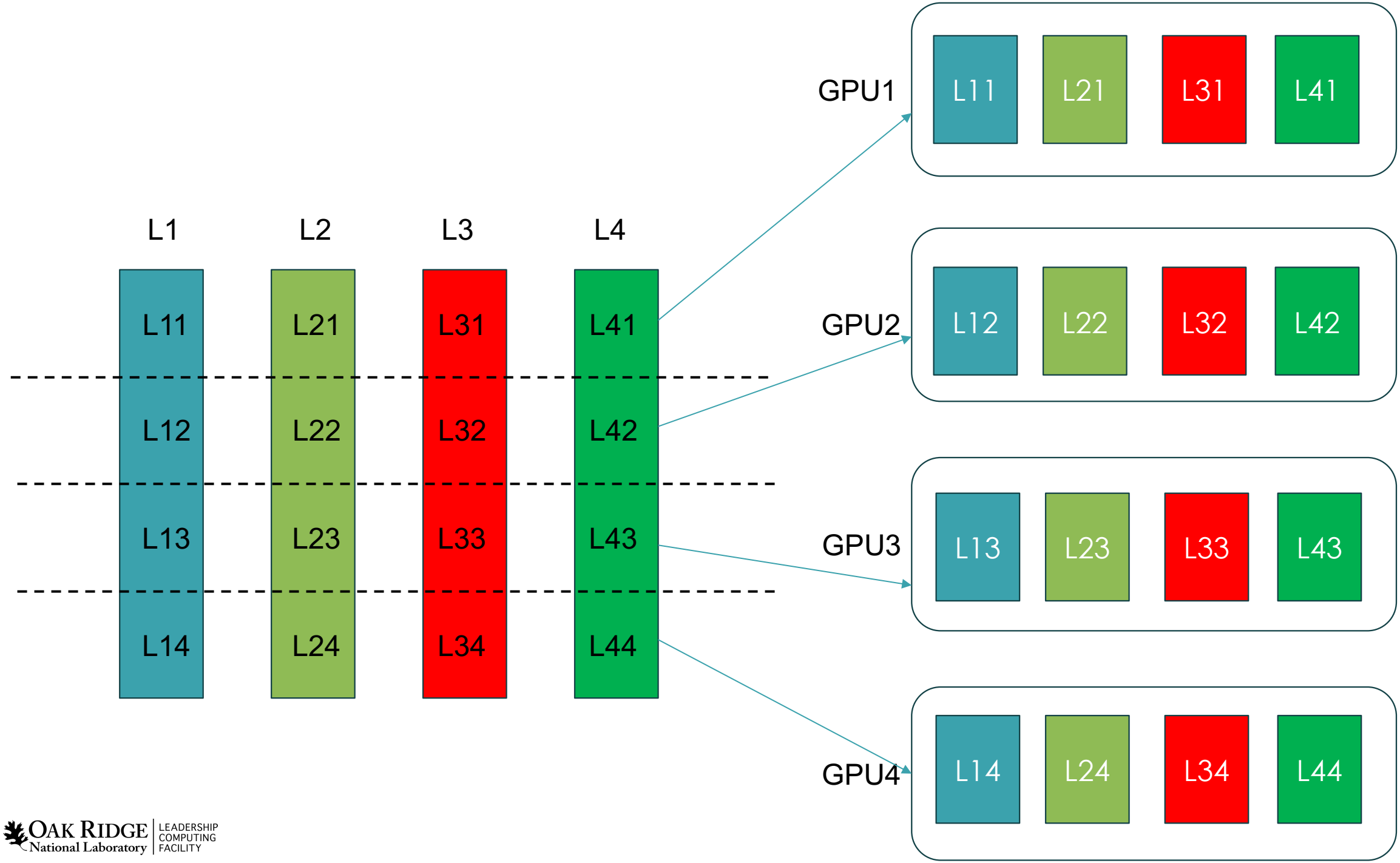


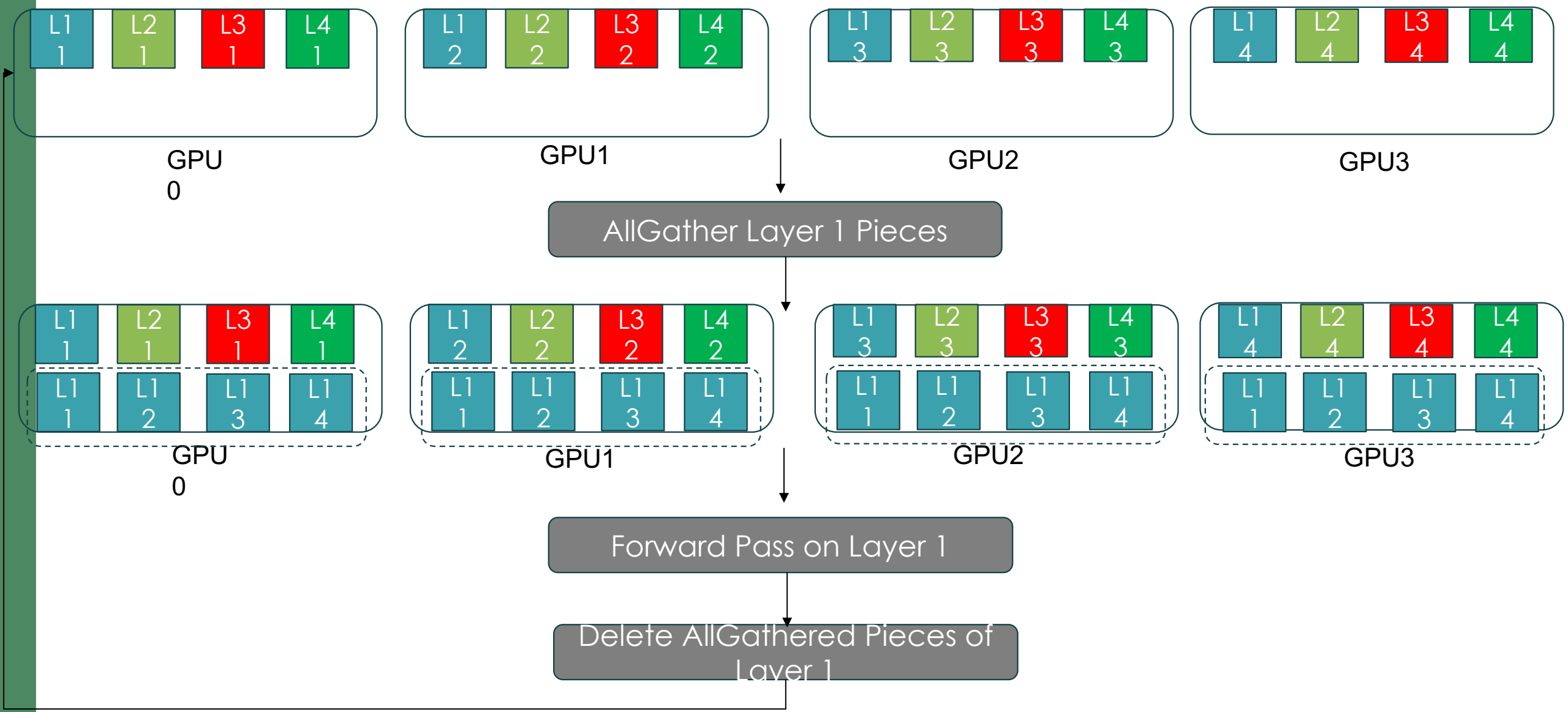
(b) Throughput vs. PP while scaling global batch size to keep the pipeline bubble ratio fixed.

- Then, we cannot use too many GPUs

Sharded Data Parallelism

- The model is too big to fit in a single GPU's memory
- One GPU has enough memory to fit a fraction of the model
- Slice the model with horizontal lines and place one horizontal slice in one GPU
- But, when a layer is being evaluated, each GPU should have the copy of that layer
- The GPU has enough memory to fit one full layer on top of its slice


L1
L2
L3
L4
L11
L21
L31
L41
L12
L22
L32
L42
L13
L23
L33
L43
L14
L24
L34
L44
GPU1
GPU2
GPU3
GPU4
L11
L21
L31
L41
L12
L22
L32
L42
L13
L23
L33
L43
L14
L24
L34
L44


L1 1
L2 1
L3 1
L4 1
L1 2
L2 2
L3 2
L4 2
L1 3
L2 3
L3 3
L4 3
L1 4
L2 4
L3 4
L4 4
GPU 0
GPU1
GPU2
GPU3
AllGather Layer 1 Pieces
L1 1
L1 2
L1 3
L1 4
Forward Pass on Layer 1
Delete AllGathered Pieces of Layer 1

Data vs. Sharded vs. Tensor vs. Pipeline Parallelism

	Data Parallelism	Sharded Data Parallelism	Tensor Parallelism	Pipeline Parallelism
Dimension of distribution	Full Model gets replicated	Horizontal Slices + one layer replicated	Horizontal Slices	Vertical slices
Model/Output	Output	Model	Output	Output
Communication frequency	Least frequent	Most frequent	Most frequent	Intermediate
Communication volume	Large volume	Small	Small	Small
Limitation	High-volume communication	Frequent low-volume communication	Frequent low-volume communication	Bubbles from insufficient overlap
Advantage	Necessary for consuming large data	Allows large model training	Allows large model training	Can hide latency with smaller bubbles

Outline

- Distributed training of LLMs
- Preparing a training software stack for Frontier
- Finding best distributed training strategies
- Training up to a Trillion parameter model on Frontier (**for a few iterations**)
- Training LLMs on OLCF resources

Preparing AI Software Stack for Frontier

- PyTorch – initially from source code and using ROCM compiler
- ROCM implementations of libraries like APEX, FlashAttention
- DeepSpeed without JIT compilation
- Utilize PyTorch's cpp_extention and cuda_extension for modifying cuda kernels to hip kernels

3D Parallelism using Megatron-DeepSpeed

- Megatron-DeepSpeed is a state-of-the-art training framework developed by Microsoft and NVIDIA
- It supports 3D+ parallelism (Tensor, Pipeline, Data, Sharded Data)
- We ported this Framework to Frontier through hipify and combining with ROCM specific packages

Porting Megatron-DeepSpeed to Frontier

- https://github.com/sajal-vt/Megatron-DeepSpeed-ORNL/tree/frontier-sd

1. Install deepspeed with prebuilt ops.
   DS_BUILD_CPU_ADAM=1 DS_BUILD_FUSED_ADAM=1
   DS_BUILD_FUSED_LAMB=1DS_BUILD_TRANSFORMER=1 DS_BUILD_AIO=1
   DS_BUILD_UTILS=1 pip install deepspeed

2. Following from the earlier step, commented out jit_fusion options in megatron/training.py
   Line 34: #from megatron.initialize import set_jit_fusion_options
   Line 100: #set_jit_fusion_options()

3. Add the following line to megatron/core/tensorparallel/layers.py – line 531, config.gradient_accumulation_fusion = False

4. In the launch script we had to set:
   export CUDA_DEVICE_MAX_CONNECTIONS=1

Outline

- Distributed training of LLMs
- Preparing a training software stack for Frontier
- Finding best distributed training strategies
- Training a Trillion parameter model on Frontier (**for a few iterations**)
- Training LLMs on OLCF resources

3D Parallelism

- A Combination of Tensor, Pipeline, and Data Parallelism
- Determine how many GPUs (world-size) you need to fit the model
- Factorize world-size into TP (tensor parallel size) and PP (pipeline parallel size)

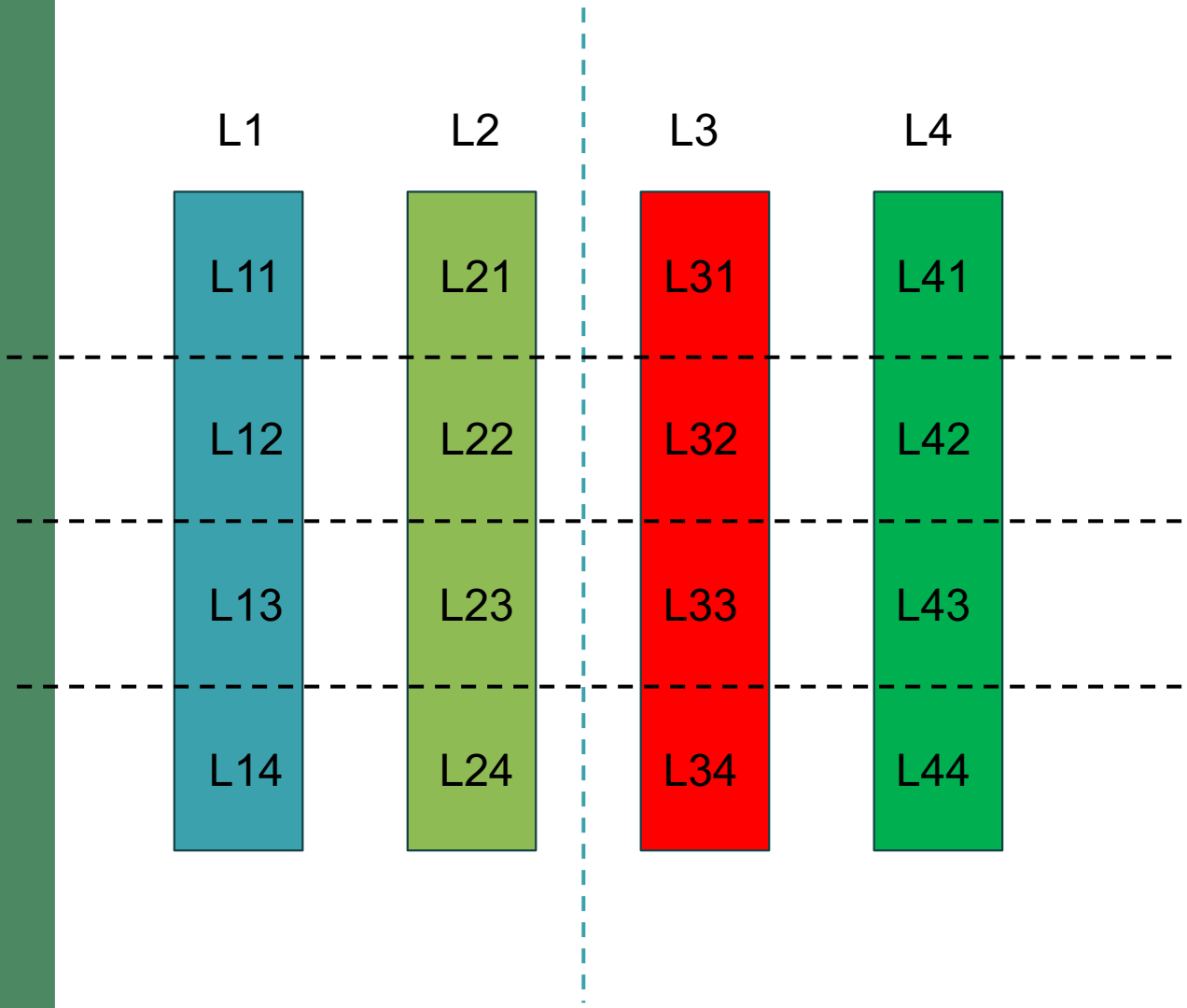
Distribution Strategy	Tunable Parameters
Tensor Parallelism	Tensor Parallel Size (TP)
Pipeline Parallelism	Pipeline Parallel Size (PP), #Microbatches (m)
Sharded Data Parallelism	ZeRO-1
Common	Micro Batch Size
Mixed Precision Training	FP16, BF16

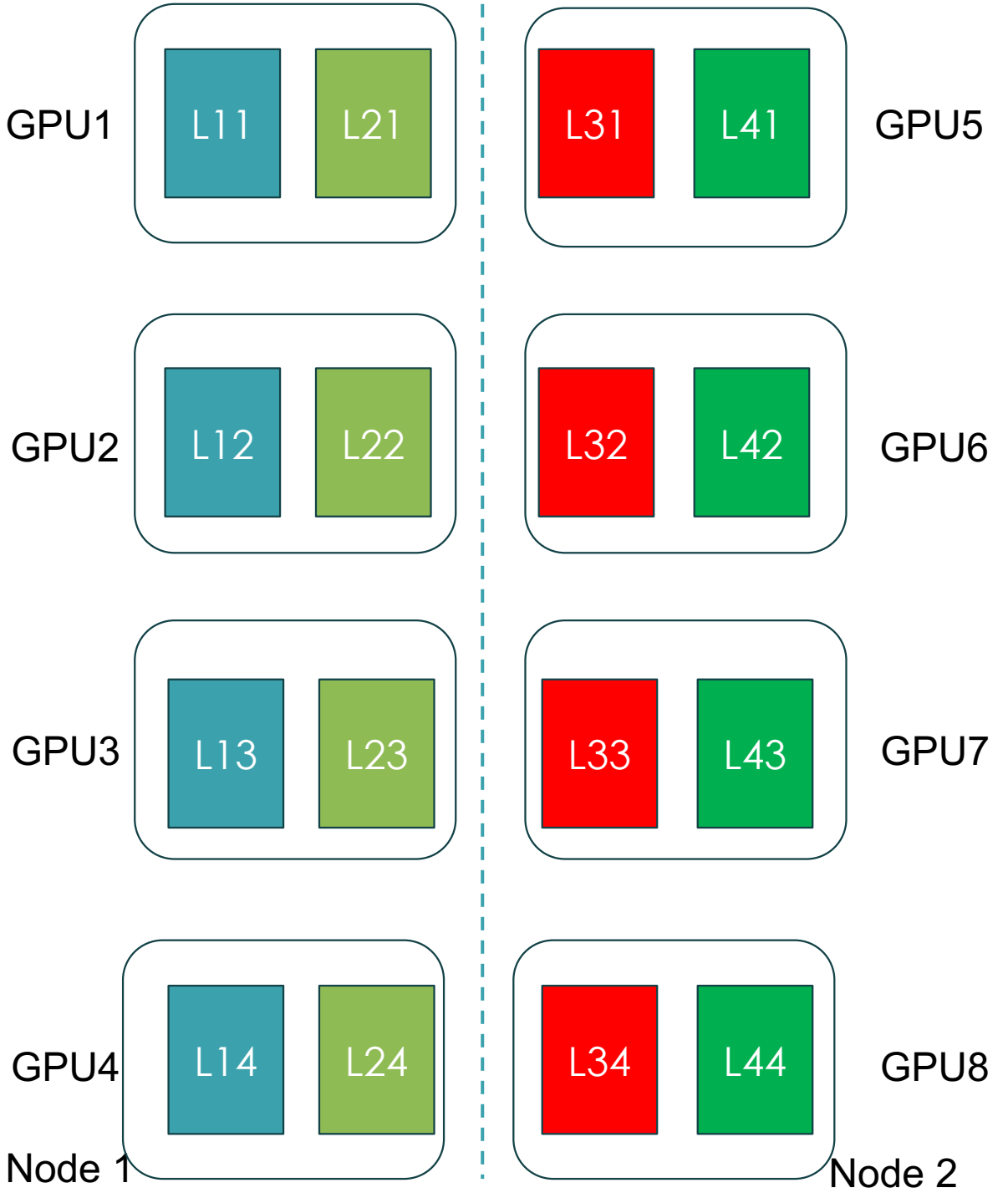
TABLE IV: Distribution Strategies and relevant tunable parameters

Hyperparameters	Range
Pipeline-parallel-size (PP)	$PP \in \{1, 2, 4, 8, 12, 16\}$
Tensor-parallel-size (TP)	$TP \in \{1, 2, 4, 8\}$
Micro-batch-size (MBS)	$MBS \in [4, 20]$
Gradient accumulation steps (GAS)	$GAS \in \{5, 10\}$
ZeRO-1 Optimizer	$ZeRO - 1 \in \{True, False\}$
Number of Nodes (NNODES)	$NNODES \in \{12, 16\}$

TABLE V: Hyperparameter Tuning for 175B Model

Hybrid (TP=4, PP=2)



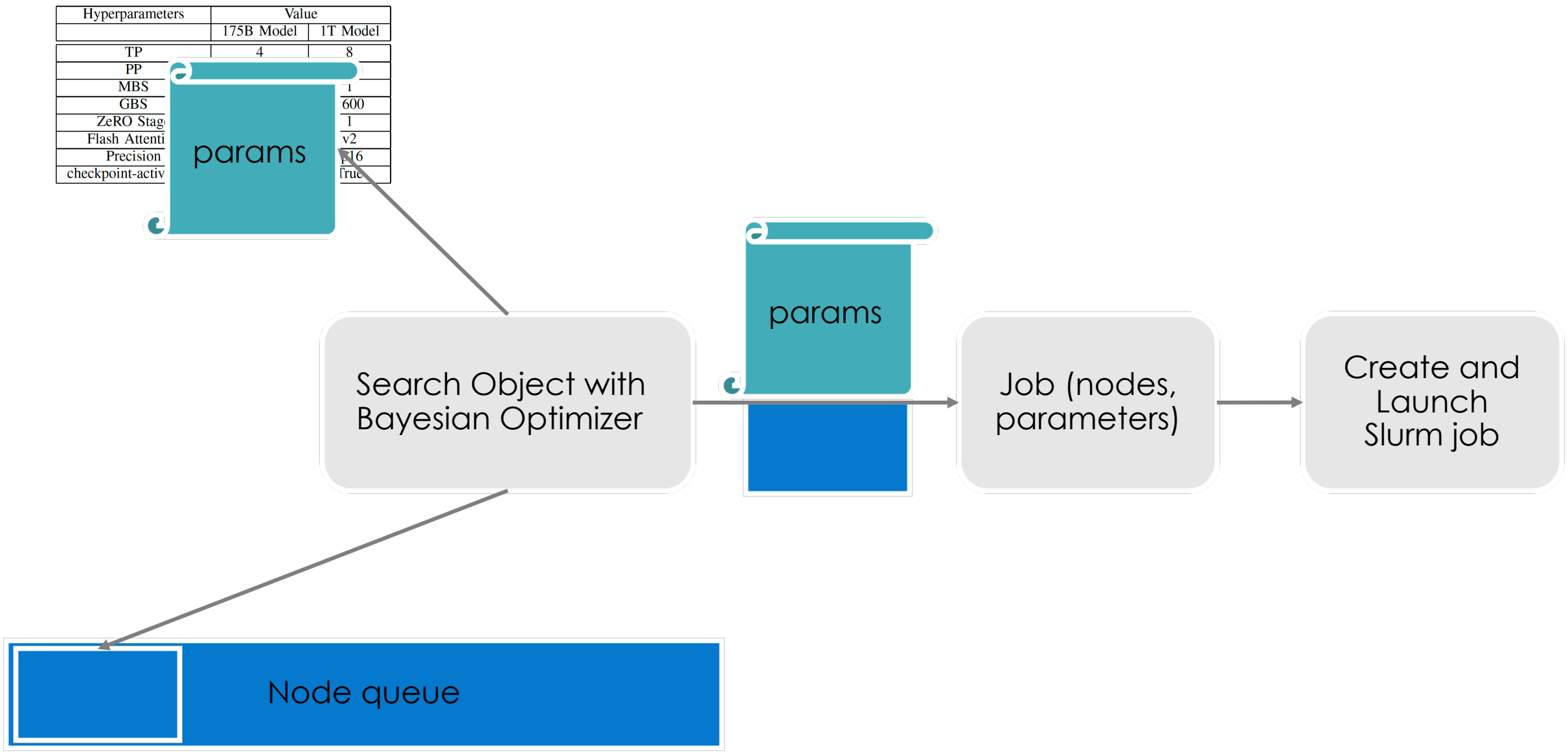


Best practices with parallelism paradigms

- Tensor Parallelism
  - Keep it within the node (TP < 8)
- Pipeline Parallelism
  - Use large number of micro-batches (But that can increase the global batch-size)
- Data Parallelism
  - Can't use too much data parallelism. A large global batch size will make the model divergence.

Hyperparameter Search using DeepHyper

Hyperparameters	Value	
	175B Model	1T Model
TP	4	8
PP		
MBS		1
GBS		600
ZeRO Stag		1
Flash Attenti		v2
Precision		A16
checkpoint-activ		True



Searching a 3D Parallelism Strategy using DeepHyper

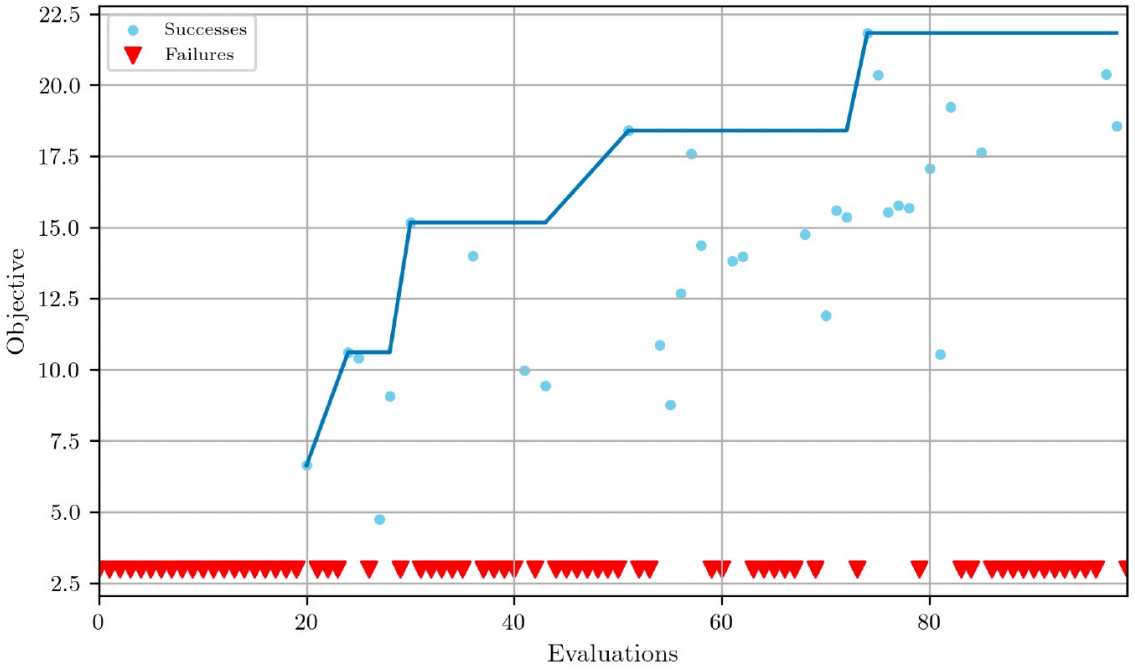


Fig. 9: Search Trajectory of DeepHyper search

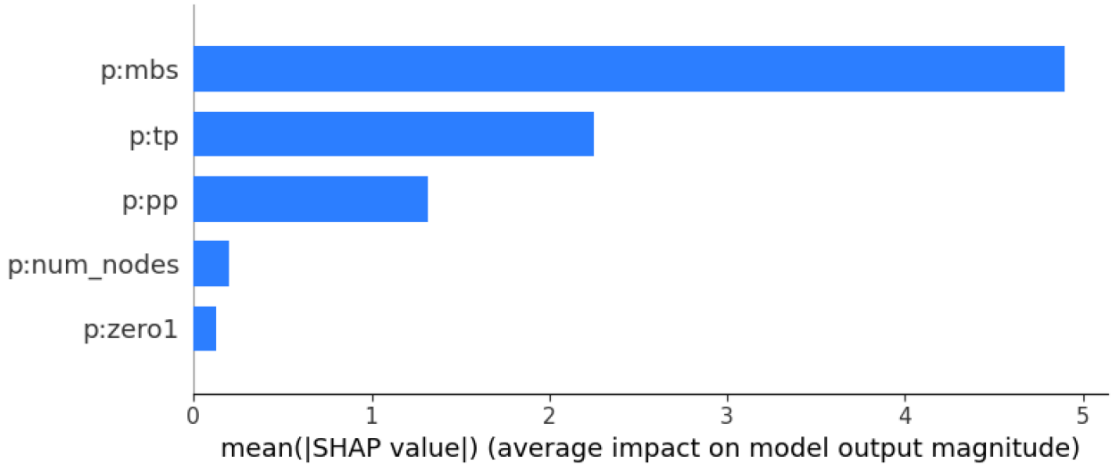


Fig. 10: Impact of various hyperparameters on training performance in terms of GPU throughput.

Outline

- Distributed training of LLMs
- Preparing a training software stack for Frontier
- Finding best distributed training strategies
- Training a Trillion parameter model on Frontier (**for a few iterations**)
- Training LLMs on OLCF resources

“Best” Strategy to Train 175B and 1T Models

Disclaimers:

1. We didn’t train any model till completion. We only trained for 10 iterations and less than 2 hours.
2. We don’t have any completely trained models

Hyperparameters	Value	
	175B Model	1T Model
TP	4	8
PP	16	64
MBS	1	1
GBS	640	1600
ZeRO Stage	1	1
Flash Attention	v2	v2
Precision	fp16	fp16
checkpoint-activations	True	True

TABLE VI: Best parameters for training a 175B model and a 1T model.

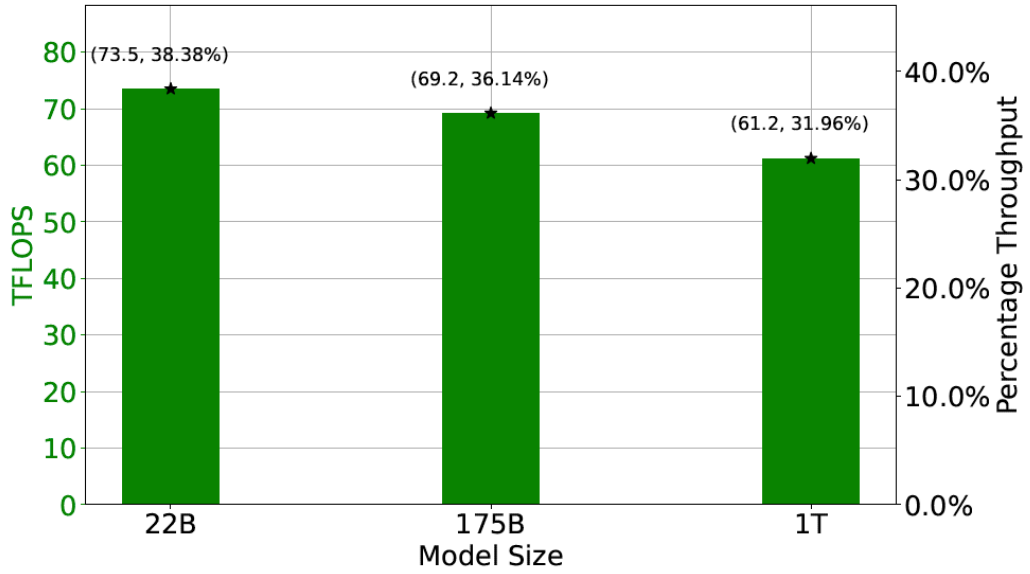
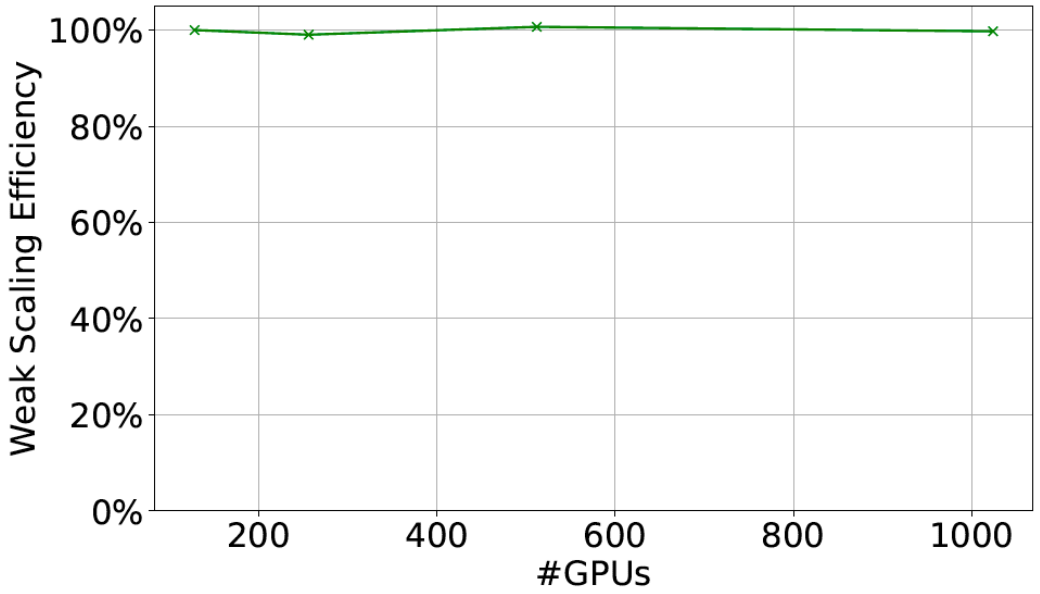
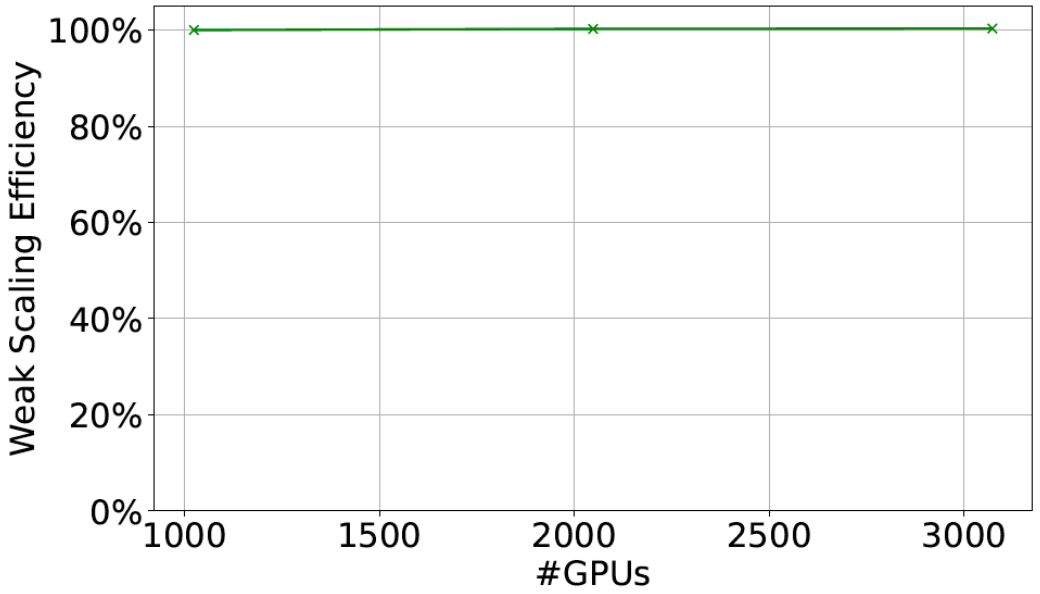


Fig. 11: MI250X Throughput for various model sizes. We report the hardware FLOPS, which are in close agreement with the model FLOPS.

Weak Scaling Performance



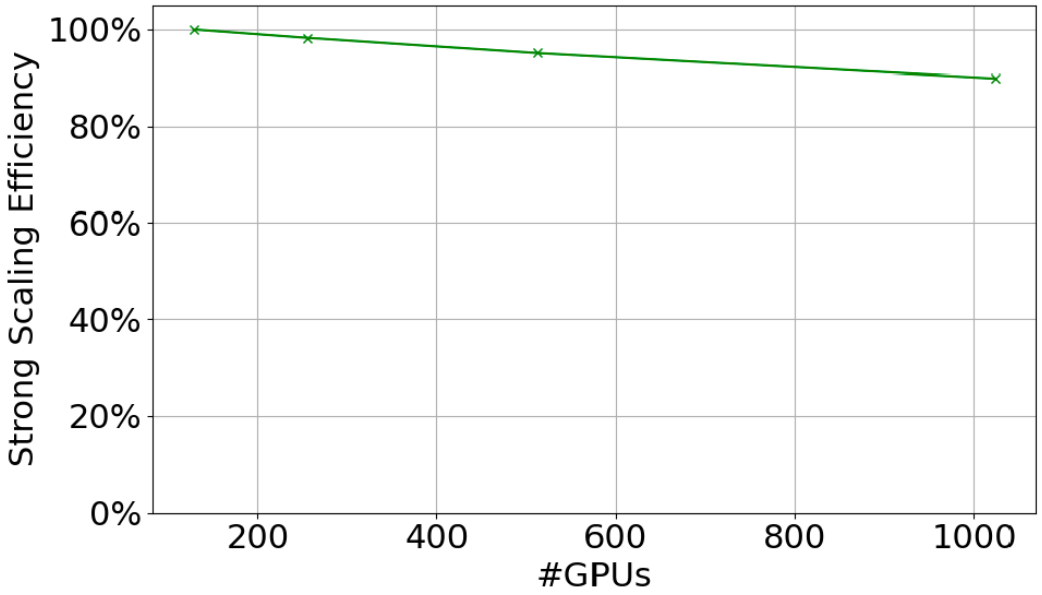
(a) Weak scaling of 175b model training by keeping per replica batch-size fixed at 640.



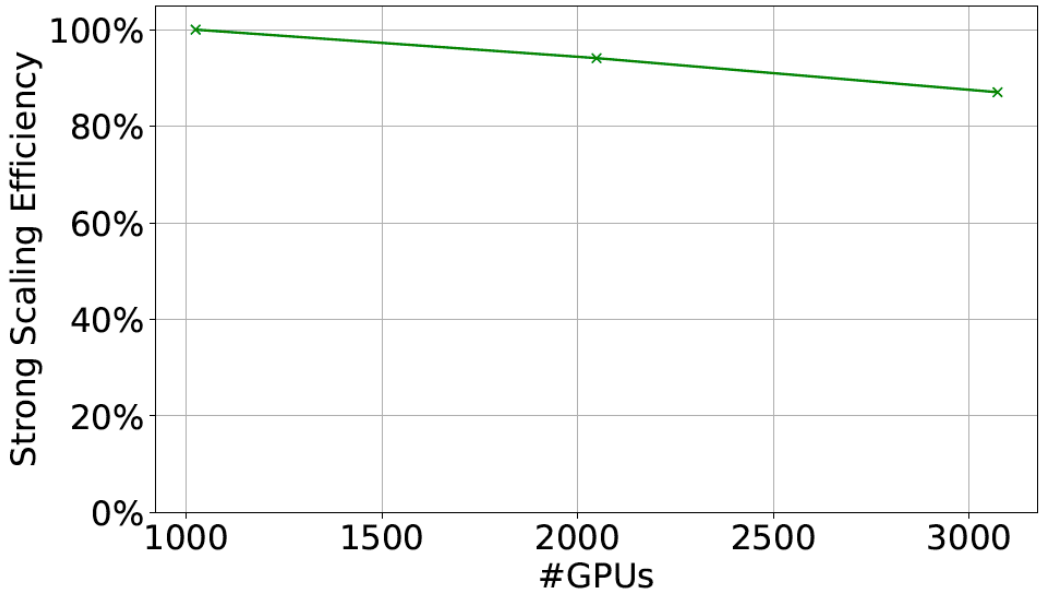
(b) Weak scaling of 1T model training by keeping per replica batch-size fixed at 1600.

Fig. 12: Weak scaling performance of 175b model and 1T model training.

Strong Scaling Performance



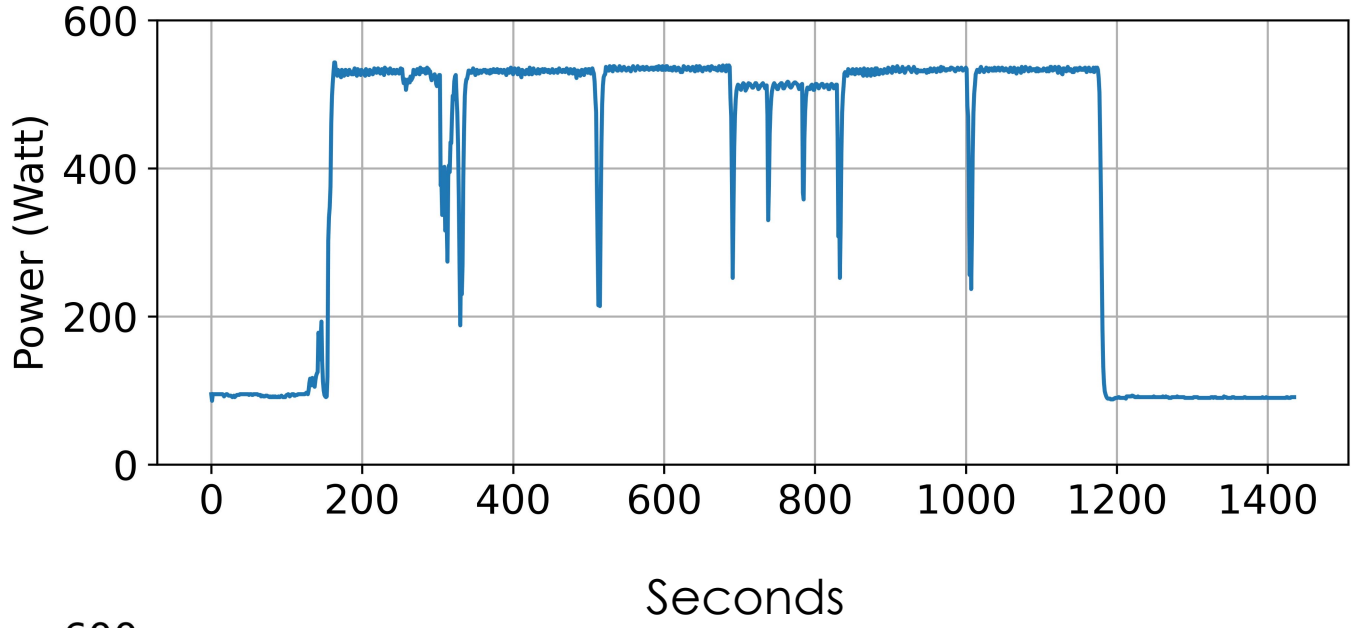
(a) Strong scaling of 175b model training by keeping the total batch size fixed at 8000. The strong scaling efficiency at 1024 GPUs is 89.93%.



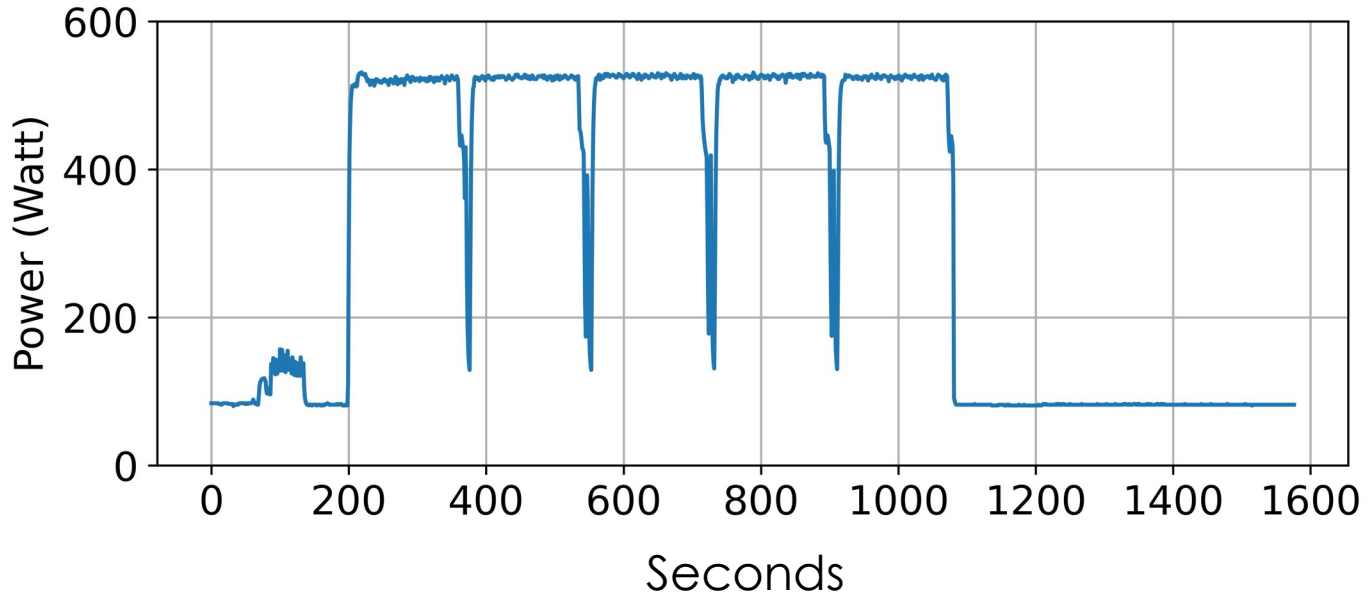
(b) Strong scaling of 1T model training by keeping a total batch-size fixed at 8016. The strong scaling efficiency at 3072 GPUs is 87.05%.

Fig. 13: Strong scaling performance of 175b model and 1T model training.

Power Consumption



22B Model training



175B Model training

Energy Budget against Model Size (~P^2)

- We used rocm-smi to measure GPU level energy consumption
- Idle GPU power is approximately 90 Watts.
- Taking iteration time, consumed tokens per iterations, average active power, and total number of MI250X GPU cards, we can estimate how much energy each model training will require.
- For 22B, 175B, and 1T parameter model, the estimated energy consumption is 284 Gigajoules, 17.65 Terajoules, and 662 Terajoules.

To Read About the Work...

Optimizing Distributed Training on Frontier for Large Language Models

Sajal Dash
Oak Ridge National Laboratory
dashs@ornl.gov

Isaac R Lyngaas
Oak Ridge National Laboratory
lyngaasir@ornl.gov

Junqi Yin
Oak Ridge National Laboratory
yinj@ornl.gov

Xiao Wang
Oak Ridge National Laboratory
wangx2@ornl.gov

Romain Egele
Université Paris-Saclay
romainegele@gmail.com

J. Austin Ellis
AMD
austin.ellis@amd.com

Matthias Maiterth
Oak Ridge National Laboratory
maiterthm@ornl.gov

Guojing Cong
Oak Ridge National Laboratory
congg@ornl.gov

Feiyi Wang
Oak Ridge National Laboratory
fwang2@ornl.gov

Prasanna Balaprakash
Oak Ridge National Laboratory
pbalapra@ornl.gov

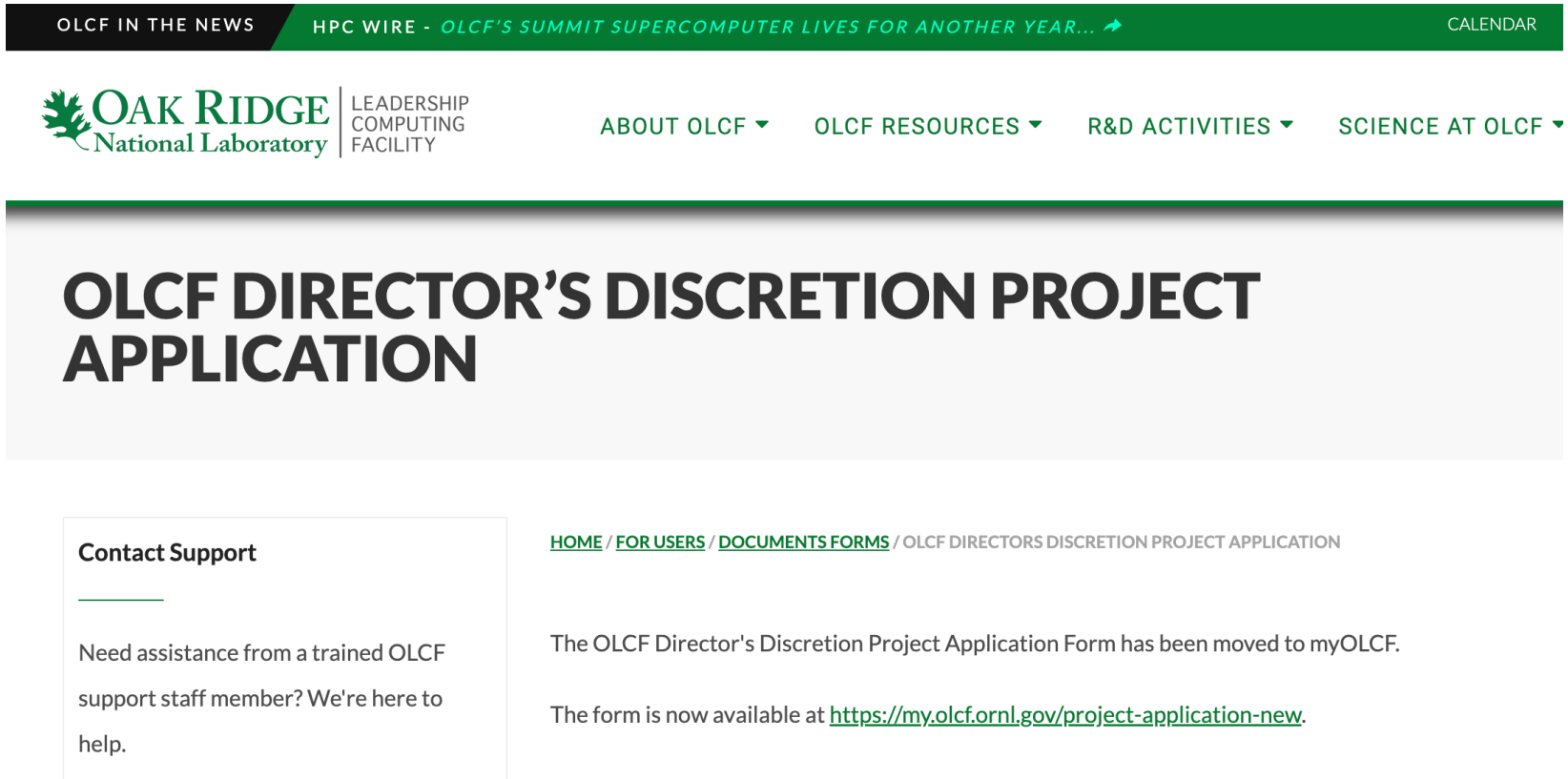
- Arxiv Link: https://arxiv.org/abs/2312.12705
- Accepted at ISC HPC, 2024

Outline

- Distributed training of LLMs
- Preparing a training software stack for Frontier
- Finding best distributed training strategies
- Training up to a Trillion parameter model on Frontier (**for a few iterations**)
- Training LLMs on OLCF resources

Getting OLCF Allocation

- https://www.olcf.ornl.gov/for-users/documents-forms/olcf-directors-discretion-project-application/



OLCF Tutorials on Distributed Training of Deep Learning Models

- Part I: Introduction and Data Parallel Training
  - https://www.olcf.ornl.gov/calendar/ai-for-science-at-scale-intro/
- Part II: Model Parallelism
  - https://www.olcf.ornl.gov/calendar/ai-training-series-ai-for-science-at-scale-part-2/

Hands On

- Training Large LLMs
  - Train a 22B model on 2 nodes
  - Train a 175B model on 16 nodes
  - Train a 1T model on 128 nodes
- Finding the best training strategies using DeepHyper
  - Find the best strategy for training a 22B model

Training Large LLMs

- git clone git@github.com:sajal-vt/Megatron-DeepSpeed-ORNL.git
- git fetch
- git branch –v –a
- git switch FA2
- sbatch –reservation=ai launch_gpt22b_bf16.slurm
- sbatch --reservation=ai launch_gpt175b_bf16.slurm
- sbatch --reservation=ai launch_gpt1T_bf16.slurm

Finding the best training strategies using DeepHyper

- git clone git@github.com:sajal-vt/Megatron-DeepSpeed-ORNL.git
- git fetch
- git branch –v –a
- git switch frontier-sd
- sbatch --reservation=ai launch_dh.frontier