MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs

Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, Xin Liu

ByteDance, Peking University

NSDI 2024 (Experience Paper)

Outline

- Background & Motivation
- Efficient Training at Scale
  - Network Performance Tuning
- Fault Tolerance and Diagnosis
- Training Troubleshooting
- Experience
- Discussion

Key Questions to Answer

- Why 10K+ GPUs?
- What is communicated?
- What does MegaScale Optimize?
- What Breaks in Production?
- How Should We Critique It?

Outline

- **Background & Motivation**
- Efficient Training at Scale
  - Network Performance Tuning
- Fault Tolerance and Diagnosis
- Training Troubleshooting
- Experience
- Discussion

LLM Training at Scale – The Core Pressure

- **Scale**: Scaling laws push model size and data size upward. Training frontier LLMs requires enormous GPU-hours, so cluster-scale training becomes economically necessary
- **Why Networking Matters:** Parallel training creates frequent cross-GPU communication. Synchronization means the slowest rank can delay the entire iteration.
- **Why Production Differs?** A weeks-long run sees failures, stragglers, configuration changes, link flaps, and software anomalies. Sustained efficiency is harder than peak efficiency.

Model FLOPs Utilization (MFU)

- **Definition**
  - MFU ≈ useful model FLOPs per second divided by theoretical peak GPU FLOPs.
- **Why Low MFU?**
  - Communication stalls
  - Pipeline bubbles
  - Kernel launch/memory overhead
  - Data loading stalls
  - Stragglers
  - Fault recovery downtime

Model FLOPs Utilization (MFU)

- MegaScale reports **55.2% MFU on a 175B model using 12,288 GPUs**, 1.34× higher than Megatron-LM in their comparison
  - **Full-stack co-design**
  - **In-depth observability**
- MegaScale is an experience paper, not a single clean algorithm

Distributed LLM training: Four Parallelisms

- **Data Parallelism:** Replicate model; shard/aggregate optimizer states, parameters, and gradients.
- **Pipeline Parallelism:** Split layers into stages; send activations and gradients between stages.
- **Tensor Parallelism:** Partition large matrix operations; use all-gather/reduce-scatter
- **Sequence Parallelism:** Shard sequence dimension for memory-heavy non-GEMM operators.

Distributed LLM training: Four Parallelisms

- **Networking View:** each parallelism generates a different communication operator, dependency, and locality preference.

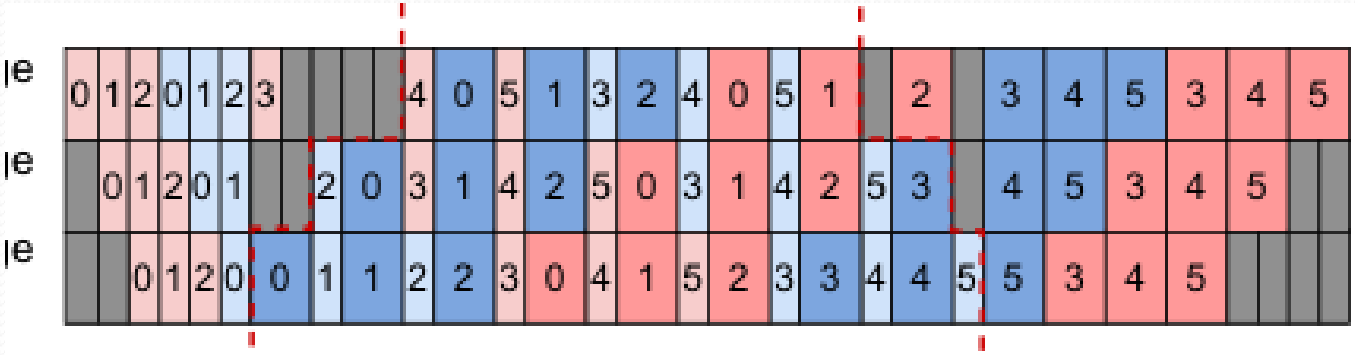
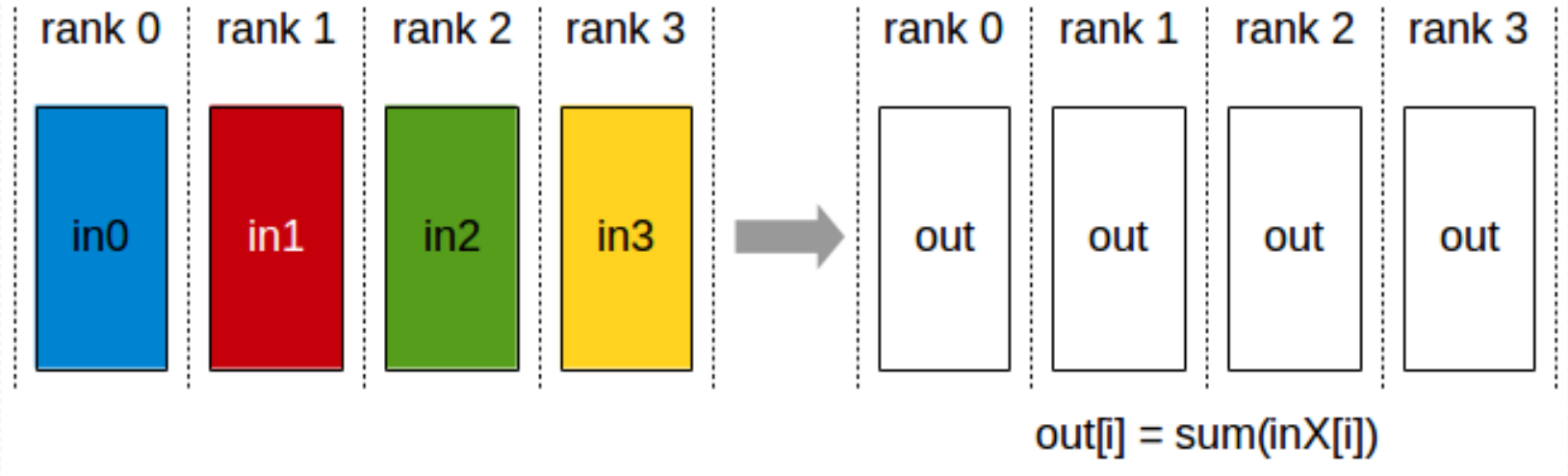


Figure 2: Interleaved 1F1B pipeline.

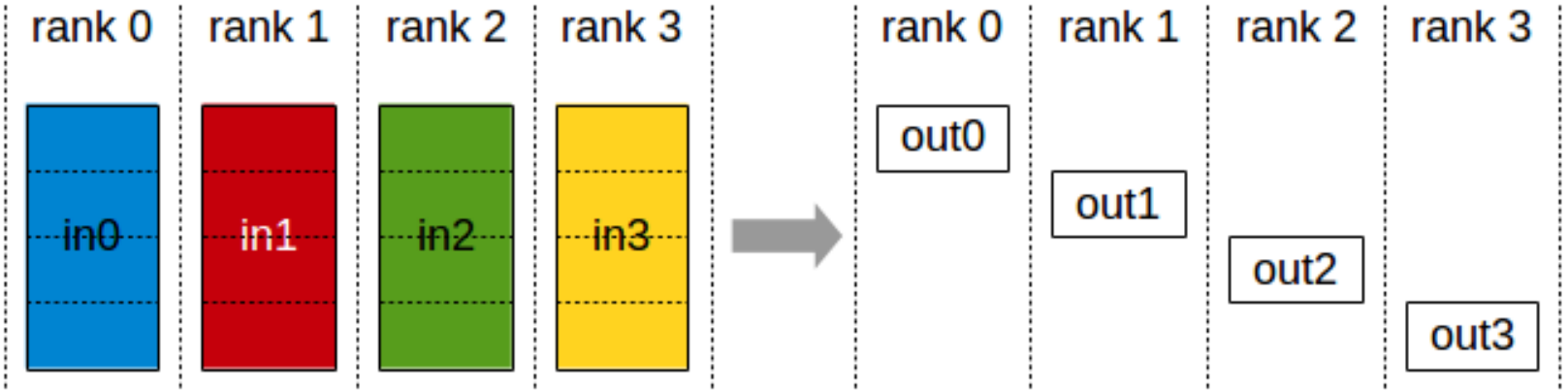
Collective Communication: Network Workload is Structured

- **All-gather:** Each participant gathers shards from others. Used for parameters/activations depending on parallelism.
- **Reduce-scatter:** Participants reduce values and keep only a shard. Used for gradients and tensor-parallel outputs.
- **Point-to-point:** Pipeline stages send activations/gradients. Dependencies create order constraints and bubbles.

All-gather


rank 0
rank 1
rank 2
rank 3
in0
in1
in2
in3
rank 0
rank 1
rank 2
rank 3
out
out
out
out
out[i] = sum(inX[i])

Reduce-scatter



Outline

- Background & Motivation
- **Efficient Training at Scale**
  - Network Performance Tuning
- Fault Tolerance and Diagnosis
- Training Troubleshooting
- Experience
- Discussion

Efficient Training at Scale

- **Algorithmic and Runtime Optimizations**
  - Parallel Transformer Block
  - Sliding Window Attention
  - LAMB Optimizer
- **Communication Overlap**
  - Overlapping in Data Parallelism
  - Overlapping in Pipeline Parallelism
  - Overlapping in Tensor/Sequence Parallelism

Efficient Training at Scale

- **Efficient Operators**
- **Data Pipeline**
  - Asynchronous data preprocessing
  - Redundant dataloader elimination
- **Collective Communication Group Initialization**
- ***Network Performance Tuning***

Parallel Transformer Block

- Reformulates the transformer block so attention and MLP computations can proceed in parallel.
- Goal: Reduce computation time without degrading quality (guided by previous work).

$$y = x + \mathrm{MLP}(\mathrm{LN}(x + \mathrm{Attention}(\mathrm{LN}(x)))) \qquad (1)$$

into

$$y = x + \mathrm{MLP}(\mathrm{LN}(x)) + \mathrm{Attention}(\mathrm{LN}(x)) \qquad (2)$$

Sliding Window Attention

- Sliding window attention is a sparse attention mechanism that employs a fixed size window surrounding each token in the input sequence.
- The computation complexity is $O(s \times w)$, where s is the input sequence length and w is the fixed window size.
- More efficient than the full self-attention: $O(s \times s)$, given that $w \ll s$.
- Empirical experience show this mechanism is useful.

LAMB Optimizer

- The **LAMB optimizer** has been demonstrated to enable the scaling of BERT's training batch size to 64K without compromising accuracy.
- Enables larger batch size in their experiments.
- Larger batch size reduces pipeline bubbles under interleaved pipeline parallelism.

Overlapping in Data Parallelism

- The *all-gather* operation is triggered prior to the forward pass of a model chunk, and the *reduce-scatter* operation commences after its backward pass.
  - Prefetch the first all-gather at the beginning of each iteration so it overlaps with data loading.
  - Launch higher-priority communication earlier.

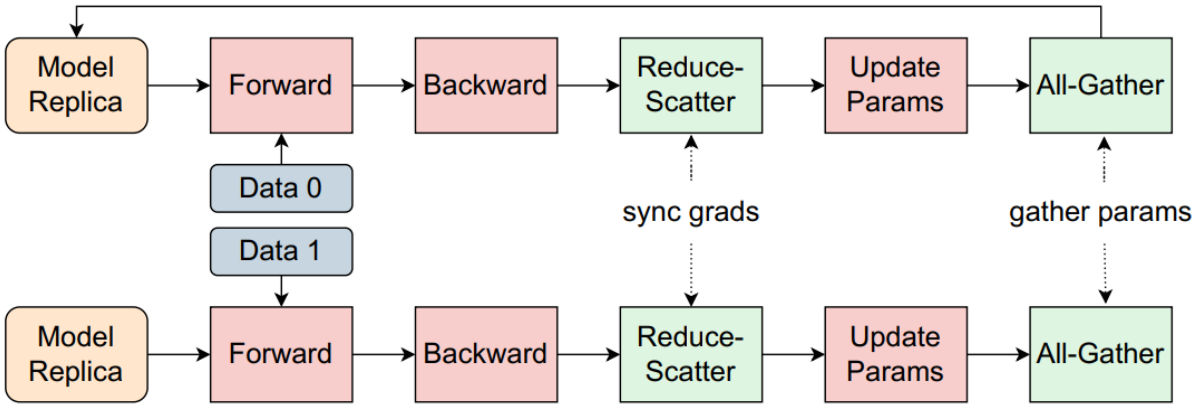


Figure 1: Data parallel training with Zero Redundancy Optimizer.

Overlapping in Pipeline Parallelism

- Forward/backward computation is often independent of adjacent send/receive operations.
- Launch send and receive asynchronously to overlap with computation.

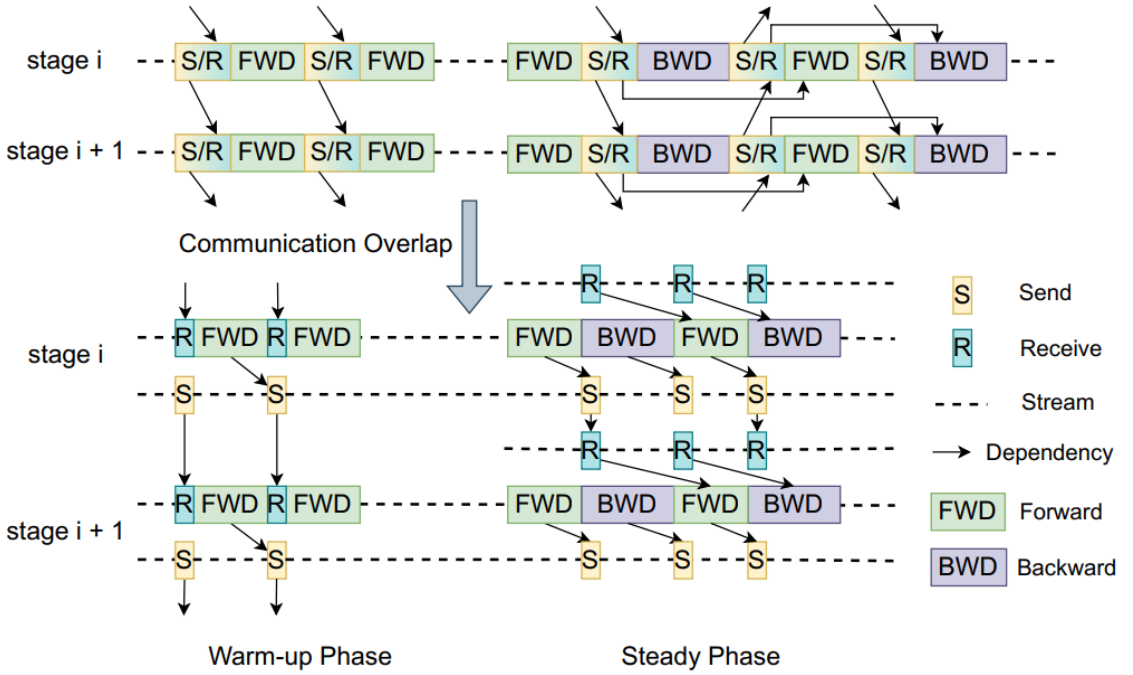
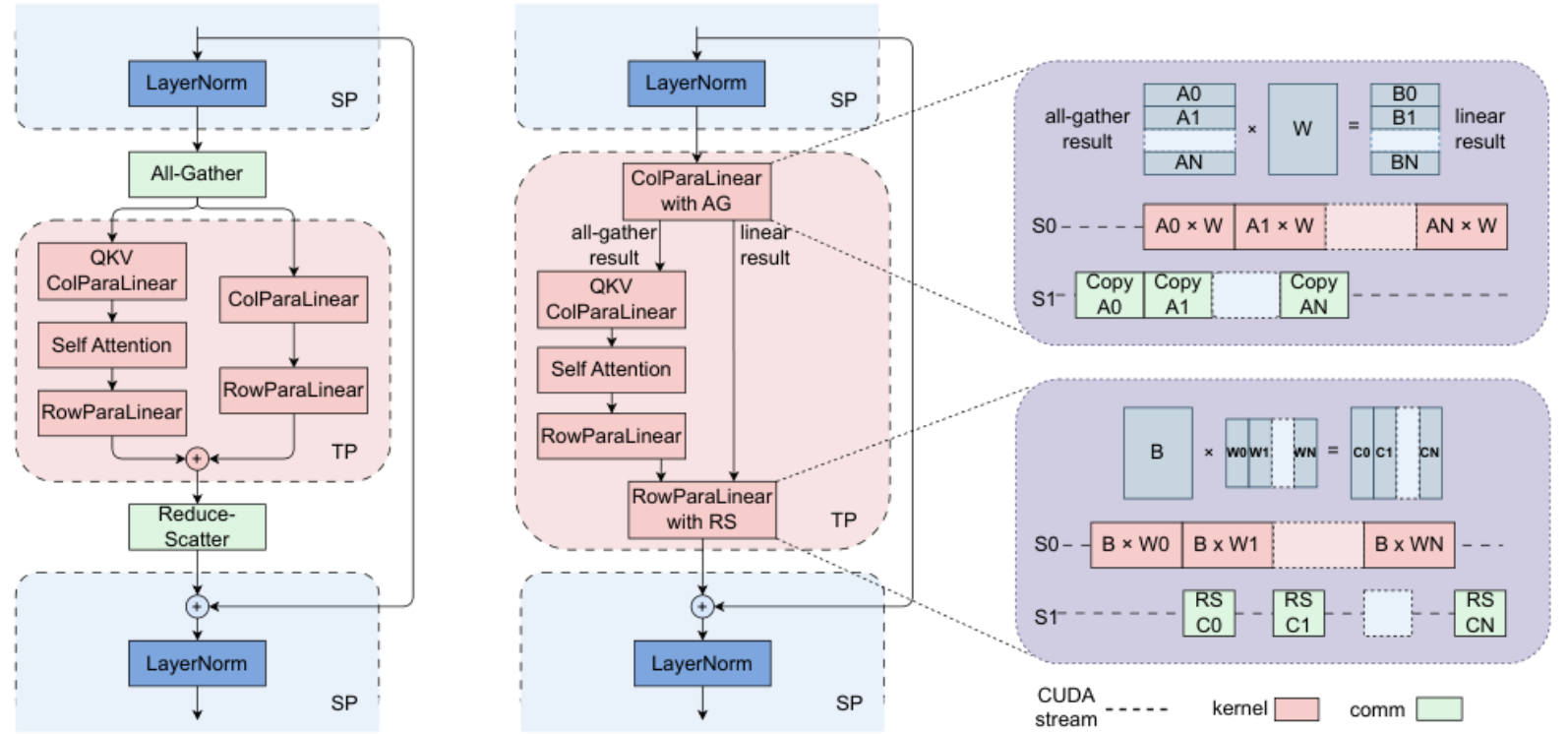


Figure 4: Overlapping communication in pipeline parallelism.

Overlapping in Tensor/Sequence Parallelism

- Use FFN path because GEMM kernels are larger and can better hide communication.
- Break GEMM into chunks, then pipeline compute and communication kernels.
- Same principle applies to backward pass.
- ***Lesson: overlap requires operator-level redesign, not only transport-level tuning.***

Overlapping in Tensor/Sequence Parallelism



(a) PTB with SP and TP (b) Fuse communication into Linears (c) Overlap communication with GEMM

Figure 3: Overlapping communication in tensor parallelism (TP) and sequence parallelism (SP) with parallel transformer block (PTB).

Efficient Operators

- Besides the optimization for GEMM operators in MegatronLM
  - Adopt **FlashAttention-2**
  - **Kernel Fusion** in LayerNorm and GeLU operator
    - Reduce the overhead associated with launching
    - Multiple kernels and aid in optimizing memory access patterns

Asynchronous Data Preprocessing

- Data preprocessing is not on the critical path (consume GPU).
- As a result, while the GPU workers are synchronizing gradients at the end of each training step, the data preprocessing for the subsequent step can start
- **Goal: Hides the preprocessing overhead**

Redundant Dataloader Elimination

- Workers in the same machine are in the same tensor-parallel group and use identical input.
- Use one loader per machine, write to shared memory, then copy to each GPU.

Collective Communication Group Initialization

- Naive *torch.distributed* group initialization **took about 1047 seconds on 2048 GPUs** in their measurement.
- TCPStore is single-threaded and blocking for synchronization. Replacing it with Redis **reduced time to 361 seconds on 2048 GPUs**.
- Incautious global barriers. Careful group initialization order reduces global barrier complexity from $O(n^2)$ to $O(n)$, yielding **<5s on 2048 GPUs** and **<30s on >10K GPUs**.

Outline

- Background & Motivation
- Efficient Training at Scale
  - **Network Performance Tuning**
- Fault Tolerance and Diagnosis
- Training Troubleshooting
- Experience
- Discussion

Network Performance Tuning

- Network Topology
- Reducing ECMP Hashing Conflicts.
- Congestion Control
- Retransmit Timeout Setting

Network Topology

- **Switch Fabric:** Three-layer CLOS-like topology built using Broadcom Tomahawk 4 switches
- **Bandwidth Details:** Each Tomahawk 4 chip provides 25.6 Tbps with 64 × 400G ports. Downlink/uplink ratio (over-subscribe ratio) is configured as 1:1
- **Observation:**
  - Small diameter and high bisection bandwidth are necessary but not sufficient
  - Synchronized collectives still expose hashing and congestion issues

Reducing ECMP Hashing Conflicts

- **Split Downlinks:** At the ToR level, split one 400G downlink into two 200G downlinks. This reduces conflict probability because uplinks are larger than downlinks.
- **Multi-rail Wiring:** Each server has eight 200G NICs connected to 8 different ToR switches. Servers sharing the same ToR set can form a locality group.
- **Placement Strategy:** Schedule data-intensive nodes under the same ToR set to reduce hop count and ECMP conflict probability.

Congestion Control

- **Problem Observed:** Default DCQCN led to congestion and elevated PFC under all-to-all communication at scale.
- **Risk:** Excessive PFC may cause head-of-line blocking, reducing network throughput for synchronized training.
- **MegaScale**: Combine Swift-style RTT measurement with DCQCN-style fast ECN response.

Retransmit Timeout Setting

- Parameters in NCCL can be set to control **retransmit timer** and **retry count**.
- Tune these parameters for fast recovery under link flapping.
- Enable the *adap_retrans* feature on the NIC to further reduce the latency.
  - *adap_retrans* helps the network adapt its timeout **dynamically** instead of triggering lengthy hard connection drops.

Outline

- Background & Motivation
- Efficient Training at Scale
  - Network Performance Tuning
- **Fault Tolerance and Diagnosis**
- Training Troubleshooting
- Experience
- Discussion

Fault Tolerance and Diagnosis

- Robust Training Workflow
- Data Collection and Analysis
- Diagnostic Tests
- Fast Checkpointing and Recovery

Robust Training Workflow

- **Driver/Executor:** One executor manages one node; a daemon periodically sends heartbeats to the driver.
- **Recovery Trigger:** Missing heartbeat or abnormal status suspends training and triggers diagnostics.
- **Resume Strategy:** Problem nodes are replaced and training resumes from the latest checkpoint.

Data Collection and Analysis

- The heartbeat messages includes the basic information of the executor, such as the IP address, the Pod name, and hardware information
- The current status of the training processes is reported, enabling the driver to promptly detect any explicit anomalies
  - stdout/stderr logs of training processes
  - RDMA traffic metrics
- A monitoring system with precision reaching the millisecond level

Diagnostic Tests

- Intra-host network tests.
  - The Loopback test measures the loopback bandwidth from all RDMA NICs (RNICs) to various intra-host endpoints, including memory nodes and GPUs.
  - RNIC-to-RNIC test examines the connectivity and bandwidth performance between different RNICs on the same host.
- NCCL tests.
  - First, *all-to-all* test among the GPUs in a single machine
  - Then, each node also conducts an all-reduce test with neighboring machines under the same ToR switch

Fast Checkpointing and Recovery

- **Two-stage checkpoint write**
  - Stage 1: each GPU worker writes on-chip states to host memory and resumes training
  - Stage 2: a background process transfers state to HDFS asynchronously
- **Optimized checkpoint read**
  - A single worker in a sharing group reads a state partition from HDFS, then broadcasts it to other workers that need the same data.

Outline

- Background & Motivation
- Efficient Training at Scale
  - Network Performance Tuning
- Fault Tolerance and Diagnosis
- **Training Troubleshooting**
- Experience
- Discussion

Training Troubleshooting

- Performance Diagnosis with CUDA Event Monitor
- 3D Parallel Training Visualization

Performance Diagnosis with CUDA Event Monitor

- Inconsistent MFU observed in large-scale training.
- Different colors denote distinct executions of the same training job

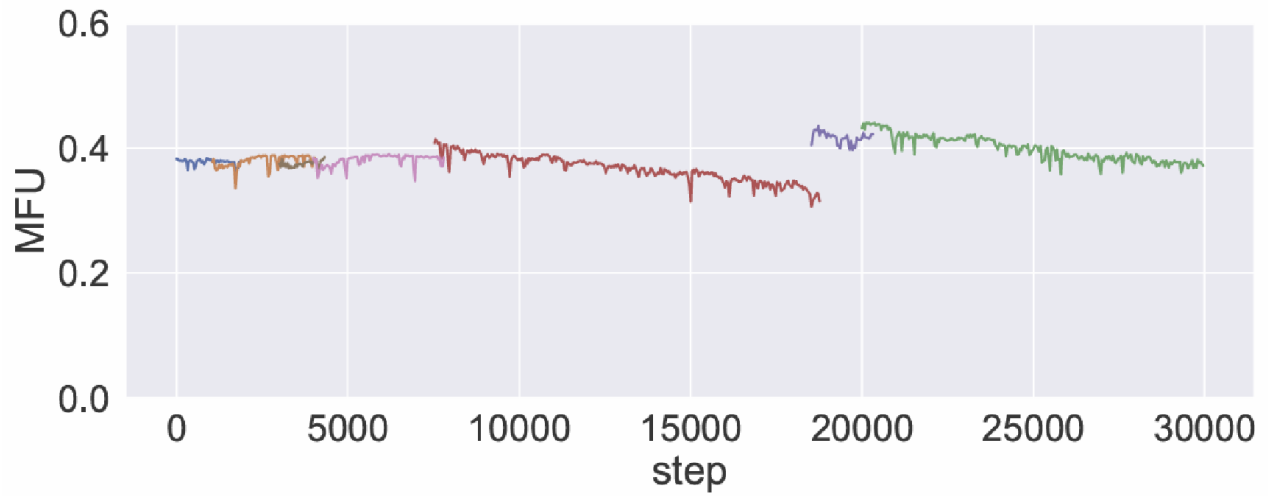


Figure 6: Inconsistent MFU observed in large-scale training. Different colors denote distinct executions of the same training job.

Performance Diagnosis with CUDA Event Monitor

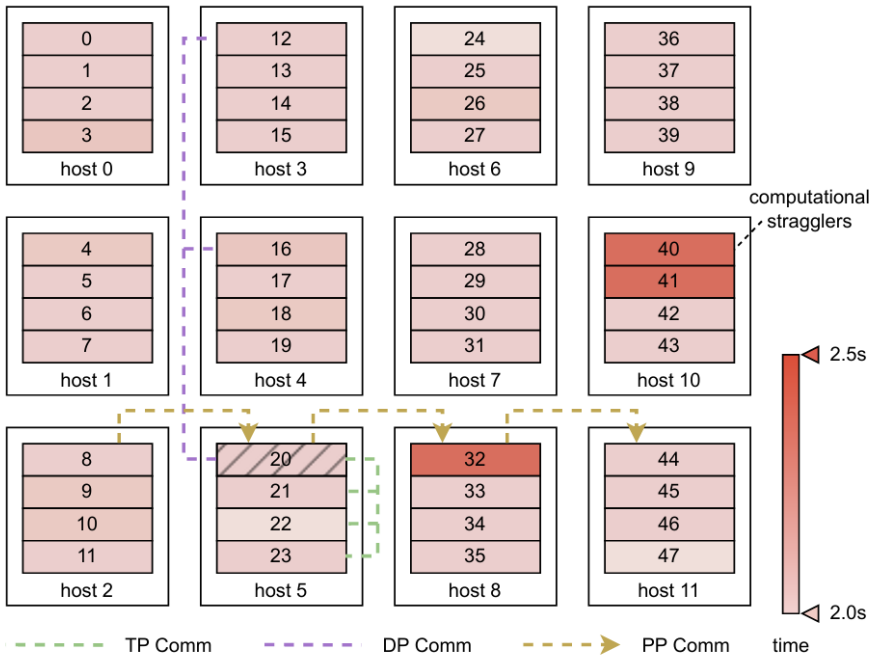


Figure 7: Performance heat-map. The color denotes the running time of the code segments on a rank. The figure also shows the 3D visualization feature, where rank 20 has been selected and the dependency across different parallelism dimensions become visible.

- Record execution time of critical code segments **using CUDA events** instead of heavy profilers.
- The heat map averages computation phase latency across steps and visualizes per-rank differences.
- The paper reports that about **0.5% of machines were substantially slower during training**.
- After excluding outliers, peak MFU became more consistent across runs.
- Lesson: single-GPU microbenchmarks may miss distributed straggler behavior.

3D Parallel Training Visualization

- **Problem**: Pipeline bubbles, synchronization points, and event ordering across ranks on a unified timeline.
- **Challenge**: Single node information is insufficient
- **Design**: Each GPU worker logs its ongoing event on timeout; logs are mapped onto the 3D topology of tensor, pipeline, and data parallelism.
- **Outcome**: Dependencies become visible when a selected event spans multiple parallel dimensions.

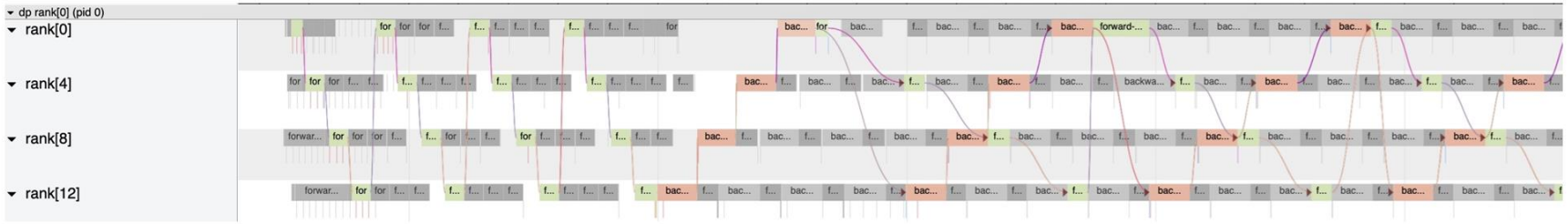


Figure 8: The trace shows events collected in a pipeline group on a unified timeline. Dependencies become visible when an event is selected.

Outline

- Background & Motivation
- Efficient Training at Scale
  - Network Performance Tuning
- Fault Tolerance and Diagnosis
- Training Troubleshooting
- **Experience**
- Discussion

Models

- 175B model: 128 heads, hidden size 12288, 96 layers, TP=8, PP=8.
- 530B model: 160 heads, hidden size 20480, 105 layers, TP=8, PP=35.

Compared Scheme

- Megatron-LM commit from January 2023, chosen by the authors for stability and feature set at that time.
- https://github.com/NVIDIA/Megatron-LM/commit/285068c8108e0e8e6538f54fe27c3ee86c5217a2

Training Performance

- **Strong Scaling**: measures how the execution time of a fixed-size problem decreases as more processors are added.
- **Weak Scaling**: evaluates how well a system maintains consistent execution time as the workload increases proportionally with the number of processors.

Training Performance - Strong Scaling

Batch Size	Method	GPUs	Iteration Time (s)	Throughput (tokens/s)	Training Time (days)	MFU	Aggregate PFlops/s
768	Megatron-LM	256	40.0	39.3k	88.35	53.0%	43.3
		512	21.2	74.1k	46.86	49.9%	77.6
		768	15.2	103.8k	33.45	46.7%	111.9
		1024	11.9	132.7k	26.17	44.7%	131.9
	MegaScale	256	32.0	49.0k	70.86	65.3%(**1.23**×)	52.2
		512	16.5	95.1k	36.51	63.5%(**1.27**×)	101.4
		768	11.5	136.7k	25.40	61.3%(**1.31**×)	146.9
		1024	8.9	176.9k	19.62	59.0%(**1.32**×)	188.5
6144	Megatron-LM	3072	29.02	433.6k	8.01	48.7%	466.8
		6144	14.78	851.6k	4.08	47.8%	916.3
		8192	12.24	1027.9k	3.38	43.3%	1106.7
		12288	8.57	1466.8k	2.37	41.2%	1579.5
	MegaScale	3072	23.66	531.9k	6.53	59.1%(**1.21**×)	566.5
		6144	12.21	1030.9k	3.37	57.3%(**1.19**×)	1098.4
		8192	9.56	1315.6k	2.64	54.9%(**1.26**×)	1400.6
		12288	6.34	1984.0k	**1.75**	55.2%(**1.34**×)	2166.3

Table 2: Strong-scaling training performance for the 175B model. We set the batch size to 6144 when training with 3072 to 12288 GPUs. For 256 to 1024 GPUs, we decrease the batch size to 768 due to GPU memory limit. We report the training time required for training 300B tokens here. The number in parentheses in the MFU column represents the speedup of MegaScale compared to Megatron-LM.

Training Performance - Weak Scaling

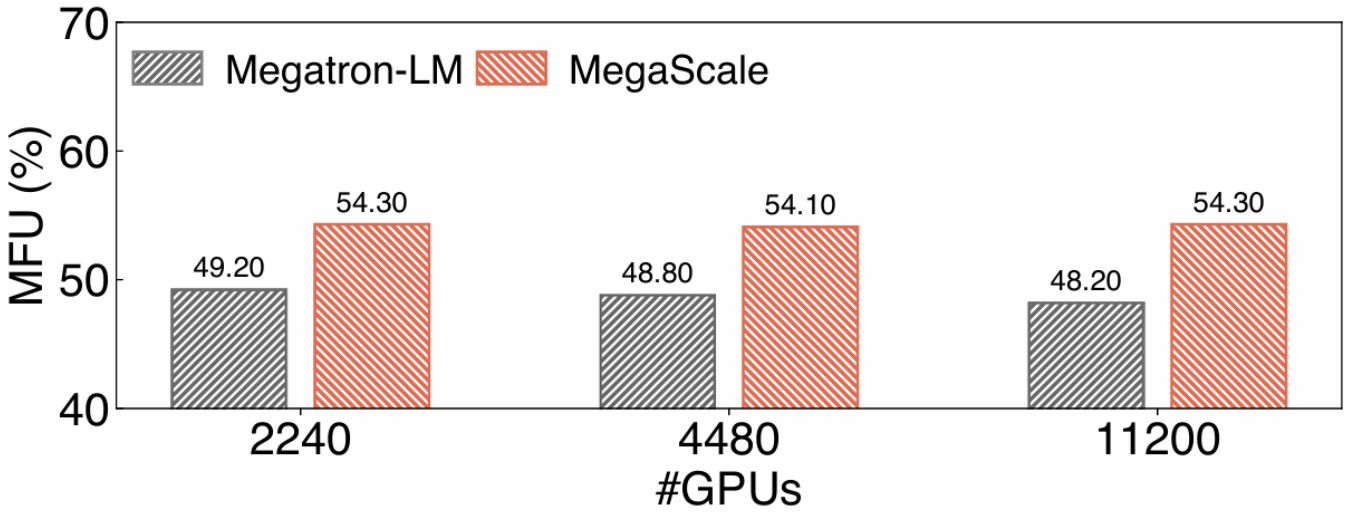
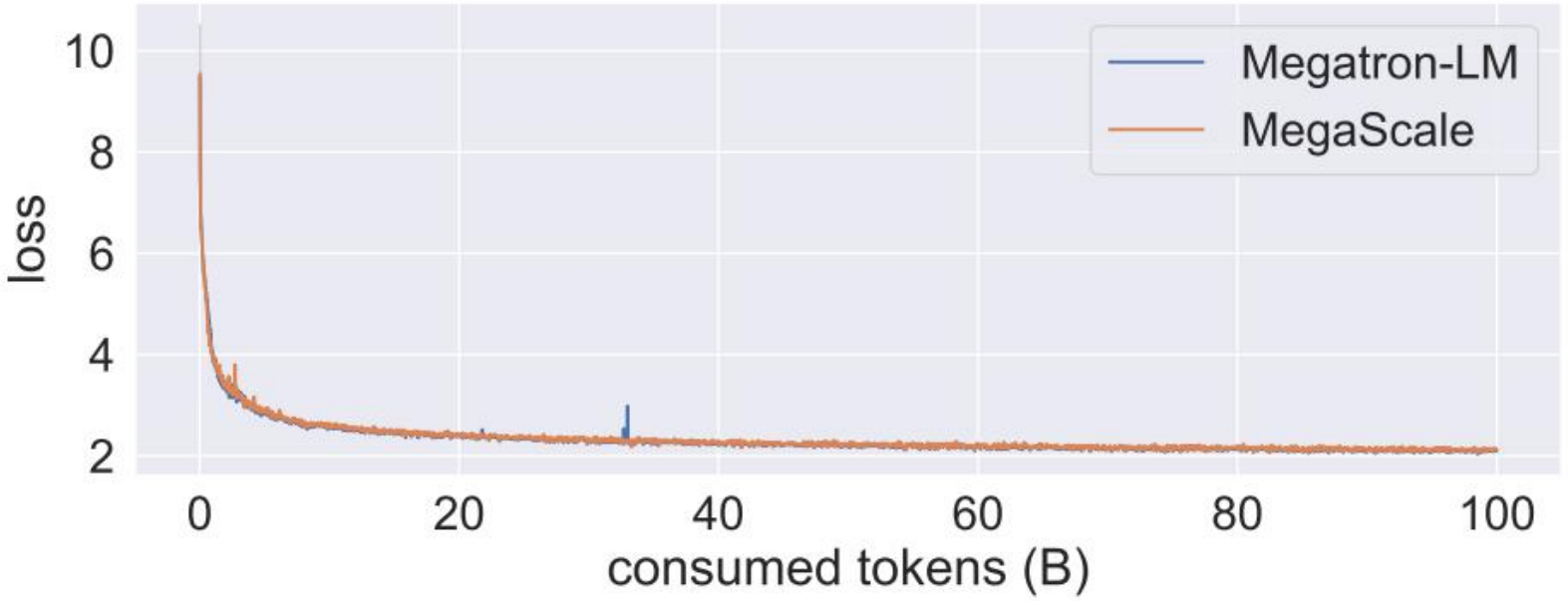


Figure 9: Weak-scaling training performance of Megatron-LM and MegaScale on the 530B model, where the batch size is scaled proportionally with the number of GPUs.

Model Convergence and Stability



(a) The training loss curve of MegaScale, which includes algorithm optimizations, in comparison with Megatron-LM.

Problems Discovered and Fixed

- MFU gradually decreased during a large production run.
- Ranks arrived at reduce-scatter at different times because some code segments fluctuated.
- Ranks arrived at reduce-scatter at different times because some code segments fluctuated.

Problems Discovered and Fixed

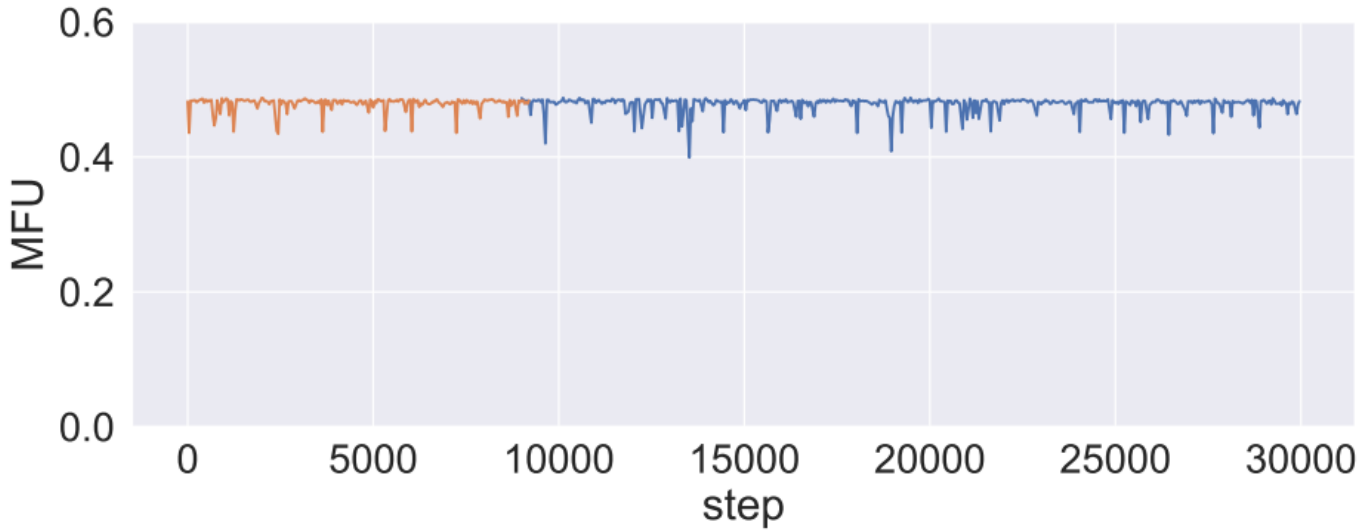


Figure 12: The MFU becomes stable after addressing the stragglers and problematic code segments. Different colors represent different training trials with the same setup.

Outline

- Background & Motivation
- Efficient Training at Scale
  - Network Performance Tuning
- Fault Tolerance and Diagnosis
- Training Troubleshooting
- Experience
- **Discussion**

Discussion

- LLM training at 10K+ GPUs is a synchronized distributed system; the slowest rank matters.
- Communication optimization requires application/runtime redesign: fusion, chunking, scheduling, and overlap.
- The network is not generic: ECMP, PFC, congestion control, placement, and RTO settings interact with collectives.
- Production success depends on observability and recovery as much as peak MFU.

Topic Review

Network Support for Machine Learning

What We Have Learned?

- **Topology**: TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs, NSDI 2023
- **Transports**: Towards Domain-Specific Network Transport for Distributed DNN Training, NSDI 2024
- **Deployment**: MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs, NSDI 2024 (Experience Paper)

Some Key Points

- TopoOpt
  - Why Fat-Tree is a good for general data center workloads but may be suboptimal for DNN training?
  - What is OCS?
  - Why AllReduce traffic is considered mutable, while model-parallel traffic is not?

Some Key Points

- MLT
  - What is the main performance problem in MLT paper?
  - What is the key observation to enable MLT?
  - How is MLT different from conventional congestion control?

Some Key Points

- MegaScale
  - How to achieve efficient training at scale?
  - How to perform network performance tuning?
  - How to enable fault tolerance and diagnosis?
  - How to perform training troubleshooting?

Reading Materials (For Group Presentation. Not in the Exam)

- LLM Training
  - Alibaba HPC: A Data Center Network for Large Language Model Training, **SIGCOMM 2024**
  - ByteScale: Communication-Efficient Scaling of LLM Training with a 2048K Context Length on 16384 GPUs, **SIGCOMM 2025**
- LLM Inference
  - Efficient Memory Management for Large Language Model Serving with PagedAttention, **SOSP 2023**
  - Distserve: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving, **OSDI 2024**

End of the Semester
Thanks!