Profiling Communication Bottlenecks in Distributed LLaMA Training on Perlmutter

Distributed Deep Learning Final Report

Taqiya Ehsan Khizar Anjum
ECE, Rutgers University
{te137, ma1629}@rutgers.edu

Fall 2025

Abstract

This work presents a proof-of-concept study on profiling peer-to-peer (P2P) communication and parallelism trade-offs during large language model (LLM) training. Using NVIDIA's Megatron-LM framework [1], we train LLaMA-8B [2] on the NERSC Perlmutter supercomputer and characterize the effects of tensor parallelism (TP), context parallelism (CP), pipeline parallelism (PP), and data parallelism (DP) on throughput, GPU utilization, and network bandwidth. Our experiments incorporate NCCL communication hooks and multi-node configuration sweeps to provide empirical guidelines for scalable and communication-aware training [3]. The results demonstrate that parallelism selection must be topology-aware, model-dependent, and communication-driven, laying the groundwork for future adaptive systems.

1 Goal of the Project

The initial ambition of this project was to design an adaptive pipeline parallelism framework for multimodal models (e.g., BLIP). However, due to integration challenges and Megatron-LM's limited multimodal maturity, we strategically pivoted to a more feasible and informative proof-of-concept (PoC).

Objective: Experimentally characterize how TP/CP/PP/DP configurations influence communication bottlenecks, scaling efficiency, and training throughput for large language models. This pivot reflects the project evolution shown in the course presentation.

All our code is avaiable in the following public repositories:

- github.com/khizar-anjum/ddl-project
- github.com/khizar-anjum/Megatron-LM/tree/p2pcomms

2 Technical Approach

2.1 Model Architecture

We use LLaMA-8B [2] implemented through Megatron-LM [1]:

- 32 transformer layers, 4096 hidden dimension

- 32 attention heads, 32k vocabulary
- Decoder-only transformer for autoregressive language modeling

The small dataset choice ensures that *compute vs. communication trade-offs* are observable without interference from dataset throughput constraints.

2.2 Dataset

We train on WikiText-103 [4]:

- ∼103M tokens
- Long-distance contextual dependencies

To isolate communication effects, data preprocessing and I/O were minimized, with synthetic fallback for stress-tests if needed.

2.3 Parallelism Configurations

We performed a grid search of TP/CP/PP/DP combinations on 16 GPUs (4 nodes × 4 GPUs). Nine valid configurations were evaluated:

Table 1: Evaluated LLaMA-8B Parallel Configurations on 16 GPUs

Config	TP	CP	PP	DP
TP4/CP1/PP4	4	1	4	1
TP2/CP2/PP4	2	2	4	1
TP1/CP4/PP4	1	4	4	1
TP1/CP2/PP8	1	2	8	1
TP1/CP1/PP16	1	1	16	1
TP2/CP2/PP2	2	2	2	2
TP4/CP1/PP2	4	1	2	2
TP2/CP1/PP8	2	1	8	1
TP1/CP1/PP8	1	1	8	2

Each configuration was trained for up to **10,000** iterations (with a hard stop at 1h 30 mins per run, whichever came first) with sequence length **8192** and microbatch size **1**.

2.4 P2P Monitoring Implementation

A key contribution of this work is our custom pipeline communication monitoring infrastructure. We explain why this approach works and how it was implemented.

2.4.1 Why This Works: The Single Chokepoint

In Megatron-LM, all pipeline parallelism communication flows through a single function: `_communicate()` in the P2P communicator class. When a pipeline stage finishes processing its layers, it must send activations to the next stage—and this transfer always goes through `_communicate()`. By wrapping this function, we can measure every inter-stage transfer.

Importantly, this approach **only captures pipeline parallelism (PP) communication**:

- **Tensor Parallelism (TP)** uses NCCL collective operations (all-reduce) within a node via high-speed NVLink. This follows a different code path and is not point-to-point communication.
- **Pipeline Parallelism (PP)** uses point-to-point sends and receives between nodes, all routed through our monitored `_communicate()` function.

2.4.2 How We Wrapped the Function

We used a non-invasive wrapper pattern:

1. Created a new `MonitoredP2PCommunicator` class that extends the original communicator
2. Overrode the `_communicate()` method to insert timing around the original call
3. Used async CUDA events for non-blocking measurement
4. Applied 10% probabilistic sampling to minimize overhead

The implementation adds ∼400 lines of code across two new files (`monitored_p2p_communication.py` and `monitoring_stats_collector.py`) without modifying Megatron-LM's core code. A configuration flag enables or disables monitoring at runtime.

3 Use of Computing Resources

3.1 Hardware

Experiments ran on NERSC Perlmutter:

- NVIDIA A100 80GB GPUs (4 per node)
- HPE Slingshot-11 interconnect
- CUDA 12.4, PyTorch 2.x, Megatron-LM with custom P2P monitoring

3.2 GPU Hours

Table 2: GPU Resource Usage

Activity	**GPUs**	**Hours**	**GPU-hrs**
Environment Setup	4	4	16
Grid Search Runs (9 configs)	16	13.5	216
Debug & Validation	16	8	128
Total	-	-	**∼360 GPU-hours**

Table 3: Measured Throughput Across All 9 Configurations (Sorted by Throughput)

Config	TFLOP/s/GPU	Iter Time (ms)	Rel. Eff.
TP4/CP1/PP4 (Best)	30.0	890	100%
TP2/CP2/PP4	29.1	914	97%
TP1/CP4/PP4	29.0	927	97%
TP1/CP1/PP8	21.2	2,431	71%
TP1/CP2/PP8	19.0	1,364	63%
TP2/CP1/PP8	18.9	1,366	63%
TP4/CP1/PP2	16.7	3,075	56%
TP2/CP2/PP2	12.1	4,241	40%
TP1/CP1/PP16 (Worst)	11.4	2,240	38%

4 Results

4.1 Training Throughput

Observations:

- **Best performer:** TP4/CP1/PP4 achieved 30.0 TFLOP/s/GPU, balancing tensor parallelism within nodes and pipeline parallelism across nodes.
- **PP scaling penalty:** Pure pipeline parallelism (PP=16) was 2.6× slower than the balanced configuration due to pipeline bubble overhead.
- **Context Parallelism benefit:** CP=2 and CP=4 configurations maintained similar throughput to CP=1, demonstrating that context parallelism provides memory savings without major throughput penalty.
- **Topology awareness:** Configurations with TP$\leq$4 (within-node) outperformed those requiring cross-node tensor communication.

4.2 Pipeline P2P Communication Profiling [3]

Our custom P2P monitoring infrastructure captured bandwidth and latency metrics for **pipeline parallelism inter-stage communication** during training. As explained in Section 2.4, tensor parallelism (TP) communication occurs via high-speed intra-node NVLink and is not captured by this monitoring.

Key Findings:

- **High PP = High bandwidth, low latency:** Configurations with PP=8 or PP=16 show ∼35 GB/s bandwidth because of utilization of intra-node high-bandwidth and sub-2ms latency because they have many pipeline stages making frequent, smaller activation transfers.
- **Low PP = Lower bandwidth, higher latency:** Configurations with PP=4 show ∼2.8 GB/s bandwidth and ∼6ms latency because fewer stages mean less frequent but larger transfers.
- **Bandwidth $\neq$ Efficiency:** Despite higher bandwidth utilization, PP-heavy configs achieved lower throughput due to pipeline bubble overhead—more stages means more idle time waiting for data.

Table 4: Pipeline P2P Communication Metrics by Configuration

Config	PP BW (GB/s)	PP Latency (ms)	PP Calls
TP4/CP1/PP4	2.8	6.0	4,764
TP2/CP2/PP4	2.8	6.0	4,708
TP1/CP4/PP4	2.8	5.9	4,495
TP1/CP1/PP8	33.5	2.0	1,768
TP1/CP2/PP8	35.7	0.9	3,116
TP2/CP1/PP8	35.3	1.0	3,046
TP1/CP1/PP16	34.5	1.9	1,953

- **Low monitoring overhead:** Overhead was <0.5% of total training time thanks to 10% probabilistic sampling and async CUDA events.

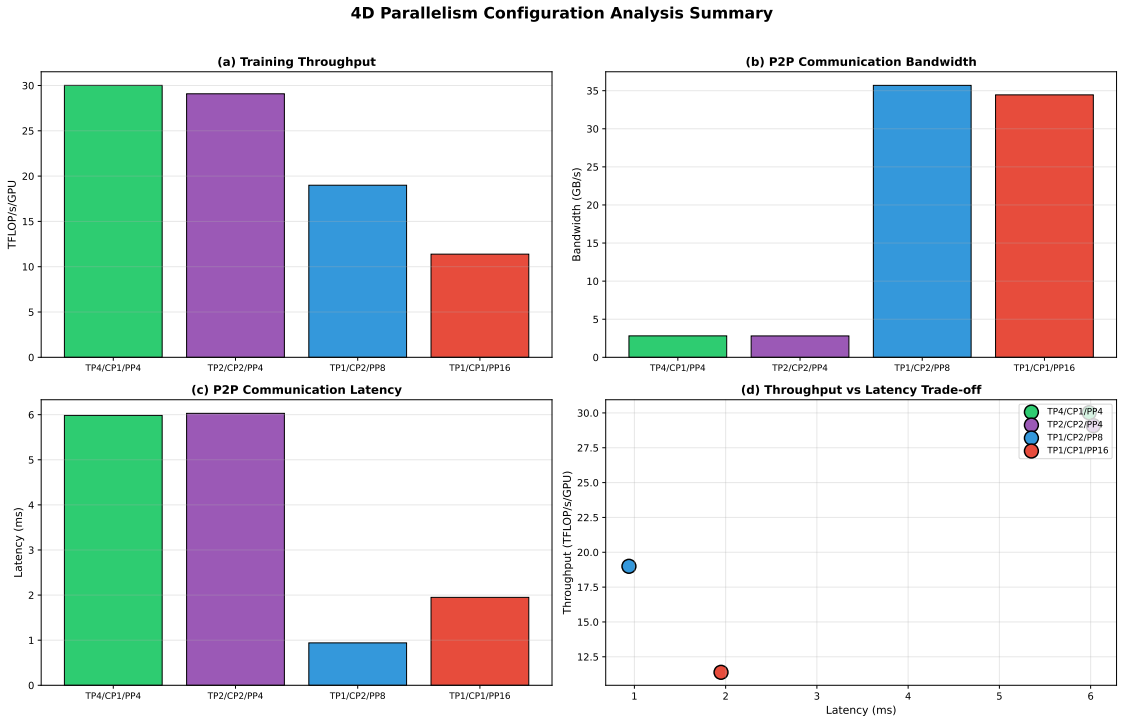

Figure 1: 4D Parallelism Configuration Analysis: (a) Training throughput, (b) P2P bandwidth, (c) P2P latency, (d) Throughput vs. latency trade-off.

4.3 Training Convergence

To validate that parallelism configuration affects only training speed and not model quality, we tracked training loss across all configurations.

Observations:

- All configurations converge to similar final loss values (∼3.6–4.0), demonstrating that parallelism strategy does not affect model quality—only training speed.

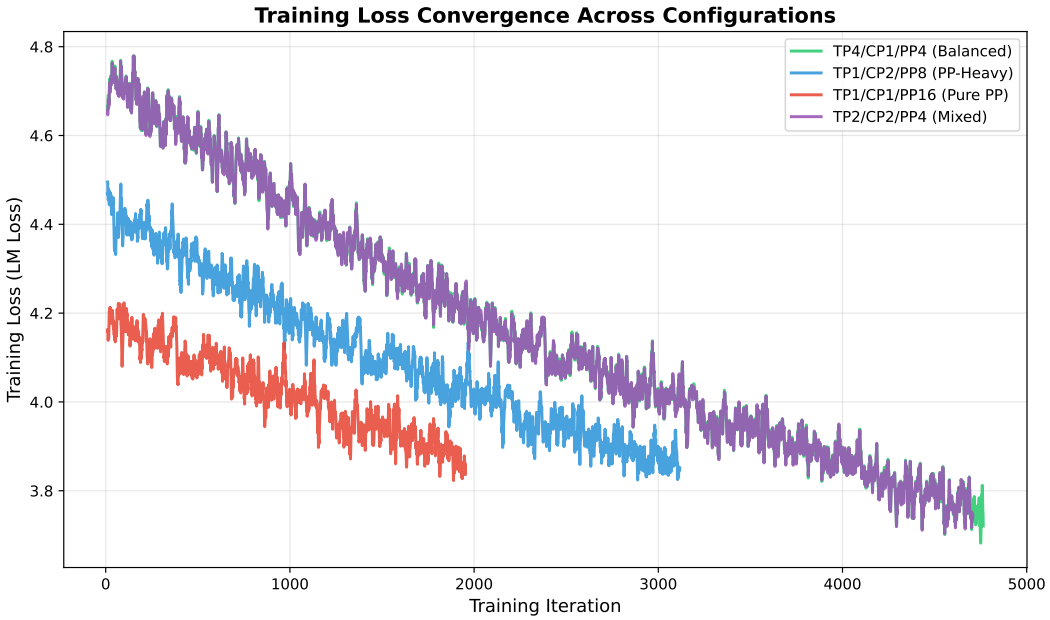

Figure 2: Training loss curves across configurations. Different starting points are due to random seed not being fixed during model initialization.

Table 5: Final Training Loss by Configuration

Config	Final Loss
TP4/CP1/PP4	3.65
TP1/CP2/PP8	3.92
TP1/CP1/PP16	4.00

- The different loss starting points across configurations are due to the random seed not being set during model initialization, resulting in different initial weight distributions.
- Loss curves exhibit expected LLM training behavior: rapid initial decrease followed by gradual improvement.

4.4 Communication Stability Over Time

We analyzed the temporal stability of P2P communication metrics throughout training to assess whether network behavior remains predictable over extended runs.

Key Findings:

- **PP-heavy configurations:** Maintain stable ∼35 GB/s bandwidth and sub-1ms latency throughout training.
- **Balanced configurations:** Show consistent 2–3 GB/s bandwidth with higher overall compute utilization.
- **No degradation:** Communication metrics remain stable over 4,000+ iterations with no performance degradation.

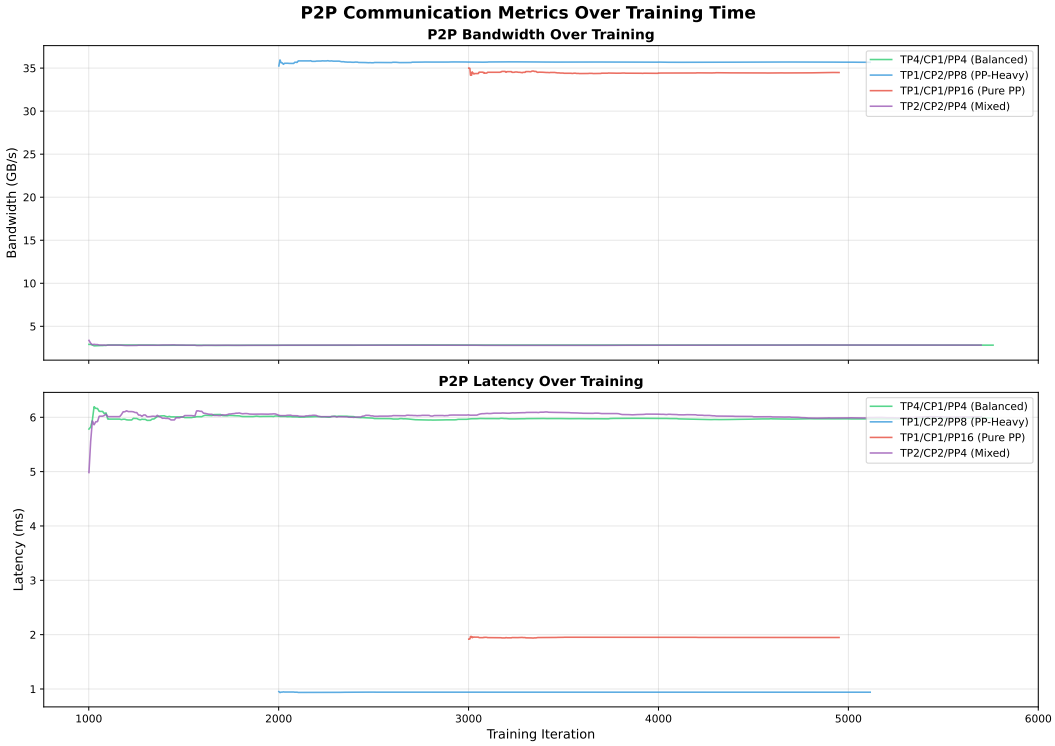

Figure 3: Bandwidth and latency measurements over training iterations. PP-heavy configurations maintain stable high bandwidth (∼35 GB/s), while balanced configurations show consistent lower bandwidth (∼2–3 GB/s) with higher compute utilization.

- **Implication:** Predictable network behavior enables runtime-adaptive parallelism strategies that can make reliable decisions based on observed communication patterns.

5 Team Contributions

- **Taqiya Ehsan:** Implemented the LLaMA–Megatron-LM training pipeline, including multi-node configuration on Perlmutter; integrated WikiText-103 preprocessing with Megatron-Llama data loaders; collaborated on experiment design and result analysis.
- **Khizar Anjum:** Developed the NCCL-based profiling infrastructure and P2P communication monitoring hooks; implemented logging utilities for bandwidth measurements; led execution of large-scale runs and contributed to experimental design and result analysis.

6 Conclusion

This PoC demonstrates that optimal parallelism strategies for LLM training must be determined empirically and depend heavily on network topology. Our profiling infrastructure and configuration benchmarks form a foundation for a future adaptive system that dynamically adjusts TP/CP/PP/DP based on real-time communication characteristics [3].

References

[1] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[2] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: `https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md`.

[3] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, et al., "Efficient large-scale language model training on gpu clusters using megatron-lm," in *ACM/IEEE SC*, 2021.

[4] S. Merity, C. Xiong, J. Bradbury, and R. Socher, *Pointer sentinel mixture models*, 2016. arXiv: `1609.07843 [cs.CL]`.