Understanding Communication Bottlenecks in Multi-Node LLM Inference

Prajwal Singhania[1], Siddharth Singh[1 3], Lannie Dalton Hough[1], Ishan Revankar[1], Harshitha Menon[2], Charles Jekel[2], Abhinav Bhatele[1]

[1]Department of Computer Science, University of Maryland [2]Lawrence Livermore National Laboratory [3]NVIDIA, Inc.

Abstract

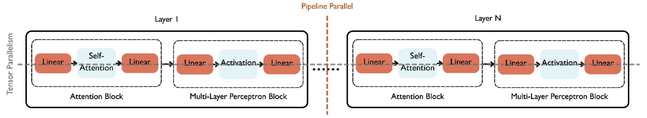
Motivation

- LLMs are rapidly growing in parameter count, requiring **inference to scale beyond a single node**.
- Current frameworks rely on **tensor parallelism (TP)** within a node and **pipeline parallelism (PP)** across nodes.
- TP has **high communication overhead**, while PP incurs **pipeline bubbles** and is unsuitable for latency critical scenarios.
- Existing frameworks are **large and complex**, making it **difficult to rapidly prototype** and study new distributed inference strategies.

Our Work

- We present **Yalis**, a **lightweight and performant** framework for offline LLM inference that is easy to use and extend.
- We study **strong scaling performance of LLM Inference,** and **identify communication bottlenecks** in tensor parallelism.

Background



Pipeline Parallelism (PP): Splits contiguous layers across devices; cannot speed up a single query due to input–output dependency; point-to-point communication.

Tensor Parallelism (TP): Splits the computation of a single layer across GPUs; all-reduce communication, incurring high communication volume.

Inference Phases

- Prefill: Model processes prompt tokens in parallel to generate the first output; compute-bound, with communication message sizes in the 100s of MBs.
- Decode: Model generates tokens one at a time, feeding each new token back as input; memory-bound, with message sizes in the 10-100s of KBs range.

Online Inference: Model serves interactive requests (e.g., chat).

Offline Inference: Model processes a fixed workload (e.g., dataset generation, evaluation or locally-run LLMs).

YALIS: Yet Another LLM Inference System Design

LitGPT [1]
Lightweight Model Definition for fast prototyping

Torch Compile [2]
Optimized kernels with reduced launch overhead

AxoNN [3]
Scalable 2D Tensor Parallelism

Support for pluggable attention backends.

Supports Paged-KV caching.

Currently optimized for offline inference with single user online inference support.

Strong Scaling LLM Inference

Goal: Study strong scaling of LLM inference on Alps (GH200) and Perlmutter (A100), and identify performance bottlenecks.

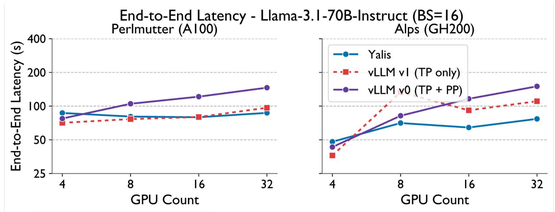
Experimental Setup:

Model: Llama-3.1-70B Instruct, Batch Size:16

Prefill Length: 2480 tokens, Decode Length: 2048 tokens

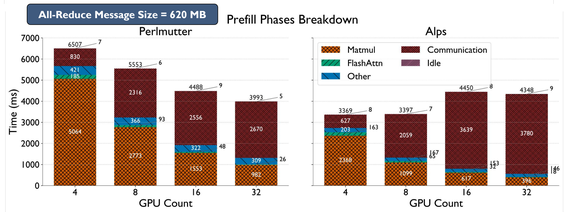
Frameworks: Yalis†, vLLM [4] v1†, vLLM v0*

(† TP intra and inter nodes, * TP intra and PP inter node)



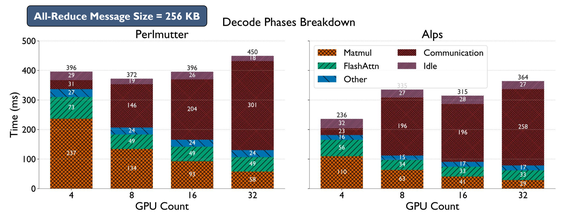
Top: End-to-end generation latency (Yalis vs. vLLM)

- Time to solution does not decrease with more GPUs.
- With PP, end-to-end latency increases on both machines.
- With TP:
  - *Perlmutter*: Latency remains almost constant.
  - *Alps:* Latency increases sharply going from a single node to multi-node, then continues to rise steadily.



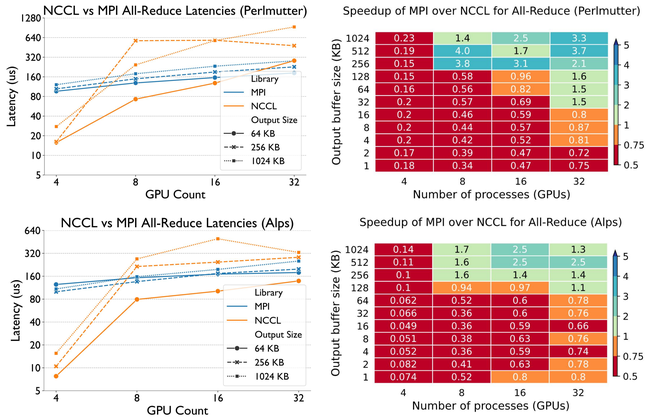
Middle & Bottom: Prefill and Decode Breakdowns for Yalis (Decode results are summed over 10 steps)

- Matmul times decrease with more GPUs.
- Communication time grows drastically beyond one node.
- Communication blow-up is more pronounced for Alps.



MPI vs NCCL Latencies in the Decode Regime

To better understand the communication bottleneck, we compared NCCL performance with MPI in the small message size (decode) regime.



- NCCL is highly optimized for intra-node communication over NVLink.
- Beyond a single node, MPI can outperform NCCL in the 256-1024 KB range.

Conclusion and Future Work

- Offline inference performance of Yalis is on par with SOTA frameworks like vLLM.
- Parallelism strategies (tensor and pipeline) scale poorly beyond a single node for offline inference.
- Tensor parallelism incurs high communication costs across nodes, especially on newer hardware like GH200.
- NCCL is efficient for small messages within a node but scales poorly across nodes; MPI can outperform it.
- **Future work**: Optimized GPU-initiated all-reduce algorithms and deeper analysis of communication bottlenecks in the prefill regime.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, through solicitation DE-FOA-0003264,"Advancements in Artificial Intelligence for Science," under Award Number DE-SC0025598.This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory (LLNL) under Contract DE-AC52-07NA27344 (LLNL-POST-2011899).
This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility, operated under Contract No. DE-AC02-05CH11231 using NERSC award DDR-ERCAP0034262.
This research is supported by the National Artificial Intelligence Research Resource (NAIRR) Pilot and the Delta advanced computing and data resource which is supported by the NSF (award NSF-OAC 2005572) and the State of Illinois.
The authors acknowledge the University of Maryland supercomputing resources made available for conducting the research reported in this paper.
This work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID lp98 on Alps.

References

[1] Lightning AI. 2023. LitGPT. https://github.com/Lightning-AI/litgpt.
[2] Meta. 2023.Torch Compile. https://docs.pytorch.org/tutorials/intermediate/torch_compile_tutorial.html.
[3] Singh, Siddharth, and Abhinav Bhatele. "AxoNN: An asynchronous, message-driven parallel framework for extreme-scale deep learning." 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2022.
[4] Kwon, Woosuk, et al. "Efficient memory management for large language model serving with pagedattention." Proceedings of the 29th symposium on operating systems principles. 2023.