ACM Conference on AI and Agentic Systems

Understanding and Improving Communication Performance in Multi-node LLM Inference

Prajwal Singhania[1], Siddharth Singh[1], Lannie Dalton Hough[1], Akarsh Srivastava[1], Harshitha Menon[2], Charles Fredrick Jekel[2], Abhinav Bhatele[1]

[1]Department of Computer Science, University of Maryland
[2]Lawrence Livermore National Laboratory

PARALLEL SOFTWARE AND SYSTEMS GROUP

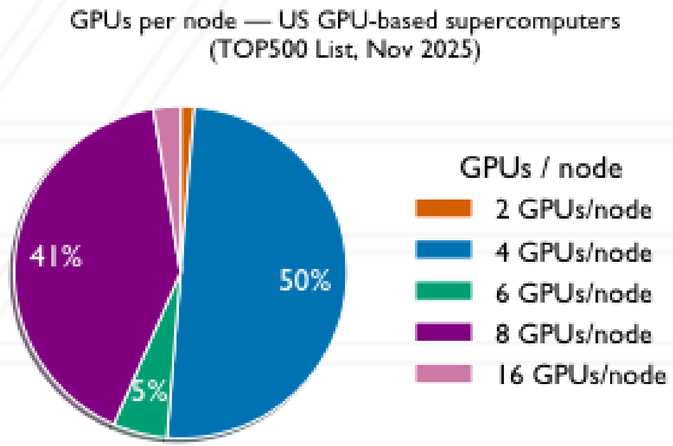
UNIVERSITY OF MARYLAND

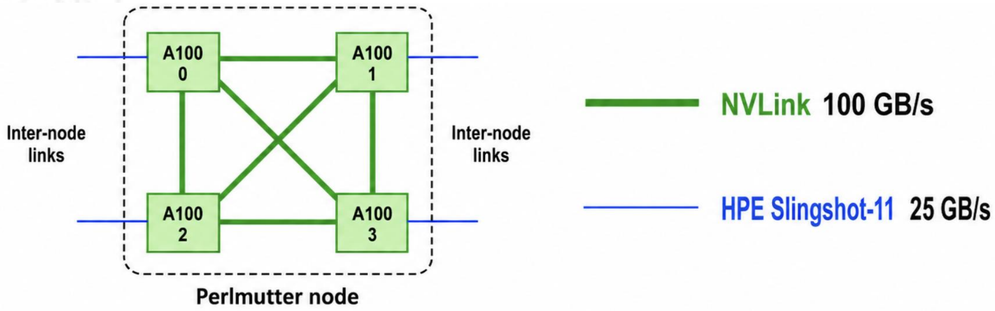
Talk Overview

1. Motivation & Background
2. Multi-Node Inference Performance Study
3. Optimizing All-reduce for TP Inference
4. Evaluation

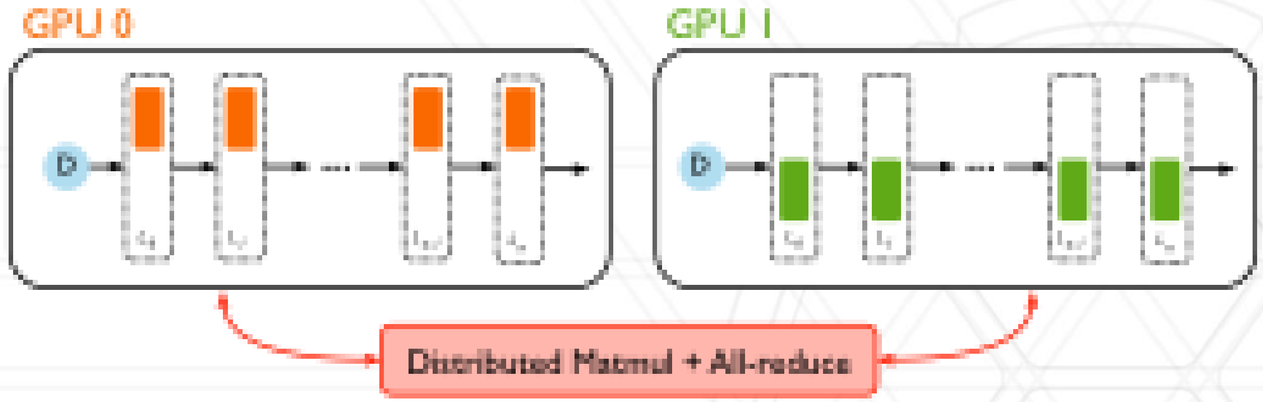
Why Study Multi-node LLM Inference?

- GPU clusters come in different shapes and forms.
  - Single node: GPUs in the same compute tray usually sharing a high b/w network like NVLink.
- With growing model size, inference spills beyond a node ↦ need *model parallelism.*
  - Llama 3.1 405B and Qwen3-235B-A22B require ≥ 16 80GB A100s for inference (4 nodes with 4 GPUs per node).
- **Gap:** Existing inference performance studies ar optimizations largely focus on singe-/two-node settings.

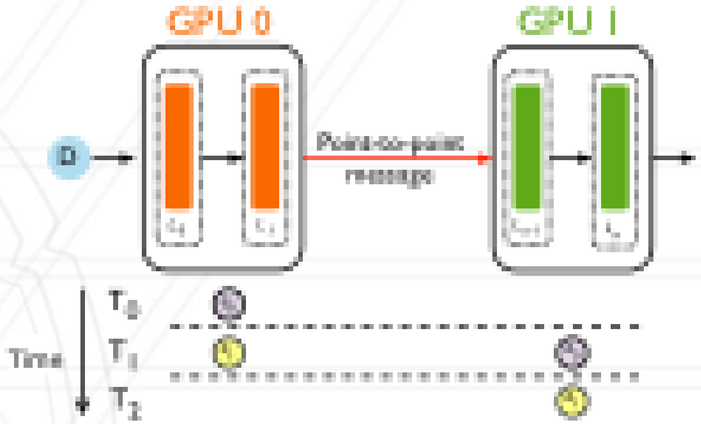




Multi-node Model Parallelism



Tensor Parallelism (TP)
Splits the computation of a single layer across GPUs
All-reduce communication



Pipeline Parallelism (PP)
Splits contiguous layers across devices
Point-to-point communication

- **Hybrid Parallelism (HP):** TP within a node and PP across nodes.
- Most inference frameworks use pure TP or HP for multi-node model parallelism.

Multi-node Inference Performance Study

- We conduct a study comparing **HP vs TP** multi-node inference deployments.
- Batched Inference Workloads: One batch of prompts completes before the the next batch is submitted.
- Strong scaling: Fixed workload with increasing GPU count.

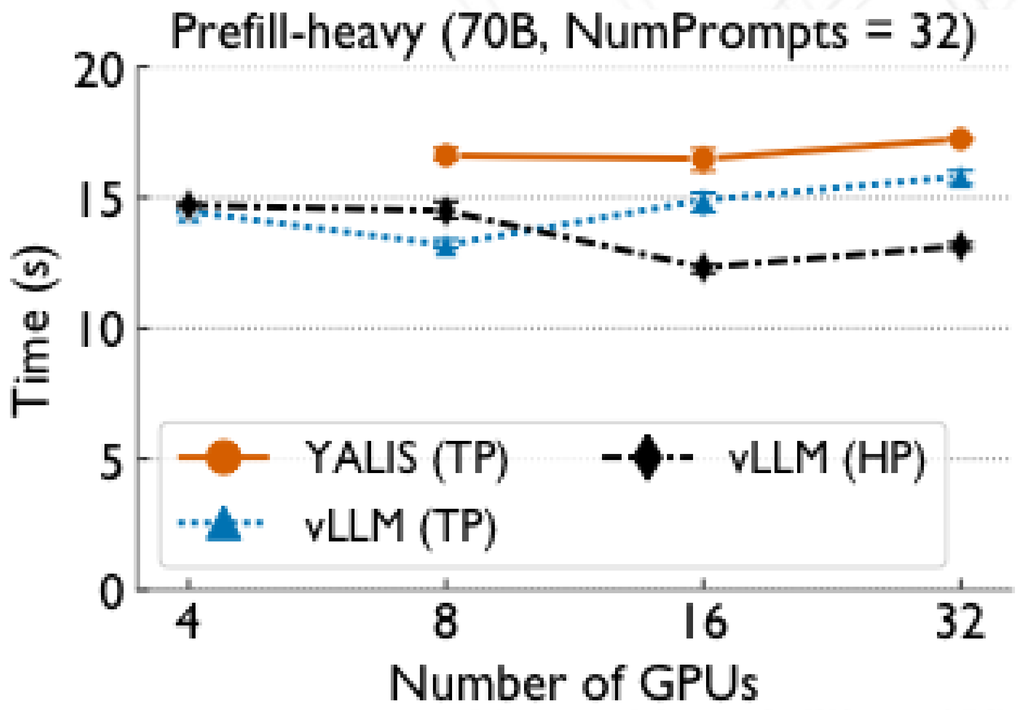
Models	Llama 70B, 405B
Workloads	Prefill-heavy (ISL=2363, OSL=128), Decode-heavy (ISL=1426, OSL=3072)
Frameworks	YALIS, vLLM (v0.11), SGLang (v0.5.1)
Machine	Perlmutter: 4x A100/node, NVLink intra-node, Slingshot-11 inter-node

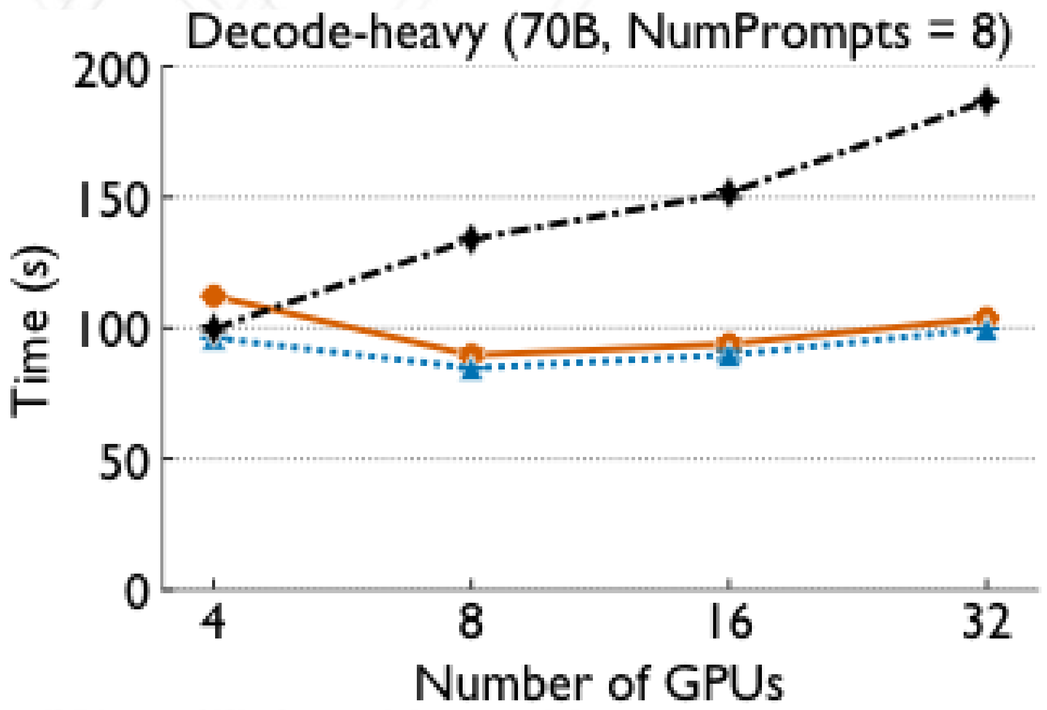
6 out of 10 TOP500 Supercomputers use this Interconnect

- **YALIS (Yet Another LLM Inference System):** Prototype inference engine developed for easier instrumentation.

Scaling Results

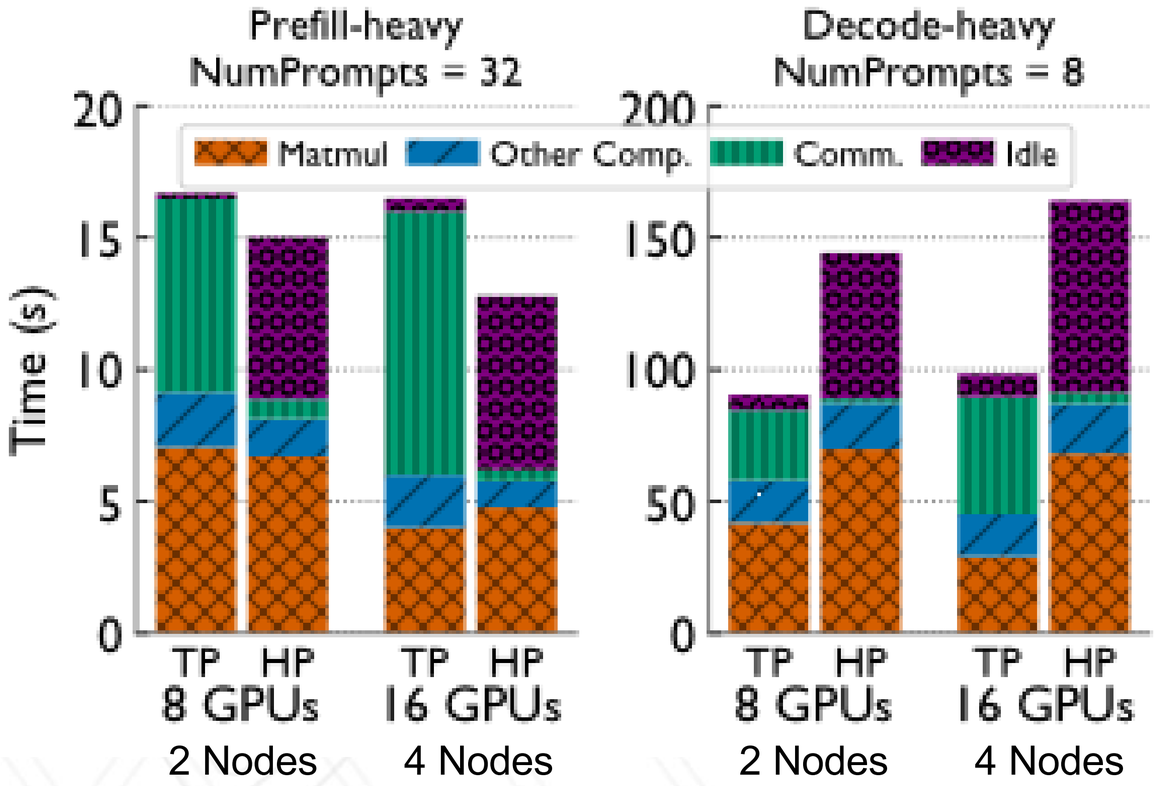
Strong Scaling Expectation: With increasing GPU count, time per batch decreases.





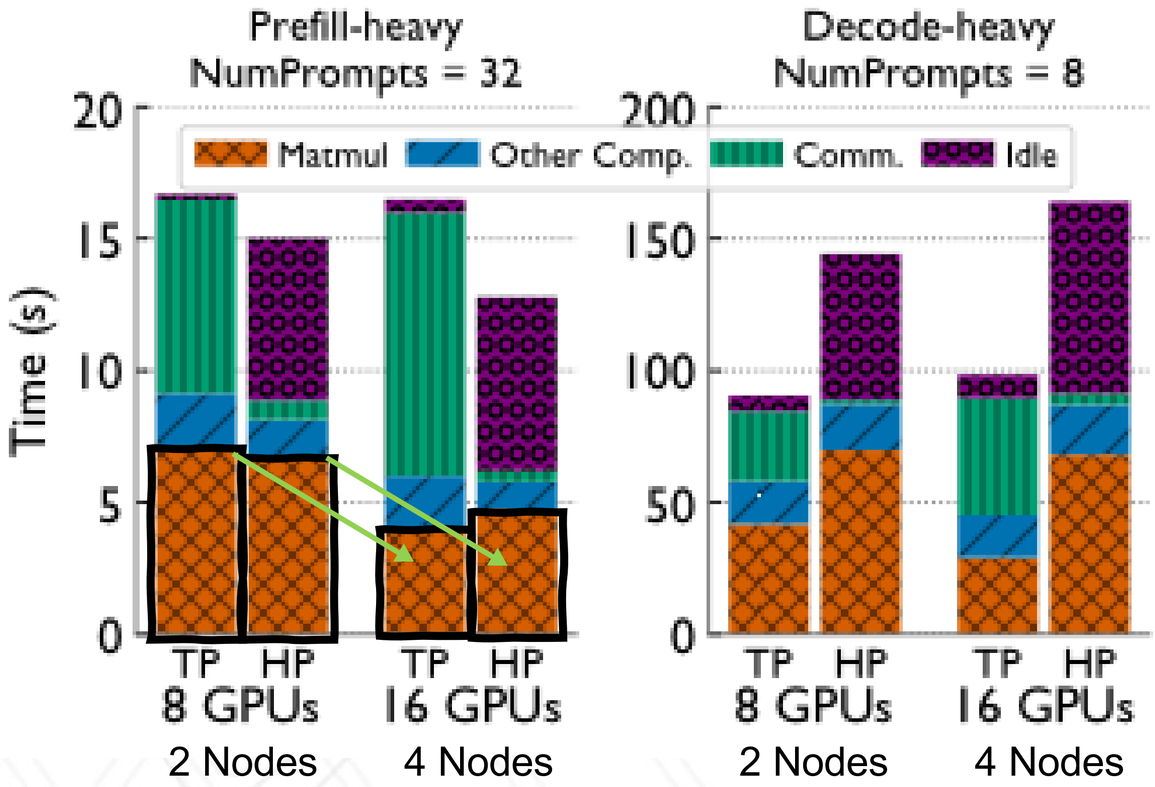
TP and HP do not scale ideally! Decode-heavy workloads favor TP

Performance Bottlenecks



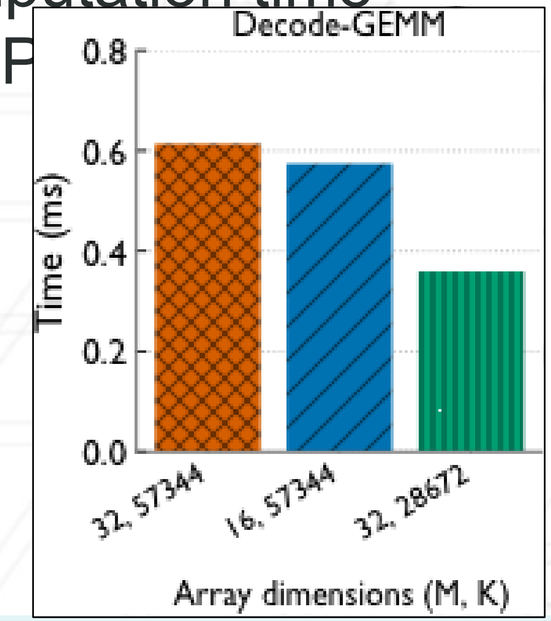
Performance Bottlenecks

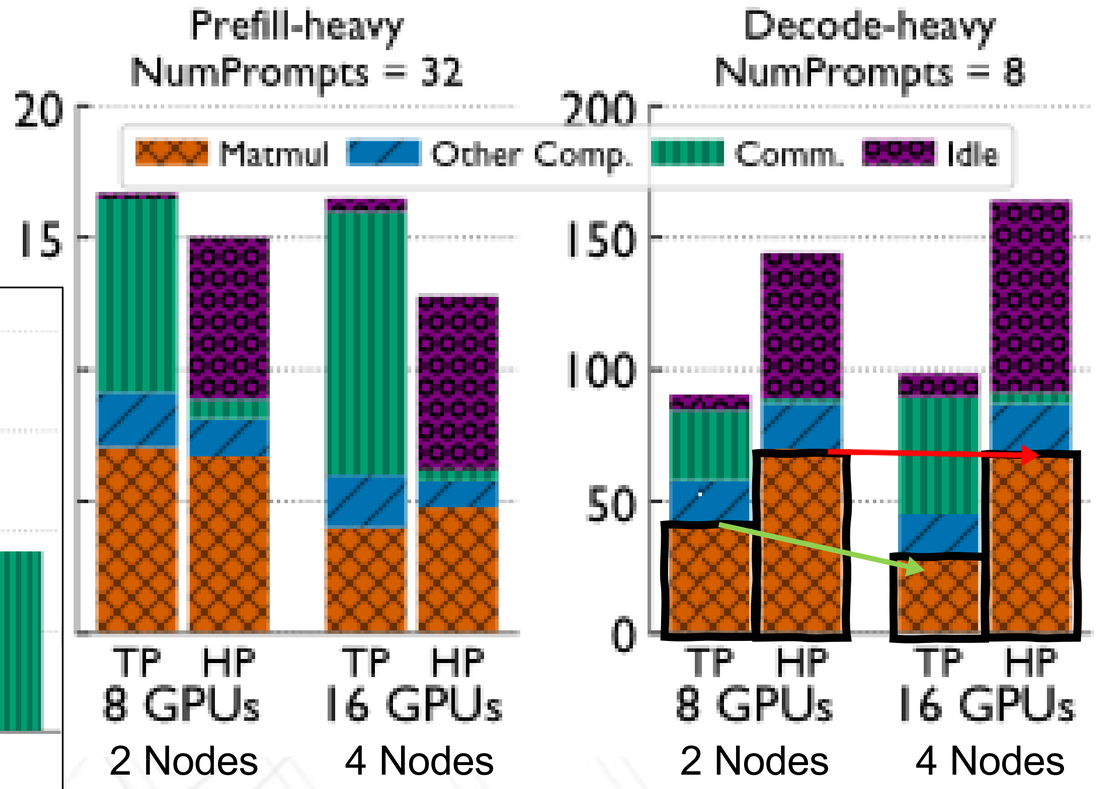
- Prefill-heavy: Computation time decreases for both TP and HP when scaling from 8 to 16 GPUs.



Performance Bottlenecks

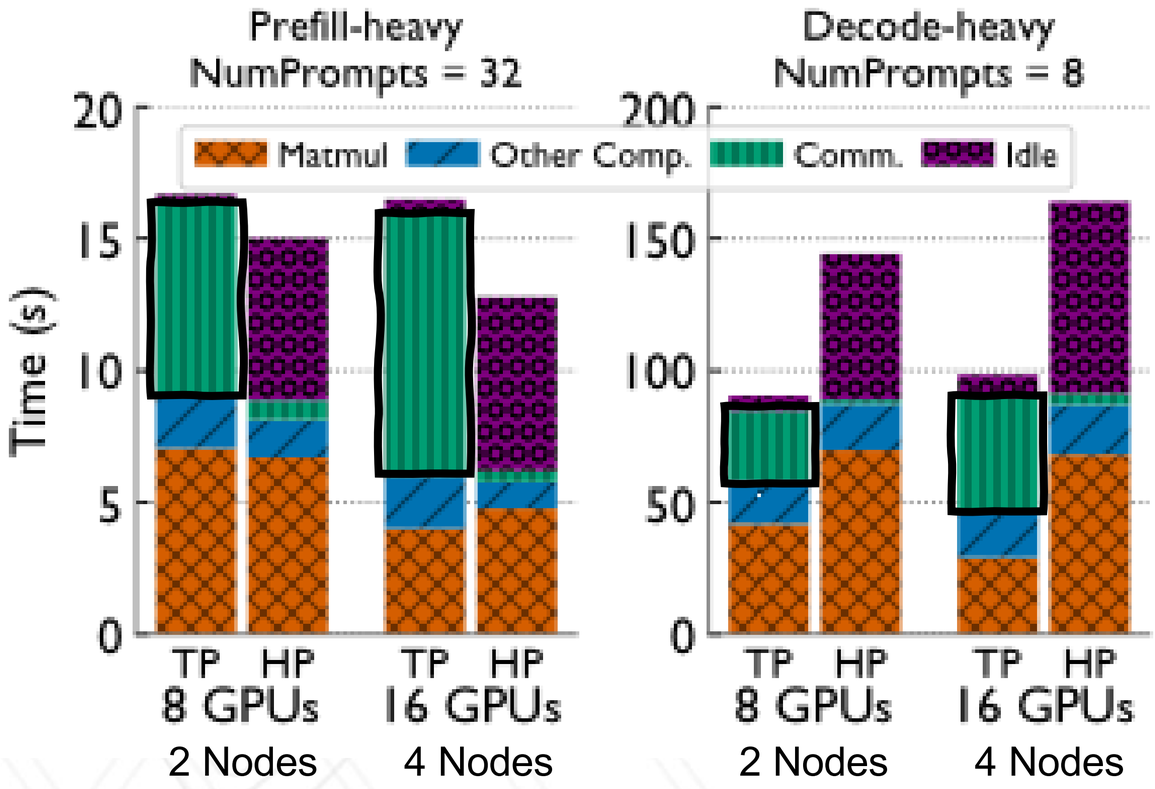
- Prefill-heavy: Computation time decreases for both TP and HP when scaling from 8 to 16 GPUs.
- Decode-heavy: Computation time decreases only for TP





Performance Bottlenecks

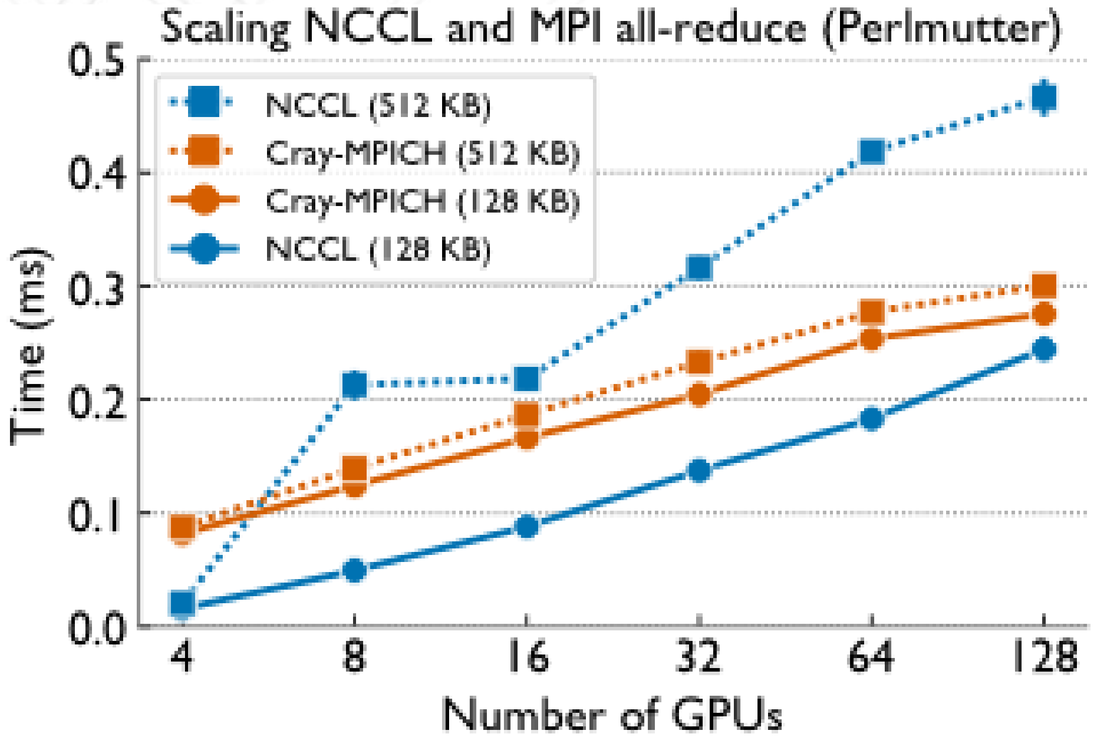
- Prefill-heavy: Computation time decreases for both TP and HP when scaling from 8 to 16 GPUs.
- Decode-heavy: Computation time decreases only for TP.
- TP has high communication overhead for both workloads, but the compute reduction makes it scale better for decode-heavy workloads.



Communication Issues in TP

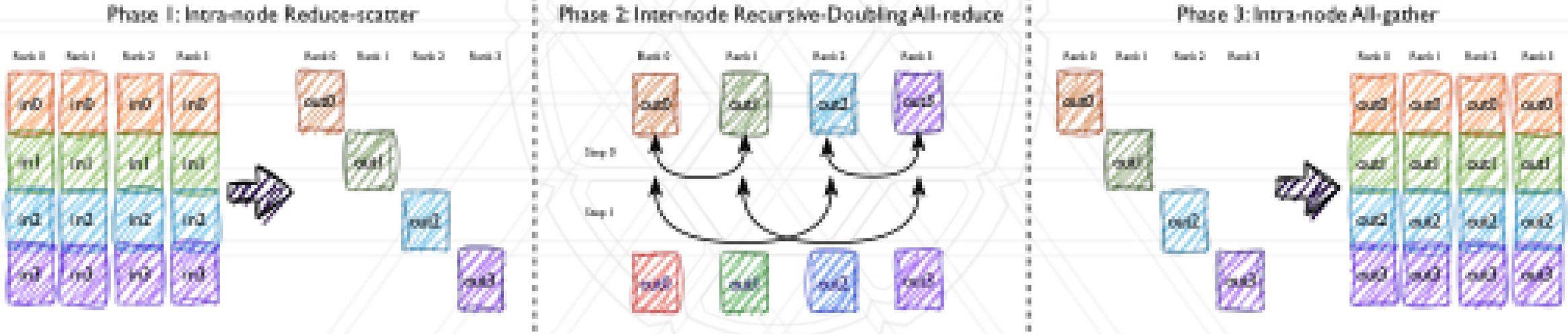
- **Focus**: Decode-heavy regime, where TP is more advantageous.
- All-reduce message sizes:
  - Prefill ↦ ~10 MB – 100 MB
  - Decode ↦ ~10 KB – 1 MB

NCCL all-reduce scales poorly across nodes for decode phase message sizes



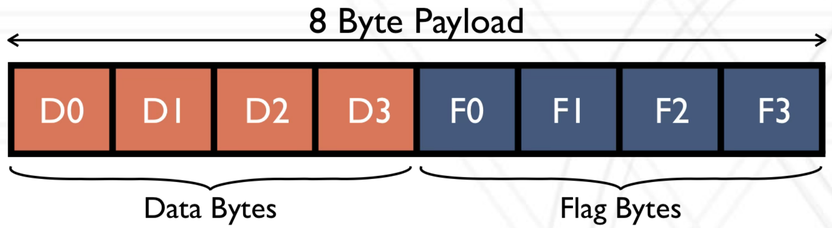
NVRAR: Optimizing All-reduce Communication

- For small message all-reduce, NCCL uses Ring/Tree all-reduce vs. MPICH uses Recursive-doubling.
  - MPI implementations are not CUDA-graph friendly and not optimized over NVLink.
  - Need an approach that is best of both worlds.
- NVRAR: NVSHMEM-based Recursive-doubling All-reduce
  - Hierarchical Design + low-latency optimizations; Compatible with CUDA-graphs.

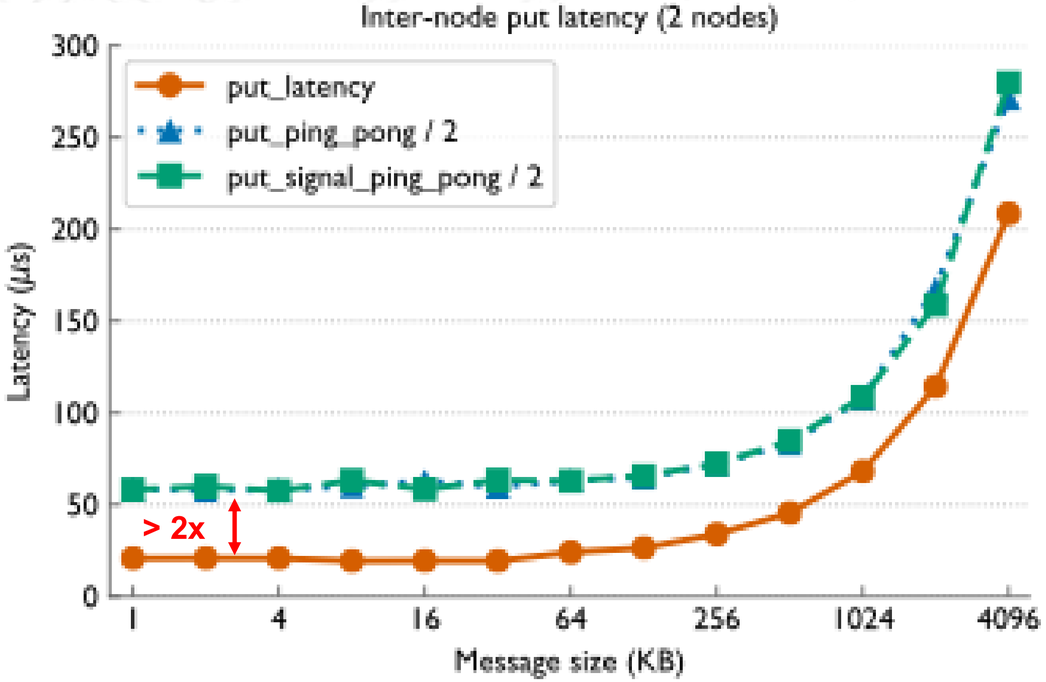


Low Latency Optimizations

A. Fused data and flag payload (NCCL's LL Protocol) to avoid explicit peer sync



- NVSHMEM's explicit signaling API is slow on Slingshot network due to software fences



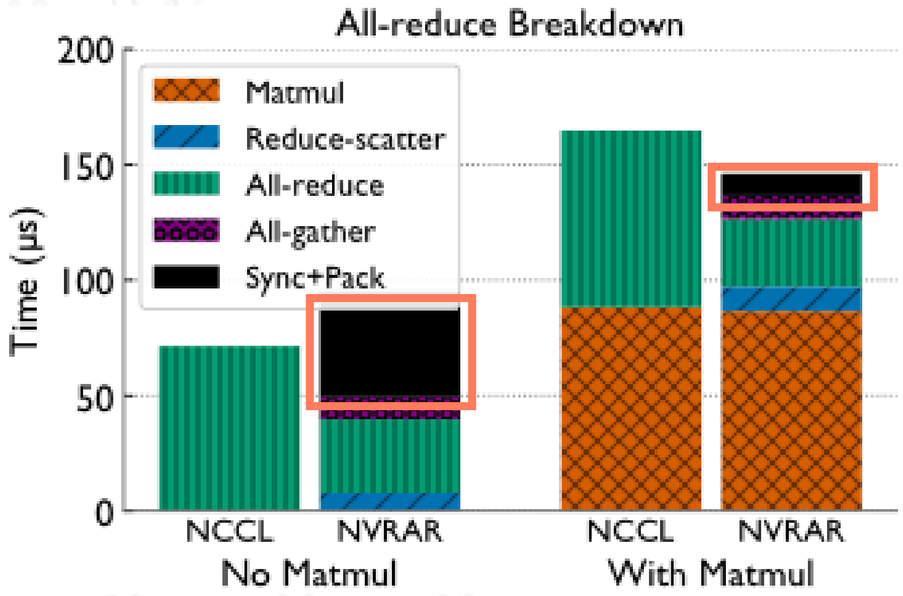
B. Chunked Non-blocking Communication

- Parallel thread-blocks and chunked non-blocking puts enable coarse-grained compute-comm overlap

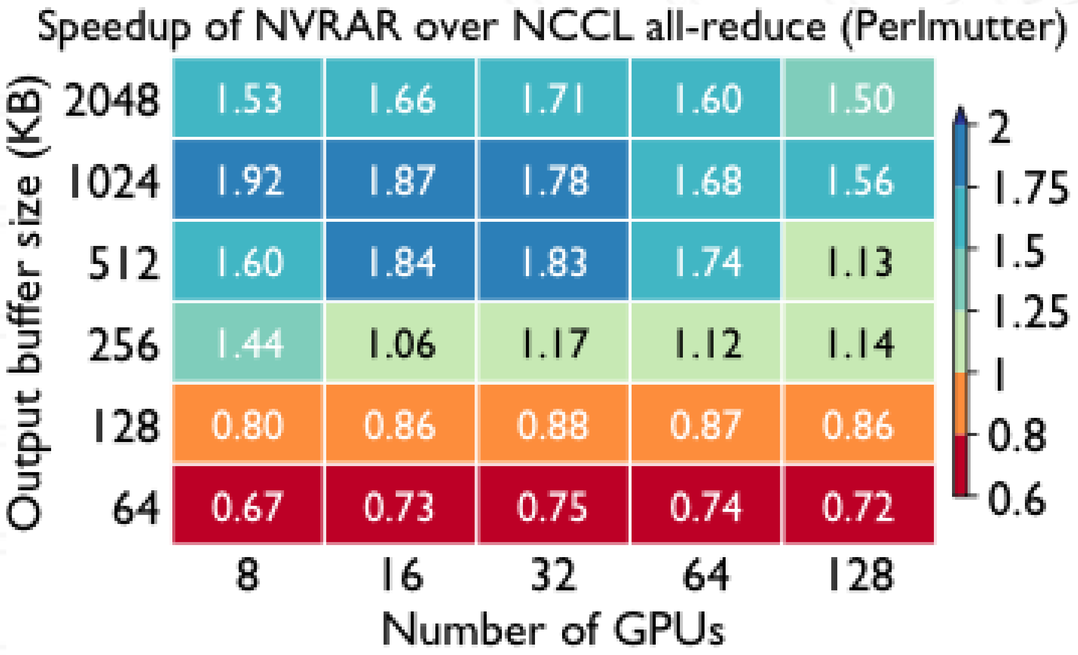
Low Latency Optimizations

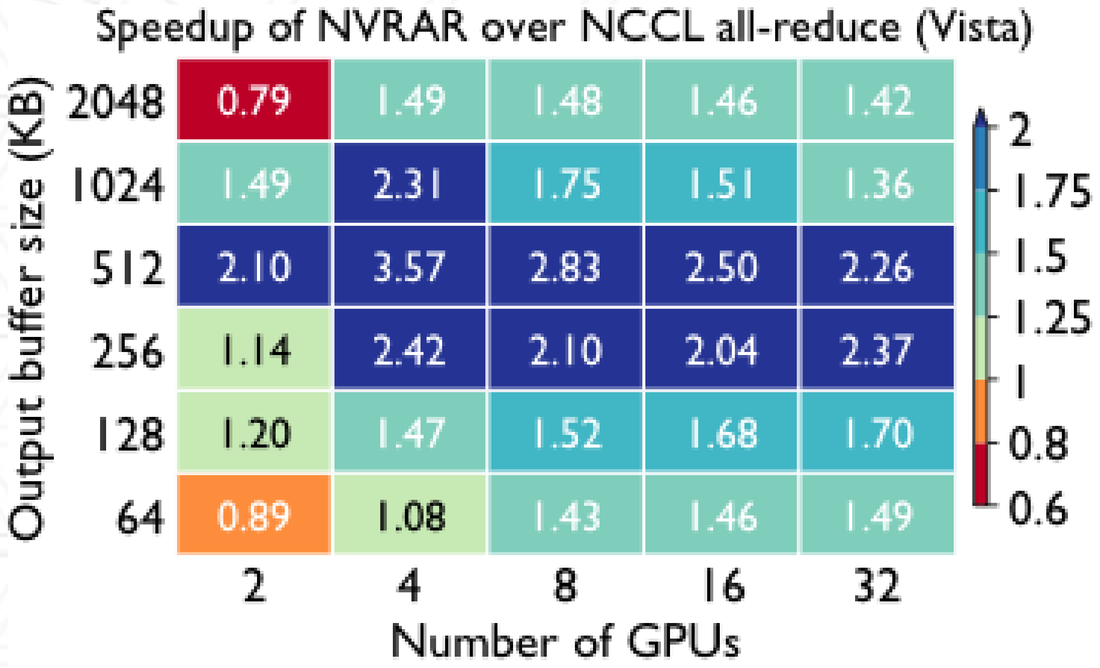
C. Deferred synchronization between successive all-reduce calls without a barrier

- Blocking synchronization at the start of an operation and no barrier at the end
  - Once a rank has it's copy of the globally reduced data, it can start compute
  - Only synchronize with peer ranks, no global synchronization



NVRAR Speedup over NCCL All-reduce





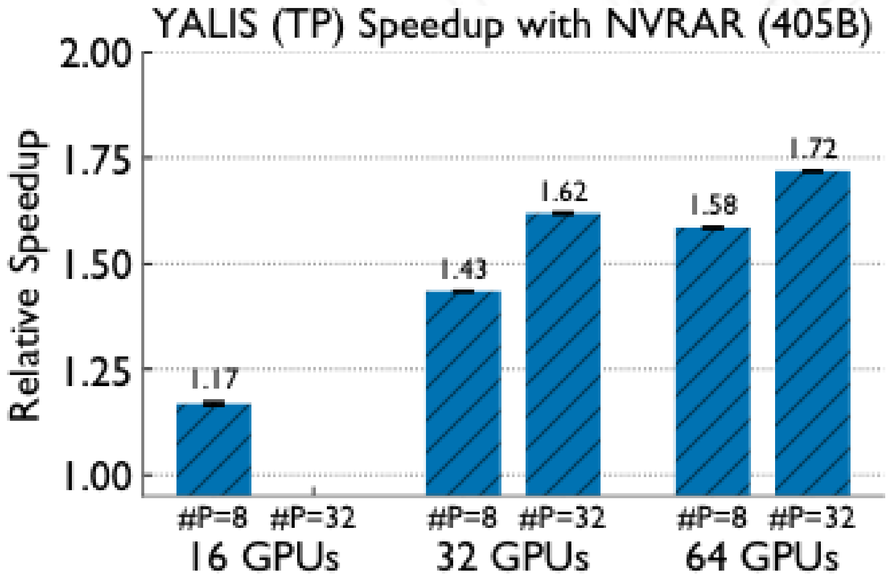
Perlmutter – 4 A100s per node, Slingshot-11

Vista – 1 GH200 per node, InfiniBand

NVRAR achieves lower latency than NCCL all-reduce on both Slingshot & IB networks

Improving Multi-node TP Inference

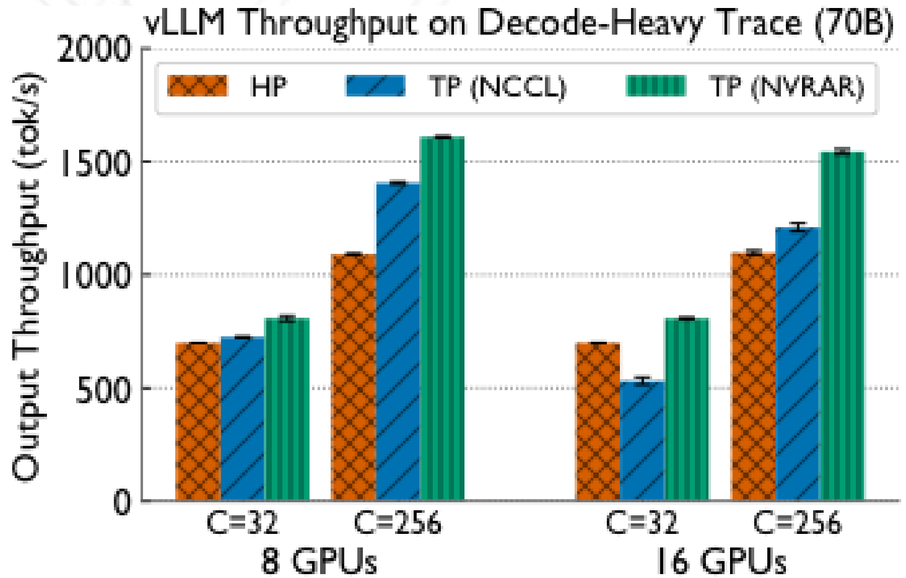
A. Batched Inference Speedups over NCCL-based TP



Decode-heavy Workload

Using NVRAR can significantly speedup batch times for multi-node TP inference

B. vLLM Throughput Improvements for Online Workloads

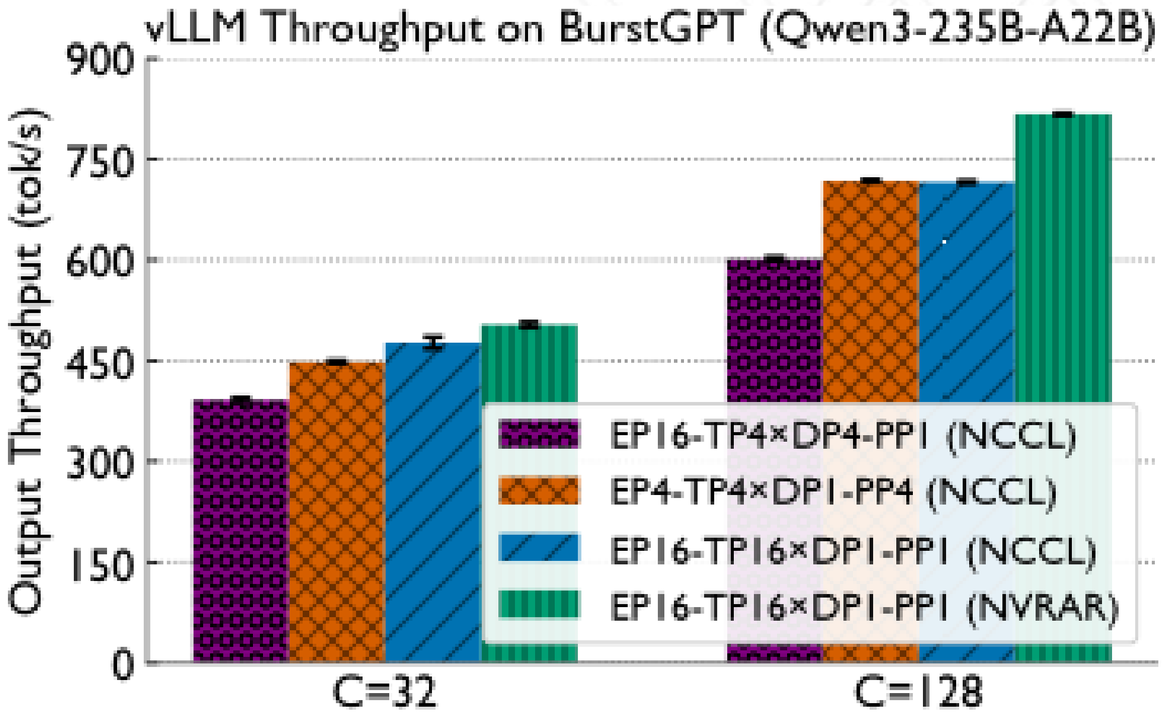


Decode-heavy Trace: Mean ISL =1024, Mean OSL = 4096

Integrated into vLLM, NVRAR improves online serving throughput over best parallel configuration

Improving Multi-node TP Inference

C. Applicability to MoE models



- MoE models use Expert Parallelism for the expert layers.
- Non-expert layers are either TP sharded or DP (data parallel) sharded
  - EP degree = TP degree x DP degree

Using NVRAR for TP sharding in the non-expert layers speeds up MoE inference

Summary

Performance Study

- Compare HP vs TP in multi-node inference deployments.
- Develop **YALIS –** an open and easy-to-instrument prototype inference engine.
- Findings:
  - TP and HP don't scale ideally.
  - Decode-heavy workloads favor TP but communication is the dominant bottleneck.
  - NCCL all-reduce scales poorly across nodes.

Optimizing TP Communication

- **NVRAR**: NVSHMEM-based recursive-doubling all-reduce with low-latency optimizations.

Evaluation

- Significant throughput improvements over NCCL-based TP & HP when integrated into vLLM.

YALIS

NVRAR

Abhinav Bhatele

Parallel Software and Systems Group

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu

github.com/hpcgroup

youtube.com/@hpc_group

x.com/hpc_group

pssg.cs.umd.edu