Understanding Inter-GPU AllReduce Performance through Hardware Telemetry

Ram Sharan Chaulagain Nathan S. Nichols Servesh Muralidharan Kalyan Kumaran
Argonne Leadership Computing Facility, Argonne National Laboratory
Lemont, IL, USA
{rchaulagain, nnichols, servesh, kumaran}@anl.gov

***Abstract*—Inter-GPU AllReduce operations traverse several tightly coupled hardware components, including the GPU, PCIe interface, NIC, and network. Each component exposes hardware telemetry for its local behavior, and no single telemetry source observes the complete communication path. A combined view is therefore needed to narrow where performance is limited and to guide optimization. We combine selected NVIDIA A100 GPU counters with Cassini NIC counters on HPE Slingshot-11 to analyze a 1 GiB inter-node AllReduce across NCCL protocols, NCCL channel counts, and GPU-aware MPI versus NCCL. Across the three studies, the joint telemetry associates performance differences with distinct behaviors: work amplification for NCCL LL, increased synchronization and reduced transfer progress at high channel counts, and slow transfer progress with additional rank imbalance for MPI. Joint GPU–NIC telemetry therefore helps distinguish different sources of AllReduce slowdown and narrows where optimization effort should be focused.**

***Index Terms*—AllReduce, MPI, NCCL, GPU telemetry, NIC counters, Slingshot**

I. Introduction

Inter-GPU AllReduce is central to distributed AI and HPC, and communication libraries provide several ways to execute it through collective algorithms, transfer protocols, and different degrees of software parallelism. These choices affect packet traffic, communication steps, synchronization, and data movement across the GPU, PCIe interface, NIC, and network. Performance limitations can therefore emerge at different points along this path. Hardware telemetry provides visibility into the behavior of these components, but counters from each device capture only a local view. This disjoint view makes it difficult to localize a slowdown, which remains a key challenge for performance analysis and optimization.

NVIDIA Nsight Compute captures GPU execution, memory traffic, PCIe transfers, and warp stalls, while Cassini counters expose NIC traffic, commands, address translation, latency, and per-rank variation. We apply both views to controlled 1 GiB AllReduce experiments on Polaris and ask a common question in three settings: *Which hardware behaviors distinguish faster and slower AllReduce configurations, and where along the communication path do these differences appear?*

Prior work characterizes NCCL protocol and collective-algorithm behavior [1]. Our focus is on how software configurations appear jointly in GPU and NIC telemetry. We examine NCCL protocols, NCCL channel counts, and GPU-aware MPI versus NCCL. Across these studies, the combined measurements expose different hardware behaviors associated with each performance gap. The accompanying poster presents the full counter panels and per-rank distributions; this summary presents the cross-layer evidence used to interpret those differences.

II. Method

Polaris nodes contain four NVIDIA A100 GPUs and two Cassini NICs connected by HPE Slingshot-11 [2], [3]. We use one MPI rank, one GPU, and one NIC per node so that measured traffic remains inter-node. Experiments span 2, 4, 8, and 10 nodes and reduce 1 GiB of double-precision data. The stack is Cray MPICH 9.0.1 with GPU Transport Layer support, NCCL 2.26, CUDA 12.9, libfabric 2.2.0rc1, and aws-ofi-nccl 1.9.1. Every run verifies the reduction and uses 100 warmup iterations.

We isolate one software parameter in each study. For protocols, Ring and the channel count remain fixed while we compare Simple, LL, and LL128. Simple is bandwidth-oriented; LL uses 4 B data + 4 B flag, and LL128 uses 120 B data + 8 B flag [1], [4]. For channels, Ring and Simple remain fixed while we sweep 1–32 channels, which partition the input across concurrent communication streams. For the library comparison, Cray MPICH debug output identifies Rabenseifner's reduce-scatter/allgather [5], [6], whereas NCCL selects Ring. Rabenseifner adds pre- and post-processing for non-power-of-two process counts.

We first use end-to-end latency to establish the performance difference between configurations. We then examine GPU and NIC telemetry together, comparing communication work, transfer progress, and per-rank balance. We interpret counter totals together with transfer rates and quantities that remain unchanged across configurations. Changes and invariants across the GPU and NIC layers help narrow where the slowdown appears. When hardware counters expose the effect but not the software operation that caused it, we use software logs and performance models to provide the missing context.

III. Results

A. NCCL Protocols: More Work for LL

At 8–10 nodes, LL requires about 550–560 ms, versus 110–115 ms for Simple and 125–130 ms for LL128. Despite this gap, LL has the lowest NIC request–response latency: most LL events fall in 0.256–0.640 μs bins, while Simple and LL128 are concentrated near 1.0–1.5 μs. Lower per-event NIC latency

therefore does not correspond to lower collective latency for this 1 GiB message.

The NIC counters show substantially more work for LL. Its 4 B data + 4 B flag format reduces payload efficiency [1]; dominant transmit bins contain roughly twice as many flits as Simple and LL128. LL also issues about $20\times$ more GET commands than Simple, more rendezvous-PUT commands, and more ATU misses, evictions, and NIC engine-cache misses. The GPU view shows a corresponding path change: LL writes 3.87 GB to host memory and 4.35 GB over PCIe, versus 2.19 GB for LL128 and 2.06 GB for Simple. Its PCIe write rate is only 2.04 GB/s, compared with 11.3 GB/s for LL128 and 14.6 GB/s for Simple. Joint telemetry therefore associates LL's gap with work amplification, additional host/PCIe movement, and slow transfer progress.

B. NCCL Channels: Parallelism without Faster Progress

Two channels provide the lowest latency at every tested scale. At ten nodes, latency is about 118 ms with two channels, 144 ms with one, and 190 ms with 32. Active SMs rise almost one-to-one with channel count, from about 1 to 31.7, but SM throughput changes only modestly. CTA-barrier stalls increase sharply and dominate the high-channel stall profile.

PCIe write volume remains nearly constant (2.05–2.12 GB), and NIC packet volume changes little. PCIe write rate peaks at two channels at 15.2 GB/s and falls to about 11.5–12.0 GB/s at 16–32 channels; NIC posted-write blocking also increases. More channels therefore increase parallelism, but beyond the optimum, higher barrier stalls, lower PCIe transfer rates, and increased NIC blocking limit communication progress.

C. MPI versus NCCL: Slow Progress and Rank Imbalance

Cray MPICH selects Rabenseifner's reduce-scatter/allgather algorithm, whereas NCCL selects Ring. At power-of-two node counts, the communication model predicts similar algorithmic costs, yet MPI is 13–20% slower at 2–8 nodes.

At eight nodes, NIC traffic is balanced across ranks and the MPI and NCCL request–response distributions largely overlap. PCIe volume is also identical: MPI reads/writes 2.35/2.00 GB, compared with 2.36/2.00 GB for NCCL. The difference is transfer progress. MPI sustains 9.58–10.39 GB/s reads and 8.14–8.83 GB/s writes, whereas NCCL reaches 17.4–18.7 and 14.6–15.9 GB/s, respectively. MPI also has lower SM throughput and is dominated by long-scoreboard stalls. Thus, at eight nodes, similar communication work progresses more slowly with MPI.

At ten nodes, Rabenseifner requires additional pre- and post-processing for the two excess ranks. The communication model predicts this additional cost, consistent with the measured MPI latency becoming about 90% higher than NCCL. NIC traffic and PCIe volume also become rank dependent: some MPI ranks move substantially more data than others, while NCCL remains comparatively uniform. MPI's average PCIe write rate drops to about 6.05 GB/s, compared with about 14.6 GB/s for NCCL.

The GPU stall profiles show substantial waiting for both implementations at ten nodes and do not isolate a single stall category that explains the large gap. The combined evidence instead associates the ten-node difference with Rabenseifner's additional non-power-of-two work, rank imbalance, and lower transfer progress. The hardware counters expose these effects, while the algorithm log and communication model identify their software origin.

IV. Discussion and Conclusion

The three studies show why the GPU and NIC views are complementary. LL changes work on both devices. In the channel sweep, NIC traffic remains nearly unchanged while GPU synchronization and PCIe progress change. In the MPI comparison, similar work at eight nodes shifts attention to transfer rate, while per-rank counters expose imbalance at ten nodes. Both changes and invariants are therefore useful for narrowing where a slowdown appears.

Hardware telemetry does not always identify the software operation that caused the observed behavior. Function-level and timeline profiling show where time is spent in software, such as a kernel or communication phase, while hardware telemetry shows how the underlying resources behave during that execution. At ten nodes, for example, counters expose uneven data movement but do not identify Rabenseifner's pre- and post-processing without algorithmic context. Logs, profiling, and performance models therefore complement telemetry.

The selected counters also do not prove saturation of a specific hardware unit: a low PCIe rate indicates slow progress through the measured path, not PCIe saturation. Joint GPU–NIC telemetry narrows the search space for further analysis and helps direct optimization to the relevant part of the communication path.

Acknowledgment

This research used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility operated under contract DE-AC02-06CH11357.

References

[1] Z. Hu, S. Shen, T. Bonato, S. Jeaugey, C. Alexander, E. Spada, J. Dinan, J. R. Hammond, and T. Hoefler, "Demystifying NCCL: An in-depth analysis of GPU communication protocols and algorithms," in *Proc. IEEE Symp. High-Performance Interconnects (HOTI)*, 2025, pp. 48–59, doi: 10.1109/HOTI66940.2025.00024.

[2] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the Slingshot interconnect," in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, Nov. 2020, pp. 1–14, doi: 10.1109/SC41405.2020.00039.

[3] B. Videau. (2024, Oct.) Polaris: Overview of hardware and software. ALCF Hands-on HPC Workshop presentation slides. Argonne Leadership Computing Facility. Accessed: Jul. 29, 2026. [Online]. Available: https://www.alcf.anl.gov/sites/default/files/2024-11/Polaris-Overview-ALCF-HandsOnWorkshop-2024.pdf

[4] NVIDIA Corporation. (2026) NVIDIA Collective Communications Library (NCCL) user guide. NCCL 2.30.7 documentation. Accessed: Jul. 29, 2026. [Online]. Available: https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/

[5] R. Rabenseifner, "Optimization of collective reduction operations," in *Computational Science—ICCS 2004*, ser. Lecture Notes in Computer Science, M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds., vol. 3036. Berlin, Germany: Springer, 2004, pp. 1–9, doi: 10.1007/978-3-540-24685-5_1.

[6] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, 2005, doi: 10.1177/1094342005051521.