Unmasking Performance Variability in GPU Codes on Production Supercomputers

Cunyang Wei, Rishi Keshav Pradeep, Abhinav Bhatele
Department of Computer Science, University of Maryland
College Park, Maryland, USA
{cunyang,keshprad}@umd.edu,bhatele@cs.umd.edu

Abstract

Modern HPC facilities increasingly rely on GPU-accelerated clusters to drive both scientific computing and AI workloads. Performance variability is a critical issue in these systems, undermining efficiency and reproducibility. While prior studies have extensively analyzed variability in CPU-centric supercomputers, large-scale investigations on GPU clusters are lacking. To address this gap, we set up a longitudinal experiment on Perlmutter and Frontier. We benchmark representative HPC and AI applications and collect detailed performance data to assess the impact of compute variability, allocated node topology, and network conditions on overall runtime. We also use a ML based approach to identify potential correlations between these factors and to forecast the execution time. Our analysis identifies network performance as the dominant source of runtime variability. These findings provide crucial insights that can inform the development of future mitigation strategies.

1 Methodology

We conducted extensive experiments on Perlmutter at NERSC and Frontier at OLCF. We repeatedly ran a suite of representative workloads to probe the systems' performance under real-world conditions. These workloads include both traditional HPC applications (AMG2023 [4, 7], MILC[1]) and AI training tasks (nanoGPT [11], DeepCAM[6]). By covering both MPI and NCCL/RCCL [10, 12], our experiments stress the network in ways typical of traditional HPC applications and modern AI workloads, respectively. We observed substantial performance variability across runs. On Perlmutter we observed up to 1.4× variability for nanoGPT, and on Frontier, DeepCAM exhibited up to 3.2× variability.

For each experimental run, we captured comprehensive performance metrics including application runtime, network counters [3], MPI profiles via `mpiP` [13] for HPC applications and using the `PyTorch Profiler` [8] to profile AI applications. We also recorded job scheduler logs to track node allocations, placement topology, and background system load.

2 Analysis of the Data

2.1 Impact of Slow GPUs on Variability

In large-scale parallel workloads, collective communications occur frequently, and overall performance often hinges on the slowest GPU. We examined the correlation between application runtime and the number of GPUs within the job's allocation that fall into the system-wide slowest 1% (based on GEMM performance). As illustrated in Figure 1 (left), our analysis reveals no discernible relationship between the quantity of allocated slow GPUs and runtime. We observed the same lack of relationship when extending this

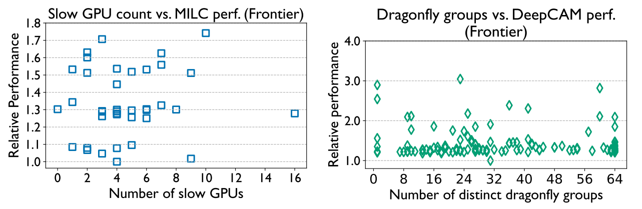

Figure 1: Left: Impact of the number of slow GPUs in our job's allocation on MILC runtime, Right: Impact of the number of dragonfly groups on DeepCAM runtime

investigation to include the slowest 10% and 30% of GPUs. This finding suggests that performance variability likely stems from other factors, such as network congestion rather than from the presence of individually slower computing elements.

2.2 Allocation Spread across Dragonfly Groups

We then investigated whether allocating a larger number of dragonfly [5] groups increases network hops, thereby impacting overall network performance. As shown in Figure 1 (right), our experiments show that the runtime of DeepCAM does not correlate with the number of dragonfly groups. Although increasing the number of dragonfly groups inherently introduces longer network paths and potentially higher hop counts, the performance implications are effectively neutralized in practice, demonstrating the robustness of the underlying network hardware and software layers.

2.3 Impact of Concurrently Running Jobs

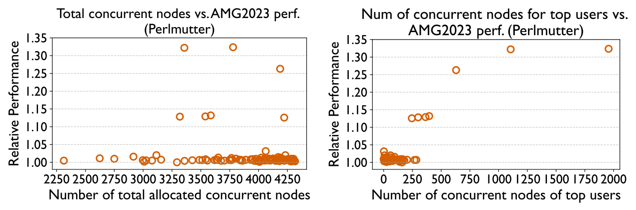

Figure 2: Left: Impact of total number of nodes allocated to relevant jobs on performance variability. Right: Impact of number of nodes allocated to top users on performance variability

Here, we investigate whether the total allocated nodes from all relevant jobs correlate with the runtime of our application. Figure 2 (left) shows that aggregate node allocation alone fails to explain performance variations, likely due to noise introduced by low communication demand tasks. To address this, we perform a refined analysis identifying *Top Users* whose allocated number of nodes correlates with our application's runtime and who concurrently request more than 32 nodes. Our analysis reveals that, as shown in Figure 2 (right), when the concurrent node allocation by these *Top Users* exceeds a certain threshold, our application's runtime consistently increases. For example, on Perlmutter, performance degradation of at least 7% for AMG2023 occurs when *Top Users* collectively occupy over 300 nodes.

3 ML-based Performance Forecasting

We used machine learning methods to investigate key factors contributing to performance variability. We collected logs from multiple runs and extracted metrics including application name, dragonfly group placement, GEMM performance, MPI/NCCL Allreduce performance, and NIC counters [3]. We used XGBoost [2, 9] regression as our primary method, with 10% of data kept for testing and 90% for training.

Feature Importances: On Perlmutter, `hni_rx_paused_0_mean` and `allreduce_2GB` stand out. The first metric tracks cycles where traffic class 0 remains paused on the receive path, suggesting the network pushes data more rapidly than the endpoint NIC can consume. Together with Allreduce performance as a proxy for overall network performance, these features show that NIC congestion and system network congestion are strong predictors of performance variation.

On Frontier, `lpe_net_match_request_0_mean`, `atu_cache_hit_derivative1_page_size_0_mean`, and `parbs_tarb_pi_non_posted_blocked_cnt_mean` dominate the prediction. These metrics track requests matched on software endpoints, cache hits, and cycles where the non-posted path is blocked, revealing that uneven load with respect to data handling drives runtime variability.

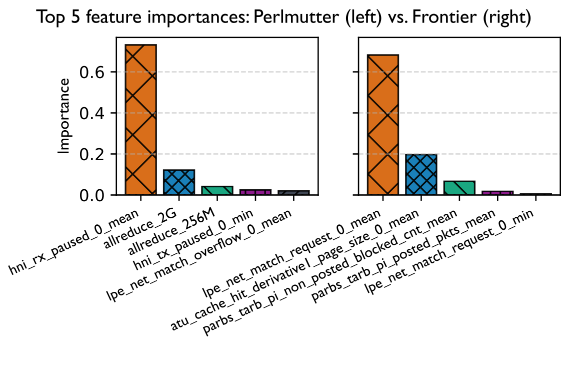

Figure 3: Feature importances based on XGBoost models.

Forecasting Application Performance: Our model predicts absolute performance reasonably well and almost perfectly captures performance variation trends. When including NIC counters, MAPE decreases significantly, especially for DeepCAM. Moreover, models trained across multiple applications exhibited advantages over individual application models, learning common performance variation patterns more robustly.

We consider these findings to be a major highlight of our work. We envision that system administrators and resource managers could use these predictions to schedule workloads more intelligently or detect early signals of congestion. Our approach demonstrates that even partial hardware counters and application-level metrics can reveal critical bottlenecks in high performance computing platforms.

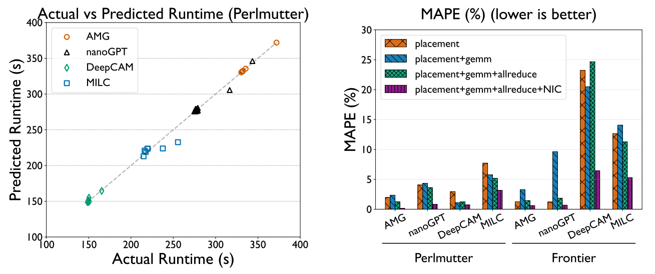

Figure 4: Analysis of XGBoost runtime predictions on Perlmutter and Frontier.

References

[1] Claude Bernard, Tom Burch, Thomas A. DeGrand, Carleton DeTar, Steven Gottlieb, Urs M. Heller, James E. Hetrick, Kostas Orginos, Bob Sugar, and Doug Toussaint. 2000. Scaling tests of the improved Kogut-Susskind quark action. *Physical Review D* 61 (2000).

[2] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. doi:10.1145/2939672.2939785

[3] Hewlett Packard Enterprise. 2024. *HPE Cassini Performance Counters*. https://cpe.ext.hpe.com/docs/latest/getting_started/HPE-Cassini-Performance-Counters.html Accessed: 2025-02-24.

[4] R.D. Falgout, J.E. Jones, and U.M. Yang. 2006. The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners. In *Numerical Solution of Partial Differential Equations on Parallel Computers*, A.M. Bruaset and A. Tveito (Eds.). Vol. 51. Springer-Verlag, 267–294.

[5] J. Kim, W. J. Dally, S. Scott, and D. Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *2008 International Symposium on Computer Architecture*. IEEE Computer Society.

[6] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, and Michael Houston. 2018. Exascale Deep Learning for Climate Analytics. arXiv:1810.01993 [cs.DC] https://arxiv.org/abs/1810.01993

[7] Ruipeng Li and Ulrike M. Yang. 2023. AMG2023. [Computer Software] https://doi.org/10.11578/dc.20230413.1. doi:10.11578/dc.20230413.1

[8] Meta. [n. d.]. PyTorch Profiler. https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html.

[9] Daniel Nichols, Alexander Movsesyan, Jae-Seung Yeom, Daniel Milroy, Tapasya Patki, Abhik Sarkar, and Abhinav Bhatele. 2024. Predicting Cross-Architecture Performance of Parallel Programs. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS '24)*. IEEE Computer Society.

[10] NVIDIA. [n. d.]. NCCL. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/overview.html.

[11] Siddharth Singh, Prajwal Singhania, Aditya Ranjan, John Kirchenbauer, Jonas Geiping, Yuxin Wen, Neel Jain, Abhimanyu Hans, Manli Shu, Aditya Tomar, Tom Goldstein, and Abhinav Bhatele. 2024. Democratizing AI: Open-source Scalable LLM Training on GPU-based Supercomputers. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '24)*.

[12] AMD ROCm Software. [n. d.]. *RCCL*.

[13] Jeffrey Vetter and Chris Chambreau. 2005. mpip: Lightweight, scalable mpi profiling. (2005).