System-Wide Roofline Profiling - a Case Study on NERSC's Perlmutter Supercomputer

Brian Austin, Dhruva Kulkarni, Brandon Cook, Samuel Williams, Nicholas J. Wright
Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
{baustin, dkulkarni, bgcook, swwilliams, njwright }@lbl.gov

***Abstract*—HPC system architects routinely use many forms of application profiling and performance modeling to evaluate hardware and software performance trade-offs. However, the focus on individual applications can leave gaps in the understanding of total system utilization because it is impractical to collect profiles and models for every combination of application and input. In this paper, we use hardware activity metrics data gathered from thousands of GPUs on NERSC's Perlmutter system to perform a roofline performance analysis of the full cross-section of a diverse scientific workload and provide quantitative empirical evidence for widely held beliefs that had previously been inferred from scattered analyses of individual applications. Specifically, we confirm the predominance of double-precision (FP64) floating point operations in scientific computing, responsible for two thirds of the total flop count; single-precision (FP32) accounts for another third while half-precision (FP16) operations are rare. Additionally, the arithmetic intensity for these operations are below the machine balance for 46% of samples and above it for 54%, which suggests near equal fractions of the workload are compute-bound and bandwidth-bound on Perlmutter's GPUs. These results stand in contrast to hardware performance trends where artificial intelligence applications are driving processors to emphasize the performance of reduced-precision operations, and gains in memory bandwidth are not keeping pace with peak processing rates.**

***Index Terms*—Profiling, Monitoring, Roofline**

I. Introduction

Designing supercomputers requires system architects to map the requirements of their workload to the potential performance capabilities of future technologies. For reasonably homogeneous workloads with a small number of applications, it may be possible to evaluate the entire application space through performance models or simulation, but the workloads at many supercomputing centers are far too diverse to be so thorough. Instead, a common alternative practice is to analyze a subset of the applications and make cautious inferences about how the remaining applications would perform on the proposed hardware, but this approach may be skewed by the selection of which codes to analyze or the biases that the architects unavoidably use when making inferences. In this paper, we use passive monitoring of GPU performance metrics to profile the Perlmutter system at NERSC. Analyzing these metrics through the roofline performance model, we obtain an empirical (non-inferential) estimate of the memory bandwidth sensitivity of the entirety of an extremely diverse workload consisting of over 650 distinct applications.

The roofline performance model combines the properties of a computational kernel and the performance capabilities of a processor to determine the maximum achievable performance of the kernel on that processor [1]. Kernels are characterized by their arithmetic intensities (AI), defined as the ratio of the number of operations (typically FLOPs) they execute and the number of bytes they transfer to/from memory. Processors are characterized by their peak operation rates and peak memory bandwidth. The roofline function defines a performance ceiling for the kernel according to its AI:

$$Roofline(AI) = \min\begin{pmatrix} AI \times PeakBandwidth, \\ PeakOps \end{pmatrix}$$

The kernel's performance may approach the roofline if it is well optimized for that architecture.

The roofline model also enables simple diagnosis of which aspects of the system constrain the kernels performance. The machine's balance point, given by the ratio of its peak ops and peak bandwidth, describes the AI that separates the two terms in the roofline function. The performance of well optimized kernels with AI below the machine balance are memory-bandwidth bound on the system and kernels with AI above the machine balance are compute bound.

Methods for determining the AI of arbitrary kernels are often costly or labor intensive. Manual counting of flops and bytes can be tedious, especially for complex kernels. Compiler-instrumentation such as Byfl [2] and binary-instrumentation tools such as Pin [3] have severe runtime overheads. The roofline analysis tools included in several commercially developed profiling tools are easy to use, but can be appieled only on a per-kernel or per-job basis and are not suitable for applying to a very large number of jobs. (Our analysis includes over 135,000 jobs.)

Recently advances in system telemetry expose a far richer view of system activity. These include the availability of interfaces to make GPU performance metrics available outside vendor-provided performance tools and the maturity of data collection infrastructure that is capable of ingesting hundreds of metrics from thousands of nodes at reasonably high sampling rates. This enables a first ever (to our knowledge) system-wide roofline analysis and our contribution of the following key findings.

- Two thirds of the floating point operations on Perlmutter use double-precision (FP64) and one third uses single-precision (FP32). Half-precision (FP16) operations are rare in this scientific computing workload.
- The median of FP64 AI measurements (3.2) is more than an order of magnitude less than the median for

FP32 AI measurements (0.06). The fraction of FP64 AI measurements that exceed the A100 machine balance and have the potential to be compute bound is 38%; for FP32, this fraction is only 21%.

- We define a "pseudo-64" operation that allows the computation of a total AI for all floating point operations. By this metric, 46% of the Perlmutter workload is memory-bound and 54% is compute bound.
- We compare the NERSC-10 benchmark suite to the Perlmutter workload and observe that although the benchmarks replicate the workload's overall balance of memory- and compute-bound samples, effects of using a finite suite are clearly visible in the shapes of the AI distribution.

The remainder of the paper is organized as follows. Section II describes the Perlmutter system and its monitoring infrastructure. Section III presents our roofline analysis of Perlmutter's GPU workload. In Section IV, we compare the full GPU workload to the NERSC-10 benchmark suite. Section V describes related work and Section VI discusses sources of uncertainty in our results.

II. System Description

Perlmutter is a HPE-Cray Shasta supercomputer installed at NERSC in 2019. The system is composed of two types of compute nodes: 3072 CPU-only nodes, and 1792 GPU-accelerated nodes, all connected by a Slingshot network [4]. Details of the Perlmutter architecture can be found online [5].

This work focuses on the GPU-accelerated nodes, each of which has one AMD EPYC 7763 "Milan" CPU [6], 256 GB of DDR4 memory, four NVIDIA A100 "Ampere" GPUs [7], and four HPE Slingshot NICs [4]. In most (1536) of the GPU nodes, the A100s have 40 GB of HBM2 memory, but a smaller number (256) have 80 GB of HBM2. Our analysis is limited to the 40 GB A100s; their peak performance specifications are listed in Table I.

Perlmutter's monitoring infrastructure collects a broad variety for real-time operational needs and post hoc analysis. The GPU performance counters used in this study were sampled using the NVIDIA Data Center Graphics Manager (DCGM) [8], aggregated using the Lightweight Distributed Metric Service (LDMS) [9], and stored in NERSC's Operations Monitoring and Notification Infrastructure (OMNI) [10]. DCGM metrics are sampled at 1 second intervals and indicate the average resource utilization during that interval.

TABLE I
Performance specification for various operations on NVIDIA 40 GB A100 GPUs, DCGM metrics for determining their utilization, and the relative frequency of their occurrence on Perlmutter.

Feature	Peak Performance	DCGM Metric	Relative Frequency
FP16	78 TF/s	`fp16_active`	0.5%
FP32	19.5 TF/s	`fp32_active`	35.9%
FP64	9.7 TF/s	`fp64_active`	33.4%
FP64 Tensor	19.5 TF/s	`tensor_active`	30.2%
HBM2	1.555 TB/s	`dram_active`	N/A

III. Workload Profile

The GPU activity metrics listed in Table I were collected for jobs that ran on Perlmutter's 40 GB GPU nodes during the month of July, 2024. These metrics report the fraction of cycles the FP16, FP32, FP64, tensor pipelines were active, or the fraction of cycles when data was sent to or received from HBM. We compared the metric values to the performance measured by mixbench [11] and mt-gemm [12] to confirm that these metrics can also be interpreted as highly accurate measures of the fractions of peak performance values listed in Table I.

These calibration experiments also revealed that for approximately 15% of the samples, all of the floating-point activity metrics (but not the DRAM activity) were exactly zero, even when running sustained temporally uniform workloads. Thus, we labeled any time samples for which all of the floating point metrics are zero as invalid and excluded them from our analysis.

We estimate the total number of floating point operations by multiplying the activity metrics by the corresponding peak performance, and summing over all samples. The relative number of operations of each type are listed in the final column of Table I. We infer that the tensor activity is predominantly due to FP64 operations because: a) there number of FP16 vector operations is not commensurate with the level tensor activity, b) the tensor cores do not support FP32 operations, and c) TF32 and integer matrix operations are presumably rare among simulation workloads. It is possible that some of the tensor operations could have used reduced precision without corresponding FP16 vector activity, for example by using the mixed-precision matrix operations in cublasLt, but this seems unlikely to be widespread given the relative novelty and lack of standardization for such interfaces. Combining the FP64 vector and tensor activity, FP64 operations constitute roughly two thirds of the workload. The remaining third is FP32, while FP16 operations are rare.

A simple estimate of the AI for FP64 operations can be made using the formula

$$AI_{FP64} = \frac{\texttt{fp64\_active} \times Peak_{FP64}}{\texttt{dram\_active} \times Peak_{HBM2}}$$

Similar formulas for FP32 and tensor operations are easily constructed. This formula will underestimate the AI if multiple types of FP operations occur during the sample window because the denominator may include bytes for other FP types, integers and instructions.

Figure 1 shows the distribution of DCGM samples (nominally one per GPU-second, excluding invalid samples) vs arithmetic intensity for FP32, FP64 and FP64 tensor operations collected from Perlmutter during July, 2024. For comparison, the performance rooflines for these operations are shown on the secondary axis. For all three precisions, the majority of samples have AI values substantially below the machine balance (marked by the dashed vertical line). The AI distribution for FP32 operations is almost always well below the machine balance (marked by the vertical dashed line) which could plausibly be explained by user preferences of FP32 for bandwidth-bound codes. The AI for FP64 operations is

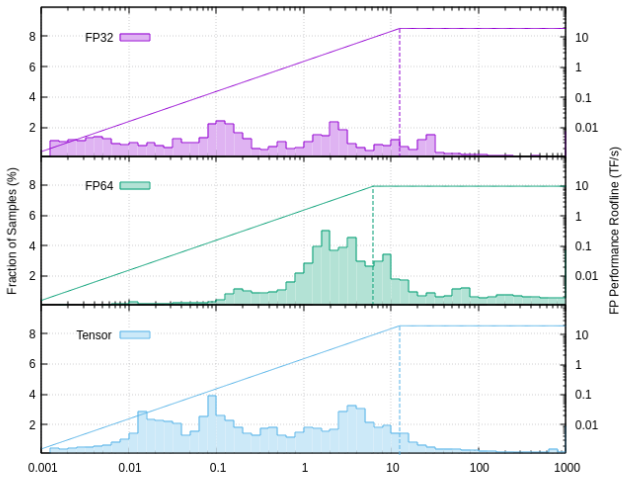

Fig. 1. Algorithmic Intensity distributions for Perlmutter's GPU workload.

qualitatively higher than FP32 operations, has a broad peak near 2 flops/byte, and a significant tail that extends far to the right of the machine balance. It is surprising that the AI for tensor operations is typically below the machine balance because matrix multiplication (the intended purpose of tensor cores) is highly amenable to data caching.

IV. Benchmark Profiles

One of the prevailing approaches to understanding the needs of diverse workloads is to select a set of representative benchmarks from the workload, profile the benchmarks to understand their performance sensitivities, and assume that the hardware requirements of the benchmarks are applicable to the broader workload. In this section, we examine the GPU-enabled benchmarks from the NERSC-10 workflow component benchmark suite and compare their characteristics to the Perlmutter workload.

The benchmark descriptions, source code and inputs were obtained from the NERSC-10 benchmarks website [13]. These benchmarks were selected to span the range of science domains, algorithmic patterns and workflow motifs running at NERSC. The four benchmarks with mature GPU implementations are:

- **MILC** is a lattice quantum chromodynamics (QCD) framework for subatomic physics [14]. The MILC workflow benchmark has two component applications identified as generation and spectrum.
- **BerkeleyGW** models the electronic structure of materials [15]. The BerkeleyGW workflow benchmark has two component applications: Epsilon and Sigma.
- **LAMMPS** is a molecular dynamics application for modeling the physical properties of materials [16].
- **DeepCAM** trains a deep neural network to classify atmospheric phenomena from previously computed climate simulations [17].

Each benchmark was run using the "small" problem size defined in the benchmark distributions, and configured to use

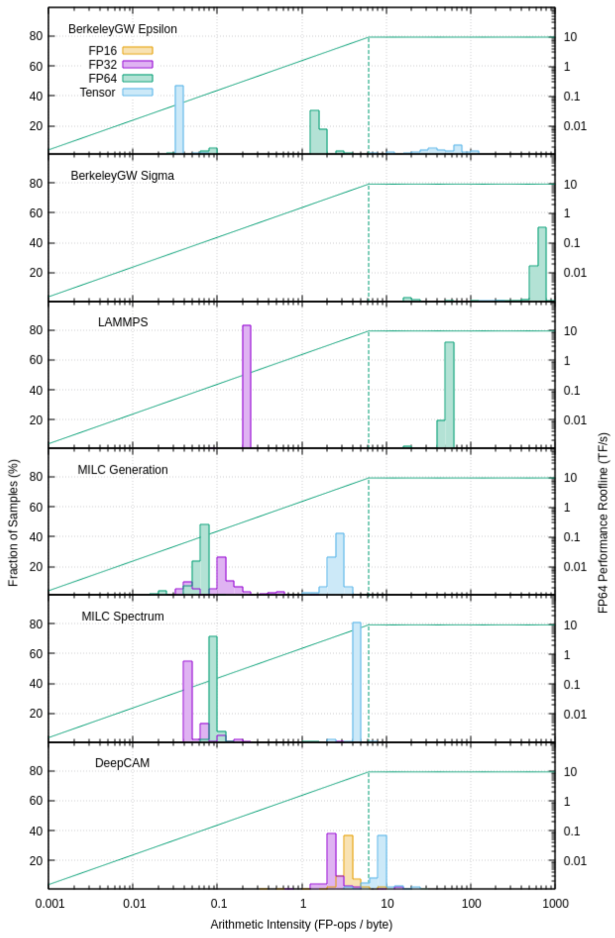

Fig. 2. Algorithmic Intensity distributions for the NERSC-10 GPU benchmarks.

4 GPUs on a single node. The DCGM command line interface was used to collect the same metrics named in Section III.

The benchmarks' distribution of DCGM samples vs. AI is shown in Figure 2. The benchmarks have distinct and diverse signatures with respect to FP types and AI. Both MILC applications, generation and spectrum, have a mix of FP32 that are near the machine balance but memory-bandwidth bound, and FP64 vector and tensor operations that are strongly bandwidth bound. BerkelyGW-Epsilon uses double precision exclusively, with memory-bound FP64 vector operations and FP64 tensor operations associated with matrix inversion that are compute-bound. BerkeleyGW-Sigma uses only FP64 vector operations, but with very high AI. LAMMPS FP64 vector operations are clearly to the right of the machine balance, while its FP32 vector operations are to the left. DeepCAM is the only benchmark that uses FP16 and the only benchmark that does not use FP64. Thus, counter to the arguments in Section III, tensor activity for DeepCAM is assumed to use FP16. DeepCAM's FP16 and FP32 operations are near the A100 machine balance, but bandwidth-bound, while its FP16 tensor operations are compute-bound. Across all of the benchmarks, the peaks of the AI distributions are

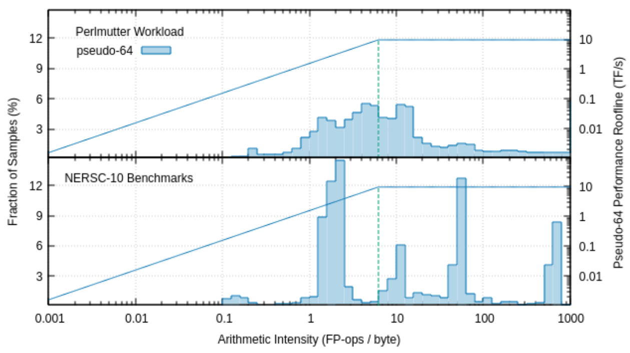

Fig. 3. Pseudo-64 Algorithmic Intensity distributions for Perlmutter's GPU workload and NERSC-10 benchmarks.

much narrower than the full workload shown in Figure 1, which suggests that although the benchmarks may represent a few important uses of the supercomputer, they do not exemplify every pattern and proportion of usage, as would be expected.

In order to compare the benchmarks to the workload using a single metric, we define the "pseudo-64" operation count as a weighted sum of 0.25x the FP16 operations, 0.5x the FP32 operations, and 1.0x the FP64 and tensor operations; which is a proxy for FPU utilization scaled by FP64 rates. Figure 3 shows the distribution of samples among pseudo-64 AI measurements for the Perlmutter workload and the average of the NERSC-10 Benchmarks. The two distributions are qualitatively different from each other. The Perlmutter workload has a single wide peak close to the machine balance with equal fractions of samples on the memory-bound and compute-bound sides of the roofline. The NERSC-10 Benchmark distribution spans the same range of AI values and also has a near equal division between memory-bound and compute-bound samples, but the samples are clustered into several narrow peaks.

V. Related Work

A large body of work uses roofline analysis to understand application performance. See References [18]–[22] for a limited sample. In all of these examples, the roofline model is applied to jobs on an individual basis. Our approach uses the roofline model to learn about the entire workload, with no attention given to individual jobs.

System monitoring research is also relevant. Most of this work focuses on either the efficient aggregation of metrics [9], [10]. Practical use of this data is typically focused on operational questions such as system health and utilization; and does not address questions of performance.

Earlier HPC workload analyses from NSF centers [23], NCAR [24] and NERSC [25] have focused largely on identifying and classifying applications and the schedulable resources they request from the job scheduler. Where workload analyses have examined utilization of resources within job allocations, it has been limited to memory capacity. They have not examined rates (memory bandwidth or FLOP performance) and have not compared those rates to a performance model. To the best of our knowledge, this paper is the first presentation of a system-wide roofline performance analysis.

VI. Discussion

Several sources of uncertainty should be considered when interpreting these results. The floating point activity metrics count any activity the pipelines and our simple estimate of the floating point count assumes the entire warp is active, but this may not be true if some threads within the warp are masked. Further, the estimate is based on the peak floating point performance, but the actual floating point count could be a factor of two lower if the code cannot take advantage of fused multiply-add pipes. Both of these effects would cause the estimated AI to be artificially low. The low sampling rate (1 Hz) may mute bursts of high floating point or memory by averaging over a long time-window, thus causing moments of particularly high (or low) AI to go undetected. Last, we have ignored many samples when all of the floating point activity metrics were exactly zero due to an unidentified problem with DCGM. A consequence of this filtering is that we are unable to distinguish between data collection errors and times when the activity is truly zero due to I/O or communication. Understanding the effect of these errors requires additional research.

Acknowledgment

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a Department of Energy Office of Science User Facility using NERSC award ASCR-ERCAP0026875. ROR: https://ror.org/05v3mvq14

References

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, apr 2009.
[2] S. Pakin and P. McCormick, "Hardware-independent application characterization," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, 2013, pp. 111–112.
[3] "Pin - A Dynamic Binary Instrumentation Tool ," https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html, 2013, Accessed: 8-31-2024", organization = "Intel".
[4] "HPE Slingshot," [Online]. Available: https://www.hpe.com/psnow/doc/PSN1012904596USEN, Accessed: 2024-08-22.
[5] "Perlmutter, a HPE Cray EX system," https://docs.nersc.gov/systems/perlmutter/architecture/, Accessed: 2024-08-22.
[6] "AMD EPYC 7763," https://www.amd.com/en/products/processors/server/epyc/7003-series/amd-epyc-7763.html, Accessed: 2024-08-22.
[7] "NVIDIA A100 Tensor Core GPU Architecture," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/A100-PCIE-Prduct-Brief.pdf, Accessed: 2024-08-22.
[8] "NVIDIA DCGM," https://developer.nvidia.com/dcgm, Accessed: 2024-08-22.
[9] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, and T. Tucker, "The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 154–165.

[10] E. Bautista, M. Romanus, T. Davis, C. Whitney, and T. Kubaska, "Collecting, Monitoring, and Analyzing Facility and Systems Data at the National Energy Research Scientific Computing Center," in *Workshop Proceedings of the 48th International Conference on Parallel Processing*, ser. ICPP Workshops '19. New York, NY, USA: Association for Computing Machinery, 2019.

[11] E. Konstantinidis and Y. Cotronis, "A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling," *Journal of Parallel and Distributed Computing*, vol. 107, pp. 37–56, 2017.

[12] "mt-gemm," https://gitlab.com/NERSC/nersc-proxies/mt-gemm, NERSC, 2021, Accessed: 7-27-2024.

[13] "NERSC-10 Benchmarks," https://gitlab.com/NERSC/N10-benchmarks, 2023, Accessed: 8-31-2024", organization = "NERSC".

[14] MIMD Lattice Collaboration and Bernard, C and others, "The MILC Code," 2010.

[15] J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, "BerkeleyGW: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures," *Computer Physics Communications*, vol. 183, no. 6, pp. 1269–1289, 2012.

[16] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Computer Physics Communications*, vol. 271, p. 108171, 2022.

[17] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, Prabhat, and M. Houston, "Exascale deep learning for climate analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. IEEE Press, 2018.

[18] M. Javed, K. Ibrahim, and X. Lu, "Performance analysis of deep learning workloads using roofline trajectories," *CCF Transactions on High Performance Computing*, vol. 1, 11 2019.

[19] K.-H. Kim, K. Kim, and Q.-H. Park, "Performance analysis and optimization of three-dimensional FDTD on GPU using roofline model," *Computer Physics Communications*, vol. 182, no. 6, pp. 1201–1207, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465511000452

[20] D. Doerfler, J. Deslippe, S. Williams, L. Oliker, B. Cook, T. Kurth, M. Lobet, T. Malas, J.-L. Vay, and H. Vincenti, "Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor," in *High Performance Computing*, M. Taufer, B. Mohr, and J. M. Kunkel, Eds. Cham: Springer International Publishing, 2016, pp. 339–353.

[21] M. Wittmann, G. Hager, R. Janalik, M. Lanser, A. Klawonn, O. Rheinbach, O. Schenk, and G. Wellein, "Multicore Performance Engineering of Sparse Triangular Solves Using a Modified Roofline Model," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018, pp. 233–241.

[22] B. Mostafazadeh, F. Marti, F. Liu, and A. Chandramowlishwaran, "Roofline Guided Design and Analysis of a Multi-stencil CFD Solver for Multicore Performance," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018, pp. 753–762.

[23] N. A. Simakov, J. P. White, R. L. Deleon, S. M. Gallo, M. D. Jones, J. T. Palmer, B. D. Plessinger, and T. R. Furlani, "A Workload Analysis of NSF's Innovative HPC Resources Using XDMoD," *ArXiv*, vol. abs/1801.04306, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:13400191

[24] "Cheyenne Workload and Usage Analysis," https://www.cisl.ucar.edu/sites/default/files/2021-10/UCAR_RFP000074_Attachment_5_Cheyenne_Workload_and_Usage_Analysis_v1.1.pdf, NCAR, 2020, Accessed: 8-31-2024.

[25] B. Austin, W. Bhimji, R. Gerber, M. Mustafa, C. Daley, H. He, R. Thomas, T. Davis, K. Konate, C. Whitney, S. Farrell, G. Lockwood, N. Wright, and Z. Zhao, "NERSC-10 Workload Analysis," https://portal.nersc.gov/project/m888/nersc10/workload/N10_Workload_Analysis.latest.pdf, NERSC, 2020, Accessed: 8-31-2024.

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 Two thirds of the floating point operations on Perlmutter use double-precision (FP64) and one third uses single-precision (FP32). Half-precision (FP16) operations are rare in this scientific computing workload.

C_2 The median of FP64 AI measurements (3.2) is more than an order of magnitude less than the median for FP32 AI measurements (0.06). The fraction of FP64 AI measurements that exceed the A100 system balance and have the potential to be compute bound is 38%; for FP32, this fraction is only 21%.

C_3 We define a "pseudo-64" operation that allows the computation of a total AI for all floating point operations. By this metric, 46% of the Perlmutter workload is memory-bound and 54% is compute bound.

C_4 We compare the NERSC-10 benchmark suite to the Perlmutter workload and observe that although the benchmarks replicate the workload's overall balance of memory- and compute-bound samples, effects of using a finite suite are clearly visible in the shapes of the AI distribution.

B. Computational Artifacts

A_1 https://doi.org/10.5281/zenodo.13853536

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1 - C_4	Figures 1-3

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

C_1 is supported by the histograms of the arithmetic intensity shown in Figure 1 (LDMS/LDMS_AI_v3.png). The histograms are provided in tabular form by files LDMS/AI_FP16A_hist.out, LDMS/AI_FP32_hist.out, LDMS_FP64_hist.out and LDMS/AI_TENSO_hist.out. The total flop counts for each type can be estimated by integrating over the histogram.

C_2 is also supported by Figure 1 and its tables. The tabular histograms include the cumulative distribution functions (CDF) not shown in the figure. The median AI values correspond to the AI at which the CDF equals 50%. The fraction of compute-bound codes corresponds to the value of the CDF at the machine balance point.

C_3 is supported by Figure 3 (LDMS/LDMS_PS_v2.png)and its tables (LDMS/AI_PS64A_hist.out,LDMS/N10bench.d1000_PS64A.hist). The fraction of compute-bound codes corresponds to the value of the CDF at the machine balance point.

C_4 is supported by Figure 3 and visual comparison of the upper and lower panels.

Expected Results

This was an observational study. There were no experiments with expected results.

Expected Reproduction Time (in Minutes)

There are types of data included in this artifact: "workload" data collected from the Perlmutter system, and "benchmark" data collected by monitoring specific jobs.

The workload data was collected passively and cannot be reproduced. Queries to the OMNI datastore to access the workload metrics are reproducible, but would require privileged system access. The Jupyter notebook for generating the workload histograms is provided in LDMS/LDMS_data.ipnb. Preparing the histograms requires approximately 1 hour on a single core of a Perlmutter login node.

The benchmark data can be reproduced with approximately 1 hour of setup time per benchmark and less than ten minutes of execution time per benchmark on one Perlmutter node.

Artifact Setup (incl. Inputs)

Hardware: This work was performed on Perlmutter, a HPE-Cray Shasta supercomputer installed at NERSC in 2019. The system is composed of two types of compute nodes: 3072 CPU-only nodes, and 1792 GPU- accelerated nodes, all connected by a Slingshot network. This work focuses on the GPU-accelerated nodes, each of which has one AMD EPYC 7763 "Milan" CPU [6], 256 GB of DDR4 memory, four NVIDIA A100 "Ampere" GPUs, and four HPE Slingshot NICs. In most (1536) of the GPU nodes, the A100s have 40 GB of HBM2 memory, but a smaller number (256) have 80 GB of HBM2. Our analysis is limited to the 40 GB A100s.

Software: Four applications from the NERSC-10 Benchmark suite were run on Perlmutter: MILC, BerkeleyGW, LAMMPS and DeepCAM. The benchmark suite is available online: https://gitlab.com/NERSC/N10-benchmarks

MILC is a lattice quantum chromodynamics (QCD) framework for subatomic physics. The MILC code can be obtained from https://github.com/milc-qcd/milc_qcd.git. We used commit 13ffa851. The GPU version of MILC relies on the QUDA library, which can be obtained from https://github.com/lattice/quda.git We used commit 35b57df9f. The instructions to compile, inputs and data needed to run and performance expectations are described in the benchmark website: https://gitlab.com/NERSC/N10-benchmarks/lattice-qcd-workflow/

BerkeleyGW models the electronic structure of materials. The BerkeleyGW project website, https://berkeleygw.org/ includes a public download option. We used a custom snapshot of their code-base that can be obtained from https://portal.nersc.gov/project/m888/nersc10/benchmark_data/BGW_input/ The instructions to compile, inputs and data needed to run and performance expectations are described in the benchmark website: https://gitlab.com/NERSC/N10-benchmarks/berkeleygw-workflow

LAMMPS is a molecular dynamics application for modeling the physical properties of materials. The LAMMPS code can be obtained from https://github.com/lammps/lammps.git. We used commit 7d5fc356fe. The instructions to compile, inputs and data needed to run and performance expectations are described in the benchmark website: https://gitlab.com/NERSC/N10-benchmarks/exaalt

DeepCAM trains a deep neural network to classify atmospheric phenomena from previously computed climate simulations. We used the following DeepCAM container: registry.nersc.gov/das/deepcam-opt:23.09.01. DeepCAM depends on numerous packages that are included in the aforementioned container. The instructions to compile, inputs and data needed to run and performance expectations are described in the benchmark website: https://gitlab.com/NERSC/N10-benchmarks/deepcam

Datasets / Inputs: The inputs and data sets for the NERSC-10 Benchmarks identified above can be obtained by following the instructions in each benchmark's web page, which are listed above.

The raw LDMS counters collected from the full-system workload are not publicly distributed due to their large size. Their histograms are provided in the LDMS directory of the artifact.

Installation and Deployment: Instructions for compiling the NERSC-10 Benchmarks are included in each benchmark's web page, which are listed above. For MILC and LAMMPS, the GNU compliler collection, version 12.3.0 was used. BerkeleyGW was compiled with the NVIDIA HPC SDK version 8.5.0. In all the above cases, the NVIDIA cuda compiler version 12.2 was used to compile cuda code. For DeepCAM, no additional compilation was needed because the executable is distributed in a container.

The batch scripts used to run each benchmark have been modified to start the dcgm daemon before each job. This is accomplished by adding a wrapper script, wrap_dcgm.sh, provided in the scripts directory of the artifact. to the job-launch (e.g. srun) command. The scripts used to launch the jobs are included in the artifact and are named dcgmi_*.sh.

Artifact Execution

Execution and post-processing of each benchmark has a similar workflow. Benchmark jobs are submitted using a batch script named with the pattern. ⟨benchmark⟩/dcgmi_*.sh (for example: LAMMPS/dcgmi_small_A100.sh). Standard output for the job is included in the artifact. Counters are written to a file named with the pattern ⟨benchmark⟩/dcgm.d1000.⟨job-id⟩.out. Histograms are produced by the script scripts/dcgmi_hist.py and written to a separate file for each floating point type: ⟨benchmark⟩/dcgm.d1000.⟨job-id⟩.out_⟨FPtype⟩.hist. Figure 3 requires an additional histogram (N10bench.d1000_PS64A.hist) that combines the individual benchmarks; this is generated by the script N10pseudo.py.

All of the figures are generated from the histogram files using the command 'gnuplot -c ¡script¿', where script corresponds to one of:

Figure 1:	LDMS/LDMS_AI.gp
Figure 2:	N10bench.gp
Figure 3:	LDMS/LDMS_PS.gp