© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

A. Khan, N. Hagerty, A.M. Karimi, M.E. Rowan, J. Lange, M. Ezell, C. Zimmer, and S. Atchley, "SORT: Slurm-Orchestrated ROCm Telemetry for Facility-Wide GPU Observability on HPC Systems," in SC26-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2026.

https://doi.ieeecomputersociety.org/10.1109/SCW72768.2026.00391

SORT: Slurm-Orchestrated ROCm Telemetry for Facility-wide GPU Observability on HPC Systems

Awais Khan[1], Nick Hagerty[1], Ahmad Maroof Karimi[1], Michael E. Rowan[2], Jack Lange[1],
Matthew Ezell[1], Christopher Zimmer[1], Scott Atchley[1]
[1]Oak Ridge National Laboratory, TN, USA, [2]Advanced Micro Devices, Inc., USA
{khana, hagertynl, karimiahmad, langejr, ezellma, zimmercj, atchleyes}@ornl.gov, michael.rowan@amd.com

***Abstract*—Nearly 300 projects use Frontier at the Oak Ridge Leadership Computing Facility each year, with approximately 30% turnover in the project population. These projects execute heterogeneous applications across CPUs and GPUs, and their computational requirements evolve as new workloads enter the facility. Planning next-generation systems requires OLCF leadership to better understand accelerator usage across the facility workload population, which existing data sources capture poorly. We present SORT, a Slurm-Orchestrated ROCm Telemetry framework that uses traditional prologue and epilogue hooks to control a ROCprofiler-SDK-based telemetry collector over the lifetime of a Slurm allocation. SORT integrates ROCprofiler-SDK counter collection with the Slurm allocation lifecycle and joins compact counter artifacts with scheduler metadata for offline analysis. Deployed on Borg, Frontier's test and development system, SORT collected 88,269 valid counter files associated with 67,373 jobs. Controlled evaluation shows minimal runtime overhead for most workloads and bounded telemetry output of approximately 16–21 KB of logical data per job at full sampling scale. The resulting telemetry supports population-level characterization of instruction, memory-request, and FP64 activity and can help identify candidate jobs for deeper profiling.**

***Index Terms*—Frontier, GPU telemetry, HPC observability, Performance counters, ROCprofiler, Slurm,**

I. INTRODUCTION

Leadership-class HPC systems increasingly rely on large GPU allocations to support scientific simulation, data analytics, and machine-learning workloads [1], [2]. Systems such as Frontier execute a heterogeneous mixture of tightly coupled MPI applications, single-node throughput workloads, multi-stage workflows, system tests, and short development jobs [3], [4], [5]. This diversity makes it difficult for facilities to characterize GPU execution behavior across the full job population. Schedulers such as Slurm [4] record requested resources, allocation size, placement, and execution metadata, but these records describe allocation intent rather than device activity observed during execution [4]. Fine-grained application profiling can expose kernel behavior and performance bottlenecks, but broad continuous deployment is constrained by data collection cost, output volume, operational complexity, and dependence on selected applications or user participation. Consequently, facilities lack a lightweight, job-attributed observability layer between scheduler accounting and detailed application profiling.

This need may become more pronounced as emerging numerical techniques, such as the Ozaki scheme [6], [7], realize higher-precision computation through combinations of lower-precision operations. Such approaches can alter the numerical-format and instruction-path mix observed across GPU workloads, increasing the value of facility-wide characterization for understanding workload evolution and informing future system design. Production monitoring commonly combines scheduler accounting, node-level logs, application logs, power and utilization telemetry, and vendor-specific monitoring tools such as NVIDIA DCGM [8], [9] and ROCm™-based interfaces [10]. These sources support longitudinal utilization analysis, power studies, resource modeling, and system-health monitoring [11], [12], [13], [14], [15], [16]. However, coarse node-level telemetry often omits job-attributed hardware-counter information about selected instruction classes, memory requests, and numerical execution paths. Attribution can also become ambiguous when an allocation contains multiple job steps or when multiple processes and GPU agents contribute to host-level measurements. Job-attributed device information can help facilities characterize the numerical formats and execution paths observed across their workload populations and identify jobs that warrant deeper investigation.

Application-oriented tools instead attach profilers, collect execution traces, or sample detailed hardware counters during selected executions. Tools such as DrGPU, LEO, and low-overhead GPU tracing frameworks support bottleneck analysis, stall attribution, and kernel-level performance investigation [17], [18], [19]. These methods are valuable for targeted performance engineering, but their collection cost, output volume, application involvement, and analysis complexity complicate uniform deployment across an entire job population. Facility-managed application instrumentation can also introduce operational complexity. Intercepting application launches or modifying user command paths can interact with existing profilers, launch wrappers, and complex job workflows. A facility-managed telemetry service must preserve application behavior, tolerate collection failures, and avoid requiring users to modify application source code or launch

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (https://www.energy.gov/doe-public-access-plan).

SC26 Workshops, November 15-20, 2026, Chicago, Illinois, USA
979-8-3195-1221-5/26/$31.00 ©2026 IEEE

commands.

A middle ground is lightweight, job-scoped GPU telemetry that records final cumulative hardware-counter snapshots for post-hoc analysis. ROCprofiler-SDK exposes AMD GPU hardware counters, while Slurm provides the allocation lifecycle and accounting metadata needed to associate those measurements with scheduler-managed jobs. Combining these sources supports broad workload characterization without modifying application source code or collecting continuous kernel-level traces. The resulting telemetry is coarser than detailed profiling, but it can characterize selected instruction paths, conservative memory-request proxies, and metric-specific activity coverage across sampled hosts. When collected consistently and joined with scheduler metadata, these signals can support operational triage by grouping jobs according to broad observed behavior and identifying potentially unusual records for deeper profiling.

To address this gap, we present SORT, a Slurm-Orchestrated ROCm Telemetry framework for job-scoped GPU observability. SORT uses Slurm prologue and epilogue hooks to initiate and finalize a ROCprofiler-SDK-based collector over the allocation lifetime. The collector produces compact job-host artifacts containing final cumulative counter snapshots, validates profiling sessions, and preserves the normal application launch path. An offline analysis pipeline joins the artifacts with Slurm accounting metadata for workload characterization and operational triage. SORT is intended for population-level characterization and retains final cumulative counter snapshots instead of execution traces or time-series measurements.

Over longer deployments, consistent measurements of numerical formats, execution paths, and memory-request behavior may support workload-aware system planning. The present study validates the collection workflow and demonstrates population-level workload characterization on Borg, the test and development system for Frontier at OLCF. Borg provides a controlled environment for validating the collection lifecycle, counter configurations, failure handling, and analysis workflow. The reported workload population is specific to Borg; deployment and validation at full Frontier scale remain future work.

This paper makes the following contributions:

1) We present SORT, a Slurm-orchestrated ROCm telemetry framework that binds low-overhead GPU counter collection to the job lifecycle without modifying application source code or launch commands.
2) We develop a facility-managed and production-oriented collection design with bounded spatial sampling, configurable counter-set partitioning, compact job-host artifacts, and validation of potentially conflicting ROCprofiler sessions.
3) We provide a post-hoc analysis pipeline that joins ROCm counter records with Slurm accounting data and derives coarse workload classes, conservative compute-to-memory proxies, and FP64 activity-coverage indicators.
4) We evaluate SORT on Borg using controlled workloads and telemetry from 67,373 jobs, demonstrating minimal runtime overhead for most evaluated workloads, bounded telemetry output, and population-level characterization of the observed Borg workload.

II. Background and Related Work

GPU workload characterization commonly combines facility-scale monitoring with application-level profiling [20], [21]. Scheduler accounting systems such as Slurm provide allocation and execution metadata, while GPU monitoring and profiling tools expose utilization, power, instruction activity, memory behavior, and execution stalls. Prior approaches differ primarily in collection scope, attribution granularity, output volume, and intended use.

A. Related Work

A longitudinal study of Perlmutter GPU telemetry characterized spatial and temporal imbalance, burstiness, and behavioral differences between machine-learning and traditional HPC workloads [11]. Other existing work combined Slurm accounting records with NVIDIA DCGM measurements to predict GPU utilization, memory use, and power consumption in heterogeneous HPC systems [12]. Production-cluster telemetry has also been used to identify execution-idle periods in which GPUs consume substantial power despite low measured activity [13]. These studies demonstrate the value of broad longitudinal telemetry for utilization, power modeling, imbalance detection, and operational planning, but primarily focus on utilization, power, and temporal behavior rather than compact job-host characterization using instruction and memory-request counters.

A second body of work targets detailed diagnosis of individual applications and kernels. DrGPU uses a top-down profiling methodology to map hardware-counter measurements to bottleneck categories and optimization guidance [17]. LEO applies cross-vendor backward slicing to trace GPU stalls to source-level causes across NVIDIA, AMD, and Intel devices [18]. Low-overhead tracing work has developed instrumentation techniques for collecting detailed GPU-kernel execution records while reducing profiling cost [19]. Instruction-based roofline modeling for AMD GPUs uses ROCprofiler counters to characterize application compute and memory behavior [22]. These approaches provide greater diagnostic detail, including kernel attribution, stall causes, and optimization guidance, but typically require selected profiling runs, application-aware collection, richer output, or analysis at process and kernel granularity.

SORT occupies the space between longitudinal facility monitoring and application-level performance diagnosis. It uses Slurm to orchestrate ROCprofiler-SDK counter collection over the allocation lifetime and joins compact job-host snapshots with scheduler metadata. Its design combines bounded spatial sampling, counter-set partitioning, profiling-session validation, and Slurm-based attribution. SORT builds on existing ROCprofiler-SDK counter interfaces rather than introducing a new hardware-counter or kernel-level bottleneck model.

B. Borg: Proof-of-Concept Test Cluster

Borg, the test and development system for Frontier, was used to demonstrate the proof-of-concept of this telemetry framework. Borg is one half of an HPE/Cray EX cabinet that is identical to Frontier and has up to 64 nodes. Each Borg compute node contains one AMD EPYC™ processor and four AMD Instinct™ MI250X accelerators. Borg uses a software stack that is closely aligned with Frontier, including ROCm, the ROCprofiler-SDK, and Slurm resource management. Borg's exact node count fluctuates, as Borg's compute blades may be moved to Frontier in place of blades that have a longer-lead time repair or need further triage.

One significant difference between Borg and Frontier is connection to the Lustre file system, Orion. Frontier's Slingshot network is directly integrated with Orion, while Borg is connected by three router nodes and may experience higher latency and lower bandwidth as a result.

Borg's workload will fluctuate dramatically over time depending on several factors: (1) how many blades from Frontier are being triaged in Borg, (2) what changes are being tested (e.g., updates to Slurm, operating system), and (3) other R&D work being performed on Borg. In a 3-month span closely aligned with the presented data in this work, nearly one third of the compute jobs can be attributed to node triage and another third to system testing and debugging. Both of these workloads change significantly over time.

Borg's system testing workload is managed by the OLCF Test Harness, the test orchestration framework used for acceptance and regression testing at OLCF [23].

C. ROCm Counter-Collection Constraints

Deploying ROCm-based device-counter collection across HPC systems introduces several practical constraints. First, the hardware restricts how many counters can be configured simultaneously within a performance-monitoring block. On the target AMD GPU platform, the sequencer (SQ) block supports at most eight simultaneously configured counters [24]. Collecting the desired FP64, FP32, FP16, and BF16 instruction counters therefore requires multiple counter configurations. Combining the resulting observations introduces an explicit sampling assumption: locations assigned different configurations must execute sufficiently homogeneous work for coarse job-level interpretation. Second, in the deployed ROCprofiler-SDK interface, the collector does not receive a direct notification when another process initiates a competing profiling session. SORT therefore monitors the monotonicity of `GRBM_COUNT` as an indirect validation signal.

Device-wide collection outside the application process scope also requires elevated counter-access privileges on the deployed system, complicating operation under normal user credentials. Linux capabilities such as `CAP_PERFMON` may be assigned to the collector binary, but this approach depends on file-system support for extended attributes and may require staging the executable on a compatible local file system. In the Borg deployment, SORT instead operates as a scheduler-managed privileged collector.

The validation interval introduces another tradeoff. More frequent polling can detect counter resets sooner but increases collection activity, whereas less frequent polling may delay detection of a competing profiling session. Together, these constraints motivate SORT's counter-set partitioning, session validation, privilege-aware deployment, and scheduler-controlled collection lifecycle. Section III describes how the framework addresses these requirements.

III. SORT Design and Integration

A. Design Goals

SORT is designed for lightweight, job-attributed GPU telemetry at facility scale, prioritizing broad coverage, compact output, and reliable attribution over kernel-level or time-resolved analysis.

- **Application transparency:** Collection must preserve the normal application launch path and require no source-code, binary, or command-line modification.
- **Scheduler-bound lifecycle:** Collection must be initiated and finalized through the Slurm allocation lifecycle so that every artifact can be associated with a job and sampled host.
- **Low operational cost:** Collection must limit runtime overhead and produce compact artifacts suitable for repeated deployment across a large job population.
- **Bounded representative coverage:** The sampling policy must distribute observations across large allocations rather than relying only on the first node, while keeping the number of collectors and artifacts within a fixed operational budget.
- **Valid and interpretable records:** The framework must detect potentially conflicting profiling sessions, retain counter-configuration context, distinguish unavailable counters from zero activity, and exclude measurements that cannot be interpreted reliably.
- **Deterministic artifact semantics:** Each output record must clearly identify its job, sampled host, counter configuration, validation status, and the scope of the retained counter values.

To satisfy these goals, SORT separates scheduler-side control from device-side counter collection. The framework emits job-host artifacts with deterministic naming, while metric derivation, workload classification, and Slurm enrichment are performed offline. Figure 1 shows an overview of the proposed SORT framework.

B. Spatial Sampling Across Slurm Allocations

Large HPC applications may distribute different computational components across rank groups and node regions. These components may execute different kernels, use different numerical formats, and exhibit different instruction and memory-request behavior. For large allocations, sampling only the first node or GPU may therefore provide an unrepresentative view of the job. SORT uses a bounded spatial-sampling policy that distributes profiling groups across the Slurm allocation while limiting the total number of collectors.

The prologue expands the allocation node list, determines each node's position within the allocation, and computes the placement of the profiling groups from the allocation size. For an allocation containing N nodes, SORT selects up to $G_{\max} = 16$ profiling groups. When G groups are used, their nominal spacing is

$$S \approx \frac{N}{G}$$

where $1 \leq G \leq G_{\max}$. The prologue converts the nominal locations to valid node indices according to its deterministic placement policy.

For allocations with sufficient node count, each profiling group uses three neighboring node positions. Config-0, Config-1, and Config-2 are assigned to the first, second, and third positions, respectively. A fully populated layout therefore uses at most

$$16 \text{ groups} \times 3 \text{ configurations} = 48 \text{ profiling locations}$$

For example, a 1,881-node allocation using 16 profiling groups has a nominal group spacing of

$$S \approx \frac{1{,}881}{16} \approx 118 \text{ nodes}$$

Each group places the three complementary counter configurations on nearby nodes, while successive groups are separated by approximately 118 nodes. This distributes the bounded set of profiling locations across the spatial extent of the allocation rather than concentrating them near its beginning.

a) Small-allocation fallback: Allocations that do not provide three distinct node positions cannot apply the three-configuration layout entirely across hosts. In this case, SORT assigns Config-0, Config-1, and Config-2 to the first three eligible profiling processes, each bound to a distinct GPU agent within the available allocation. For example, in a single-node job using all eight GPU agents, the first three eligible profiling processes collect Config-0, Config-1, and Config-2 from three distinct agents. The remaining five agents execute normally but are not directly sampled. The job therefore provides coverage of all three counter configurations while directly observing only three of its eight GPU agents.

The small-allocation fallback prioritizes counter-set coverage over complete GPU coverage. More generally, different configurations are collected from different nodes or GPU agents and therefore do not represent simultaneous measurements from one device. The resulting characterization assumes that sampled locations are sufficiently representative of the job; heterogeneous rank roles, pipeline parallelism, or asymmetric communication may violate this assumption.

C. Counter-Set Partitioning

ROCm hardware limits the number of counters that can be configured simultaneously within the same performance-monitoring block. On the target AMD GPU platform, the SQ block supports at most eight simultaneously configured counters within one profiling configuration [25]. Because the desired metric set contains more SQ counters than one profile can support, SORT partitions the counters across multiple external configuration files. Each selected profiling process loads one counter configuration at startup and binds its collection context to the corresponding GPU agent. This partitioning expands coverage across numerical formats and selected execution paths while keeping each device-level profile within the hardware counter limit.

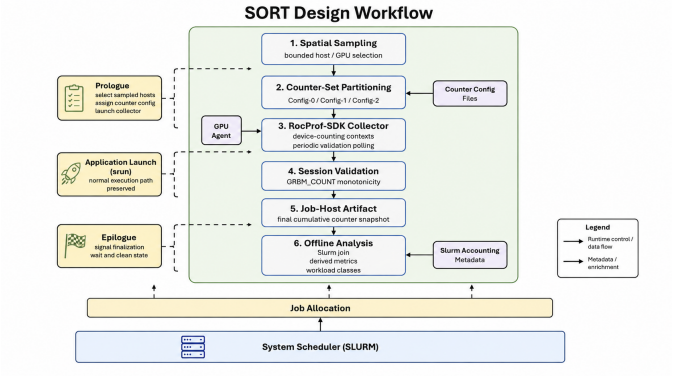


Fig. 1: An overview of the proposed SORT framework.

The current deployment uses three counter configurations:

- **Config-0** captures selected FP64 VALU addition, multiplication, and FMA instruction counters, FP64 MFMA activity, and selected TCC read-request counters.
- **Config-1** captures selected FP32 VALU addition, multiplication, and FMA instruction counters together with FP32 MFMA activity.
- **Config-2** captures selected FP16 VALU addition, multiplication, and FMA instruction counters together with FP16 and BF16 MFMA activity.

Each configuration also includes `GRBM_COUNT`, which is used as an indirect validation signal for possible profiling-session interference. An observed decrease in `GRBM_COUNT` marks the corresponding session as potentially invalid.

a) Configuration assignment: Configuration placement follows the spatial-sampling policy described in Section III-B. Because different configurations may be collected from different nodes or agents, downstream analysis treats them as complementary sampled observations rather than an exact per-GPU instruction mix.

D. Slurm-Orchestrated Collection Lifecycle

SORT binds counter collection to the Slurm allocation lifecycle through scheduler *Prologue* and *Epilogue* scripts as shown in Figure 1. This design allows the facility to initiate and finalize collection without modifying user applications, launch commands, or individual `srun` invocations. On each allocated node, the prologue reads scheduler-provided metadata, including `SLURM_JOBID`, `SLURM_JOB_NUM_NODES`, `SLURM_NODELIST`, and `SLURMD_NODENAME`. The prologue expands the node list and determines the local node's index within the allocation. It then applies the spatial sampling policy described in Section III-B and, for selected locations, assigns the corresponding counter configuration and launches the

collector. Because node selection and configuration assignment depend only on the allocation node list and node index, the resulting placement is deterministic for a given allocation. Note that the nodes that are not selected by the sampling policy return from the prologue without starting a collector. After launch, the collector records its process identifier in node-local storage using the Slurm job identifier as part of the file name. This state allows the epilogue to locate the collector associated with the completing allocation. At allocation completion, the epilogue reads the collector PID from node-local storage and sends `SIGUSR1` to request finalization. The collector treats `SIGUSR1`, `SIGTERM`, and `SIGINT` as controlled shutdown requests. On receipt, it stops sampling, validates the session state, and writes the accumulated counter artifact. The epilogue waits for finalization and removes the node-local PID file after the collector has exited or the configured wait period has expired. The scheduler hooks are designed to avoid disrupting the user workload. Failure to select a node, load a configuration, locate a PID file, or finalize a collector is recorded as a telemetry failure and does not modify the application launch path. The job therefore continues or terminates according to normal Slurm behavior even when counter collection is unavailable. The hooks operate at allocation scope rather than individual job-step scope. Consequently, a job containing multiple `srun` steps produces an accumulated artifact covering all activity observed between prologue initialization and epilogue finalization. The current implementation does not separate counter values by Slurm step. Each finalized artifact is named and stored using the Slurm job identifier and host identity, providing deterministic job-host attribution for downstream analysis.

E. Device-Level Counter Collection and Reduction

SORT uses the ROCprofiler-SDK device-counting interface to collect hardware counters from the GPU agents visible on each selected host. At startup, the collector discovers all visible GPU agents and creates one device-counting context for each agent. On MI250X systems, each graphics compute die is exposed as a separate ROCm agent, so one selected host may contribute measurements from multiple agents. For each counter listed in the selected configuration file, the collector queries the counters supported by the target agent and constructs the corresponding ROCprofiler profile configuration. The resulting profile configurations are cached and reused during periodic sampling, avoiding repeated profile construction in the collection loop.

At startup, the collector reads the selected configuration file and interprets each nonempty line as a counter name. After initialization, the collector records an initial sample and polls the configured counters once per second until it receives a controlled shutdown signal. Intermediate samples are used for session validation and are not retained as a time series. A single ROCprofiler counter query may return records indexed by hardware dimensions such as compute unit or shader engine. For each sample, the collector reduces records sharing the same counter name across the returned hardware dimensions. At finalization, the collector writes the dimension-reduced cumulative counter values for each counter and GPU agent.

These values indicate that selected classes of activity occurred during the allocation, but they do not reveal which kernel, process, or Slurm step generated that activity. At finalization, the collector writes the accumulated values together with the counter configuration, host identity, job identifier, and validation status to the job-host artifact. Performing reduction inside the collector and retaining only the final cumulative snapshot bounds output size independently of job duration and avoids storing a continuous counter trace.

a) Device-level reduction and finalization: SORT enumerates all GPU agents visible through ROCprofiler-SDK and creates one device-counting context per agent. A daemon process receives one counter configuration, and that same configuration is applied to every GPU agent visible to the process. ROCprofiler may return multiple records for one counter because a counter can contain several hardware dimensions. For each GPU agent and sampling point, the daemon groups records by counter name and sums their values across those dimensions. This reduction produces one value per selected counter for the sampled agent.

SORT samples counters once per second during the allocation. Intermediate samples are used to validate the profiling session by checking the monotonicity of `GRBM_COUNT`; they are not stored as a time series or summed across sampling intervals. When the epilogue sends `SIGUSR1`, the daemon takes a final sample and writes the dimension-reduced counter values to the job-host artifact. The resulting values are final accumulated hardware-counter snapshots rather than sums of periodic samples.

F. Profiling-Session Validation

Concurrent profiling sessions may reset, reconfigure, or otherwise invalidate the hardware-counter state observed by SORT. In the deployed ROCprofiler-SDK interface, SORT does not receive a direct notification when another process starts a competing profiling session. SORT therefore uses `GRBM_COUNT` as an indirect session-validity signal.

During an uninterrupted collection session, the observed value is expected to be monotonic for each GPU agent. At initialization, the collector records a baseline `GRBM_COUNT` value for each visible GPU agent. At each sampling interval, the collector compares the current value with the previous value for the same agent. An unchanged or increasing value is accepted, while a decrease is treated as evidence that the counter state may have been reset or disturbed. A decrease does not uniquely identify the cause, but it is treated conservatively as possible interference from another profiling session. If a decrease is observed for any monitored agent, the collector marks the corresponding job-host session as invalid. This conservative policy favors data validity over coverage by excluding records whose counter state cannot be trusted. The monotonicity check detects only interference that produces an observable decrease in `GRBM_COUNT`. The

periodic samples used for the monotonicity check are transient validation observations. They are neither retained as a time series nor added together; only the final cumulative snapshot is written to the artifact. It cannot prove that an accepted session was free from all counter conflicts or other collection errors. The validation status is retained in the output artifact and later used by the dataset-construction pipeline to exclude invalid records.

IV. Experimental Methodology

A. Borg Deployment Environment

SORT was deployed on Borg, the Frontier Test and Development System at OLCF, described in Section II-B. SORT runs with system-level privileges, with Slurm prologue and epilogue hooks controlling its lifecycle and ROCprofiler-SDK providing access to configured GPU counters.

B. Controlled Evaluation Methodology

We evaluate SORT's runtime overhead, telemetry volume, and collection reliability using workloads from the OLCF Test Harness, the system-testing framework used for acceptance and regression testing at OLCF [23]. The harness includes full scientific applications, collaborative user-provided tests, synthetic benchmarks, and microbenchmarks, including applications such as AthenaPK [26], WarpX [27], and MFC [28], [29]. For each workload, we compare two collection modes:

1) **No profiling**, in which the application executes without ROCprofiler-based counter collection;
2) **SORT**, in which Slurm prologue and epilogue hooks manage the system-level collector over the allocation lifetime.

Each workload and collection mode is executed five times. We report the mean runtime across repetitions. Runtime is measured using the elapsed time of the primary Slurm job step. We additionally record total Slurm allocation time as a secondary measurement because it may include variability from scheduler-side operations, initialization and finalization, and Borg's storage path to Orion.

C. Production Dataset Construction

The production dataset was constructed from job-scoped counter artifacts collected on Borg. Each artifact is associated with a Slurm job identifier, host, counter configuration, and validation status. The construction pipeline parses these records, excludes unusable artifacts, groups valid records by job, and joins the resulting data with available Slurm accounting metadata.

The initial corpus contained 67,373 numeric job directories, each treated as a candidate Slurm job. The scan discovered 88,959 counter files. Artifacts that were empty or explicitly marked invalid by the profiling-session validation mechanism were excluded, leaving 88,269 valid counter files associated with 67,373 jobs, as summarized in Table I. For jobs with multiple valid counter files, each job-host record is preserved separately and linked through the common Slurm job identifier. This preserves spatial information across sampled hosts while allowing job-level aggregation when required by a particular analysis. The assigned counter configuration is also retained so that measurements are interpreted only within the corresponding counter set.

TABLE I: Summary of the collected Borg counter dataset.

Dataset Metric	Count
Jobs with valid counter data	67,373
Counter files discovered	88,959
Valid counter files	88,269
Invalid or empty counter files	690
Jobs containing one counter file	55,645
Jobs containing multiple counter files	11,728
Maximum counter files associated with one job	48

TABLE II: Elapsed-time distribution of the Borg jobs.

Elapsed-Time Bucket	Jobs	Jobs (%)	Cumulative Jobs
$\leq$ 1 min	65,936	97.87	65,936
1–5 min	139	0.21	66,075
5–30 min	238	0.35	66,313
30–120 min	1,059	1.57	67,372
$>$ 2 hr	1	$<$ 0.01	67,373
Total	**67,373**	**100.00**	**67,373**

The resulting dataset supports two analysis levels. The job-host level preserves the sampled host and counter configuration of each valid artifact, while the job level associates all valid records with common Slurm accounting metadata. Analyses of spatial coverage operate on job-host records, whereas population-level workload analyses use job-level groupings.

D. Dataset Characteristics

a) Job-duration distribution: Table II shows that 97.87% of the retained jobs complete within one minute. The dataset therefore primarily reflects Borg development, testing, and workflow-fragment activity rather than sustained Frontier production workloads. These short executions may disproportionately capture initialization, early termination, failures, or partial workflow behavior rather than sustained application phases.

b) Counter-file multiplicity: Among the retained jobs, 82.59% provide a single sampled location and 17.41% provide observations from multiple sampled locations. Single-file jobs support coarse instruction-path and memory-request characterization but cannot reveal variation across sampled hosts. Multi-file jobs enable analysis of activity coverage and spatial variation when comparable counter configurations are available. File multiplicity does not necessarily indicate repeated observations of the same counter set because sampled hosts may be assigned different configurations. Analyses requiring a particular metric therefore use only records whose assigned configuration contains the corresponding counters. For metric-specific coverage, records assigned other configurations are treated as unavailable rather than inactive.

E. Derived Metrics and Workload Classification

We derive job-level metrics from the validated counter records for coarse workload characterization. Because counter

configurations are distributed across sampled hosts, these metrics represent spatially sampled observations rather than simultaneous measurements from every GPU in a job. Each metric is computed only from records containing the required counters.

a) Estimated non-MFMA FP64 operations: For Config-0 records, we estimate FP64 operations executed through the VALU path using the addition, multiplication, FMA, and transcendental instruction counters. Let C_{add}, C_{mul}, C_{fma}, and C_{trans} denote the corresponding accumulated instruction counts. We compute

$$O_{\mathrm{FP64}} = C_{\mathrm{add}} + C_{\mathrm{mul}} + 2C_{\mathrm{fma}} + C_{\mathrm{trans}} \tag{1}$$

The factor of two accounts for the multiplication and addition performed by each FMA instruction. This estimate excludes FP64 MFMA activity, which is analyzed separately.

b) Conservative known-byte estimate: We derive a lower-bound traffic proxy from the available TCC request-width counters. Let C_{32} and C_{64} denote the observed 32-byte read requests and 64-byte write requests, respectively. The known-byte volume is

$$B_{\mathrm{known}} = 32C_{32} + 64C_{64} \tag{2}$$

Because the configured counters do not capture all HBM traffic, request widths, cache effects, or data reuse, B_{known} is interpreted as a conservative traffic proxy rather than an absolute memory-bandwidth measurement.

c) Conservative compute-to-memory proxy: For jobs with positive values of both O_{FP64} and B_{known}, we compute

$$I_{\mathrm{known}} = \frac{O_{\mathrm{FP64}}}{B_{\mathrm{known}}} \tag{3}$$

Because both quantities are incomplete estimates, I_{known} is used only for comparative stratification and should not be interpreted as conventional arithmetic intensity. For visualization, we use

$$\log_{10}(I_{\mathrm{known}}) = \log_{10}\left(\frac{O_{\mathrm{FP64}}}{B_{\mathrm{known}}}\right) \tag{4}$$

Jobs with zero or unavailable values in either term are excluded from the logarithmic analysis.

d) Instruction-path composition: We distinguish activity executed through conventional VALU instructions from activity executed through MFMA instructions. For a record with available FP64 counters, the observed FP64 instruction-path fractions are

$$f_{\mathrm{VALU}} = \frac{C_{\mathrm{add}} + C_{\mathrm{mul}} + C_{\mathrm{fma}} + C_{\mathrm{trans}}}{C_{\mathrm{VALU}} + C_{\mathrm{MFMA}}}$$

and

$$f_{\mathrm{MFMA}} = \frac{C_{\mathrm{MFMA}}}{C_{\mathrm{VALU}} + C_{\mathrm{MFMA}}}$$

where

$$C_{\mathrm{VALU}} = C_{\mathrm{add}} + C_{\mathrm{mul}} + C_{\mathrm{fma}} + C_{\mathrm{trans}}$$

These fractions describe the observed distribution of selected FP64 instruction classes. They do not represent complete GPU utilization or the fraction of application runtime spent in each execution path.

e) FMA contribution: Within the estimated non-MFMA FP64 operation count, the fraction of work attributed to FMA is

$$f_{\mathrm{FMA,ops}} = \frac{2C_{\mathrm{fma}}}{C_{\mathrm{add}} + C_{\mathrm{mul}} + 2C_{\mathrm{fma}} + C_{\mathrm{trans}}}$$

This operation-weighted definition differs from the fraction of FP64 instructions that are FMA because each FMA contributes two arithmetic operations.

f) FP64 activity coverage: For each job, let $\mathcal{F}_{64}$ denote the set of valid files whose assigned configuration contains the FP64 counters required by the analysis. A file $i \in \mathcal{F}_{64}$ is considered FP64-active when at least one selected FP64 VALU or MFMA counter is positive. The active-file fraction is

$$A_{\mathrm{FP64}} = \frac{\sum_{i \in \mathcal{F}_{64}} \mathbf{1}\left[O_{\mathrm{FP64},i} > 0 \vee C_{\mathrm{MFMA},i} > 0\right]}{|\mathcal{F}_{64}|} \tag{5}$$

Files assigned configurations without FP64 counters are excluded from both the numerator and denominator, and jobs without FP64-capable files are excluded from this analysis. For single-file jobs, $A_{\mathrm{FP64}} = 1$ indicates activity only at the available sampled location; multi-file jobs provide additional information about whether observed FP64 activity is spatially concentrated.

g) Workload classification: We assign jobs to one of seven coarse workload classes: matrix-FP64-heavy, mixed, memory-intense, low-activity, FMA-heavy, compute-intense, and integer-heavy. These classes are intended for population-level stratification rather than application identification or bottleneck diagnosis. The classes characterize different patterns in the derived counter metrics. Matrix-FP64-heavy jobs exhibit dominant FP64 MFMA activity. FMA-heavy jobs exhibit a large FMA contribution within the non-MFMA FP64 path. Memory-intense and compute-intense jobs occupy the lower and upper regions, respectively, of the conservative compute-to-memory metric I_{known}. Integer-heavy jobs exhibit dominant selected INT64 activity, while low-activity jobs contain little or no activity in the selected instruction and memory-request counters. Jobs with substantial measured activity that do not exhibit one of these dominant characteristics are grouped into mixed class.

The resulting workload classes are descriptive labels derived from the same metrics later visualized in the evaluation. Consequently, separation among classes in the compute-to-memory plots illustrates the resulting population partition rather than serving as an independent validation of the labels.

V. Evaluation Results

A. Operational Cost and Collection Reliability

We evaluate the runtime overhead of SORT using a suite of tests commonly executed on Borg during normal regression testing (e.g., after a software update). These tests span the following applications: CosmoFlow [30], rocHPL [31], AthenaPK [26], Cholla [32], [33], GenASiS [34],

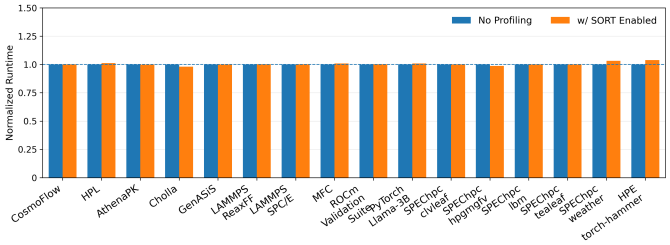


Fig. 2: Runtime overhead measurement for compute job steps of several key applications using SORT, normalized to the no-profiling configuration for each workload. Small negative differences are attributed to normal runtime variability and are not interpreted as performance improvements due to SORT.

LAMMPS [35], MFC [29], [28], ROCm Validation Suite [36], SPEChpc [37], HPE's TorchHammer [38], and a locally-developed PyTorch Llama-3B training workload.

a) Runtime Overhead: Figure 2 compares the mean runtime of the evaluated workloads with and without SORT. For most workloads, the difference is below 1%, indicating minimal application-level overhead. The main exceptions are SPEChpc miniweather and HPE torch-hammer, which exhibit larger runtime differences. We also examined total Slurm allocation time and observed differences of up to approximately 30s for some workloads. Total allocation time includes effects outside the primary application step, including scheduler-side operations and variability associated with Borg's connection to Orion. We therefore use primary Slurm job-step elapsed time as the principal measure of application-level overhead.

b) Data Volume and Collection Reliability: SORT produces compact telemetry with storage bounded by its spatial-sampling policy. Each individual counter artifact contains approximately 350–430 bytes of logical telemetry data. Because a profiling location may generate multiple counter artifacts, we also report storage at the counter-set level. A complete three-configuration counter set occupies approximately 1.0–1.35 KB of logical file content. On Borg, jobs typically generate one or two counter sets because of their smaller allocation sizes. Under the maximum sampling configuration, SORT permits up to 16 counter sets, corresponding to 48 individual counter files and approximately 16–21 KB of logical telemetry data per job. The physical storage consumed by these files may be larger because small files occupy space according to the underlying file-system allocation granularity.

B. Observed Workload Mix

We next examine how the retained Borg jobs are distributed across the workload classes defined in Section IV-E. Figure 3 summarizes the classified workload population by job count and aggregate measured counter volume.

By job count, matrix-FP64-heavy is the largest class, followed by mixed, memory-intense, and low-activity jobs. Together, these four classes account for approximately 98.35% of the retained population, while the FMA-heavy, compute-intense, and integer-heavy classes are comparatively small. These labels describe the counter activity observed during each retained job and do not necessarily identify the underlying application, particularly for short or partial executions. Aggregate counter volume provides a different view of the workload population. The mixed class contributes the largest estimated non-MFMA FP64 operation volume, while the mixed, memory-intense, and matrix-FP64-heavy classes contribute most of the observed known TCC-request bytes. Matrix-FP64-heavy jobs contribute relatively little to the non-MFMA FP64 estimate because their dominant FP64 activity is captured separately by the MFMA counter.

TABLE III: Workload-class and derived counter volumes.

Class	Jobs	Non-MFMA FP64 ops	Known TCC bytes
Matrix FP64 heavy	23,011	2.03×10^{15}	3.34×10^{17}
Mixed	18,320	1.12×10^{17}	5.68×10^{17}
Memory intense	16,350	5.81×10^{15}	4.64×10^{17}
Low activity	8,577	0	0
FMA heavy	1,093	5.84×10^{15}	6.34×10^{16}
Compute intense	16	3.81×10^{12}	5.45×10^{9}
Integer heavy	6	4.97×10^{8}	1.90×10^{9}

Because the workload classes are derived from these same counter features, Figure 3 illustrates the resulting population partition rather than providing an independent validation of the class distinctions.

C. Compute and Memory Characteristics

Figure 4 compares estimated non-MFMA FP64 operations with conservative known TCC-request bytes for jobs with positive values of both metrics. The observed jobs span several orders of magnitude in both dimensions, indicating substantial variation in measured FP64 VALU work and memory-request activity across the retained workload population. Although the workload classes occupy different regions of the space, substantial overlap remains. This is expected because classification also depends on MFMA, FMA, integer, and minimum-activity conditions that are not represented by the two plotted axes.

Figure 4(b) shows the distribution of estimated non-MFMA FP64 operations per conservative known TCC-request byte on a logarithmic scale. Most included jobs fall below one estimated operation per known byte, with the largest concentration between approximately 10^{-4} and 10^{0} estimated non-MFMA FP64 operations per known byte. Because both the operation count and known-byte estimate are incomplete measurements, this ratio is used only as a comparative workload-characterization metric. It should not be interpreted as conventional arithmetic intensity or as a complete measure of memory traffic. Because the memory-intense and compute-intense classes are derived partly from this ratio, their separation in Figure 4 reflects the underlying classification metric. In addition, the compute-intense and integer-heavy classes contain very few jobs, so their apparent separation should not be interpreted as representative of a broad workload population.

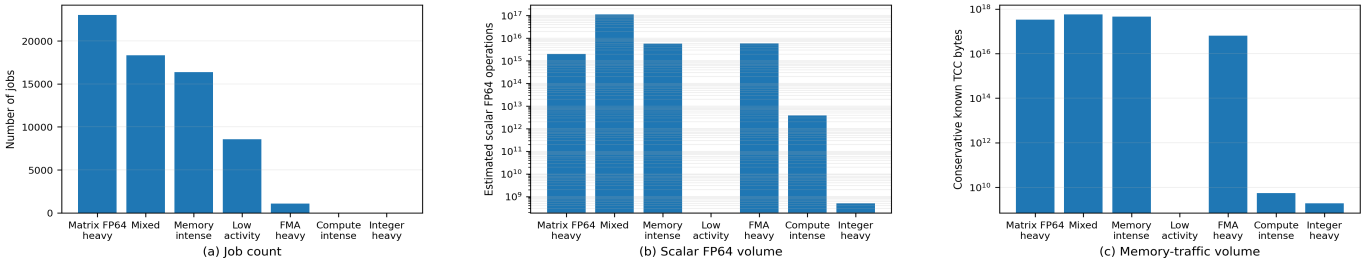


Fig. 3: Observed workload-class distribution and aggregate derived counter volume. (a) reports the number of jobs assigned to each class, whereas (b) and (c) report aggregate estimated non-MFMA FP64 operations and conservative known TCC-request bytes, respectively. Both (b) and (c) use logarithmic y-axis.

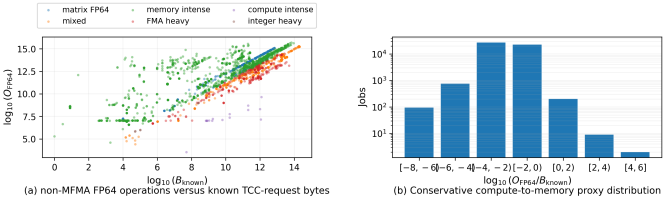


Fig. 4: Compute and memory characteristics of jobs with positive estimated non-MFMA FP64 operation counts and positive known TCC-request byte estimates. (a) compares the two quantities on logarithmic axes. (b) reports the distribution of the conservative compute-to-memory proxy $\log_{10}(O_{\mathrm{FP64}}/B_{\mathrm{known}})$ using a logarithmic count axis.

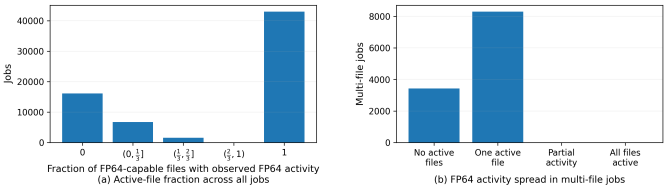


Fig. 5: Distribution of FP64-active counter files. (a) reports the active-file fraction across all jobs. (b) separates the 11,728 multi-file jobs according to the number of files with observed FP64 activity.

D. FP64 Activity Coverage

We next examine how FP64 activity is distributed across the sampled job-host records using the activity-coverage metric defined in Section IV-E. Figure 5(a) reports the fraction of FP64-capable records with observed FP64 activity for each job. Among the retained jobs, 42,953 exhibit FP64 activity in every available record, while 16,122 exhibit no observed FP64 activity. The remaining 8,298 jobs contain both active and inactive records, indicating that observed FP64 activity is concentrated within only part of the sampled coverage. For jobs with a single FP64-capable record, an active fraction of one indicates activity only at that sampled location. Figure 5(b) therefore focuses on jobs with multiple FP64-capable records. Among the 11,728 multi-file jobs, 3,430 have no FP64-active files, 8,297 have exactly one FP64-active file, and only 1 job has FP64 activity in more than one sampled file.

Because only one configuration in each three-configuration sampling group contains the FP64 counters, the number of FP64-capable records may be smaller than the total number of counter files associated with a job. The concentrated FP64 activity may reflect heterogeneous rank roles, non-FP64 execution at some sampled locations, short or failed GPU phases, or incomplete temporal and spatial coverage. An inactive FP64-capable record therefore indicates only that the selected FP64 counters remained zero at that sampled location. The metric should be interpreted as an FP64 activity-coverage indicator rather than a direct measure of GPU utilization or load balance. Because the artifacts contain final cumulative counter snapshots, the analysis also cannot determine whether sampled locations were active simultaneously.

ACKNOWLEDGMENTS

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (https://www.energy.gov/doe-public-access-plan). Also, AMD, the AMD Arrow logo, Instinct, ROCm, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

VI. CONCLUSION

This paper presents SORT, a Slurm-orchestrated ROCm telemetry framework for lightweight, job-scoped GPU observability on HPC systems. Deployed on Borg, SORT collected 88,269 valid counter files associated with 67,373 jobs while introducing minimal runtime overhead for most evaluated workloads and bounding telemetry output to approximately 16–21 KB of logical data per job at full sampling scale. The resulting data supports coarse workload classification, conservative compute-to-memory characterization, and FP64 activity-coverage analysis, providing a practical basis for population-level workload characterization and selection of jobs for deeper profiling. The current evaluation reflects Borg's development and system-testing workload population, while full Frontier-scale deployment and validation remain future work.

References

[1] E. A. León, J. Glenski, M. Stock, K. McMahon, W. Loewe, C. Snyder, L. Kaplan, S. Vadlamani, T. I. Mattox, T. D'Hooge *et al.*, "Breaking the system noise barrier at exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2025, pp. 411–436.

[2] A. Khan, H. Sim, S. S. Vazhkudai, A. R. Butt, and Y. Kim, "An analysis of system balance and architectural trends based on top500 supercomputers," in *The International Conference on High Performance Computing in Asia-Pacific Region*, ser. HPCAsia '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 11–22. [Online]. Available: https://doi.org/10.1145/3432261.3432263

[3] A. Khan, J. R. Lange, N. Hagerty, E. F. Posada, J. Holmen, J. B. White, A. Harris, V. M. Vergara, C. Zimmer, and S. Atchley, "An evaluation of the effect of network cost optimization for leadership class supercomputers," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1–16.

[4] SchedMD, "Slurm sacct Documentation," https://slurm.schedmd.com/sacct.html, accessed for background on Slurm accounting fields.

[5] G. Esposito, J.-D. Guerrero-Balaguera, J. E. R. Condia, M. S. Reorda, M. Barbiero, and R. Fortuna, "Gpu under pressure: Estimating application's stress via telemetry and performance counters," *arXiv preprint arXiv:2511.05067*, 2025.

[6] H. Ootomo, K. Ozaki, and R. Yokota, "Dgemm on integer matrix multiplication unit," *The International Journal of High Performance Computing Applications*, vol. 38, no. 4, pp. 297–313, 2024.

[7] K. Ozaki, Y. Uchino, and T. Imamura, "Ozaki scheme ii: A gemm-oriented emulation of floating-point matrix multiplication using an integer modular technique," *The International Journal of High Performance Computing Applications*, p. 10943420261467787, 2025.

[8] "Nvidia dcgm — nvidia developer," [Online; accessed 2026-07-31]. [Online]. Available: https://developer.nvidia.com/dcgm

[9] NVIDIA Corporation, "nvidia-smi - NVIDIA System Management Interface program," http://manpages.ubuntu.com/manpages/precise/man1/alt-nvidia-current-smi.1.html.

[10] "Amd rocm — amd rocm 7.14.0," [Online; accessed 2026-07-31]. [Online]. Available: https://rocm.docs.amd.com/en/latest/

[11] O. Cankur, B. Austin, and A. Bhatele, "Longitudinal analysis of gpu workloads on perlmutter," *arXiv preprint arXiv:2502.18680*, 2025.

[12] B. Oztop, D. Kulkarni, Z. Zhao, A. K. Coskun, and K. Konate, "A practical two-stage framework for gpu resource and power prediction in heterogeneous hpc systems," *arXiv preprint arXiv:2604.02158*, 2026.

[13] Y. Lei, J. Fernandez, V. Kypriotis, D. Skarlatos, E. Strubell, J. Sherry, and D. Vosler, "The energy cost of execution-idle in gpu clusters," *arXiv preprint arXiv:2604.04745*, 2026.

[14] C. Yang, R. Gayatri, T. Kurth, P. Basu, Z. Ronaghi, A. Adetokunbo, B. Friesen, B. Cook, D. Doerfler, L. Oliker *et al.*, "An empirical roofline methodology for quantitatively assessing performance portability," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 2018, pp. 14–23.

[15] P. Sinha, A. Guliani, R. Jain, B. Tran, M. D. Sinclair, and S. Venkataraman, "Not all gpus are created equal: characterizing variability in large-scale, accelerator-rich systems," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 01–15.

[16] V. Oles, A. Schmedding, G. Ostrouchov, W. Shin, E. Smirni, and C. Engelmann, "Understanding gpu memory corruption at extreme scale: The summit case study," in *Proceedings of the 38th ACM International Conference on Supercomputing*, 2024, pp. 188–200.

[17] Y. Hao, N. Jain, R. Van der Wijngaart, N. Saxena, Y. Fan, and X. Liu, "Drgpu: A top-down profiler for gpu applications," in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, 2023, pp. 43–53.

[18] Y. Xia and J. Mellor-Crummey, "Leo: Tracing gpu stall root causes via cross-vendor backward slicing," *arXiv preprint arXiv:2604.20032*, 2026.

[19] S. Darche and M. R. Dagenais, "Low-overhead trace collection and profiling on gpu compute kernels," *ACM Transactions on Parallel Computing*, vol. 11, no. 2, pp. 1–24, 2024.

[20] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476223

[21] E. Sencan, D. Kulkarni, A. Coskun, and K. Konate, "Analyzing gpu utilization in hpc workloads: Insights from large-scale systems," in *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration*, ser. PEARC '25. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3708035.3736010

[22] M. Leinhauser, R. Widera, S. Bastrakov, A. Debus, M. Bussmann, and S. Chandrasekaran, "Metrics and design of an instruction roofline model for amd gpus," *ACM Transactions on Parallel Computing*, vol. 9, no. 1, pp. 1–14, 2022.

[23] N. Hagerty and D. Dietz, "Experiences integrating database support into the olcf test harness," in *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC Workshops '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 662–668. [Online]. Available: https://doi.org/10.1145/3731599.3767426

[24] Advanced Micro Devices, Inc., "ROCProfiler Documentation," https://rocm.docs.amd.com/_/downloads/rocprofiler/en/docs-6.2.2/pdf/, 2024, rOCm 6.2.2 documentation. The counter-collection documentation states that the SQ block can handle only eight counters at a time.

[25] AMD, "ROCprofiler-SDK Documentation," https://rocm.docs.amd.com/projects/rocprofiler-sdk/, accessed for background on ROCprofiler device-counter collection.

[26] P. Grete, J. C. Dolence, J. M. Miller, J. Brown, B. Ryan, A. Gaspar, F. Glines, S. Swaminarayan, J. Lippuner, C. J. Solomon, G. Shipman, C. Junghans, D. Holladay, J. M. Stone, and L. F. Roberts, "Parthenon—a performance portable block-structured adaptive mesh refinement framework," *The International Journal of High Performance Computing Applications*, vol. 37, no. 5, pp. 465–486, 2023. [Online]. Available: https://doi.org/10.1177/10943420221143775

[27] J.-L. Vay, M. Acciarri, A. Almgren, L. D. Amorim, I. Andriyash, J. R. Angus, T. Antoun, D. Belkin, D. Bizzozero, A. Blelly, S. E. Clark, E. Dammak, L. Fedeli, A. Formenti, M. Garten, L. Ge, L. Giacomel, K. Gott, R. E. Groenewald, D. Grote, J. Gu, C. Harrison, M. Haseeb, A. Huebl, R. Jambunathan, I. Kara-Mostefa, H. Klion, P. Kumar, R. Lehe, B. Loring, T. Marks, P. Miller, A. Myers, J. Pech, E. Rheaume, G. Richardson, M. E. Rowan, R. T. Sandberg, P. Scherpelz, A. Sinn, O. Shapoval, D. Terzani, M. Thévenet, K. Weichman, E. Yang, N. Zaim, W. Zhang, Y. Zhao, K. Z. Zhu, and E. Zoni, "WarpX: An advanced Particle-In-Cell code," https://blast-warpx.github.io, 2018, https://doi.org/10.5281/zenodo.4571577. [Online]. Available: https://doi.org/10.5281/zenodo.4571577

[28] B. Wilfong, H. Le Berre, A. Radhakrishnan, A. Gupta, D. J. Vickers, D. Vaca-Revelo, D. Adam, H. Yu, H. Lee, J. R. Chreim, M. Carcana Barbosa, Y. Zhang, E. Cisneros-Garibay, A. Gnanaskandan, M. Rodriguez Jr., R. D. Budiardja, S. Abbott, T. Colonius, and S. H. Bryngelson, "MFC 5.0: An exascale many-physics flow solver," *Computer Physics Communications*, vol. 322, p. 110055, 2026.

[29] A. Radhakrishnan, H. Le Berre, B. Wilfong, J.-S. Spratt, M. Rodriguez Jr., T. Colonius, and S. H. Bryngelson, "Method for portable, scalable, and performant GPU-accelerated simulation of multiphase compressible flow," *Computer Physics Communications*, vol. 302, p. 109238, 2024.

[30] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Karna, D. Moise, S. J. Pennycook, K. Maschoff, J. Sewall, N. Kumar, S. Ho, M. Ringenburg, Prabhat, and V. W. Lee, "CosmoFlow: Using deep learning to learn the universe at scale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '18. IEEE, 2018, pp. 819–829.

[31] "Amd rocm rochpl," [Online; accessed 2026-08-11]. [Online]. Available: https://github.com/ROCm/rocHPL

[32] E. E. Schneider and B. E. Robertson, "Cholla: A new massively parallel hydrodynamics code for astrophysical simulation," *The Astrophysical Journal Supplement Series*, vol. 217, no. 2, p. 24, apr 2015. [Online]. Available: https://doi.org/10.1088/0067-0049/217/2/24

[33] B. Villasenor, B. Robertson, P. Madau, and E. Schneider, "Effects of photoionization and photoheating on ly forest properties from cholla cosmological simulations," *The Astrophysical Journal*, vol. 912, no. 2, p. 138, may 2021. [Online]. Available: https://doi.org/10.3847/1538-4357/abed5a

[34] R. D. Budiardja and C. Y. Cardall, "Genasis basics: Object-oriented utilitarian functionality for large-scale physics simulations (version

4)," *Computer Physics Communications*, vol. 281, p. 108505, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465522002247

[35] 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," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[36] "Amd rocm validation suite," [Online; accessed 2026-08-11]. [Online]. Available: https://github.com/ROCm/ROCmValidationSuite

[37] H. Brunst, S. Chandrasekaran, F. M. Ciorba, N. Hagerty, R. Henschel, G. Juckeland, J. Li, V. G. M. Vergara, S. Wienke, and M. Zavala, " First Experiences in Performance Benchmarking with the New SPEChpc 2021 Suites ," in *2022 22nd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2022, pp. 675–684. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CCGrid54584.2022.00077

[38] "Hpe torch-hammer: Strike while the gpu is hot," [Online; accessed 2026-08-11]. [Online]. Available: https://github.com/HPE/torch-hammer