SC24: International Conference for High Performance Computing, Networking, Storage and Analysis | 979-8-3503-5291-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/SC41406.2024.00071

A Workflow Roofline Model for End-to-End Workflow Performance Analysis

Nan Ding, Brian Austin, Yang Liu, Neil Mehta, Steven Farrell, Johannes P. Blaschke, Leonid Oliker,
Hai Ah Nam, Nicholas J. Wright, Samuel Williams
Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
{nanding, baustin, neilmehta, jpblaschke, loliker, hnam, njwright, swwilliams}@lbl.gov

***Abstract*—As next-generation experimental and observational instruments for scientific research are being deployed with higher resolutions and faster data capture rates, the fundamental demands of producing high-quality scientific throughput require portability and performance to meet the high productivity goals. Understanding such a workflow's end-to-end performance on HPC systems is formidable work. In this paper, we address this challenge by introducing a Workflow Roofline model, which ties a workflow's end-to-end performance with peak node- and system-performance constraints. We analyze four workflows: LCLS, a time-sensitive workflow that is bound by system external bandwidth; BerkeleyGW, a traditional HPC workflow that is bound by node-local performance; CosmoFlow, an AI workflow that is bound by the CPU preprocessing; and GPTune, an auto tuner that is bound by the data control flow. We demonstrate the ability of our methodology to understand various aspects of performance and performance bottlenecks on workflows and systems and motivate workflow optimizations.**

***Index Terms*—Workflow Roofline Model, End-to-end Workflow, Performance Analysis, Performance Evaluation**

I. Introduction

Scientific workflows are a cornerstone of modern scientific computing and are used widely across scientific domains [1]. Over the decades, many workflow patterns have been developed, ranging from a simple collection of tasks [2] and sets of distributed applications with intermediate key-value pairs [3] and object ordering [4] to more sophisticated iterative chains of MapReduce jobs [5]. High-Performance Computing (HPC) workflows [6], often known as interconnecting computational and data manipulation steps, also expand their archetypes to high-performance AI workflow [7] such as training and inference, and cross-facility workflow [8] which requires rapid data analysis and real-time steering, etc.

Workflow management [9] and monitor systems aiding in the automation of those tasks emerged, such as IPDD [10], panorama [11] and Ramses [12]. However, they are often required to deploy a set of tools and software to collect fine-grained workflow traces and then leverage expensive simulations to diagnose performance. Such a method lacks the usability and flexibility to analyze workflows without traces, as well as a lack of insights for the supercomputing facilities. Traditionally, it's common sense that understanding and tuning the application performance is a challenge. It is important to stress that an application is only one task within a workflow. Thus, analyzing and understanding a workflow's end-to-end performance to drive optimization becomes an even more formidable work.

In this paper, we propose a Workflow Roofline model, which ties a workflow's performance with node- and system-performance constraints. The contributions in this paper include:

- Definition of Workflow Roofline ceilings that characterize the peak workflow performance.
- Creation of System boundaries that define a range of attainable performance.
- Development of a workflow characterization methodology that incorporates number of parallel tasks, workflow makespan (latency), and workflow throughput into the Workflow Roofline model.
- Development of a workflow execution characterization methodology that allows easy visualization of potential performance constraints.
- Evaluation of the Workflow Roofline Model and methodology on four workflows: LCLS (data analysis, system external bound), BerkeleyGW (traditional HPC, node bound), CosmoFlow (hyperparameter tuning, node HBM bound) and GPTune (auto-tuner, control flow bound).

II. Related Work

The common sense of a workflow performance bottleneck is I/O. Li *et al.* [13] and Zhang *et al.* [14] leverages simulation and analysis workloads within the workflow to understand the I/O performance. In-situ, in-transit and the combination of them are widely explored to achieve the trade-off between performance and data movement cost [15]–[20]. Haldeman *et al.* [21] and Rodero *et al.* [22] evaluated the performance and power/energy trade-offs of different data movement strategies for in-situ processing. Poeschel *et al.*optimized the file-based HPC workflows using streaming data pipelines with openPMD and ADIOS2 [23]. There are also work on improving task scheduling [24]–[26].

As workflows become diverse, cross-facility workflows raise attention. These workflows are being deployed with higher resolutions and faster data capture rates, creating a big data crunch that modest institutional computing resources cannot handle. In the meantime, these big data analysis pipelines also

This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231 and used resources of the National Energy Research Scientific Computing Center (NERSC) which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

require near real-time computing and have higher resilience requirements than the simulation and modeling workloads more traditionally seen at HPC centers. As such, containerize [8], [27]–[30] has been explored to improve the resilience. Hybrid HPC and cloud environment has also been discussed to understand the portability and performance benefits [29], [31], [32].

Some efforts have been made to collect, profile, and online monitor for real scientific workflows with domain knowledge [33]–[43]. These traces provide fine-grained insights on CPU and memory usage, I/O operations, job dependencies and etc. However, there is limited insights on guiding end-to-end performance optimization and system implications.

Ben-Nun *et al.* demonstrate that the fundamental demands of producing high-quality scientific requires portability and performance to meet the high productivity goals [44]. The Workflows Community Summits [45]–[47] also discuss that it is worth designing workflows that exploit the hybrid architectures, which are the key to high performance. Tail-tolerant techniques were demonstrated that make it possible to achieve higher system utilization without sacrificing service responsiveness [48].

Eventually, various optimizations and the corresponding profiling methods have been explored with the support of domain knowledge. Unfortunately, these existing works are case by case. The insights gained from one workflow cannot be easily propagated to others. The toolchains deployed on one system may not be easily and properly deployed on other machines. Therefore, one crucial field that is missing from the previous work is a unified methodology to quickly and intuitively tell the potential bottlenecks for end-to-end workflow execution, and drive optimizations.

III. Workflow Roofline Model

The traditional Roofline model [49] characterizes a kernel's performance in GigaFLOPs per second (GFLOP/s) as a function of its arithmetic intensity (AI). The AI is expressed as the ratio of floating-point operations performed to data movement (FLOPs/Bytes). For a given kernel, we can find a point on the X-axis based on its AI. The Y-axis represents the measured GFLOP/s. This performance number can be compared against the bounds set by the peak compute performance (Peak GFLOPs) and the memory bandwidth of the system (Peak GB/s) to determine what is limiting performance: memory or compute.

Whereas the traditional Roofline model and its variants [50]–[52] can be quite effective in the above regard, such refinement is misplaced for workflows that are the orchestrated sequences of these applications along with data handling and processing steps. In order to affect the Workflow Roofline analysis, we need to target a different set of metrics. First, we count the number of parallel tasks. One task is considered as one job in the workflow. It can be a large MPI application or small script, depending on how the workflow developers design them. Such metric allows us to both identify throughput bottlenecks, and, when refining the workflow task orders, critical path tuning. Makespan [53](or latency), the time cost of a workflow, is another critical performance factor for workflows. A time-sensitive workflow usually has a deadline by when results need to be available to the dispatcher. When the execution time exceeds the expectation, one need to diagnose the poor performance with the guidance of the potential bottlenecks. Finally, the nature of workflows makes choosing lightweight metrics a critical factor in workflow profiling. As such, we characterize the data volume and floating-point operations at node-level, and data volume at the system level for workflow characterization.

Although workflow developers with domain knowledge can use available performance tools [54], [55] to diagnose the performance bottlenecks discussed above, the Workflow Roofline Model provides an approachable means of characterizing performance bottlenecks in a single figure.

We describe the methods of characterizing architectures, workflows, and performance interpretation in the rest of this section.

A. Architecture Characterization

First, we describe how we define the Workflow Roofline ceilings. For consistency, we use Perlmutter [56], a Hewlett Packard Enterprises Cray EX Supercomputer at the National Energy Research Scientific Computing Center (NERSC), when describing the methodology. However, the model and terminology are applicable to other system architectures. The system details of Perlmutter can be found later in Section IV-A.

Node ceilings (diagonal): Each Perlmutter GPU node consists of four NVIDIA's A100 GPU (A100) [57], and each GPU is connected to the CPU via PCIe 4.0 at 25 GB/s/direction. As such, the theoretical node PCIe bandwidth is $4 \times 25 = 100$ GB/s/direction for the data transfer between the host CPU and GPUs (PCIe bytes in Figure 1). Similarly, the node peak TFLOPS is the aggregated peak performance of the four GPUs (Compute Flops in Figure 1). Ultimately, a workflow can have multiple node ceilings which represents the compute and data motion performance upper bound.

System ceilings (horizontal): The all-NVMe file system [58] on Perlmutter is directly integrated on to the same Slingshot network as the compute nodes. Perlmutter has a total of four I/O groups on the dragonfly network, and each I/O group is directly connected to each compute group via 100 GB/s [58]. Therefore, one can get 5.6 TB/s (14 GPU groups $\times$ 4 I/O groups $\times$ 100 GB/s) for the system internal (loading data from file system) bandwidth ceiling. Alternatively, for the workflows that leverage Message Passing Interface (MPI) to do the data transfer, we leverage the NIC performance to set the ceilings. Each Perlmutter GPU node has four PCIe 4.0 NICs which provides 100 GB/s/direction in total. As such, a workflow can also have multiple system ceilings which represents different ways of data transfer.

The Workflow Roofline Model is described in Eq.(1). A workflow's performance, characterized in number of tasks per second (TPS), is a function of the number of parallel tasks,

floating-point operations at node-level, and data volume at node- and sytem-level with performance boundaries of the peak node performance (node TFLOPS and node GB/s), system peak bandwidth (sys GB/s). "Number of parallel Tasks" for a workflow is defined as the number of tasks that can be executed in parallel.

It is important to stress that, unlike the traditional Roofline model, the absolute machine peak is hidden behind the ceilings because the achieved performance upper bound varies as different data volumes are performed, i.e., data volume happens in the system (Bytes_sys) and within a node (Bytes_node) and floating-point operations (Flops_node).

$$TPS \leq \min \begin{cases} \frac{\text{Number of parallel tasks}}{\text{Bytes\_sys / Peak sys GB/s}} \\ \frac{\text{Number of parallel tasks}}{\text{Bytes\_node / Peak node GB/s}} \\ \frac{\text{Number of parallel tasks}}{\text{Flops\_node / Peak node TFLOPS}} \\ \textit{other ceilings not subjected to the above} \end{cases} \quad (1)$$

System parallelism wall: The number of parallel tasks is not an infinite number, and it can be bound by the available resource in the system (or queue). Imagine a task uses 64 nodes, and thus, the number of parallel tasks is limited to $\frac{1792}{64} = 28$ on Perlmutter GPU partition. Therefore, we introduce a vertical task parallelism wall, which is defined as the available number of nodes divided by the required number of nodes per task.

Figure 1 shows the resultant Workflow Roofline ceilings on Perlmutter GPU partition. We assume one terabyte data is loaded via file system at 5.6 TB/s (upper horizontal) and one terabyte data per compute node is transferred via the NICs at 100 GB/s (lower horizontal). The node performance boundaries (diagonals) assume 4 GB data is transferred and 100 GFLOPs are performed. The task in the workflow uses 64 nodes, and thus the task parallelism boundary (vertical wall) is 28. The grey area is unattainable due to the above system constraints. The upper direction represents a shorter makespan, and the upper right direction indicates a higher throughput.

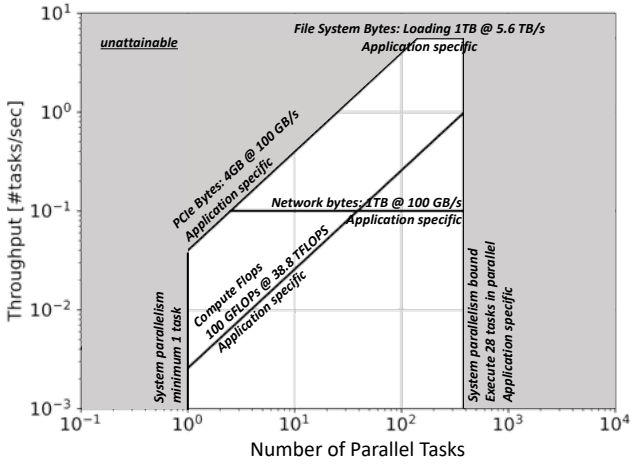


Fig. 1: Workflow Roofline Model. The achieved performance area is limited by the application-specific performance bounds derived from finite node and system resources. The Upper direction represents a shorter makespan, and the upper right direction indicates a higher throughput. Changing system or node bandwidths shift the ceilings.

B. Workflow Characterization

Mirroring the previous section that characterizes a system performance capabilities, in this section, we describe the methodology we employ to characterize workflow execution in terms of the Workflow Roofline Model.

Number of parallel tasks: Recall that a workflow inherently contains several kinds of tasks. For example, a typical scientific workflow may include applications spanning simulation, results gathering, data analysis, and visualization. Therefore, tasks, in our definition, can be large MPI applications or small scripts, depending on how the workflow developers design them. The x-axis (the number of parallel tasks), as defined, is the number of concurrently executing tasks in such a workflow, whether big or small. Thus, increasing the number of MPI processes (MPI strong scaling) within an application does not increase the number of parallel tasks.

Throughput and makespan: We rely on the timing report from the workflow itself to collect the makespan (queue wait time is not included). The number of parallel tasks and total number of tasks within that makespan can be obtained from the workflow description, e.g., sbatch [59] and Workflow Description Language (WDL) [60]. Thus, the achieved throughput can be calculated using $\frac{\text{total number of tasks}}{\text{makespan}}$. The corresponding x-axis is the number of parallel tasks.

Node ceilings: We quantify the data volume loading to the node and the number of floating-point operations performed within a node. The two diagonal ceilings can be described as $\frac{\text{Number of tasks}}{\frac{\text{Flops\_node}}{\text{peak node TFLOPS}}}$ and $\frac{\text{Number of tasks}}{\frac{\text{Bytes\_node}}{\text{peak node GB/s}}}$ as shown in Equation 1 and Figure 1. Note that the Node Flops ceiling is not necessarily higher than the Node Bytes ceiling. The positions depend on the ratio of the number of flops (or Bytes) to the corresponding system peak.

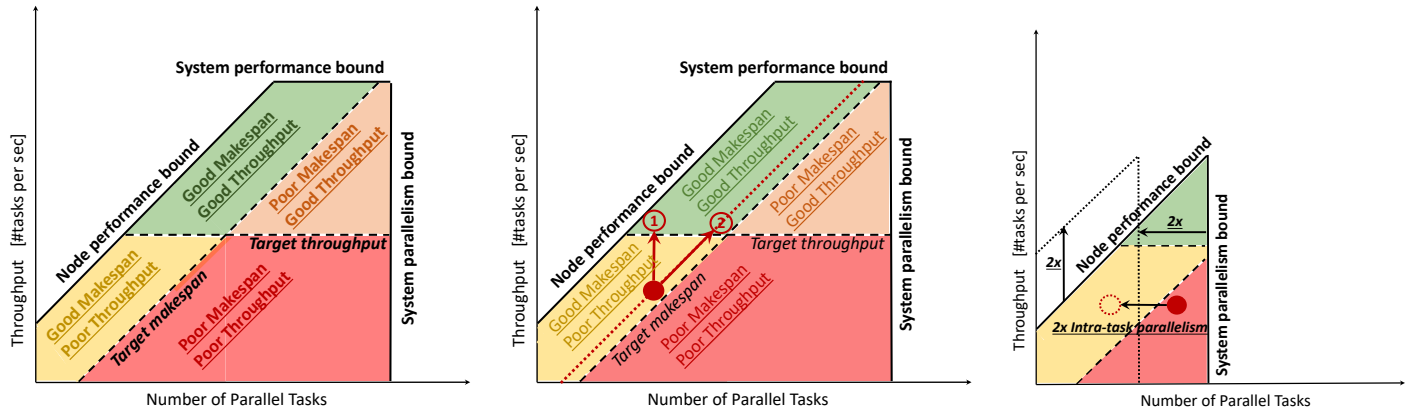
Shared system ceilings: Similar to the node ceilings, the system ceilings are formalized in the same fashion, i.e., number of parallel tasks divided by the ratio of the data volume loading to the system (Bytes_sys) to the peak system bandwidth (GB/s).

System parallelism bound: The system parallelism wall is characterized by $\frac{\text{Number of available nodes}}{\text{Number of required nodes}}$. For instance, one can imagine a workflow that uses 1024 nodes on Perlmutter. In that case, the system parallelism is limited to one ($\lfloor \frac{1792}{1024} \rfloor = 1$) on Perlmutter.

C. Driving Workflow Optimizations

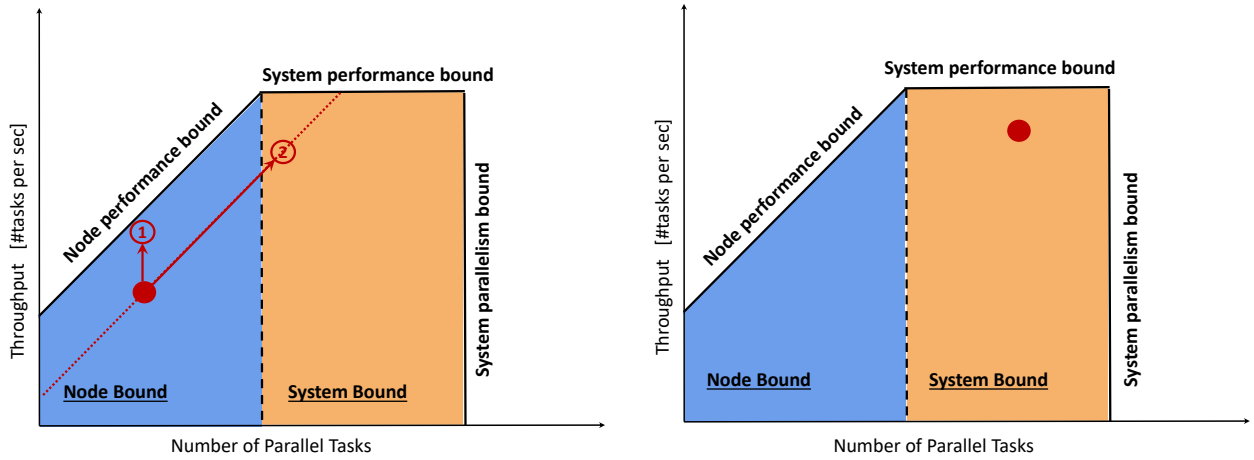
Interpretation of the insights to drive optimization matters to broad communities, including users, developers, and HPC vendors. In this section, we demonstrate the capabilities of driving optimizations using the Workflow Roofline model.

We categorize workflows into three kinds. One is time-sensitive workflows. Those workflows usually have a clear



(a) Integrate target makespan and throughput (b) Motivate two optimization directions (c) Tell infeasible optimization directions

Fig. 2: The Workflow Roofline interpretation for time- and throughput-sensitive workflows. (a) Divide the attainable area in the Workflow Roofline model into four zones according to the target makespan and throughput. (b) Assuming an empirical workflow dot meets the target makespan but far from the target throughput, the Workflow Roofline model motivate two optmization directions: ① improve latency and ② increase task parallelism. (c) A smaller machine or queue limits may prohibit optimization ②. Double the intra-task parallelism and halve the number of parallel tasks (dotted circle), the system parallelism wall will move to the left by 2× and the node ceiling will move to the upper direction by 2× accordingly.



(a) Number of Parallel Tasks Bound (b) Bandwidth Bound

Fig. 3: The Workflow Roofline interpretation for other workflows: (a) Node bound, and (b) System bound.

expectation of the target makespan [8]. The second category is throughput-sensitive workflow. Batch results are more meaningful for those workflows while time is not sensitive. The third group is other workflows. These workflows do not have a specific target makespan and throughput, but they may have very limited resources or budgets. However, their demands on workflow performance can be considered the same as throughput-sensitive workflows: finish more tasks in that limited resource.

Figure 2 shows how the Workflow Roofline model motivates optimizations for time- and throughput-sensitive workflows. We first integrate the target makespan and throughput (dotted lines) into the Workflow Roofline plot in Figure 2a. Thus, according to the target makespan and throughput, the attainable performance area is divided into four zones: good makespan good throughput (green), good makespan poor throughput (yellow), poor makespan good throughput (orange) and poor makespan poor throughput (red). When plotting the empirical workflow performance, the Workflow Roofline model intuitively shows the satisfied and unsatisfied metrics (makespan and throughput) to drive future optimizations.

Imagine a workflow that has a satisfied makespan but suffers from poor throughput. As Figure 2b shows, the empirical workflow dot is in the yellow zone. The Workflow Roofline model motivates two directions: one can keep reducing the workflow makespan to move the dot to the upper direction to meet the target throughput. Alternatively, one can increase the number of parallel tasks to move the dot to the upper right

direction.

One may decrease makespan to achieve a throughput target by either optimizing the code for iso-parallelism (performance or scalability), or increase the intra-task parallelism. Doing the latter reduces available parallelism and may preclude optimization ②. For example, if one double the intra-task parallelism and halve the number of parallel tasks (dotted circle), the system parallelism wall will move to the left by $2\times$ and the node ceiling will move to the upper direction by $2\times$ accordingly, as Figure 2c shows. Therefore, if one can't guarantee perfect scalability, then the makespan-system parallelism intercept will fall lower, i.e., the more you shift to intra-task parallelism, the easier it is to hit makespan targets, but the harder it is to hit throughput targets.

Figure 3 shows another interpretation for the workflows that do not have a clear makespan or throughput target. The attainable performance area is divided into two parts: node bound (blue) and system bound (orange). Let's imagine an empirical workflow dot falls into the blue zone as plotted in Figure 3a. The Workflow Roofline model motivates two optimization directions. First, one can improve the node efficiency to achieve a shorter makespan (the dot moves to the upper direction). Second, one can achieve higher throughput by increasing the number of parallel tasks (the dot moves diagonally up). Figure 3b plots the system bound case in which the empirical workflow dot falls into the orange zone. It indicates the potential performance bottleneck of the workflow is system bandwidth.

Eventually, the Workflow Roofline model can provide node- and system-performance insights, and thereby provides quick guidance on the optimization directions.

D. Workflow Roofline vs. Original Roofline

The original Roofline model characterizes a kernel's performance against the bounds set by the peak FLOPS and memory bandwidth. It is a fine-grained on-node model to tune detailed computing and memory performance. Conversely, the Workflow Roofline model is a coarse-grained model for workflows and the whole system throughput. The visualization may look very similar to the original Roofline – hence why they are both called rooflines – but they differ in many aspects, including metrics, usage, and insights.

The original Roofline's key metrics are bandwidth (e.g., cache thru network), performance, and data locality (e.g., arithmetic intensity). The Workflow Roofline focuses on system level metrics, such as task concurrency, shared file system bytes, and shared system network bytes, to understand the workflow's latency and throughput. The Workflow Roofline allows us to understand the code's strong and weak scaling and data transfer (file system) performance among tasks.

The ceilings of the Workflow Roofline model contain node-local (memory, PCIe bandwidth, compute) and system-wide (global network, global filesystem, etc.) performance bounds. The traditional Roofline model examines the interaction between the former in the context of a program's node-local execution, while the Workflow Roofline model incorporates the latter to analyze workflow performance on an HPC system.

Unlike the original Roofline, which defines machine-specific performance bounds, the Workflow Roofline model extracts workflow-specific performance bounds based on application-specific performance bounds derived from an HPC system's finite node and system resources.

The original Roofline model principally aims to identify whether a kernel's performance is limited by computation or data movement. The Workflow Roofline model provides new insights into whether a workflow's throughput and makespan are limited by system internal I/O bound, system external I/O bound, node-local performance, or task concurrency. The original Roofline could be the next step in analysis if a workflow is bound by node-local performance rather than the global network or filesystem.

IV. Workflow Roofline in Practice

In this section, we describe our system and workflow characterization, and use the Workflow Roofline Model to evaluate and analyze four workflows.

A. System Characteristics

System Configurations in this paper were obtained via the architecture white paper of Perlmutter [56] at NERSC. The Perlmutter GPU partition (PM-GPU) comprises nodes with one AMD Milan CPU and four NVIDIA A100 GPUs. Thus, each compute node provides a peak of $9.7 \cdot 4 = 38.8$ TFLOPS. The system offers a peak of 100 GB/s for node interconnections. The Perlmutter CPU partition (PM-CPU) consists of nodes with two AMD Milan CPUs, providing a 5 TFLOPS peak per node, 204.8 GB/s memory bandwidth per CPU, and a peak of 25 GB/s for node interconnections. Perlmutter has a total of four I/O groups on the dragonfly network, and each I/O group is directly connected to each compute group via 100 GB/s. Therefore, the PM-GPU can achieve 5.6 TB/s file system peak (14 GPU groups $\times$ 4 I/O groups $\times$ 100 GB/s), and PM-CPU can achieve 4.8 TB/s (12 CPU groups $\times$ 4 I/O groups $\times$ 100 GB/s).

Cori Haswell (Cori-HSW) was used for LCLS as one of the demonstrated architectures. Cori-HSW is a Cray CX40 system, with each Burst Buffer (BB) node providing 6.5 GB/s (total 140 BB nodes, 910 GB/s). Note that Cori-HSW was deprecated, but the lessons learned from it are beneficial.

B. Workflow Characteristics

Due to the complexity of workflows, the case studies we examined required a variety of approaches to measure or estimate the node-local FLOPS and Bytes. This section summarizes the high-level methods of node- and system-performance metrics.

Table I summarizes our methods to characterize node- and system-performance metrics for different workflows. LCLS (Linac Coherent Light Source) is a data analysis workflow using Free Electron Lasers (XFELs) to determine the molecular structure and function of unknown samples (such as COVID-19 viral proteins). The wall clock time is reported in work [8].

TABLE I: Node- and System-Performance Characteristics

	LCLS	BerkelyGW	CosmoFlow	GPTune
Wall clock time	reported [8]	Measured	Measured	Measured
Node FLOPs	NA	reported [61]	NA	NA
CPU/GPU Bytes	Analytical model	reported [61]	Measured	Measured
Node PCIe Bytes	NA	NA	Analytical model	NA
System Network Bytes	NA	reported [61]	NA	NA
File System Bytes	Analytical model	reported [61]	Analytical model	Measured

The CPU bytes and file system bytes are characterized using an analytical model with domain knowledge. BerkeleyGW (BGW) is a many-body perturbation theory code for excited states. The code has been thoroughly optimized and was selected as a finalist for the 2020 Gordon Bell Prize. The timing used in this paper is measured on PM-GPU, and the performance metrics are reported in work [61]. CosmoFlow is a machine learning training benchmark from the MLPerf HPC benchmark [62]. We use a CosmoFlow throughput benchmark [63] with an average of twenty-five epochs per model. The numbers used in this paper are measured on PM-GPU nodes. GPTune is an autotuning framework that relies on multitask and transfer learnings to help solve the underlying black-box optimization problem using Bayesian optimization methodologies [64]. The wall clock time and node CPU bytes are reported from GPTune and the tuned application SuperLU_DIST [65]. The system-wide bytes are characterized using the input matrix size and the meta data size.

C. Workflow Analysis

1) LCLS: Figure 4 presents the LCLS workflow skeleton. The critical path length is two and it has five parallel tasks (A-E) at level 0. At Level 0, each task is a parallel application with thousands of MPI ranks. The data that needs to be loaded into the system from the external storage is 1 TB per task, and the output is 1 GB per task. These five tasks solve the same problem. Due to the various input data quality, the five tasks may run with different algorithms and different time costs. At level 1, task F performs a merge of the five output files.

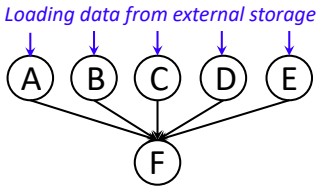


Fig. 4: LCLS Workflow skeleton. The critical path length is two, with five parallel tasks (A-E) at level 0, and each task is a parallel application with thousands of MPI ranks. Each task needs to load input data from the external storage.

Figure 5 presents the Workflow Roofline for LCLS on Cori-HSW and its time breakdown. The target makespan of LCLS was ten minutes in the year 2020. Thus, the target throughput is to finish the six tasks within those 10 minutes. During this time, it was observed that contention on the network, and filesystem with other workflows resulted 5× reduced performance from one day to another. Consequently, the two dots represent two cases, called "Good Days" and "Bad Days". The "Good Days" means one can load input data from external storage at an average rate of 1 GB/s, and the entire workflow can finish in 17 minutes as shown in Figure 5b. Correspondingly, the "Bad Days" indicates the network contention intensifies and decreases to 0.2 GB/s. Thus, the workflow needs 85 minutes to finish analyzing all the data. One can immediately notice that the external data loading bound the LCLS performance in both cases (the two dots overlapped with their system external boundary). The red dot (Bad Days) is well below the green dot (Good Days). The Workflow Roofline also tells that even with the average 1GB/s system external bandwidth, one can never meet the target.

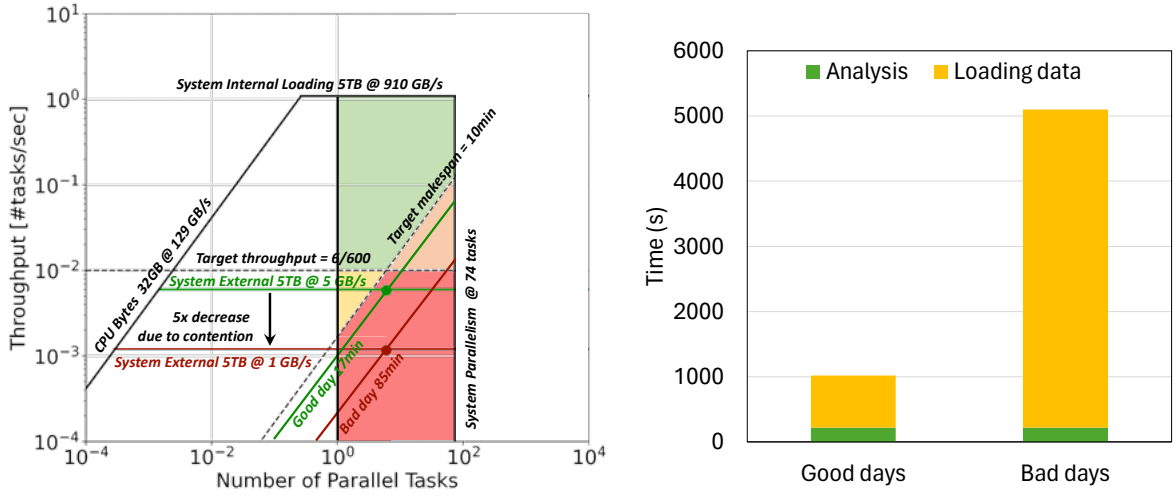
Figure 6 plots the implications of PM-CPU. Imagine that one loads the input data from the external storage using the data transfer node (DTN [66]). Each DTN node provides 25 GB/s for transfers to the internet. Ideally, one can load all 5 TB data in 3.4 minutes. Even with an ideal data transfer speed, it has very limited room for optimization of the makespan. The red horizontal system boundary is slightly above the target throughput dotted line. The system internal bandwidth is far on the top. It indicates the system internal bandwidth is not the bottleneck. Since there is no QOS on the network, the achievable system performance may drop off to 5 GB/s (5× decrease as observed in work [8]). In that case, one can never meet the targets due to the limited external data transfer speed.

A common fact across the two architectures is that resource contention can lower the system bandwidth ceiling, ultimately a bottleneck for LCLS to meet the targets. It emphasizes the importance of an end-to-end quality of service (QOS) to utilize the available quality of storage system (QSS) better.

2) BGW: BGW has one parallel task per level with a total of two levels. We refer "Task E" as Epsilon, and "Task S" as Sigma. Sigma needs to take Epsilon's output as its input.

We show the Workflow Roofline of BGW using the problem size of Si998 [61]. The total number of flops is 1164 PFLOPs and 3226 PFLOPs for task E and task S, respectively. The node ceiling can then be derived from the ratio of the number of flops per node and the node peak flops. For example, the node ceiling of using 64 nodes is $\frac{1164/64+3226/64}{9.7\cdot 4}$, where 9.7 TFLOPS is the FP64 peak of A100 GPU and Perlmutter has four A100 GPUs per node. The communication volume is constant regardless of the number of MPI ranks for a single batch [61]. In the tested case, there are 256 batches in total. Thus, the total communication volume in the system is fixed in strong scaling tests. Thus, the system network ceiling in Figure 7 can then be derived from the ratio of the communication volume and the aggregated node interconnection peak bandwidth: $\frac{\text{total communication volume}}{N\times 100\ \text{GB/s}}$, where N is the number of nodes. The data amount that is loaded from the file system is 70 GB. Therefore, there's a second system ceiling that refers to the data movement from the file system.

Figure 7a shows the BGW Workflow Roofline model using 64 nodes per task. Since BGW has only one task per level,



(a) Workflow Roofline on Cori-HSW

(b) Time breakdown

Fig. 5: The Workflow Roofline for time-sensitive workflow LCLS on Cori-HSW. (a) Resource contention can lower the system bandwidth ceiling, ultimately a bottleneck for LCLS to meet the targets. (2) Time breakdown. Loading data from the external storage is the bottleneck.

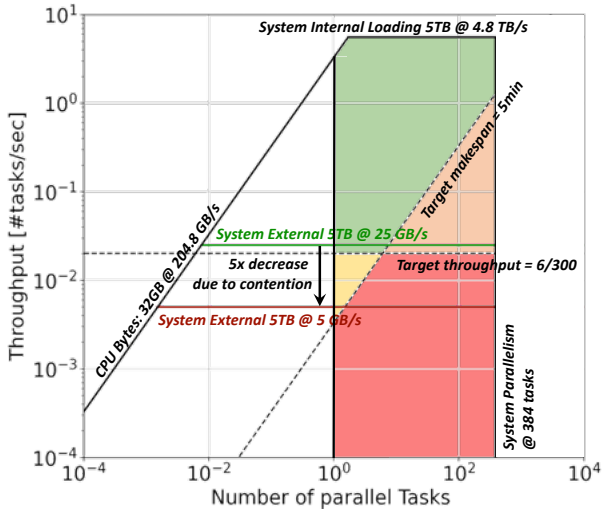


Fig. 6: The Workflow Roofline for LCLS on PM-CPU. A common fact is that resource contention can lower the system bandwidth ceiling, ultimately a bottleneck for LCLS to meet the targets. It emphasizes the importance of an end-to-end QOS to utilize the available QSS better.

its number of parallel tasks equals to one. Corresponding, the system parallelism wall on the right (x-axis equals 28) shows the maximum number of parallel tasks when scaled to the full system. When using 1024 nodes in Figure 7b, the system parallelism wall moves to the left, from 28 to 1 ($\lfloor \frac{1792 \text{ nodes}}{1024 \text{ notes}} \rfloor$). The two cases in Figure 7a and Figure 7b represent two scenarios: the single result is urgent (using 1024 nodes), and batch results are meaningful while time is not sensitive (64 nodes). In other words, one can get the one result back in minutes using 1024 nodes (fast but low throughput) for single urgent result. Alternatively, one can wait batch results in hours using 64 nodes running multiple instances (slow but high throughput).

Figure 7c is the task view of the Workflow Roofline model. It can guide developers and users for finer-grained optimization. The red color refers to Epsilon and the blue color represents Sigma. The number of nodes is differentiated by the light and dark color. The light blue and light red in Figure 7c uses 64 nodes, while the dark blue and dark red represent the case using 1024 nodes. One can immediately tell that the workflow makespan is dominated by Task S (blue circles) because it is at the lowest location. Recall that the lower location in the y-direction indicates a longer makespan in the Workflow Roofline model. For the case using 1024 nodes (upper part in Figure 7c), even though the two dots are crowed together, one can still tell that Task S (dark blue) takes slightly longer than Task E (dark red). In addition, Task E is farther away from the node ceiling than Task S. One can improve the node efficiently of Task E to improve workflow makespan.

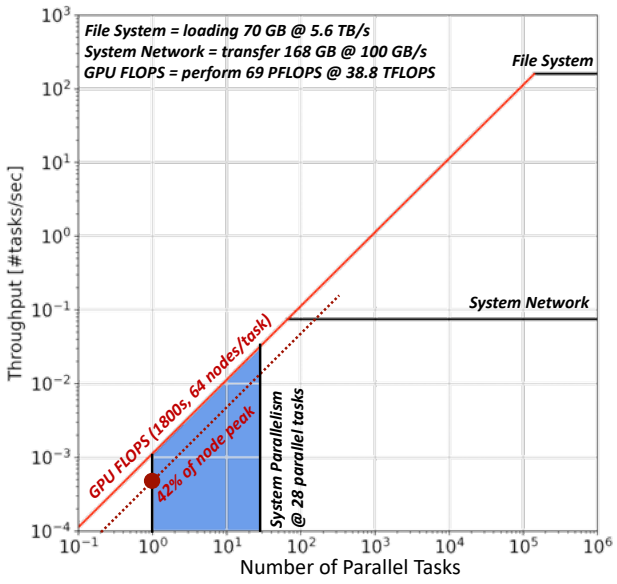
Figure 7d plots the Gantt chart of BGW using 64 nodes and 1024 nodes. The critical path remains the same at scales, while the critical path length varies due to time cost.

3) CosmoFlow: We focus on the throughput evaluation for CosmoFlow. In the throughput benchmark measurement [63], [67], we run multiple instances (twenty-five epochs per instances on average) of the model concurrently and report the performance as the number of epochs per second. In this context, it can be thought of as a proxy for a hyperparameter tuning workflow.

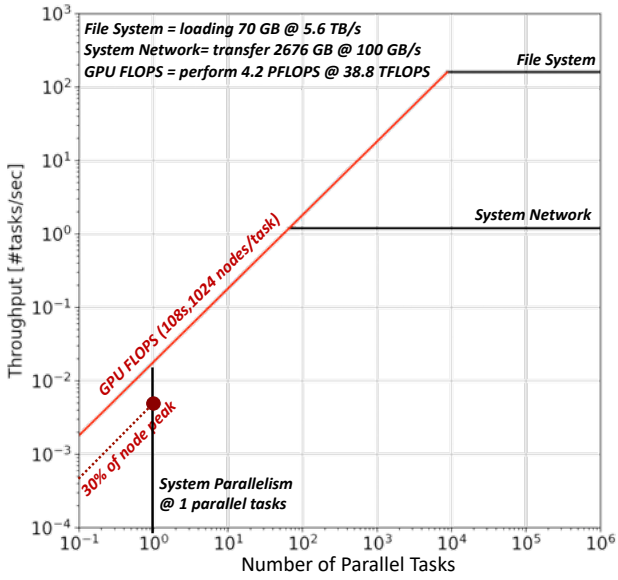
The training data set is 2 TB. After loading it, the code decompresses the 2TB data into 10TB and then transfers them from CPU to GPU via PCIe at 100 GB/s/node. Therefore, the PCIe Bytes in our test case are 80 GB per node ($\frac{10 \text{ TB}}{128 \text{ nodes}}$),

which denotes the diagonal PCIe makepan ceiling of 0.8s ($\frac{80\ \text{GB}}{100\ \text{GB/s}}$) in Figure 8. There are 2^{19} samples in total, and each sample requires 6.4 GB HBM data movement [68]. Thus, the diagonal HBM makespan ceiling can be calculated by $\frac{6.4\ \text{GB}\times 2^{19}\ \text{samples}}{1555\ \text{GB/s}\times 4\ \text{GPUs}\times 128\ \text{nodes}} = 4.2$ s.

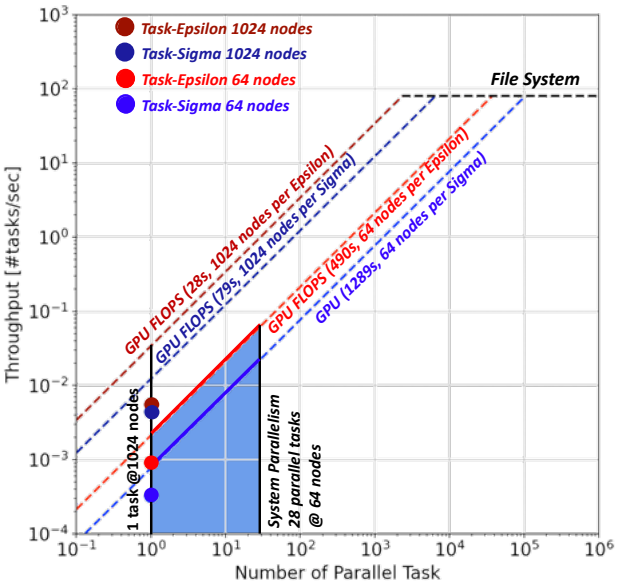
The empirical dots in Figure 8 are measured using 128 PM-GPU nodes per instances while increasing the number of instances (x-axis). There's one copy of the training data set on Perlmutter. Therefore, all models must load their inputs from the same training data set. As the number of instances increases, the throughput (y-axis, the number of epochs per second) increases proportionally. When running multiple instances, the number of epochs may come from different training. The total time cost to finish all models



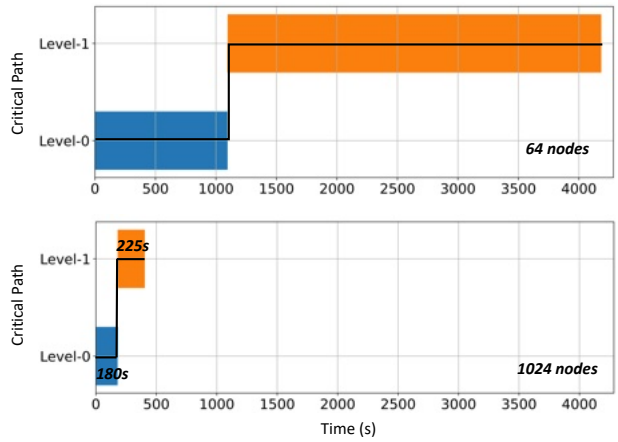
(a) Workflow Roofline on Perlmutter for BGW using 64 nodes



(b) Workflow Roofline on Perlmutter for BGW using 1024 nodes



(c) The Task View of Workflow Roofline on Perlmutter for BGW



(d) Gantt Chart on Perlmutter for BGW

Fig. 7: The Workflow Roofline for traditional HPC workflow BGW. (a) The node peak FLOPS ultimately bounds its performance as the empirical dots are close to the node diagonal ceiling. (b) Using more nodes per task pushes the system parallelism bound to the left (less available parallel tasks) and the system network ceiling to the upper direction (increased aggregate system network bandwidth). (c) The task view of the Workflow Roofline motivates future code optimization – the lower the dot is, the longer makespan it has. Dotted lines can never be achieved due to system parallelism limitation. (d) The Gantt chart for BGW. The critical path (solid back line) remains the same as it scales.

equals the duration between the earliest start time and the latest end time. Recall that there are 1536 GPU nodes (except 256 large memory nodes). Thus, one can run 12 models concurrently at maximum. Recall that Perlmutter has a peak of 5.6 TB/s file system peak. Thus, the file system ceiling is denoted as $\frac{2\ \text{TB}}{5.6\ \text{TB/s}}$. Note that the dots with an x-axis smaller than twelve instances are derived from the throughput measurements. Therefore, the resource contention could be over-provision which is the reason that we observe a linear relationship between the number of instances and the throughput.

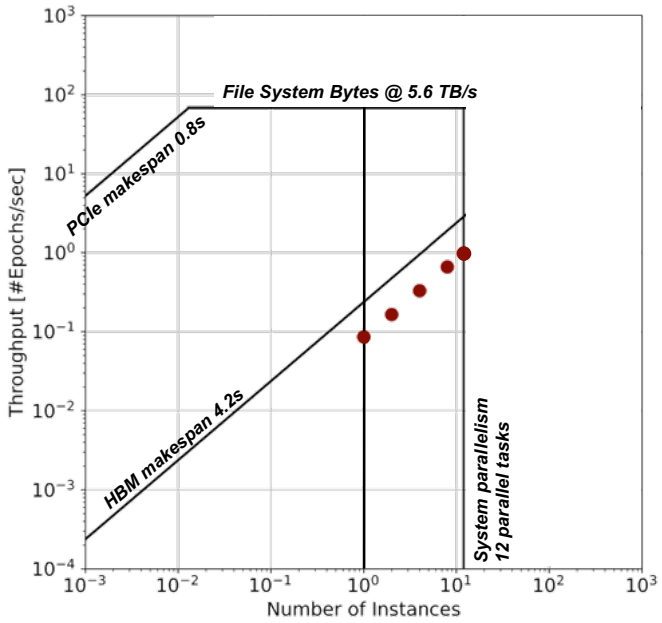
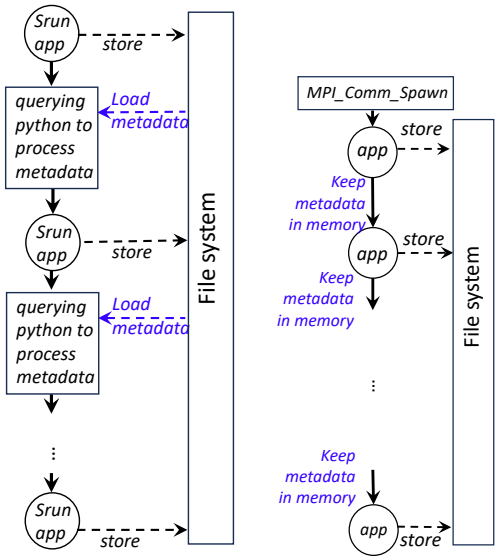


Fig. 8: Workflow Roofline for Cosmoflow on PM-GPU. We run twelve instances concurrently and HBM is ultimately the limitation.

4) GPTune: Figure 9a and Figure 9b show the two control flows or work modes) of GPTune: RCI and Spawn. RCI refers to using bash to control each iteration. This mode relies on interaction with the metadata (or the log file) from the file system in each autotuning iteration. RCI keeps querying Python for proposing the new samples for evaluation, calling "srun" to generate evaluation results and communicate them back to Python via the log file. Thus, every iteration requires a "srun" command and loads the entire metadata from the file system.

Spawn means that the entire tuning needs only one "srun", and the iterations are controlled via "MPI_Comm_Spawn". MPI_Comm_Spawn is the means by which MPI processes can create siblings. The spawning processes and spawned processes reside in two different communicators. Nonetheless, they can communicate together via the inter-communicator returned. In GPTune, the spawning process handles the meta in the memory, and the spawned processes call the superLU_DIST driver interface. Thus, each application run does not need to load metadata from the file system because those



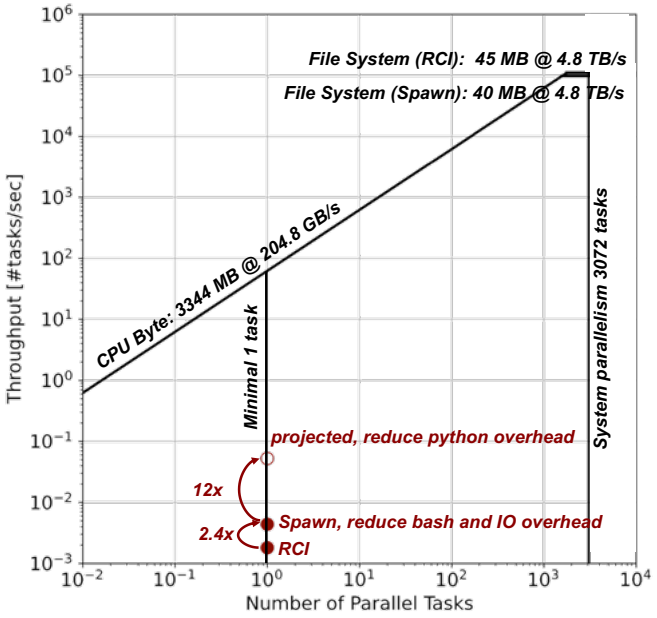
(a) RCI Control flow: via bash (b) Spawn Control flow: via spawn

Fig. 9: The workflow skeleton of GPTune. (a) RCI keeps querying python for loading the meta data from file system, and processing those data from next srun. (b) Spawn leverages MPI_Comm_Spawn to call the superLU_DIST driver and keeps the meta data in memory.

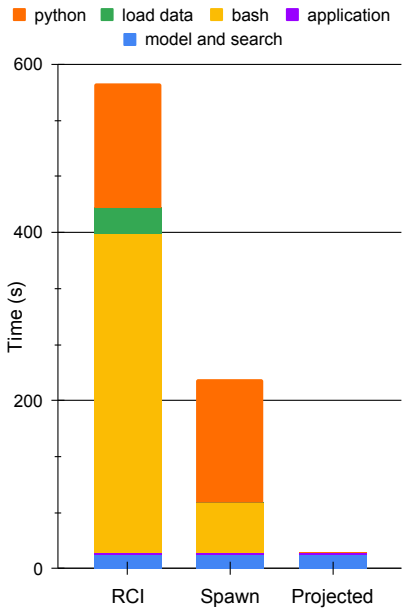
metadata are already in the memory. Therefore, the Spawn mode can save the bash overhead and the I/O time. Note that the application runs are serialized in GPTune due to the data dependencies.

Figure 10a plots the Workflow Roofline for GPTune on Perlmutter. The tuned application is SuperLU_DIST with a matrix size of 4960×4960. It tuned forty samples. Since all applications are serialized, the number of parallel tasks equals to one. The RCI mode takes 553s while the Spawn mode takes 228s (reduced batch and I/O time) as Figure 10b shows. One can immediately observe that from the Workflow roofline model, the Spawn dot is above the RCI dot. The Workflow Roofline model also suggests getting an extra 12× speedup by reducing the Python overhead (open dot). The Workflow Roofline model also indicates that the I/O pattern and concurrency play a more important role than I/O volume in this case: the two system bounds (horizontal, characterized by data volume) are very close to each other, but time cost in GPTune varies: 30s for RCI and 0.02s for Spawn.

GPTune usually prefers to use a representative benchmark to tune the parameters to get a result within an acceptable time. The sparse matrix size in real-world applications varies from hundreds to millions. Correspondingly, the application time varies from milliseconds to seconds. Note that we use a relatively small matrix to highlight the implications of different control flows. One can imagine that the bash and python overhead (500s in Figure 10b) still takes 50% of the time if the tuning benchmark takes 13s for each run (13s×40 samples=520s).



(a) Workflow Roofline model on Perlmutter



(b) Time breakdown

Fig. 10: The Workflow Roofline for GPTune on PM-CPU. (a) The Workflow Roofline for GPTune on Perlmutter. Using Spawn mode achieves a higher throughput than RCI because it reduces the bash and I/O time. The projected performance upper is 12× faster than Spawn by reducing the Python overhead. (b) Time breakdown of GPtune.

V. Conclusion

In this paper, we developed and applied a methodology for analyzing end-to-end workflow performance using the Roofline model. This allows us to analyze both node performance (FLOPS, data movement in CPU/GPU, etc.) and system performance (data transfer via interconnect, file system, etc.), thereby expanding the applicability of Roofline to workflow domains. It also allows us to analyze potential performance bottlenecks (node-bound vs. system-bound), thereby guiding workflow performance optimizations. The insights and recommendations we gained from the case studies are threefold. The first, intended for system architects posits that if a time-sensitive workflow like LCLS is dominated by a system's external bandwidth, then going for a faster computing unit is a bad idea. Increasing a workflow's computation speed by 10× makes no difference from today's observation that the system's external bandwidth still bounds LCLS. Instead, system architects should work on the network and storage QOS, which is essential for providing optimal system services. The second one is for workflow developers. For example, if one has a workflow like GPTune, which is dominated by Python library loading time (captured as a diagonal representing overhead time), it is worth considering using containers to avoid such overheads. Workflow users and workflow management teams might focus on the third insight: depending on the urgency of tasks, one can schedule the urgent ones on a large scale to get one single result back quickly or merge non-urgent tasks into a batch job to get batch results in a longer time.

The Workflow Roofline model has two limitations we will address in our future work. One is that the total number of tasks, or critical path length, is hidden in the y-axis (throughput). Therefore, learning whether the poor pipeline strategy limits the workflow's performance is not intuitive. The second one is the overhead of on-node workflow execution characterization. Since tasks may be large applications, the on-node profiling overhead, such as memory bytes, could be significant. Thus, users must manually profile the representative ranks or use an analytical model for workflow execution characterization.

VI. Acknowledgement

This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231 and used resources of the National Energy Research Scientific Computing Center (NERSC) which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The future of scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 159–175, 2018.

[2] D. T. Moustakas, P. T. Lang, S. Pegg, E. Pettersen, I. D. Kuntz, N. Brooijmans, and R. C. Rizzo, "Development and validation of a modular, extensible docking program: Dock 5," *Journal of computer-aided molecular design*, vol. 20, pp. 601–619, 2006.

[3] J. Ekanayake, S. Pallickara, and G. Fox, "Mapreduce for data intensive scientific analyses," in *2008 IEEE Fourth International Conference on eScience*. IEEE, 2008, pp. 277–284.

[4] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[5] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.

[6] "NERSC-10 Workflow Archetypes White Paper," https://www.nersc.gov/assets/NERSC-10/Workflows-Archetypes-White-Paper-v1.0.pdf, 2023.

[7] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook *et al.*, "Cosmoflow: Using deep learning to learn the universe at scale," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 819–829.

[8] J. P. Blaschke, A. S. Brewster, D. W. Paley, D. Mendez, A. Bhowmick, N. K. Sauter, W. Kröger, M. Shankar, B. Enders, and D. Bard, "Real-time xfel data analysis at slac and nersc: a trial run of nascent exascale experimental data analysis," *arXiv preprint arXiv:2106.11469*, 2021.

[9] R. F. da Silva, H. Casanova, K. Chard, T. Coleman, D. Laney, D. Ahn, S. Jha, D. Howell, S. Soiland-Reys, I. Altintas *et al.*, "Workflows community summit: Advancing the state-of-the-art of scientific workflows management systems research and development," *arXiv preprint arXiv:2106.05177*, 2021.

[10] N. Tallent, R. Friese, J. Suetterlein, J. Strube, M. Schram, T. Elsethagen, L. DeLaTorre, M. Halappanavar, K. J. Barker, K. K. Van Dam *et al.*, "Integrated end-to-end performance prediction and diagnosis for extreme scientific workflows," , Santa Cruz, Tech. Rep., 2017.

[11] C. D. Carothers, "Predictive modeling and diagnostic monitoring of extreme science workflows," Rensselaer Polytechnic Inst., Troy, NY (United States), Tech. Rep., 2019.

[12] "RAMSES: Robust Analytical Models for Science at Extreme Scales," https://www.anl.gov/mcs/ramses-robust-analytical-models-for-science-at-extreme-scales#:~:text=The%20aim%20of%20the%20RAMSES,in%20extreme%2Dscale%20science%20environments., 2023.

[13] M. Li, S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman, "Functional partitioning to optimize end-to-end performance on many-core architectures," in *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–12.

[14] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi, "Enabling in-situ execution of coupled scientific workflow on multi-core platform," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 1352–1363.

[15] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci *et al.*, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–9.

[16] M. Taufer, S. Thomas, M. Wyatt, T. M. A. Do, L. Pottier, R. F. da Silva, H. Weinstein, M. A. Cuendet, T. Estrada, and E. Deelman, "Characterizing in situ and in transit analytics of molecular dynamics simulations for next-generation supercomputers," in *2019 15th International Conference on eScience (eScience)*. IEEE, 2019, pp. 188–198.

[17] M. Dreher and B. Raffin, "A flexible framework for asynchronous in situ and in transit analytics for scientific simulations," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 277–286.

[18] B. Friesen, A. Almgren, Z. Lukić, G. Weber, D. Morozov, V. Beckner, and M. Day, "In situ and in-transit analysis of cosmological simulations," *Computational astrophysics and cosmology*, vol. 3, no. 1, pp. 1–18, 2016.

[19] K. Moreland, R. Oldfield, P. Marion, S. Jourdain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld *et al.*, "Examples of in transit visualization," in *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, 2011, pp. 1–6.

[20] L. Pottier, R. F. da Silva, H. Casanova, and E. Deelman, "Modeling the performance of scientific workflow executions on hpc platforms with burst buffers," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2020, pp. 92–103.

[21] G. Haldeman, I. Rodero, M. Parashar, S. Ramos, E. Z. Zhang, and U. Kremer, "Exploring energy-performance-quality tradeoffs for scientific workflows with in-situ data analyses," *Computer Science-Research and Development*, vol. 30, pp. 207–218, 2015.

[22] I. Rodero, M. Parashar, A. G. Landge, S. Kumar, V. Pascucci, and P.-T. Bremer, "Evaluation of in-situ analysis strategies at scale for power efficiency and scalability," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2016, pp. 156–164.

[23] F. Poeschel, J. E, W. F. Godoy, N. Podhorszki, S. Klasky, G. Eisenhauer, P. E. Davis, L. Wan, A. Gainaru, J. Gu *et al.*, "Transitioning from file-based hpc workflows to streaming data pipelines with openpmd and adios2," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, 2021, pp. 99–118.

[24] Q. Wu, D. Yun, X. Lin, Y. Gu, W. Lin, and Y. Liu, "On workflow scheduling for end-to-end performance optimization in distributed network environments," in *Job Scheduling Strategies for Parallel Processing: 16th International Workshop, JSSPP 2012, Shanghai, China, May 25, 2012. Revised Selected Papers 16*. Springer, 2013, pp. 76–95.

[25] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li, "End-to-end delay minimization for scientific workflows in clouds under budget constraint," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 169–181, 2014.

[26] M. Kaur and S. Kadam, "Bio-inspired workflow scheduling on hpc platforms," *Tehnički glasnik*, vol. 15, no. 1, pp. 60–68, 2021.

[27] K. B. Antypas, D. J. Bard, J. P. Blaschke, R. Shane Canon, B. Enders, M. A. Shankar, S. Somnath, D. Stansberry, T. D. Uram, and S. R. Wilkinson, "Enabling discovery data science through cross-facility workflows," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 3671–3680.

[28] J. Ejarque, R. M. Badia, L. Albertin, G. Aloisio, E. Baglione, Y. Becerra, S. Boschert, J. R. Berlin, A. D'Anca, D. Elia *et al.*, "Enabling dynamic and intelligent workflows for hpc, data analytics, and ai convergence," *Future generation computer systems*, vol. 134, pp. 414–429, 2022.

[29] G. Li, J. Woo, and S. B. Lim, "Hpc cloud architecture to reduce hpc workflow complexity in containerized environments," *Applied Sciences*, vol. 11, no. 3, p. 923, 2021.

[30] J. González-Abad, Á. López García, and V. Y. Kozlov, "A container-based workflow for distributed training of deep learning algorithms in hpc clusters," *Cluster Computing*, vol. 26, no. 5, pp. 2815–2834, 2023.

[31] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "Streamflow: cross-breeding cloud with hpc," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1723–1737, 2020.

[32] S. Jha, V. R. Pascuzzi, and M. Turilli, "Ai-coupled hpc workflows," *arXiv preprint arXiv:2208.11745*, 2022.

[33] W. E. Denzel, J. Li, P. Walker, and Y. Jin, "A framework for end-to-end simulation of high-performance computing systems," *Simulation*, vol. 86, no. 5-6, pp. 331–350, 2010.

[34] A. Clum, M. Huntemann, B. Bushnell, B. Foster, B. Foster, S. Roux, P. P. Hajek, N. Varghese, S. Mukherjee, T. Reddy *et al.*, "Doe jgi metagenome workflow," *Msystems*, vol. 6, no. 3, pp. 10–1128, 2021.

[35] D. Król, R. F. da Silva, E. Deelman, and V. E. Lynch, "Workflow performance profiles: development and analysis," in *Euro-Par 2016: Parallel Processing Workshops: Euro-Par 2016 International Workshops, Grenoble, France, August 24-26, 2016, Revised Selected Papers 22*. Springer, 2017, pp. 108–120.

[36] M. L. Mondelli, M. T. de Souza, K. Ocana, A. De Vasconcelos, and L. M. Gadelha Jr, "Hpsw-prof: a provenance-based framework for profiling high performance scientific workflows," in *Proceedings of Satellite Events of the 31st Brazilian Symposium on Databases (SBBD 2016), SBC*, 2016, pp. 117–122.

[37] R. Ferreira da Silva, W. Chen, G. Juve, K. Vahi, and E. Deelman, "Community resources for enabling and evaluating research on scientific workflows," in *IEEE International Conference on e-Science. eScience*, vol. 14, 2014.

[38] D. Garijo, P. Alper, K. Belhajjame, O. Corcho, Y. Gil, and C. Goble, "Common motifs in scientific workflows: An empirical analysis," *Future generation computer systems*, vol. 36, pp. 338–351, 2014.

[39] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future generation computer systems*, vol. 29, no. 3, pp. 682–692, 2013.

[40] L. Ramakrishnan and D. Gannon, "A survey of distributed workflow characteristics and resource requirements," *Indiana University*, pp. 1–23, 2008.

[41] L. Versluis, R. Mathá, S. Talluri, T. Hegeman, R. Prodan, E. Deelman, and A. Iosup, "The workflow trace archive: Open-access data from public and private computing infrastructures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2170–2184, 2020.

[42] G. Papadimitriou, C. Wang, K. Vahi, R. F. da Silva, A. Mandal, Z. Liu, R. Mayani, M. Rynge, M. Kiran, V. E. Lynch *et al.*, "End-to-end online performance data capture and analysis for scientific workflows," *Future Generation Computer Systems*, vol. 117, pp. 387–400, 2021.

[43] D. Ghoshal, B. Austin, D. Bard, C. Daley, G. Lockwood, N. J. Wright, and L. Ramakrishnan, "Characterizing scientific workflows on hpc systems using logs," in *2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2020, pp. 57–64.

[44] T. Ben-Nun, T. Gamblin, D. S. Hollman, H. Krishnan, and C. J. Newburn, "Workflows are the new applications: Challenges in performance, portability, and productivity," in *2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2020, pp. 57–69.

[45] R. F. Da Silva, H. Casanova, K. Chard, I. Altintas, R. M. Badia, B. Balis, T. Coleman, F. Coppens, F. Di Natale, B. Enders *et al.*, "A community roadmap for scientific workflows research and development," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2021, pp. 81–90.

[46] R. F. da Silva, H. Casanova, K. Chard, D. Laney, D. Ahn, S. Jha, C. Goble, L. Ramakrishnan, L. Peterson, B. Enders *et al.*, "Workflows community summit: Bringing the scientific workflows community together," *arXiv preprint arXiv:2103.09181*, 2021.

[47] R. F. da Silva, R. M. Badia, V. Bala, D. Bard, P.-T. Bremer, I. Buckley, S. Caino-Lores, K. Chard, C. Goble, S. Jha *et al.*, "Workflows community summit 2022: A roadmap revolution," *arXiv preprint arXiv:2304.00019*, 2023.

[48] J. Dean and L. A. Barroso, "The tail at scale software techniques that tolerate latency variability are vital to building responsive large-scale web services."

[49] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Commun. ACM*, vol. 52, no. 4, 2009.

[50] N. Ding and S. Williams, *An instruction roofline model for gpus*. IEEE, 2019.

[51] C. Yang, T. Kurth, and S. Williams, "Hierarchical roofline analysis for gpus: Accelerating performance optimization for the nersc-9 perlmutter system," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 20, p. e5547, 2020.

[52] N. Ding, M. Haseeb, T. Groves, and S. Williams, "Evaluating the performance of one-sided communication on cpus and gpus," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1059–1069.

[53] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei, and M. Chen, "Cost and makespan-aware workflow scheduling in hybrid clouds," *Journal of Systems Architecture*, vol. 100, p. 101631, 2019.

[54] J. Dongarra, K. London, S. Moore, P. Mucci, and D. Terpstra, "Using papi for hardware performance monitoring on linux systems," in *Conference on Linux Clusters: The HPC Revolution*, vol. 5. Linux Clusters Institute, 2001.

[55] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular hpc i/o characterization with darshan," in *2016 5th workshop on extreme-scale programming tools (ESPT)*. IEEE, 2016, pp. 9–17.

[56] "Perlmutter Architecture ," https://docs.nersc.gov/systems/perlmutter/architecture/, 2023.

[57] J. Choquette, E. Lee, R. Krashinsky, V. Balan, and B. Khailany, "3.2 the a100 datacenter gpu and ampere architecture," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 48–50.

[58] G. K. Lockwood, A. Chiusole, L. Gerhardt, K. Lozinskiy, D. Paul, and N. J. Wright, "Architecture and performance of perlmutter's 35 pb clusterstor e1000 all-flash file system," 2021.

[59] "Sbatch," https://docs.nersc.gov/jobs/#sbatch, 2023.

[60] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich *et al.*, "Common workflow language, v1. 0," 2016.

[61] M. Del Ben, C. Yang, Z. Li, F. H. da Jornada, S. G. Louie, and J. Deslippe, "Accelerating large-scale excited-state gw calculations on leadership hpc systems," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–11.

[62] "MLPerf Training: HPC Benchmark Suite," https://mlcommons.org/benchmarks/, 2024.

[63] "Cosmoflow benchmark," https://github.com/mlcommons/hpc_results_v3.0, 2024.

[64] Y. Liu, W. M. Sid-Lakhdar, O. Marques, X. Zhu, C. Meng, J. W. Demmel, and X. S. Li, "Gptune: Multitask learning for autotuning exascale applications," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 234–246.

[65] Y. Liu, N. Ding, P. Sao, S. Williams, and X. S. Li, "Unified communication optimization strategies for sparse triangular solver on cpu and gpu clusters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–15.

[66] "Data Transfer Nodes," https://docs.nersc.gov/systems/dtn/, 2024.

[67] "Cosmoflow benchmark implementation," https://github.com/mlcommons/hpc_results_v3.0/tree/main/HPE%2BLBNL/benchmarks/cosmoflow/implementations/cosmoflow-pytorch, 2024.

[68] K. Z. Ibrahim, T. Nguyen, H. A. Nam, W. Bhimji, S. Farrell, L. Oliker, M. Rowan, N. J. Wright, and S. Williams, "Architectural requirements for deep learning workloads in hpc environments," in *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 2021, pp. 7–17.

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

- C_1 Definition of Workflow Roofline ceilings that characterize the peak workflow performance.
- C_2 Creation of System boundaries that define a range of attainable performance.
- C_3 Development of a workflow characterization methodology that incorporates number of parallel tasks, workflow makespan (latency), and workflow throughput into the Workflow Roofline model.
- C_4 Development of a workflow execution characterization methodology that allows easy visualization of potential performance constraints.
- C_5 Evaluation of the Workflow Roofline Model and methodology on four workflows: LCLS (data analysis, system external bound), BerkeleyGW (traditional HPC, node bound), CosmoFlow (hyperparameter tuning, node HBM bound) and GPTune (auto-tuner, control flow bound).

B. Computational Artifacts

Note that all computational artifacts are archived under a single DOI.

- A_1 https://sandbox.zenodo.org/doi/10.5072/zenodo.44625

Artifact ID	Contributions Supported	Related Paper Elements
A_1	all	Figure 1,5-8,10

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

All computational artifacts are archived under A_1. It contains eight scripts for plotting the Workflow Roofline models. The relation to the contributions and the corresponding paper elements of those scripts are listed below.

Script Name	Contributions Supported	Related Paper Elements
example.py	C_1 and C_2	Figure 1
WRF_LCLS_HSW.py	C_3,C_4 and C_5	Figure 5(a)
WRF_LCLS_PM.py	C_3,C_4 and C_5	Figure 6
WRF_BGW_64.py	C_3,C_4 and C_5	Figure 7(a)
WRF_BGW_1024.py	C_3,C_4 and C_5	Figure 7(b)
WRF_BGW_task.py	C_3,C_4 and C_5	Figure 7(c)
WRF_COSMO_PM.py	C_3,C_4 and C_5	Figure 8
WRF_GPTUNE_PM.py	C_3,C_4 and C_5	Figure 10(a)

Expected Results

Plots the Workflow Roofline model frames for the evaluated workflows without text labels.

Expected Reproduction Time (in Minutes)

The expected computational time of plotting all Workflow Roofline model plots is within 1 min.

The expected computational time of measuring time, and other performance metric varies from minutes to hours. Note that the queue time is not counted here.

Artifact Setup (incl. Inputs)

Hardware: Plotting the Workflow Roofline model can be done at any hardware. We plotted the figures on a MacBook laptop.

Software: Python3.10

Datasets / Inputs: Plotting a Workflow Roofline model requires two kinds of input: system input and workflow input. The system input can be obtained from the system white paper. Due to workflow diversity, the workflow inputs can be very different. We obtain the workflow inputs by using the analytical model with domain knowledge. We listed our inputs below. Please note that all the inputs are also documented in the scripts. "E" represents the empirical numbers, and "A" indicates the numbers obtained from the analytical models.

1) Perlmutter (PM) Peak Performance:

- File system bandwidth (GPU partition): 5.6 TB/s
- File system bandwidth (CPU partition): 4.8 TB/s
- Network bandwidth: N $\times$ 100 GB/s
- Total number of GPU node: 1792

- Total number of CPU node: 3072
- CPU-GPU PCIe bandwidth per GPU node: 4 × 25 GB/s
- Compute performance per GPU node: 4 × 9.7 TFLOPS
- Memory bandwidth per GPU node: 4 × 1555 GB/s
- Memory bandwidth per CPU node: 2 × 204.8 GB/s
- System external Bandwidth: 25 GB/s

2) Cori Haswell (HSW) Peak Performance:

- Peak burst buffer node bandwidth: 910 GB/s
- Total number of CPU node: 2388
- Peak memory bandwidth per CPU node: 129 GB/s

3) LCLS Workflow Inputs:

- (A) Number of tasks: 6
- (A) Number of parallel tasks: 5
- (A) Target makespan (latency) in 2020: 600 sec
- (A) Target throughput in 2020: 6 tasks/ 600 sec
- (A) Loaded data volume from external: 5 tasks × 1 TB
- (A) Data volume per CPU node: 32 GB
- (A) Number of processes per task: 1024
- (A) Target makespan (latency) in 2024: 300 sec
- (A)Target throughput in 2024: 6 tasks/ 300 sec
- (E) Makespan (latency) in 2020: reported from LCLS
- (E) Time cost (end-to-end) in good days: 17 min
- (E) Time cost (end-to-end) in bad days: 85 min

4) BerkeleyGW (BGW) Workflow Inputs:

- (A) Number of tasks: 2
- (A) Number of parallel tasks: 1
- (A) Loaded data volume from file system: 70 GB
- (A) Number of processes per task: 64 (or 1024)
- (A) Network (MPI) volume per GPU node: 2676 GB (or 168 GB)
- (A) GPU FLOPS per node: 69 PFLOPS (or 4.2 PFLOPS)
- (E) Time cost: 4184.86 sec (or 404.74 sec)

5) CosmoFlow Throughput (COSMO) Workflow Inputs:

- (A) Number of tasks: 12
- (A) Number of parallel tasks: 12
- (A) Loaded data volume from file system: 2 TB per task
- (A) Number of GPU nodes per task: 128
- (A) HBM time: 4.2 sec
- (E) Time cost of each instance: at line 15-16 in WRF_COSMO_PM.py

6) GPTune Workflow Inputs:

- (A) Number of tasks: 1
- (A) Number of parallel tasks: 1
- (A) Loaded data volume from file system: 45 MB
- (A) Number of CPU nodes per task: 1
- (E) CPU Bytes: 3344 MB per CPU socket
- (E) Time cost: at line 17 in WRF_GPTUNE_PM.py

Installation and Deployment: One need to install the workflow to obtain the reported timing. The BerkeleyGW requires PrgEnv-nvhpc, cray-hdf5-parallel, cray-fftw, cray-libsci, and python modules on Perlmutter.

The LCLS workflow repo is: https://github.com/cctbx/cctbx_project. Please note that we refer the execution time on Cori Haswell in the paper but Cori HaswellW was deprecated. The timings we reported in the paper are not reproducible anymore.

One can follow the description in CosmoFlow Throughput repo (https://github.com/mlcommons/hpc_results_v3.0.git). In the ../cosmoflow-pytorch directory, run with:

source configs/config_pm_128x4x1.sh

sbatch -N ${Number_of_nodes} -t ${Time} run_pm.sub

One can follow the instructions in GPTune repo: https://github.com/gptune/GPTune.git. The *config_perlmutter.sh* installs GPTune with mpi, python, compiler, cudatoolkit and cmake modules on Perlmutter. Note that you need to set"proc=milan #(CPU nodes) or gpu #(GPU nodes)","mpi=openmpi or craympich" and"compiler=gnu". Setting mpi=craympich will only support the RCI mode.

Artifact Execution

The scripts are independent of each other. The scripts follow *WRF_${WORKFLOW_NAME}_${MACHINE}.py* naming convention. For example, WRF_LCLS_HSW.py is the workflow roofline plot for LCLS on Cori Haswell, while WRF_LCLS_PM.py is the plot script for LCLS on Perlmutter.

Execution command: *python3.10 script_name*. For instance, *python3.10 WRF_LCLS_HSW.py*.

Artifact Analysis (incl. Outputs)

Workflow Roofline scripts: The scripts' outputs are Workflow Roofline model plots. The PDF output follows the same naming convention as the Python script. For instance, the output of WRF_LCLS_PM.py is WRF_LCLS_PM.pdf, which is the Workflow Roofline model for the LCLS workflow on Perlmutter.

The timing is reported at the end of each workflow. Note that we use analytical models to estimates the data movement and flops of the evaluated workflows.

Artifact Evaluation (AE)

A. Computational Artifact A_1

Artifact Setup (incl. Inputs)

The Workflow Roofline plots repo can be reproduced on Chameleon Cloud using image "Ubuntu22.04-WorflowRoofline-20240624".

Artifact Execution

First, create an instance using image "Ubuntu22.04-WorflowRoofline-20240624" on Chameleon Cloud, and then ssh $cc@\langle$floating IP address$\rangle$.

After logging in, enter the "workflow_roofline_scripts" directory, and then run "sh plot_all_figures_wo_text.sh". You will find eight *.pdf* files in that folder. They are the plots of the Workflow Roofline figures without text labels.

Artifact Analysis (incl. Outputs)

The expected results are the plots of the Workflow Roofline models without text labels. Since we use analytical models to characterize the workflows, all the detailed metadata, such as flops, bytes, and machine peaks, are contained in the plotting scripts. The relation to the outputs and the corresponding paper elements of those scripts are listed below.

Script Name	Script Outputs	Related Paper Elements
example.py	example.pdf	Figure 1
WRF_LCLS_HSW.py	WRF_LCLS_HSW.pdf	Figure 5(a)
WRF_LCLS_PM.py	WRF_LCLS_PM.pdf	Figure 6
WRF_BGW_64.py	WRF_BGW_64.pdf	Figure 7(a)
WRF_BGW_1024.py	WRF_BGW_1024.pdf	Figure 7(b)
WRF_BGW_task.py	WRF_BGW_task.pdf	Figure 7(c)
WRF_COSMO_PM.py	WRF_COSMO_PM.pdf	Figure 8
WRF_GPTUNE_PM.py	WRF_GPTUNE_PM.pdf	Figure 10(a)