SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis | 979-8-3503-5554-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/SCW63240.2024.00231

Vendor-neutral and Production-grade Job Power Management in High Performance Computing

Naman Kulshreshtha
Clemson University
Clemson, USA
nkulshr@clemson.edu

Tapasya Patki
Lawrence Livermore National Laboratory
patki1@llnl.gov

Jim Garlick
Lawrence Livermore National Laboratory
garlick1@llnl.gov

Mark Grondona
Lawrence Livermore National Laboratory
grondona1@llnl.gov

Rong Ge
Clemson University
Clemson, USA
rge@clemson.edu

***Abstract*—Power management and energy efficiency are critical research areas for exascale computing and beyond, necessitating reliable telemetry and control for distributed systems. Despite this need, existing approaches present several limitations precluding their adoption in production. These limitations include, but are not limited to, lack of portability due to vendor-specific and closed-source solutions, lack of support for non-MPI applications, and lack of user-level customization.**

We present a job-level power management framework based on Flux. We introduce `flux-power-monitor` and demonstrate its effectiveness on the Lassen (IBM Power AC922) and Tioga (HPE Cray EX235A) systems with a low average overhead of 0.4%. We also present `flux-power-manager`, where we discuss a proportional sharing policy and introduce a hierarchical FFT-based dynamic power management algorithm (FPP). We demonstrate that FPP reduces energy by 1% compared to proportional sharing, and by 20% compared to the default IBM static power capping policy.

I. Introduction

Power management and energy efficiency are critical research areas in exascale computing and beyond. With the advent of Artificial Intelligence and Machine Learning (AI/ML) based scientific workflows [9], [37], the power demands for High-performance computing (HPC) systems are expected to increase. Similar trends are also observed in the cloud community [12]. These practical concerns, combined with the desire for improved sustainability in supercomputing [18], are necessitating a need for scalable production-grade power telemetry and dynamic power control frameworks.

Significant research has been conducted in HPC and Cloud power management with techniques focusing on energy-efficiency research [11], [15], [19], [30], static and dynamic power management [13], [16], [24], [31], [35], [38]–[40], and hardware overprovisioning [23], [28]. Notable production-grade and open-source solutions include runtime systems such as Intel GEOPM [15] and EAR [11] that focus on application-specific optimizations for traditional MPI-based applications, and power-aware job scheduling plugins provided through SLURM [4], [5], [10], [31], [32]. Many other fragmented vendor-specific solutions have emerged [20], [33].

Despite the demonstrated benefits of these approaches, their adoption in production-grade systems that are in the Top500 has been incredibly slow. Each of these existing approaches have at least one and often many of the following limitations – they are specific to a particular system (locked in to a specific hardware vendor or device type) [4], [5], [31], [32], are not open-source [20], are lacking support for modern non-MPI workflows [11], [15], are not customizable from a power policy perspective [20], [32], or are not portable across HPC and Cloud setups [20], [32], [33]. Furthermore, these existing approaches are typically designed for system administrators with elevated privileges. User-level support for telemetry and dynamic power management is unavailable.

In this paper, we address the above challenges and present a job-level power management framework based on the Flux [6] framework[1]. Flux is an open-source resource management framework developed by Lawrence Livermore National Laboratory (LLNL) and will be deployed on the El Capitan supercomputer. Flux has been designed to support not just traditional HPC systems, but also cloud computing infrastructure, including integrations with Kubernetes [21], [34].

The Flux-based job power management framework presented in this paper is vendor-neutral, scalable, and extensible. It leverages interfaces provided by Variorum [27], a vendor-agnostic and open-source power management library. As a result, it supports multiple microarchitectures across Intel (CPU/GPU), AMD (CPU/GPU), IBM, ARM and NVIDIA platforms, including support for all three DOE exascale systems. It comprises of two modules, the `flux-power-monitor` for job-level power telemetry and the `flux-power-manager` for both static and dynamic job power management. It's flexible design for both telemetry and control goes beyond traditional MPI applications, applying to anything that can be launched under a Flux job including non-MPI frameworks such as Charm++ [2], Python-based AI/ML workflows, groups of sequential programs launched together, or even arbitrary applications self-launched under an allocation. As part of the Flux framework, it inherently supports user-level telemetry and power management policy customization, where different users can choose different power-aware scheduling policies within their respective allocations. To the best of our knowledge, this is the first vendor-neutral,

[1]Our framework is available at `https://github.com/flux-framework/flux-power-mgr`. The `flux-power-monitor` module is production-quality and is slated for a Fall 2024 release. The `flux-power-manager` is experimental as we do not intend to use power capping on our production systems yet. Each site has different power management objectives and our goal with this work is to demonstrate capability and not advocate one policy over another.

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00231

open-source, scalable, extensible and low-overhead job-level power management framework. We believe this will enable both the HPC and cloud community to test, integrate and adopt dynamic power management policies in production.

The paper is organized as follows. We provide a brief background of the tools and applications used in this paper in Section II. We then present the design of `flux-power-monitor` and `flux-power-manager` in Section III. We present a proportional power sharing policy and a novel Fast Fourier Transform (FFT) based dynamic power capping(FPP) algorithm. In Section IV, we first demonstrate the effectiveness and vendor neutrality of `flux-power-monitor` across two diverse HPC systems at LLNL, Lassen (IBM Power AC922 system, #57 on Top500) and Tioga (HPE Cray EX235A system, #187 on Top500). We discuss how it achieves a low overhead of under 0.4% on average. Then, we evaluate `flux-power-manager` on the Lassen system showcasing both static and dynamic power capping capabilities and demonstrate how FPP reduces energy consumption by 1.2% with less than a 0.8% performance loss on average, when compared to the proportional power sharing policy; and by 20% with a 1.58x performance gain when compared to default node-level power capping from IBM. Finally, we include a discussion on production challenges encountered and summarize our work.

II. Background

A. HPC Systems

We base our experiments on two systems, Lassen and Tioga, located at LLNL. Lassen is a 23-petaflop supercomputer with IBM AC922 server nodes. Each dual-socket node has 44 IBM Power9 cores, 4 NVIDIA Volta GPUs, 256 GB of CPU memory and 64 GB of high bandwidth (HBM2) GPU memory. It has a total of 792 nodes, and is connected with Mellanox 100 Gb/s Enhanced Data Rate (EDR) InfiniBand. Power telemetry is supported with in-band power sensors reported at the node, socket (CPU cores), memory and GPU levels by the On-Chip Controller (OCC) at 500 micro-second granularity [1]. Node-level power telemetry is directly supported in hardware and includes uncore components.

IBM supports node-level power capping on this architecture with the OpenPower Abstraction Layer (OPAL) firmware. The maximum power per node is 3050 W, and the minimum possible soft power cap (not guaranteed by hardware) is 500 W [27]. The soft power cap is typically only applicable if GPUs are not utilized. The minimum possible hard power cap with GPU activity (guaranteed by hardware) is 1000 W. For any given node power cap, the default IBM algorithm estimates the power share for the GPU and CPU and determines a maximum power cap for the GPUs. The ratio of distribution can be modified using the Power Shifting Ratio (PSR), which ranges from 0% to 100% on each socket. In this paper, the PSR is always set to 100 (default), implying maximum power share to the GPUs. GPU power capping is also supported directly, through NVIDIA Management Library (NVML). Each GPU has a peak power of 300 W and minimum power of 100 W.

Tioga is a 5.8-petaflop system with HPE Cray EX235a server nodes. This is an early access system for El Capitan with a total of 32 nodes. Each node has a single-socket AMD Trento processor with 64 cores, along with AMD Instinct MI250X accelerator modules. The MI250x system consists of four Open Compute Platform Accelerator Modules (OAMs) on each node. Each OAM package consists of two Graphics Compute Dies (GCDs), each of which constitutes one GPU device in the system, resulting in 8 GPUs per node on Tioga [7]. The two GCDs in the package are connected via four AMD Infinity Fabric links running at a theoretical peak rate of 25 GT/sec, giving a bidirectional peak transfer bandwidth of 400 GB/sec for the same.

Power telemetry is supported at the CPU-level and the OAM-level (not per-GPU, but combined across two GPUs). The hardware does not report memory or uncore power. The underlying power management dials include machine-specific registers (MSRs) as opposed to sensors. The E-SMI library is used on the CPU side in conjunction with the Host System Management Protocol (HSMP) and `amd-energy` kernel modules, and ROCm interfaces are used on the GPU side [27]. Power capping, while supported in the actual hardware at the CPU- and the OAM-level, has not been enabled for users on this early access system. Details on maximum or minimum node power limits are unavailable. The maximum power at the OAM-level is 560 W (across 2 GPUs).

The native system resource manager on Tioga is Flux. On Lassen, however, the native resource manager is IBM Spectrum LSF. Enabling Flux on Lassen for our experiments was non-trivial, which we discuss in Section V.

B. Flux

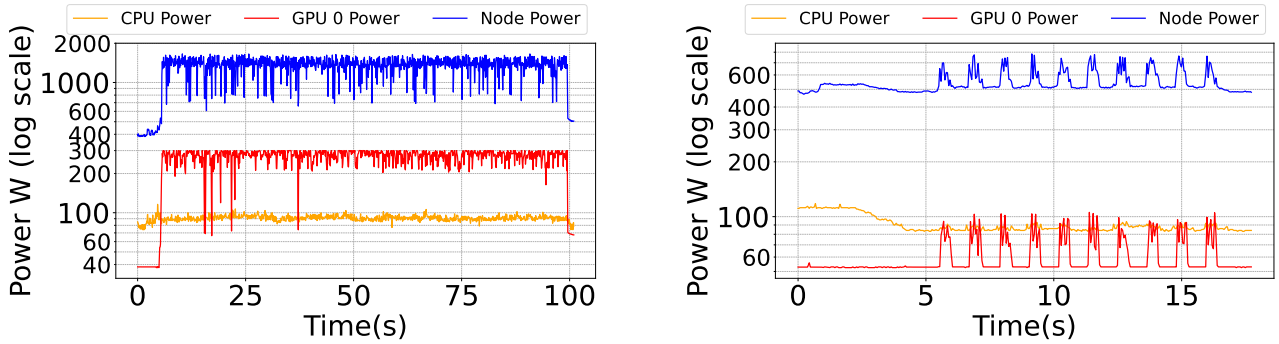
Flux [6] is a flexible resource management framework designed for the next-generation of distributed systems, ranging from HPC systems, to cloud, to converged computing setups. Flux enables hierarchical scheduling to improve throughput and is based on a novel graph-based scheduling approach [25].

A `flux-broker` is a message broker daemon [3] that is launched on each node. A Flux *instance* is similar to an allocation of physical resources, and is a set of `flux-broker` processes that form a Tree-Based Overlay Network (TBON) for distributed communication. Flux components communicates by exchanging messages over the overlay network.

A system-level Flux instance manages all the resources, users, and high-level policies for a distributed system, such as an HPC cluster. When a user requests a job, they are allocated their own user-level Flux instance, allowing them to customize the scheduling policy within their instance. On non-system-level Flux setups, it can be bootstrapped under other resource managers like SLURM or LSF. This means Flux can operate within the constraints of these existing schedulers, enabling a more flexible, hierarchical job scheduling system. In this work, we use flux-core v0.63, released in June 2024.

C. Variorum

Variorum is an open-source, extensible, vendor-neutral library for exposing power and performance capabilities



(a) LAMMPS on Lassen

(b) Quicksilver On Lassen

Fig. 1: Power consumption timeline for LAMMPS and Quicksilver on Lassen, collected on a single node with all four GPUs.

of low-level hardware dials across diverse architectures in a user-friendly manner [27]. Developed at LLNL, it is a 2023 R&D100 winner that enables power telemetry and capping on over twenty different CPU and GPU microarchitectures from different vendors (Intel, AMD, IBM, ARM, and NVIDIA). It can be integrated easily with higher-level system software such as schedulers and runtime systems [10]. Integration with Flux utilizes three Variorum APIs, the `variorum_get_node_power_json` API for vendor-neutral telemetry, the `variorum_cap_best_effort_node_power_limit` API for node-level power capping, and the `variorum_cap_each_gpu_power_limit` for GPU power capping. Node-level power capping varies on each architecture. On Intel and AMD systems, while CPU-level and GPU-level power caps can be set directly, no direct node-level power capping is available in hardware. As a result, *best effort* power capping at the node level distributes power uniformly across available sockets. IBM AC922 allows for direct node-level power capping.

D. Applications

We use five applications in this paper: four of these are GPU-enabled MPI applications, and one is a CPU-only Charm++ application. All applications have been compiled with `gcc-12.1.2` and `CUDA 12.2.0` for Lassen, and with `clang-17.0.1` and with `ROCm 6.0.3` on Tioga. LAMMPS [36] is a classical molecular dynamics simulation code. We use this as a strongly scaled application with `newton=on` (pairwise bonded interactions) and use the `ML-Snap` package to achieve high GPU utilization. GEMM is a generalized matrix multiplication application that is weakly scaled. We use the GEMM kernel from RajaPerf [17]. Quicksilver [29] is a weakly-scaled proxy application for the Monte Carlo transport code based on Mercury. Laghos [14] is a weakly-scaled application that solves time-dependent Euler equations using unstructured high-order finite element spatial discretization. Both Quicksilver and Laghos require task partitioning based on the number of MPI ranks. These are set as `(2,2,1)` for 4 ranks, `(2,2,2)` for 8 ranks, `(2,2,4)` for 16 ranks, `(4,4,2)` for 32 ranks, and `(4,4,4)` for 64 ranks, in the `x,y,z` dimensions. NQueens is a CPU-only Charm++ application based on the popular chessboard puzzle that solves for a situation where no queens attack each other. Table I shows the inputs we used with these applications.

In Figure 1 (log-scale), we show the power usage over time for LAMMPS and Quicksilver on Lassen, from data collected on a single node utilizing all 4 GPUs. Power from only one socket (CPU) and one GPU is shown here for readability, along with the total node power. The other socket, as well has other three GPUs have a similar trend. LAMMPS and GEMM are highly compute-bound and consume more power per node (and per GPU) as a result. Quicksilver and Laghos are not as compute bound. Only Quicksilver depicts periodic phase behavior. Not all applications show this, for example, GEMM, LAMMPS and NQueens have a relatively flat power timeline without any swings. Laghos has some phase behavior, albeit very minor in terms of the magnitude of swings observed in power. It spends most of the time on the CPU and very little on the GPU. We don't show these timelines here due to lack of space, but these are discussed in Section IV.

Application	Scaling	Input
LAMMPS	Strong	`-v nx 64 -v ny 64 -v nz 64`
GEMM	Weak	`--sizefact 700 -repfact 50`
Quicksilver	Weak	Derived based on number of ranks, with a base mesh size of `16` and `300` particles per mesh with `nsteps=40`.
Laghos	Weak	`-pt {task-partition} -m {input-mesh} -rp 2 -tf 0.6 -no-vis -pa -d cuda --max-steps 40`
NQueens	Weak	`+p160`, with 14 queens, `grainsize=1000`

TABLE I: Input parameters for each application.

III. Design and Implementation

`Flux-power-monitor` and `flux-power-manager` are implemented as *modules* within the Flux framework. A module [3] represents a service that is implemented as a dynamically-loaded broker plugin. Modules have their own

thread of control, are event-driven, and interact with Flux exclusively via messages.

A. `Flux-power-monitor`

This module is implemented as a control loop where each node periodically collects power data, but is not aware whether a job is running on it or not, making it *stateless*. An external client requests data for a specific job's power from the module. Based on this request, power data is aggregated and reported back to the client. By keeping the power monitor stateless, the overhead of power telemetry is significantly reduced.

The `flux-power-monitor` has the three components. A `root-agent` that runs in the Flux daemon at the root of the Flux Tree-Based Overlay Network (TBON). This agent is primarily responsible for communicating with the external client. All nodes have a `node-agent`, which collects power data using Variorum. This data includes a timestamp and instantaneous component-level (CPU, GPU, memory) power. This data is stored in the form of a circular buffer of a fixed size. The size of the buffer, as well as the sampling rate, are configurable by the user. By default, power data is collected every 2 seconds, and a buffer size of 43.4MB is used (stores 100,000 instances of the Variorum JSON object).

An external `client` requests job-level telemetry from the `flux-power-monitor`. This is a Python script that takes a job identifier and generates the associated details, such as the nodes allocated to the job, and the start and end time of the job. On receiving the client's request, the `root-agent` requests aggregated power data from the respective `node-agent(s)` on which the job executed, which is then relayed to the client. This data is presented to the user in the form of a CSV file, along with a column specifying whether the module had a complete data set for the job or a partial one, depending on when the data was flushed out.

B. `Flux-power-manager`

The `flux-power-manager` module is designed to be a hierarchical and dynamic power management system. It is *state-aware*, implying that it has complete knowledge of the state of the cluster and the jobs executing on it at any given point in time. Its hierarchical operation allows it to adjust power at various levels.

This module has three main components: the `cluster-level-manager` operates across all nodes and jobs, and is responsible for ensuring that the total power consumption of the cluster does not exceed a specified constraint. It runs on the root node. In the simplest use case, where an HPC system has no power constraint, it allocates the theoretical peak power to each node and performs no power capping. When system power is constrained, each job is allocated power in proportion to its requested node count (see Subsection III-B1). This *job-level power limit* is the maximum power that the entire job can consume at any point in time.

The `job-level-manager`, which also runs on the root node, receives the *job-level power limit* and distributes the power equally to each node associated with the job (*node-level-power-limit*). Additionally, it maintains the complete state of the current job.

The `node-level-manager` is present on each node and is responsible for setting both node and GPU-level power caps. It also tracks the power usage on the node at regular intervals by collecting power data in a separate thread. All three components communicate using RPCs over the Flux TBON.

When using proportional power allocation, the `node-level-manager` receives the *node-level power limit* and enforces it directly. The `node-level-manager` can also utilize dynamic power management policies, such as ones based on past power history, measured performance counters, or other progress metrics. We introduce a Fast Fourier Transform (FFT) based dynamic power management policy (FPP), which aims to identify application phases (*period* of an FFT) and adjusts the power cap on each GPU on the node accordingly.

1) Proportional Sharing Policy: For a given cluster with N nodes and a global power cap of P_G, its current state is defined as the k currently executing jobs with a combined power allocation of P_k ($\sum_{x=1}^{k} P_x$) and currently allocated nodes N_k (resulting in a per-node power allocation of $P_n = P_k/N_k$).

When a new job J_i requests N_i nodes, the `cluster-level-manager` will first try to allocate the maximum possible power to each node by determining if the available power ($P_{avail} = P_G - P_k$) is sufficient for doing so. If the available power is not sufficient, it will proportionally redistribute power to *all* jobs, where the per-node power allocation will be $P_n = P_G/(N_k + N_i)$. The new job will be assigned $P_i = N_i * P_n$ amount of power.

2) Fast Fourier Transform Based Policy: Algorithm 1 shows an overview of FPP. The key advantage of using FFT is its ability to determine the period of a signal. This enables us to identify periodic phase behavior in applications. This policy integrates with the `node-level-manager` to dynamically determine and set per-GPU power caps. The algorithm above sets the values for `P_reduce`, `powercap_levels`, and `Max_GPU_Cap` assuming a GPU similar to the NVIDIA Volta GPU. These values are customizable. The algorithm is executed on a per-GPU basis, allowing for non-uniform power distribution among GPUs on the same node. While we utilize this policy on GPUs, it is device-agnostic from a logistical perspective, and can be easily extended to be utilized for socket-level or memory-level power capping.

The `FFT-GET-PERIOD` procedure continuously monitors power data, updating every thirty seconds to predict the application's period with increasing accuracy. The `GET-GPU-CAP` procedure adjusts the GPU-level power cap based on the difference in the FFT period. If the difference exceeds a specified threshold, the power is increased. Power adjustments cease when the delta falls below the convergence threshold. If the delta lies between the change and convergence thresholds, the power is decreased, indicating that the application is not

Algorithm 1 FFT-Power-Policy (FPP) (Per-GPU)

procedure FFT-GET-PERIOD
  $buf \leftarrow$ empty buffer ▷ Initialize buffer
  while job is running **do**
    STOREPOWERDATA(buf)
    if time elapsed $\geq$ 30 seconds **then**
      $T \leftarrow$ FINDPERIOD(buf)
      Store T
    end if
  end while
end procedure

procedure GET-GPU-CAP($T_{cur}, P_{cap_{prev}}, P_{cap_{cur}}, T_{prev}$)
  $converge_{th} \leftarrow 2$ seconds
  $change_{th} \leftarrow 5$ seconds
  $P_{reduce} \leftarrow 50$ W
  $F_{converge} \leftarrow$ False
  $powercap_{levels} \leftarrow [10, 15, 25]$ W
  $\Delta \leftarrow T_{cur} - T_{prev}$
  $\Delta_{abs} \leftarrow |\Delta|$
  if $P_{cap_{prev}}$ is None **or** $F_{converge}$ is true **then**
    return $P_{cap_{cur}}$
  end if
  if $\Delta_{abs} \leq converge_{th}$ **then**
    $F_{converge} \leftarrow$ true
    return $P_{cap_{cur}}$
  else if $\Delta < 0$ **and** $converge_{th} < \Delta_{abs} < change_{th}$ **then**
    return $P_{cap_{cur}} - P_{reduce}$
  else
    return $P_{cap_{cur}} + powercap_{levels}[\min(\frac{\Delta_{abs}}{5}, 2)]$
  end if
end procedure

procedure MAIN
  $powercap_{time} \leftarrow 90$ sec
  $T_{prev} \leftarrow 0$ ▷ Initialize T_{prev}
  $P_{cap_{prev}} \leftarrow$ None
  $Max\_GPU\_Cap \leftarrow 300$ W ▷ Vendor-specified maximum
  $GPU\_Power\_Lim \leftarrow$ Derived max. cap from node-level limit
  $P_{cap_{cur}} \leftarrow \min(Max\_GPU\_Cap, GPU\_Power\_Lim)$
  while $time \mod powercap_{time} = 0$ **do**
    $T_{cur} \leftarrow$ FFT-GET-PERIOD
    $P_{cap_{next}} \leftarrow$ GET-GPU-CAP($T_{cur}, P_{cap_{prev}}, P_{cap_{cur}}, T_{prev}$)
    Set GPU power cap to $P_{cap_{next}}$
    Reset FFT buffer
  end while
end procedure

significantly affected by the current power cap. The goal of FPP is to improve energy efficiency without negatively affecting application performance.

IV. RESULTS

In this section, we first discuss vendor-neutral telemetry followed by an overhead analysis on different systems. Then, we present a detailed set of scenarios explaining static and dynamic power capping, comparing the FPP algorithm with the proportional power sharing algorithm. We also discuss how `flux-power-manager` can be utilized for non-MPI applications and show the impact on a real job queue.

A. Vendor-neutral telemetry with `flux-power-monitor`

Figure 2 shows the aggregated power information for two applications from the Lassen and Tioga clusters. Applications on Lassen are scaled from 1–32 nodes. On Tioga, they are scaled from 1–8 nodes. Lassen supports direct power measurements at the node, memory, CPU, and GPU levels. Tioga supports power measurements only at the CPU- and OAM-level (2 GPUs per OAM). Memory and total node power cannot be directly measured. The node power reported is thus a conservative estimate, and is the sum of the measured power on the single socket (CPU) and the four OAMs.

The `flux-power-monitor` module aggregates values (sampled every 2 seconds) for each node and provides per-node data for the job. In the graphs, we also average across the nodes for better readability. For weakly scaled applications (Quicksilver and Laghos), the average per-component power consumption is fairly similar when scaling from 1–32 nodes. In contrast, for LAMMPS, which is a strongly scaled application, the power consumption decreases as the problem size decreases when going from 1–32 nodes. Most of this reduction in power consumption comes from the GPU-level.

We also note that Tioga consumes more absolute power than Lassen on the same application with the same inputs at the same node count (for the four MPI applications in our experiments). Recall that the node power reported here on Tioga doesn't include memory and uncore power, so the actual node power consumption (which we do not have a way of measuring) is expected to be even higher. This higher power consumption is because we are utilizing a total of 8 GPUs per node as opposed to 4 GPUs per node on Lassen.

Tioga is expected to be more energy efficient than Lassen as it provides higher floating-point-operations-per-Watt (FLOPS per Watt). In Table II, we show the absolute runtime for three applications at 4 and 8 nodes on Lassen and Tioga, along with average per-node energy. We observe that for LAMMPS, overall energy reduced by 21.54% on Tioga compared to Lassen. For Laghos, the energy per-node increased by 139% on Tioga, but this is an expected result as the task count on Tioga was twice that of Lassen (8 GPUs instead of 4 GPUs) and the problem size was scaled based on the number of tasks.

Quicksilver demonstrated anomalous behavior in terms of execution time. On Lassen, the runtime was of the order of 12–14 seconds. On Tioga, we would have expected an execution time of about 24–28 seconds due to doubling of tasks and weak scaling. However, we observed the range to be 102–106 seconds, which was unexpected. This is being investigated by the development team and is attributed to its HIP variant. We do not report energy data for comparison as a result.

B. Overhead of `flux-power-monitor`

The `flux-power-monitor` module executes as a separate thread within the Flux framework. As a result, its influence and impact on application performance is expected

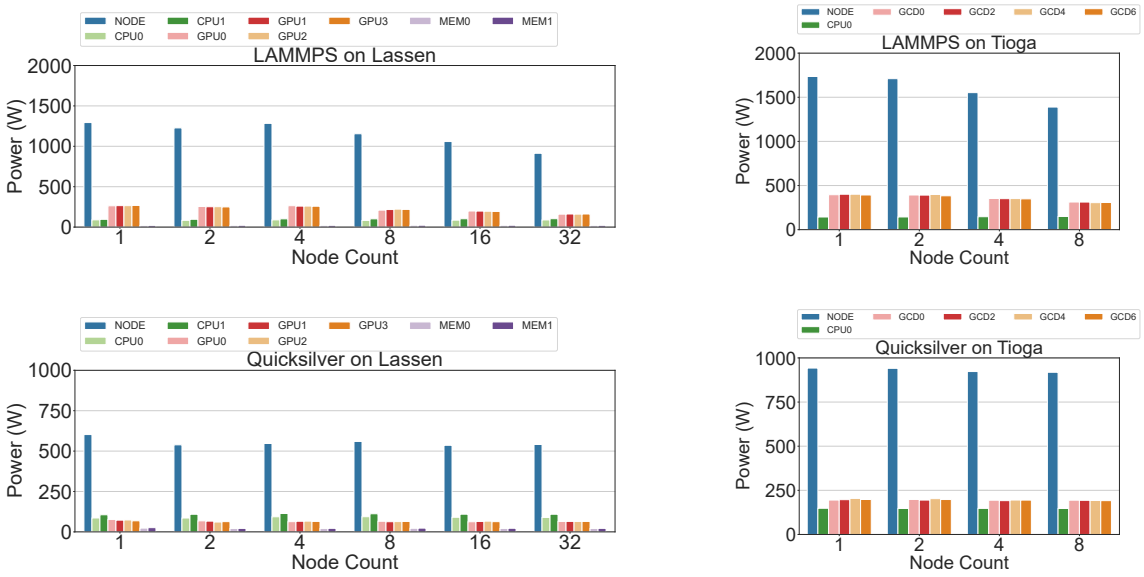


Fig. 2: Power data for various applications on Lassen and Tioga

Application	Node Count	Lassen Task Count	Tioga Task Count	Lassen Runtime (s)	Tioga Runtime (s)	Lassen Avg. Power (W) (per-node)	Tioga Avg. Power (W) (per-node)	Lassen Avg. Energy (kJ) (per-node)	Tioga Avg. Energy (kJ) (per-node)
LAMMPS	4	16	32	77.17	51.00	1283.74	1552.40	99.07	79.17
LAMMPS	8	32	64	46.33	29.67	1155.08	1388.99	53.51	41.21
Laghos	4	16	32	12.55	26.71	472.91	530.87	5.94	14.18
Laghos	8	32	64	12.62	26.81	469.59	532.28	5.93	14.27
Quicksilver	4	16	32	12.78	102.03*	546.99	915.82	-	-
Quicksilver	8	32	64	13.63	106.15*	559.64	924.85	-	-

TABLE II: Performance reported by each application at 4 and 8 nodes on Lassen and Tioga systems.
*We do not compare the energy of Quicksilver across Lassen and Tioga due to anomalous execution time of the HIP variant.

to be minimal. In order to better understand its overhead, we measured the execution time of three applications with and without the `flux-power-monitor` loaded. Applications were scaled from 1–32 nodes on Lassen, and 1–8 nodes on Tioga. Each measurement was repeated six times.

In Figure 3, we show the percentage slowdown in the execution time for each application at different node counts, averaged over the six repetitions. We observe that the average overhead is 1.2% on Lassen and 0.04% Tioga.

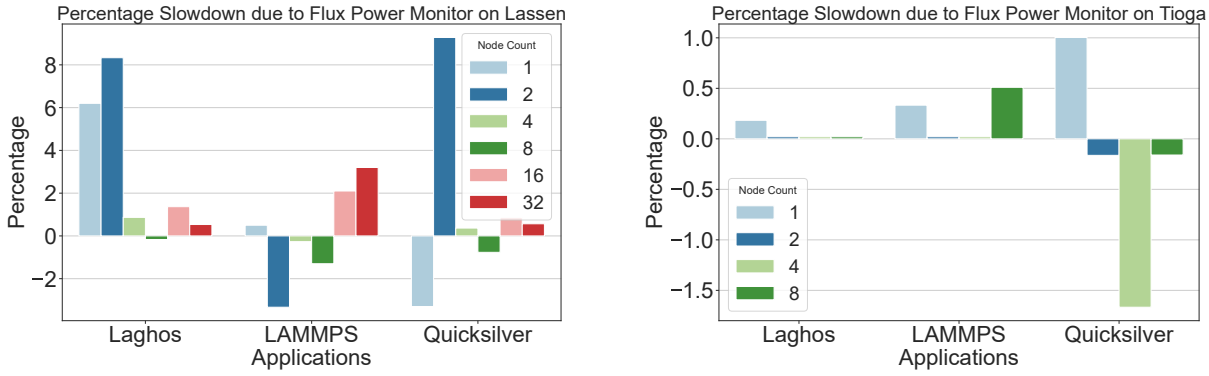
On Lassen, we observed that lower node counts ranging from 1–2 nodes (blue bars in Figure 3) had a higher overhead for some applications. With Laghos, we saw an average overhead of 6.2% on one node and 8.2% on two nodes, and with Quicksilver, we saw an average overhead of 9.3% on two nodes. In order to investigate this, we looked at each repeated run, which we show in Figure 4. The Y-axis in Figure 4 shows a box-plot of the raw execution time in seconds for each application, over the six repeated runs. The X-axis indicates whether or not `flux-power-monitor` was loaded.

Here, we observed significant run-to-run variability in Laghos and Quicksilver at these lower node counts (over 20%), even when the `flux-power-monitor` module was not loaded. This run-to-run variation persisted for Laghos when the `flux-power-monitor` module was loaded. This variation explains the slowdowns we see on Lassen at low node counts on these two applications. We attribute these exceptional observations to run-to-run variability due to issues such as jitter from operating system daemons [22] or the influence of other user's jobs due to congestion [8].

There were also some situations where we observed a minor speedup, such as with LAMMPS and Quicksilver on Lassen, and Quicksilver on Tioga. We don't believe that using `flux-power-monitor` can result in applications speeding up. We did not analyze these further, and we attribute these to run-to-run variation as well.

C. Static Job-Level Power Allocation

We establish a baseline for dynamic power management (Section IV-D) by first discussing static power capping. We leverage the Lassen system for these experiments as node-level (as well as GPU-level) power capping capability is available for users. Furthermore, Lassen reports node power directly, including memory and uncore component power. We assume



(a) Lassen

(b) Tioga

Fig. 3: Measured across six repeated runs, the average overhead of `flux-power-monitor` is 1.2% on Lassen and 0.04% Tioga. Lower node counts (blue bars) on Lassen demonstrated run-to-run variability.

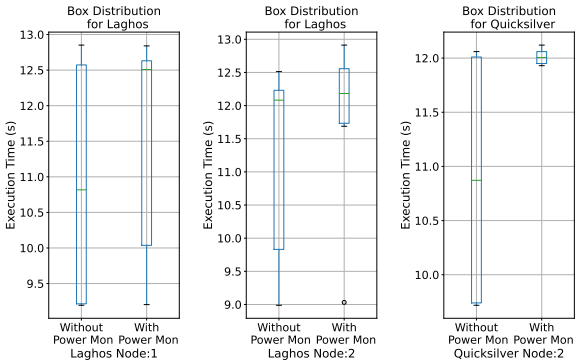


Fig. 4: Run-to-run variation is observed in Laghos and Quicksilver at low node counts.

an idle node power consumption of 400 W based on our measurements, and power capping is enforced with Variorum.

We do not account for network power in this analysis. Network power is typically in a tightly bounded range [26], and we assume that it is a constant that can be deducted from both the cluster-level and the node-level proportionally.

For this subsection and the next we design our experiments to reflect the availability of total system power in practice. Systems could be *unconstrained*, where *all* their components can operate at full power simultaneously. Systems can also be *power-constrained* (hardware-overprovisioned [28]), where all components cannot operate at full power simultaneously.

We consider an 8-node cluster for our evaluation. We study the performance and power usage of two HPC applications, GEMM and Quicksilver, when executing as a 6-node and 2-node job respectively, with a 10x problem size for Quicksilver and double the iteration count for GEMM (with respect to Table I). We focus on two applications to provide detailed analysis of performance and power management.

Use Case	Node-Level Cap (W)	Derived Max. GPU Cap (W)	Maximum Power Usage (kW)	Average Power Usage (kW)
Unconstrained	3050	300	10.66	8.9
Power-constr.	1200	100	6.05	5.1
Power-constr.	1800	216	8.68	7.2
Power-constr.	1950	253	9.5	7.9

TABLE III: Static power allocation on an 8-node Lassen allocation using IBM-provided node-level power capping.

We consider two use cases. The first is an *unconstrained* system, where the cluster-level power bound is 24,400 W (eight nodes at 3050 W each). The second is a *power-constrained* system, where the cluster-level power bound is set to 9600 W. With uniform and static allocation, this comes to 1200 W per node. At any point of time during cluster operation, a power management policy under effect (static or dynamic) should not violate (that is, exceed) these cluster-level power bounds. The overall objective is to compare different power management policies in terms of their overall energy use as well as individual application performance slowdowns, especially in a power-constrained system.

Table III shows the measurements from an unconstrained system (without any power capping) as well as from a power-constrained system when executing DGEMM and Quicksilver on 6 and 8 nodes, respectively. We show the impact of three different IBM node-level power cap values: 1200, 1800 W and 1950 W per node. We report the maximum cluster-level power usage (maximum power measured, summed across all nodes at all points in time when sampled every 2 seconds) and the average cluster-level power usage. We also show the maximum derived power cap per GPU, which is determined by IBM when a node-level power cap is set.

Our first observation is that in an unconstrained scenario the maximum power usage was only 10,657 W (10.66 kW) out of an allowed 24.4 kW despite using all four GPUs across the two applications, and with a highly compute-bound application such as GEMM executing on 6 of the 8 nodes. This is a

common trend in HPC systems, as most systems are worst-case power provisioned [28]. We observe on Lassen that most applications used less than 60% of provisioned power over multiple years of operation [26].

Next, we observe that IBM's default node-level power capping algorithm is extremely conservative. With a cluster-level power bound of 9600 W, we would have expected the measured maximum power usage (column 4 in Table III) to be close to 9.6 kW when each node had a power cap of 1200 W. Despite setting the PSR to 100%, indicating maximum possible power to GPUs, IBM's algorithm capped each GPU to 100 W. The maximum power usage was 6048 W (6.05 kW), well below the cluster-level bound of 9.6 kW.

As a result, we tested several other IBM node-level power caps to identify when the measured maximum power usage is close to the cluster-level bound of 9.6 kW. We found that at 1950 W per node, the maximum power usage is 9.5 kW, bringing it close to our desired constraint. We thus use the 1200 W and 1950 W node-level power caps as our baseline for the `flux-power-manager` experiments in Section IV-D.

We also show usage data at a node power cap of 1800 W. Here, we observe that the average per-node energy when executing GEMM and Quicksilver was 784 kJ, as opposed to 827 kJ when the power cap is set at 1950 W. We note that this node-level power cap was optimal for the combination of applications we were executing, but we do not have a clear explanation for why this was the case. It is also important to note that we found this optimal value through a manual sweep, which is not ideal in terms of determining the appropriate static cap for a certain set of jobs on a specific hardware. Our choice of baselines (1200 W and 1950 W node power cap) do not impact the comparative results for the dynamic power policies.

D. Dynamic Job-Level Power Management

We build on the experiments from Section IV-C to evaluate the effectiveness of the proportional sharing and FPP algorithms. Table IV compares the performance of GEMM and Quicksilver for unconstrained, static (at 1950 W), proportional share, and FPP. Along with the application performance, maximum node power usage (across all nodes) and the average energy per node are also shown. Quicksilver experiences minimal changes in its performance under a power constraint. GEMM, being highly compute bound, experiences a slowdown under the power-constrained use case. The advantage of a proportional sharing approach is its ability to allocate power based on the number of nodes, sharing power when needed and reclaiming it when a job finishes. Compared to IBM's default approach (1200 W node power cap, static capping), overall energy improved by 19% with an almost 1.59x performance gain. Compared to the other static policy (1950 W node power cap), energy improved by about 5.4%, due to reduced power draw. With proportional share, GEMM receives additional power when Quicksilver is not running, as shown in Figure 5. We depict only one node each for each application, as other nodes behave similarly. FPP reduces energy even further, improving energy by 1.2% when compared to the proportional sharing (performance degradation of 0.8%), by 6.6% when compared to static policy at 1950 W node power cap (performance degradation of 4.5%), and by 20% when compared to IBM's default approach (performance gain of about 1.58x). The timeline for FPP is shown in Figure 6.

The less than expected improvements of the two dynamic policies when compared to the static policy with node power cap of 1950 W can be attributed to GEMM being highly compute intensive. Any reduction in power degrades performance significantly, so there is less opportunity to save power while preserving performance. For applications that are less compute bound, a greater improvement in energy efficiency is expected.

We also want to acknowledge the fact that an FFT-based approach does not work with applications that do not exhibit periodic phase behavior. With GEMM, FPP first tries to reduce power but sees that the period doubles and instantly gives back the power to the application. In the case of Quicksilver, the impact of power capping itself is quite small (compared to the unconstrained scenario), so FPP converges early. We also did not explore FPP parameters, such as the power capping interval (which was set to 90 seconds) or the ranges for power caps (we picked 50 W for power reduction, and 10–25 W for our steps as shown in Section III-B2) in this paper. Exploring this research space for phase-based applications and workflows is part of our future work.

E. Impact on a Job Queue

We finally demonstrate the impact of the proportional sharing and FPP policies on a real job queue with 10 jobs on a 16-node allocation on Lassen. Our job queue uses a random mix of the four applications, with each application requesting between 1–8 nodes. The job queue had 3 jobs with Laghos, 2 with Quicksilver, 3 with LAMMPS and 2 with GEMM, making it a mostly compute-intensive queue. We opt for a small cluster and job queue as these are actual runs and not simulations. Using a job queue with hundreds of jobs could take multiple hours or days of experiments across different use cases. Flux schedules these jobs as any regular resource manager would. Our results show that both the proportional sharing policy and FPP, the overall `makespan` of the queue (difference between when the last job ends and when the first job was submitted) remains same with a value of 1539 seconds. The average per-job energy-per-node improved by 1.26% with FPP.

F. Demonstration on Non-MPI Applications

Figure 7 shows how proportional power capping can be applied to any Flux job, whether or not it utilizes MPI. The figure shows a Charm++ NQueens application running on 2 nodes, alongside GEMM which executes on 6 nodes. As expected, GEMM power consumption drops when the NQueens application enters the system.

V. Discussion

We encountered significant challenges during this study. First, setting up Flux on Lassen was non-trivial. Lassen uses

Use Case and Policy	Node Power Cap (W)	Max. Node Power Usage (W)		Exec. Time (s)		Avg. Node Energy (kJ)	
		GEMM	QS	GEMM	QS	GEMM	QS
Unconstr.	3050	1523	952	548	348	726	177
Constr. IBM default	1200	841	820	1145	359	805	160
Constr. Static	1950	1330	975	564	347	652	175
Constr. Prop. Shar.	1950	1343	939	597	347	612	170
Constr. FPP	1950	1325	951	602	350	598	174

TABLE IV: Comparison of static and dynamic power capping.

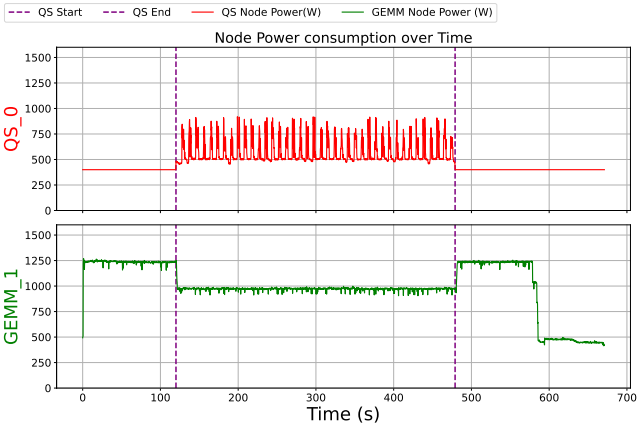


Fig. 5: **Proportional Power Sharing Policy** GEMM receives additional power when Quicksilver is not executing.

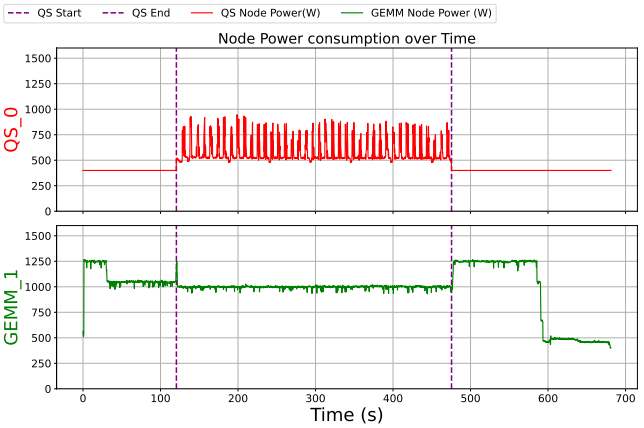


Fig. 6: **FFT Based Power Policy (FPP):** FPP algorithm converges quickly for both application, as there is not a lot of opportunity to save power while preserving performance.

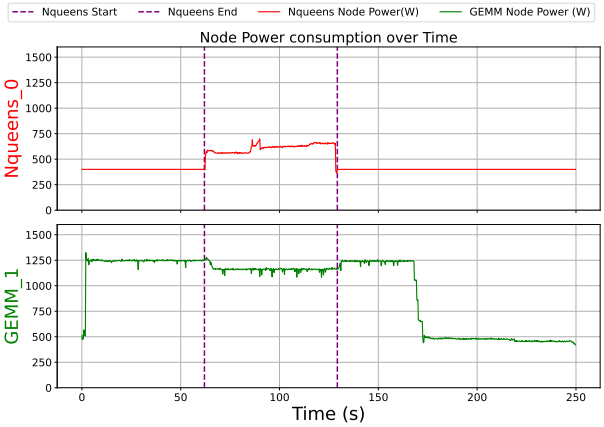


Fig. 7: **Proportional Power Capping**: GEMM and Charm++ NQueens application.

IBM's Spectrum MPI, which relies on PMIx. Further, there is limited support for updated packages and modules. Installing Flux involved installing a custom version of Open-PMIx and bootstrapping Flux accordingly. As we were evaluating on an early access system with Tioga, obtaining high-performance HIP variants for the applications was another challenge. For example, we could not obtain a HIP variant for SW4lite. Quicksilver results were unexpected, as we discussed in Section IV, and Kripke execution failed on the Tioga system. Application developers and domain scientists are still addressing these issues at the time of writing this paper. We discussed the anomalies with IBM's node-level power capping in our results. In addition to these, we observed that on some nodes at a low node-level power cap (1200 W), NVIDIA GPU power capping failed intermittently, either picking up the last set power cap or defaulting to the maximum power cap. Supercomputing sites often rely on vendors to provide reliable power capping for power-constrained scenarios. We observed in our experiments that this is often not the case. Adopting dynamic power capping techniques in production requires these algorithms to be effective as per stated guidelines. Documentation on granularities of power capping, error bounds, and steady state convergence is sparse in the public domain, delaying adoption of such techniques at scale.

VI. Summary and Future Work

We presented a vendor-neutral, production quality, and low overhead job-level power management framework based on Flux. We discussed `flux-power-monitor` for telemetry and demonstrated its effectiveness on the Lassen and Tioga systems with a low overhead of 0.4%. With `flux-power-manager` we presented FPP, a proportional share and a novel, hierarchical FFT-based dynamic power management policy called. Our results show that FPP reduces job-level energy consumption by 1% compared to proportional share policy, and by 20% compared to the default IBM static power capping policy. Our future work involves exploring

various parameters for FPP, studying diverse job queues in converged computing setups, and power-performance optimizations for complex scientific workflows.

ACKNOWLEDGMENTS

We thank Dr. Barry Rountree for his contributions to an early version of `flux-power-monitor`, and Dr. Thomas Scogland for his insightful review and suggestions. We are also grateful to the Flux team for their debugging help. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-868257), and partially supported by U.S. National Science Foundation under Grants CCF-1942182.

REFERENCES

[1] IBM OCC Firmware Interface Specification for Power9, 2019. URL: https://github.com/open-power/docs/blob/P9/occ/OCC_P9_FW_Interfaces.pdf.

[2] Charm++: Parallel Programming with Migratable Objects, 2024. URL: https://charm.cs.illinois.edu/research/charm.

[3] Flux RFC Index: RFC 3/Flux Message Protocol and RFC 5/Flux Broker Modules, 2024. URL: https://flux-framework.readthedocs.io/projects/flux-rfc/en/latest/spec_5.html.

[4] PowerAPI Reference Implementation and Plugins, 2024. URL: https://github.com/pwrapi/pwrapi-ref.

[5] REGALE: Open Architecture for Exascale Supercomputers, 2024. URL: https://regale-project.eu/.

[6] Dong H. Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. Flux: A next-generation resource management framework for large hpc centers. In *2014 43rd International Conference on Parallel Processing Workshops*, pages 9–17, 2014.

[7] AMD INSTINCT MI250 Microarchitecture. AMD Instinct MI250 Microarchitecture, 2024. ROCm Documentation. URL: https://rocmdocs.amd.com/en/latest/conceptual/gpu-arch/mi250.html.

[8] Abhinav Bhatele, Kathryn Mohror, Steven H. Langer, and Katherine E. Isaacs. There goes the Neighborhood: Performance Degradation due to Nearby Jobs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, New York, NY, USA, 2013. Association for Computing Machinery.

[9] Harsh Bhatia, Francesco Di Natale, Joseph Y. Moon, Xiaohua Zhang, Joseph R. Chavez, Fikret Aydin, Chris Stanley, Tomas Oppelstrup, Chris Neale, Sara Kokkila Schumacher, Dong H. Ahn, Stephen Herbein, Timothy S. Carpenter, Sandrasegaram Gnanakaran, Peer-Timo Bremer, James N. Glosli, Felice C. Lightstone, and Helgi I. Ingolfsson. Generalizable coordination of large multiscale workflows: Challenges and learnings at scale. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.

[10] Christopher Cantalupo, Jonathan Eastep, Siddhartha Jana, Masaaki Kondo, Matthias Maiterth, Aniruddha Marathe, Tapasya Patki, Barry Rountree, Ryuichi Sakamoto, Martin Schulz, et al. A Strawman for an HPC powerstack. Technical report, 2018. URL: https://www.osti.gov/biblio/1466153.

[11] Julita Corbalan and Luigi Brochard. EAR: Energy management framework for supercomputers. Technical report, 2019. URL: https://www.bsc.es/sites/default/files/public/bscw2/content/software-app/technical-documentation/ear.pdf.

[12] Alex de Vries. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194, 2023. URL: https://www.sciencedirect.com/science/article/pii/S2542435123003653.

[13] Jianru Ding and Henry Hoffmann. DPS: Adaptive Power Management for Overprovisioned Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3581784.3607091.

[14] Veselin A. Dobrev, Tzanio V. Kolev, and Robert N. Rieben. High-order curvilinear finite element methods for lagrangian hydrodynamics. *SIAM Journal on Scientific Computing*, 34(5):B606–B641, 2012.

[15] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. Global extensible open power manager: A vehicle for HPC community collaboration on co-designed energy management solutions. In Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes, editors, *High Performance Computing*, pages 394–412, Cham, 2017. Springer International Publishing.

[16] Neha Gholkar, Frank Mueller, Barry Rountree, and Aniruddha Marathe. Pshifter: Feedback-based dynamic power shifting within HPC jobs for performance. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 106–117, 2018.

[17] Richard D. Hornung, Holger E. Hones, and USDOE National Nuclear Security Administration. RAJA Performance Suite, 9 2017. URL: https://www.osti.gov//servlets/purl/1394927.

[18] Baolin Li, Rohan Basu Roy, Daniel Wang, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. Toward Sustainable HPC: Carbon Footprint Estimation and Environmental Implications of HPC Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, 2023.

[19] Aniruddha Marathe, Peter E Bailey, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R de Supinski. A run-time system for power-constrained HPC applications. In *High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings 30*, pages 394–408. Springer, 2015.

[20] Marcel Marquardt, Jan Mader, Tobias Schiffmann, Christian Simmendinger, and Torsten Wilde. Powersched: A HPC System Power and Energy Management Framework. Technical report, 2018. URL: https://cug.org/proceedings/cug2023_proceedings/includes/files/pap113s2-file1.pdf.

[21] Daniel J. Milroy, Claudia Misale, Giorgis Georgakoudis, Tonia Elengikal, Abhik Sarkar, Maurizio Drocco, Tapasya Patki, Jae-Seung Yeom, Carlos Eduardo Arango Gutierrez, Dong H. Ahn, and Yoonho Park. One step closer to converged computing: Achieving scalability with cloud-native hpc. In *2022 IEEE/ACM 4th International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, pages 57–70, 2022.

[22] Oscar H. Mondragon, Patrick G. Bridges, Scott Levy, Kurt B. Ferreira, and Patrick Widener. Understanding Performance Interference in Next-generation HPC Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16. IEEE Press, 2016.

[23] Pratyush Patel, Esha Choukse, Chaojie Zhang Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, and Ricardo Bianchini. POLCA: Power oversubscription in LLM cloud providers. arXiv:2308.12908 [cs.DC], Aug 2023.

[24] Tirthak Patel and Devesh Tiwari. Perq: Fair and efficient power management of power-constrained large-scale computing systems. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, page 171–182, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Tapasya Patki, Dong Ahn, Daniel Milroy, Jae-Seung Yeom, Jim Garlick, Mark Grondona, Stephen Herbein, and Thomas Scogland. Fluxion: A scalable graph-based resource model for hpc scheduling challenges. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W '23, New York, NY, USA, 2023. Association for Computing Machinery.

[26] Tapasya Patki, Adam Bertsch, Ian Karlin, Dong H. Ahn, Brian Van Essen, Barry Rountree, Bronis R. de Supinski, and Nathan Besaw. Monitoring large scale supercomputers: A case study with the lassen supercomputer. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 468–480, 2021.

[27] Tapasya Patki, Stephanie Brink, Aniruddha Marathe, Barry Rountree, Kathleen Shoga, and Eric Green. Variorum: Vendor-Agnostic Computing Power Management. Technical report, LLNL, 2023. URL: https://ipo.llnl.gov/sites/default/files/2023-08/Final_variorum-rnd-100-award.pdf.

[28] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th international ACM conference on International conference on supercomputing (ICS13)*, pages 173–182, Jun 2013.

[29] David Richards, Patrick Brantley, Shawn Dawson, Scott Mckenley, and Matthew O'Brien. Quicksilver, version 00, 3 2016. URL: https://www.osti.gov/biblio/1313660.

[30] Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: making DVS practical for complex HPC applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, page 460–469, New York, NY, USA, 2009. Association for Computing Machinery.

[31] Ryuichi Sakamoto, Thang Cao, Masaaki Kondo, Koji Inoue, Masatsugu Ueda, Tapasya Patki, Daniel Ellsworth, Barry Rountree, and Martin Schulz. Production hardware overprovisioning: Real-world performance optimization using an extensible power-aware resource management framework. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 957–966, 2017.

[32] Alejandro Sanchez and Morris Jette. SLURM Workload Scheduling and Power Management. Technical report, 2018. URL: https://slurm.schedmd.com/SLUG18/power_management.pdf.

[33] Parul Singh and Krishnasuri Narayanam. Empowering Efficiency: PEAKS - Orchestrating Power-Aware Kubernetes Scheduling. Technical report, 2024. URL: https://www.youtube.com/watch?v=WPrSnZ4lyjw.

[34] V Sochat, A Culquicondor, A Ojea, and D Milroy. The flux operator [version 1; peer review: 2 approved]. *F1000Research*, 13(203), 2024. doi:10.12688/f1000research.147989.1.

[35] Tapan Srivastava, Huazhe Zhang, and Henry Hoffmann. Penelope: Peer-to-peer Power Management. In *Proceedings of the 51st International Conference on Parallel Processing*, ICPP '22, New York, NY, USA, 2023. Association for Computing Machinery.

[36] 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.*, 271:108171, 2022. doi:10.1016/j.cpc.2021.108171.

[37] Brian Van Essen, Hyojin Kim, Roger Pearce, Kofi Boakye, and Barry Chen. LBANN: Livermore big artificial neural network HPC toolkit. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, MLHPC '15, New York, NY, USA, 2015. Association for Computing Machinery.

[38] Sean Wallace, Xu Yang, Venkatram Vishwanath, William E. Allcock, Susan Coghlan, Michael E. Papka, and Zhiling Lan. A data driven scheduling approach for power management on hpc systems. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 656–666, 2016.

[39] Pengfei Zou, Tyler Allen, Claude H. Davis, Xizhou Feng, and Rong Ge. Clip: Cluster-level intelligent power coordination for power-bounded systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 541–551, 2017. doi:10.1109/CLUSTER.2017.98.

[40] Pengfei Zou, Ang Li, Kevin Barker, and Rong Ge. Indicator-directed dynamic power management for iterative workloads on gpu-accelerated systems. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 559–568, 2020. doi:10.1109/CCGrid49817.2020.00-37.