Lawrence Livermore National Laboratory

7000 East Avenue
Livermore CA 94550

Contact
Tapasya Patki

R&D100 Award Nominee Variorum

Prepared for:

2023 R&D 100 Award Entry

LLNL-MI-849523

Prepared by LLNL under Contract DE-AC52-07NA27344.

Variorum: Vendor-Agnostic Computing Power Management

1. PRODUCT/SERVICES CATEGORIES

A. Title

Variorum: Vendor-Agnostic Computing Power Management

B. Product Category

Software/Services | Special Recognition: Green Tech

2. R&D 100 PRODUCT/SERVICE DETAILS

A. Primary submitting organization

Lawrence Livermore National Laboratory

B. Co-developing organizations

n/a

C. Product brand name

Variorum

D. Product Introduction

This product was introduced to the market between January 1, 2022, and March 31, 2023. This product is not subject to regulatory approval.

E. Price in U.S. Dollars

Free. Variorum is open source and free to users.

F. Short description

Pushing supercomputers to their limits requires a deeper understanding of power and energy than standard software and operating systems allow. Variorum provides robust interfaces that measure and optimize computation at the physical level: temperature, cycles, energy, and power. With Variorum, administrators and users can efficiently and effectively use computing resources.

G. Type of institution represented

Government or independent lab/institute

H. Submitter's relationship to product

Product developer

I. Photos

- Variorum logo
- Vendor-neutral diagram
- Comparison matrix – software
- Comparison matrix – hardware

J. Video

Available on YouTube: https://youtu.be/rgJGgPERBao

3. PRODUCT/SERVICE DESCRIPTION

A. What does the product or technology do?

Supercomputers can enable amazing research and scientific discoveries, but they require a sophisticated coordination of application codes, diverse hardware components, and system software. Optimizing and managing power and energy on such large-scale supercomputers is critical for many reasons. Efficient systems can process more user requests and improve the pace of scientific discovery. Additionally, understanding the power and energy costs allows for better utilization and environmentally friendly supercomputing practices.

An HPC system's power and performance are affected by configurations and requirements at different levels of the system, and each level presents challenges—and opportunities—for optimization. The most granular level is on the individual nodes, where the most impactful monitoring, managing, and optimizing of power and performance can (and should) occur via precision tuning of a multitude of low-level options, or "dials" as on a radio or sound engineer's board. This lowest level is the key to the hierarchy above it. These low-level dials, however, are complex and vary significantly across vendors and are often poorly documented. While some dials are obscured by specific hardware and software configurations, others have nonstandard interfaces that lack portability to other systems and are challenging to integrate into user applications.

Variorum is an extensible, vendor-neutral software library for exposing the power, energy, and performance capabilities of low-level hardware dials across diverse architectures in a user-friendly manner. It is part of the U.S Department of Energy's (DOE) Exascale Computing Project (ECP)[21]—specifically, the Argo Project[22] in which Variorum is a key component for node-level power management in the high-performance computing (HPC) PowerStack Initiative[24]. Variorum provides a rich set of vendor-neutral application programming interfaces, or APIs, such that the user can query or control hardware dials without needing to know the underlying vendor's implementation (e.g., machine-specific registers [MSRs] or sensor interfaces). These APIs enable supercomputing users to gain a better understanding of power, energy, and performance through various metrics. Additionally, the APIs enable system software to control hardware dials to optimize for a particular goal. As an open-source tool, Variorum is widely accessible to diverse users of supercomputers, bare-metal cloud infrastructure, as well as researchers who study power and energy management on their personal laptops. Variorum focuses on ease of use and reduced integration burden in scientific applications and workflows, and has enabled support for all three upcoming U.S. exascale supercomputers—El Capitan at Lawrence Livermore National Laboratory (LLNL), Aurora at Argonne National Laboratory, and Frontier at Oak Ridge National Laboratory—and many other HPC systems.

Variorum and the PowerStack

Opportunities for power and performance optimization exist at all levels in a supercomputer's software stack: Power-aware job schedulers[20], runtime system[15,25] as well as large-scale monitoring frameworks[26] allow for balancing power allocations between multiple users and among different tasks of a parallel application. We refer to such interoperable power-aware software as the PowerStack. Figure 1 depicts an interoperable PowerStack and its various components.

Each level in the PowerStack provides options for adaptive and dynamic power management and may operate as an independent entity if needed depending on the requirements of the supercomputing site under consideration. Site-specific requirements such as cluster-level power bounds, user fairness, or job priorities are translated into inputs to the job scheduler. The job scheduler (or resource manager) chooses power-aware scheduling plugins to ensure compliance, with the primary responsibility being management of allocations across multiple users and diverse workloads. Such allocations (physical nodes and job-level power bounds) serve as inputs to a fine-grained, job-level runtime system to manage the application, in turn relying on vendor-agnostic node-level measurement and control mechanisms.

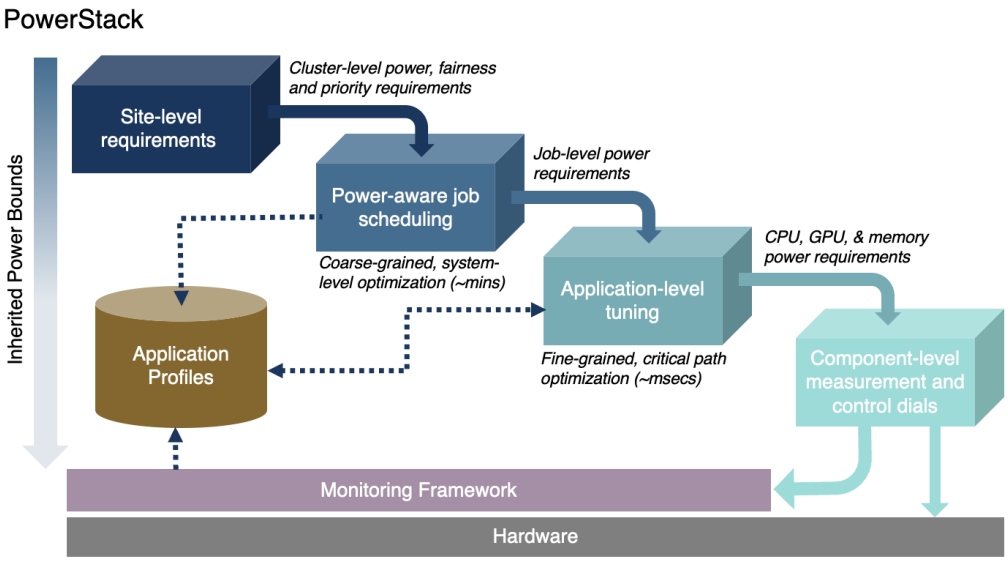

Figure 1: Overview of an HPC PowerStack.

Optimizations at multiple levels in a PowerStack can only be made possible with the help of dials that exist at the lower, compute-node level, such as those that allow for measurement of energy and temperature; tuning of central processing unit (CPU) or graphics processing unit (GPU) frequency; or setting of CPU, GPU, and memory power caps. However, these low-level dials vary significantly across vendors and can be incredibly complex to understand and tune. For example, Intel processors use MSRs to track and manage power and energy; AMD processors use a mailbox-based design; and IBM processors utilize sensors combined with file system firmware interfaces. Similarly, NVIDIA GPUs and AMD GPUs expose these dials through vendor-specific interfaces and associated libraries, such as the `nvidia-smi` or `rocm-smi`.

Even within the same vendor, dials can vary significantly with each new generation of processors. From a user's perspective, understanding these differences is extremely challenging; even if they painstakingly do understand them, incorporating these dials into their source code results in a lack of portability across supercomputing platforms. Several other low-level dials for power capping, energy and thermal management, clock frequency management, and other features present similar issues. Vendor architectures also vary in terms of the components such as differing numbers of CPUs and GPUs, and memory units, which further adds to the complexity of fine-tuning the associated dials. Figure 2 shows depicts four vendors with varying node-level components as well as dials.

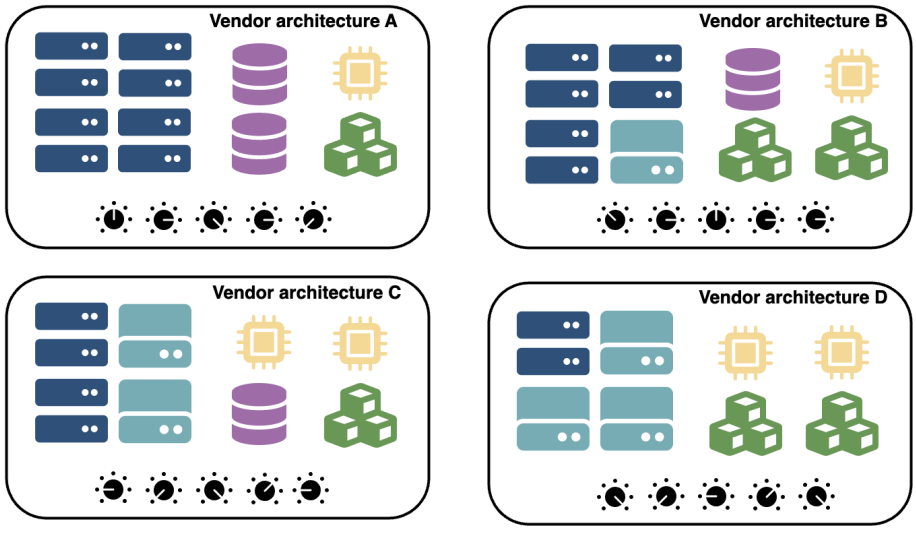

Figure 2: Each vendor architecture has different components (e.g., cores, GPUs, memory, artificial intelligence accelerators) as well as different dials and underlying mechanisms for tuning.

Supercomputer users have diverse backgrounds and include application and domain scientists, research software engineers, and system administrators. Often, users are unfamiliar with low-level architecture details across different vendors, and some power and performance dials require the user to have elevated access privileges. As a result, accessing power and performance optimization dials that are complex and vendor-specific can be chaotic, unwieldy, and error prone from the users' perspective. A representation of these dials and their access challenges is shown in Figure 3, where some users do not have access to certain dials at all, while others have to adapt their code in a vendor-specific manner each time they try to build on a new system, drastically reducing their productivity.

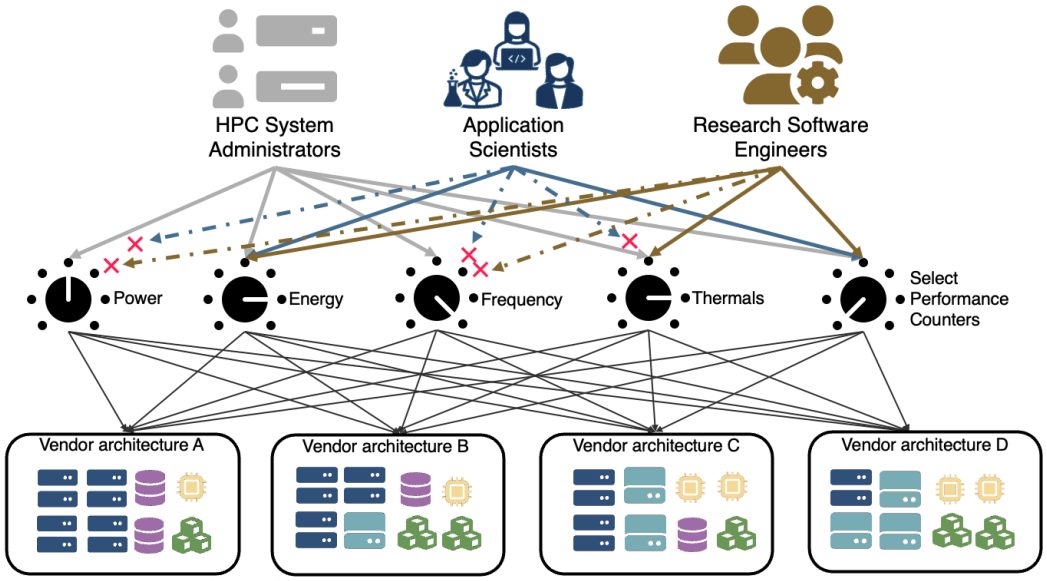

Figure 3: Optimization dials are unwieldy and complex for users to access.

Variorum solves this problem and provides users with easy access to these dials. It has been integrated successfully into all levels of an interoperable PowerStack. As shown in Figure 4, Variorum's API abstracts out the details of the vendor-specific implementations and makes dials available to both general and advanced users in a portable manner. These dials allow for measurement and control of various physical features on processors and accelerators, such as power, energy, frequency, temperature, and performance counters. With Variorum, supercomputing users can tune these dials through a common, user-friendly interface without needing to know a vendor's specific implementation details.

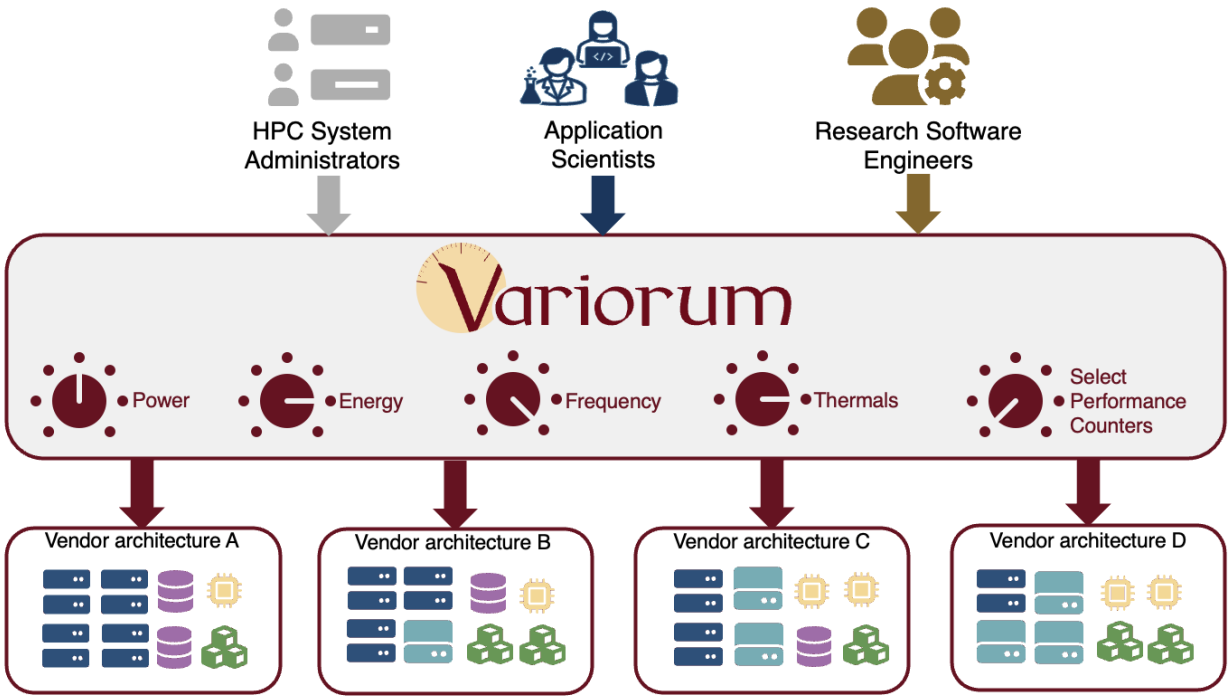

Figure 4: Variorum provides easy-to-use, vendor-neutral access to diverse set of users and integrates easily with other levels and components of a PowerStack.

Variorum adapts to an HPC system's new performance features, deprecates old features among hardware generations, and enables performance optimization toward a particular goal (e.g., to enforce a power limit). When combined with complementary technologies, Variorum ensures production-safe and user-space (non-root) access to these vital low-level dials. Currently, we support several different vendors and multiple generations of their microarchitectures.

For the development of Variorum, we focused on a set of important requirements extracted from our learnings and previous experiences with the development of power management mechanisms:

- **Create device-agnostic APIs:** We do not want the user to have to know or understand how to interface with each vendor and each device. The Variorum library is built for a target architecture, which may be heterogeneous, and can collect data from each device through a single front-facing interface.

- **Provide a simple interface:** We want users and tool developers to be able to not only collect information from the underlying hardware, but also to easily control various features provided by the vendor.

- **Easily extend capabilities to new devices and generations within a device:** We want to easily to support new features, deprecate old features among generations of devices, and adapt features that may have different domains of control from one generation to another (i.e., sockets, cores, threads).

- **Easily integrate with applications, workflows, and system tools:** We want Variorum's interfaces to be simple and portable from the perspective of supporting integrations for many broad use cases with varying optimization goals, including those from application scientists as well as system administrators.

Example: Print Power Limit API

Variorum can be built for each vendor's architecture with the popular CMake or Spack[27] package management and installation tools. Building Variorum creates the `libvariorum` library, the `powmon` monitoring tool, and several Variorum examples. Users can quickly get started with examples of each of our rich APIs, their usage with MPI and OpenMP programming standards, examples with Fortran and Python bindings, and an example of software tools and workflows that may be integrated with Variorum's JavaScript Object Notation (JSON) API.

Variorum provides a suite of robust APIs for printing outputs, managing power levels, enabling/disabling certain features, topology querying, integrating and more. These are described later on and documented on our ReadTheDocs page. In this section, we show one example API.

The example below shows how Variorum's API can be easily used to obtain the power limit of a particular platform in a vendor-neutral manner. The source code below shows an example based in C, but similar code can also be written with Python or Fortran using Variorum bindings. The `variorum_print_power_limit` API prints the output in a tabular format that can be filtered and parsed by a data analysis framework such as R or Python.

The `variorum_print_power_limit` API prints the power limits available on the platform and informs the user about the *maximum* power cap that they can set on that particular platform. This helps the user determine the range in which the power can be controlled and allocated on that particular platform. Because there are differences in internal mechanisms for each vendor, the output varies slightly based on the platform, but the front-facing user-level API stays the same. The API also indicates units as well as other important parameters for the user's consideration when available. In the example below, the user makes a function call to this API in an easy and portable manner. All they need to do is include the `variorum.h` library in their application code and then add one line to obtain the power limit information: `ret = variorum_print_power_limit();`. We show the output obtained with this example program from different platforms below as well.

```
#include <stdio.h>
#include <variorum.h>
int main(int argc, char **argv)
{
    int ret;
    ret = variorum_print_power_limit();
    return ret;
}
```

On an IBM Power9 platform, the output of the above may look similar to the following. Here, you see the hostname of the system, its current power limit, and its maximum and minimum power limits (all in watts or W). Additionally, we see the power-shifting ratio (PSR)[30], which is a specific socket-level feature that IBM Power9 platform provides to set the ratio of power between the CPU and the GPU. The units for this PSR feature are reported as a percentage value, and it is available on both sockets (denoted by _0 and _8 in the example output below).

```
_POWERCAP Host CurrentPower_W MaxPower_W MinPower_W PSR_CPU_to_
GPU_0_% PSR_CPU_to_GPU_8_%

_POWERCAP lassen3 3050 3050 500 100 100
```

On an NVIDIA GPU platform, such as LLNL's Lassen supercomputer, the output may look similar to the following. Here, we see the host as well as the device ID for the GPU, along with the power limit in watts. Lassen has four NVIDIA GPUs per node, so the ID and limit of each GPU is shown in the output.

```
_GPU_POWER_LIMIT Host Socket DeviceID PowerLimit_W
_GPU_POWER_LIMIT lassen1 0 0 300.000
_GPU_POWER_LIMIT lassen1 0 1 300.000
_GPU_POWER_LIMIT lassen1 1 2 300.000
_GPU_POWER_LIMIT lassen1 1 3 300.000
```

On an Intel platform, the output of this example would be similar to the following. Useful information such as the hostname, the socket (processor) ID, power limits, and time windows for enforcement of these are shown in Variorum's output. On Intel systems, the time window limit needs to be specified along with the power limit, so this information is displayed. Intel systems utilize MSRs for obtaining information, so the offset and actual bit values of the registers are also provided, which may be helpful for advanced users. Additionally, memory power limit (marked as `DRAM_POWER_LIMIT`) as well as information about units of conversion is shown as output, as it is available on this platform—but not available on other platforms such as IBM Power9 or NVIDIA GPUs.

```
_ PACKAGE _ POWER _ LIMITS Offset Host Socket Bits PowerLimit1 _ W TimeWindow1 _
sec PowerLimit2 _ W TimeWindow2 _ sec
_ PACKAGE _ POWER _ LIMITS 0x610 thompson 0 0x7851000158438 135.000000 1.000000
162.000000 0.007812
PACKAGE _ POWER _ LIMITS 0x610 thompson 1 0x7851000158438 135.000000 1.000000
162.000000 0.007812
_ DRAM _ POWER _ LIMIT Offset Host Socket Bits PowerLimit _ W TimeWindow _ sec
_ DRAM _ POWER _ LIMIT 0x618 thompson 0 0x0 0.000000 0.000977
_ DRAM _ POWER _ LIMIT 0x618 thompson 1 0x0 0.000000 0.000977
_ PACKAGE _ POWER _ INFO Offset Host Socket Bits MaxPower _ W MinPower _ W
MaxTimeWindow _ sec ThermPower _ W
_ PACKAGE _ POWER _ INFO 0x614 thompson 0 0x2f087001380438 270.000000 39.000000
40.000000 135.000000
_ PACKAGE _ POWER _ INFO 0x614 thompson 1 0x2f087001380438 270.000000 39.000000
40.000000 135.000000
_ RAPL _ POWER _ UNITS Offset Host Socket Bits PowerUnit _ W EnergyUnit _ J
TimeUnit _ sec
_ RAPL _ POWER _ UNITS 0x606 thompson 0 0xa0e03 0.125000 0.000061 0.000977
_ RAPL _ POWER _ UNITS 0x606 thompson 1 0xa0e03 0.125000 0.000061 0.000977
```

Despite these differences between vendors and platforms, from a user's perspective, the front-facing API stays the same, as shown in the first example above. Variorum makes it extremely easy for users to port their application code across various architectures.

System Software Integrations

Variorum has been successfully integrated with production-level system software in the PowerStack including resource managers that help with hardware allocations for scientific applications (e.g., Flux[20], a 2021 R&D100 winner), runtime systems that optimize the critical path of applications (e.g., Intel GEOPM [Global Energy Optimizer and Power Manager][15]), large-scale system monitoring frameworks (e.g., OVIS LDMS [Lightweight Distributed Metric Service][26]), as well as application profiling and portability tools (e.g., Caliper[5] and Kokkos[28]).

Flux[20], an R&D100 2021 winner, is a flexible framework for resource management deployed on several HPC clusters at LLNL and will be the key resource manager for the upcoming El Capitan exascale supercomputer. The framework consists of a suite of projects, tools, and libraries that may be used to build site-custom resource managers for HPC centers. Flux helps manage multiple user applications on shared clusters including tasks such as allocating physical nodes to user job requests, accounting for their projects, and setting job priorities.

The Intel Global Extensible Open Power Manager (Intel GEOPM)[15] is a framework for exploring power and energy optimizations targeting heterogeneous platforms and MPI applications. GEOPM supports use cases such as reading hardware counters and setting hardware controls with platform-independent syntax using a command-line tool on a particular compute node. GEOPM dynamically coordinates hardware settings across all compute nodes for an MPI application, enabling critical path optimizations.

Caliper[5] is a program instrumentation and performance measurement framework. This software allows the user to bake performance analysis capabilities directly into applications and activate them at runtime. Caliper is primarily aimed at HPC applications but works for any C/C++/Fortran program on Unix or Linux platforms. The Kokkos[28] C++ Performance Portability EcoSystem is a solution for writing modern C++ applications in a hardware-agnostic way. The EcoSystem consists of three main components: the Kokkos Core Programming Model, the Kokkos Kernels Math Libraries, and the Kokkos Profiling and Debugging Tools. Kokkos is also a part of the DOE's ECP.

LDMS[26], an R&D100 2015 winner, is a low-overhead, low-latency framework for collecting, transferring, and storing metric data on a large, distributed computing system. Metric information can be updated by a kernel module that runs only when applications yield the processor and are transported using operations similar to Remote Direct Memory Access (RDMA), resulting in minimal overhead and noise during data collection.

Variorum is integrated with all of these system software tools, which means users and system software developers can focus on application results instead of the tuning and portability of low-level dials across different vendor platforms and various system software (see Figure 5).

Figure 5 (left): Variorum has been integrated successfully in all levels of a PowerStack with supported system software such as Flux, Kokkos, Caliper, GEOPM, and LDMS.

Variorum not only provides power management features for system-level software, but it can also enable scientific applications to directly monitor system power during execution. For example, the Multiscale Machine-Learned Modeling Infrastructure (MuMMI) workflow[29], which is an award-winning workflow for cancer research on the Sierra supercomputer, can interface with Variorum at both the simulation level and the Flux job manager level.

The MuMMI workflow has been designed to enable *multiscale* simulations of RAS biology to understand its role in the initiation of cancer. Over 30% of the total cancers in the world have been attributed to the RAS family of cancer-causing genes. Yet, the true role of RAS in the cancer initiation is not well understood, limiting the ability to design drugs targeting RAS. Exploring RAS biology on cell membranes requires unraveling molecular interactions at high resolutions but at biologically relevant sizes and time scales. As illustrated in Figure 6, MuMMI framework couples a single macroscale simulation that spans large time- and length-scales with several thousand molecular dynamics (MD) simulations that provide high-resolution data for small spatiotemporal regions. The framework uses machine learning (ML) to evaluate the macroscale simulation and select regions of interest to spawn the corresponding MD simulations. ML-based sampling allows exploration of the configuration space significantly more effectively resulting in a scope of exploration that is not achievable using only brute force calculations. Finally, *in situ* analysis of the conjugate gradient (CG) simulations provide feedback to improve the parameterization of the macroscale simulation on the fly. With Variorum, MuMMI researchers do not need to understand the low-level details of power management, and can use Variorum's *powmon* monitoring tool or its Flux integration to make their codebase more energy efficient.

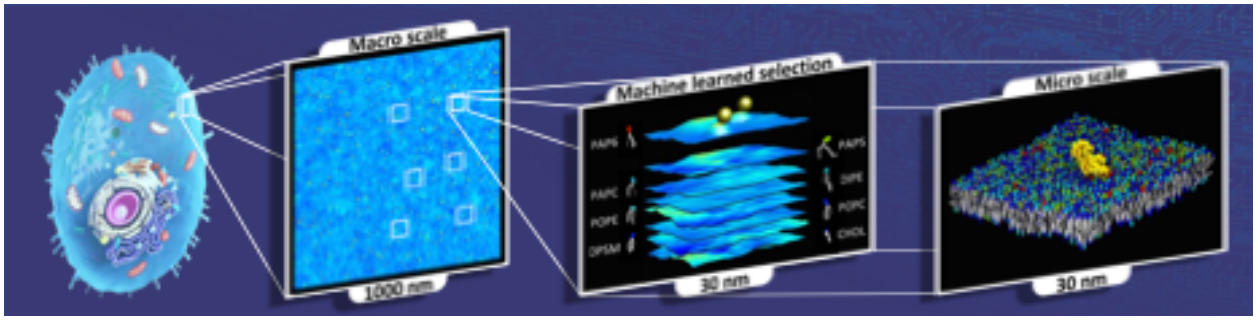

Figure 6: Overview of the MuMMI cancer workflow which shows a combination of machine-learning driven macro- and micro-scale simulations to understand the role of RAS in cancer. Variorum can be used to understand power characteristics of MuMMI and other workflows.

Power Management Use Cases Enabled by Variorum

Variorum enables an end-to-end, vendor-neutral PowerStack, which supports many important use cases in large-scale distributed computing, including those from application scientists, system administrators,

and academic and industry researchers. We discuss these use cases as well as some open challenges in power and energy management, which benefit from the use of low-level optimizations and fine tuning of dials with Variorum and its aforementioned integrations.

Optimization of Scientific Workflow Performance, Power, and Energy

When it comes to large-scale power and energy management, several technical aspects are not well understood and are active areas of ongoing research and development. One such aspect is the incorrect assumption that giving more power to an application will always improve its performance and that enforcing a power cap will always slow an application and hinder its performance. While this is true for CPU-bound and computationally intense applications such as High Performance LINPACK, it does not apply to most scientific workflows and applications. Many applications exhibit specific dynamic phase behaviors and tend to be more bound by memory, I/O (input/output), and network usage. Scientific applications differ from one another in their memory, communication, I/O requirements, and phase behaviors. Being able to steer power correctly based on an individual application's characteristics and its associated phase behaviors is critical for improving overall performance as well as efficient utilization of power—both for a user and the system in terms of energy efficiency.

Figure 7 shows an example of this effect. This dataset shows several well-known parallel application benchmarks running on an Intel Sandy Bridge allocation with 8 nodes, where each node has 2 processors and 8 cores per processor (16 cores per node)[31]. The maximum power limit (peak power) on each processor (2 on each node) is 115 W, and the minimum power limit per processor is 51 W. The x-axis shows the power cap enforced in watts on each processor, while the y-axis shows the relative execution time (or runtime) of the application under the power cap across the 8-node allocation. Lower is better on this graph, which indicates that the application ran faster. OpenMP and MPI are two ways to leverage processor parallelism; the former uses threads, and the latter uses processes. For some of the applications where we could select both mechanisms, we have indicated this choice in the graph as a suffix.

Reading this graph from right to left—maximum power cap to minimum power cap—allows for comparison of performance of individual applications. Some applications, such as LULESH or AMG-MPI, incur a significant performance slowdown when they are given less power. On the other hand, for many applications such as MiniFE-OpenMP or CoMD, there is no impact on application's execution time when a power cap is applied. This means that reducing the power supplied to the application by more than a factor of 2 (from 115 W to 51 W per processor) had no impact on its performance, providing a significant opportunity to save energy costs as well as improve energy efficiency for these applications.

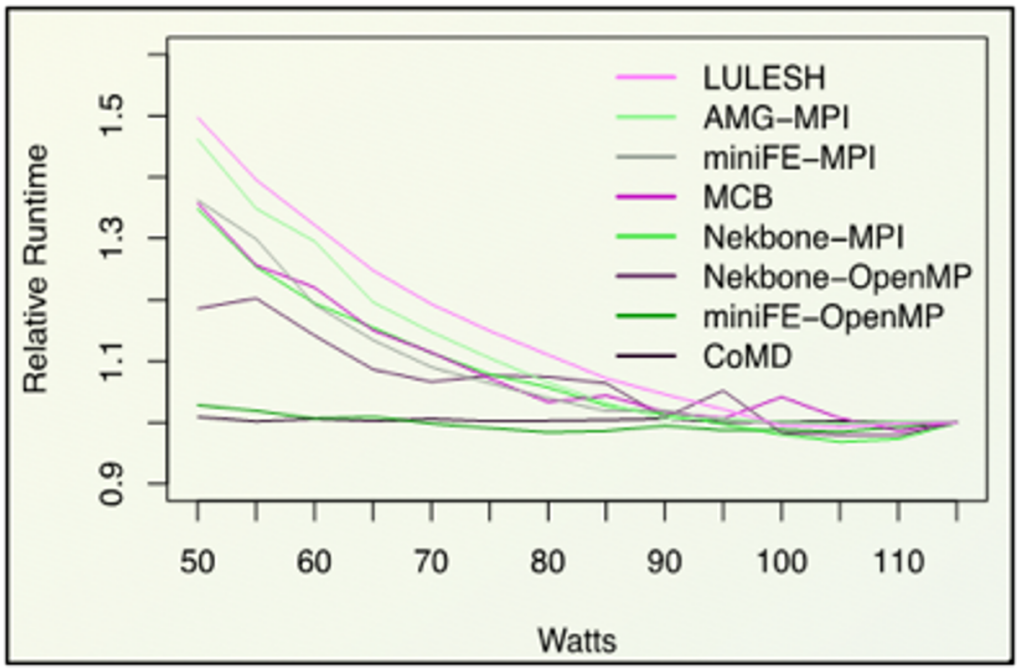

Figure 7: Applications behave differently when power caps are applied. For some applications, their performance is unimpacted by setting a power cap, providing tremendous opportunity for energy savings and cost reduction.

Similar trends are observed with scientific workflows that run on multiple GPUs and at scale, such as the MuMMI workflow described previously. The MuMMI workflow makes significant use of GPUs on the Lassen and Sierra supercomputers, both of which are some of the world's fastest supercomputers hosted at LLNL. A key part of MuMMI includes MD simulations, which use a code called `ddcMD`[32] to provide the data of interest. The `ddcMD` code is run almost exclusively on GPUs to support the multiscale simulations and maximize utilization on the Lassen/Sierra clusters.

Figure 8 shows the impact of GPU power capping on the `ddcMD` code when running on Lassen. Each compute node has 4 NVIDIA Volta GPUs along with 44 CPU cores. The x-axis here is a power cap in watts, and the y-axis is the clock rate or frequency in megahertz (MHz), which is a key indicator of application performance. Higher is better on this graph: A higher frequency indicates better performance (lower execution time) for the application. A total of 4600 profiles of `ddcMD` are shown in this graph, with approximately 200–500 profiles per power cap. The variation observed in their execution time is also shown. The maximum power cap (peak power) on each GPU is 300 W, and the minimum is 100 W. Dropping the per-GPU power from 300 W to 175 W has no impact on the execution time or performance of `ddcMD`. Capping the GPUs on Lassen to 175 W results in cluster-wide power savings of about 382 kW with no performance

slowdown. This translates to 254.6 kWh of energy savings and $43,000 of cost savings per day (assuming 7 cents per kWh), which can have a significant impact on the supercomputing center's energy efficiency. Variorum's integration with LDMS for fine-granularity data collection is critical to identifying opportunities for such savings.

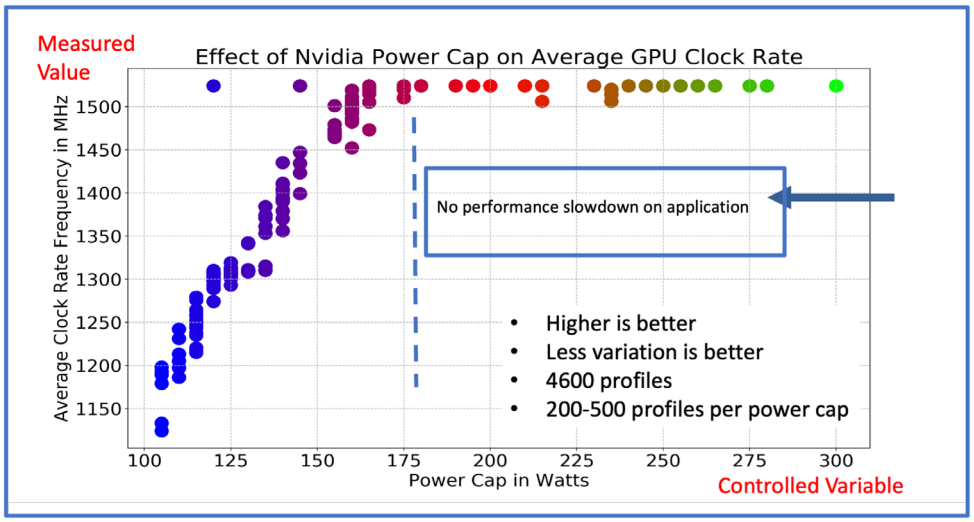

Figure 8: Impact of GPU power capping on ddcMD performance shows that 254.6 kWh of energy can be saved per day with intelligent power capping.

Another important aspect of power and energy management of scientific applications is that of optimizing the critical path under a system-level power constraint. For a parallel application, the critical path is the longest path through its functions, which are typically associated with sections of code that require the most amount of time and resources within the application. Speeding up the critical path can speed up the application significantly. Because applications behave differently under a power budget, selecting the optimal way to distribute an application's parallel tasks during its execution can have a significant impact on this performance. This technique, often referred to as configuration selection, requires a runtime system such as Conductor[25,33,34] (a research-level runtime system developed at University of Arizona) or GEOPM to be present to balance power and performance dynamically between application ranks. Variorum's integration with GEOPM allows for selection of configurations across diverse architectures.

Figure 9 shows a Pareto frontier graph[25] with different configurations (number of cores per node, varying from 2 to 16) and their power consumption, as well as execution time, for the Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) application[35]. LULESH is a proxy application for a mission-critical code that simulates the Sedov blast problem

and approximates the hydrodynamics equations discretely by partitioning the spatial problem domain into a collection of volumetric elements defined by a mesh. As can be seen clearly from the graph, a set of optimal configurations for LULESH are based on the amount of power available; determining these and selecting them at runtime requires integration with power-balancing software such as GEOPM or Conductor.

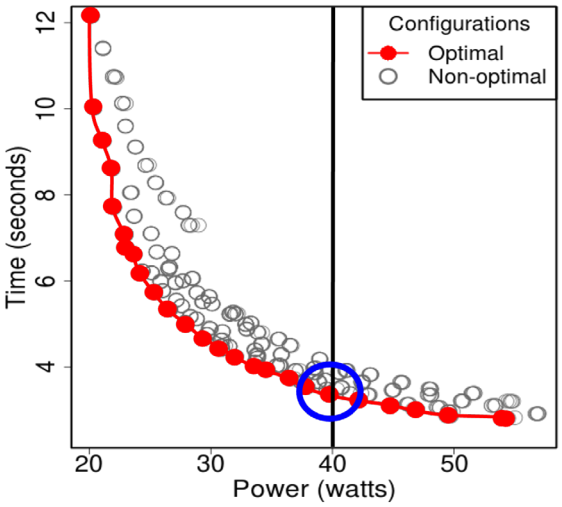

Figure 9 (left): Optimal configurations and power-performance curve for the LULESH proxy application indicates the importance of critical path optimization.

Improving System Power Utilization

Modern supercomputers are designed to be worst-case power provisioned, with the assumption that all nodes in the system should be able to operate at peak power simultaneously. While this system design and power allocation strategy is useful for a few power-hungry benchmarks such as High-Performance LINPACK[36], it often leads to inefficient use of power for most HPC production applications that fail to utilize the allocated peak power on a node, as shown below.

A key limitation of worst-case provisioning in supercomputing design is underutilization of the power infrastructure. This can be validated from large-scale system power data, such as that collected on the Quartz supercomputer at LLNL. The Quartz supercomputer is a 2604-node production cluster consisting of Intel Broadwell nodes with 36 cores per node. The cluster is organized in 42 racks with 62 nodes per rack. Figure 10 shows total power consumption of the Quartz cluster in megawatts on the y-axis and time samples taken every 3 minutes on the x-axis during a 67-month period. Quartz is provisioned for 1.35 MW peak procured power; however, the figure shows that only about 61% of this power is utilized on a regular basis. Other supercomputers at LLNL, as well as at other major supercomputing centers within the U.S. and globally, report similar trends. In our dataset, we observe that 39% of the power is left unutilized, which means there is room for either saving energy by using power capping or improving utilization and throughput with dynamic power management.

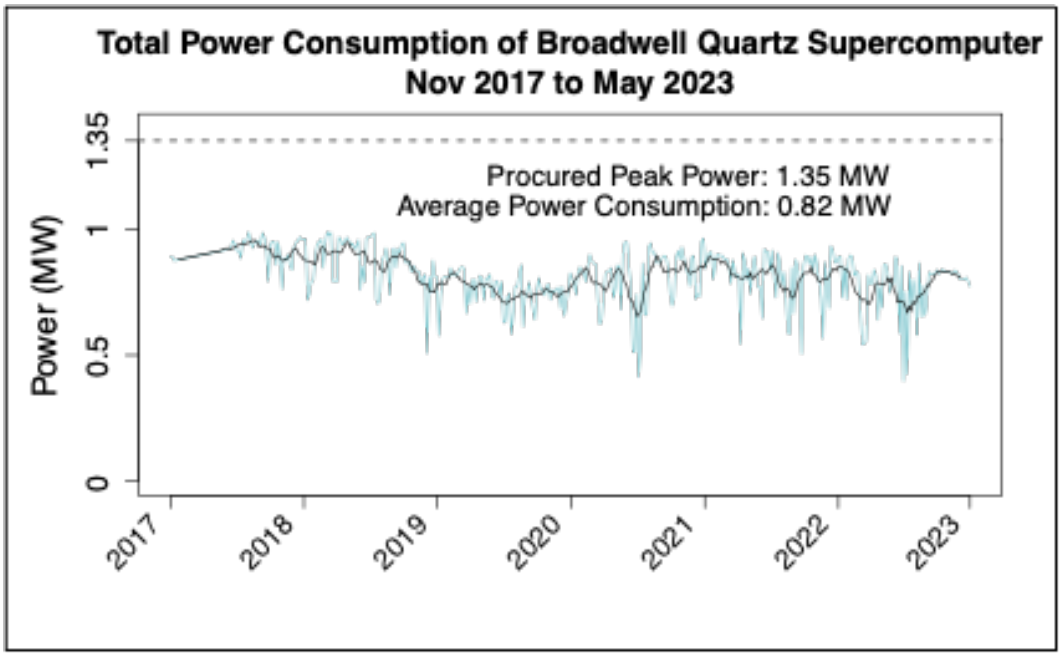

Figure 10: Power utilization on the Quartz supercomputer.

Ideally, supercomputing centers would utilize their procured power fully to accomplish more science, especially as power is a limited and expensive resource. A more flexible and efficient design approach is to build a reconfigurable system that has more capacity (nodes) under the same site-level power constraint and can adapt to applications' requirements. Such a system can provide limited power to a large number of nodes or peak power to a smaller number of nodes, or use an alternative allocation in between based on application characteristics. This approach is referred to as hardware overprovisioning[37] and has been an active research area. For overprovisioning to be successful, and to effectively utilize power with energy-aware scheduling of applications, power must be dynamically managed and intelligently allocated to applications that require it the most. This requires power-aware scheduling or power-aware resource management[31,38,39,40] which can be enabled by designing intelligent allocation policies within the Variorum and Flux integration. Flux can leverage Variorum's power management capability and best-effort power capping API to allocate power based on the job's usage, allowing the system to be more energy efficient.

Detecting and Mitigating Power Swings

Another important use case for large-scale power management is that of power swings that occur due to applications exhibiting distinct phase behaviors such as compute, communication (network traffic), I/O and checkpointing, or global synchronizations—all of which are common scenarios in parallel simulations and codes. Figure 11 shows an example of power swings with

the Livermore Big Artificial Neural Network toolkit (LBANN) [41] running at full scale on the Sierra supercomputer, which is one of top 10 fastest supercomputers in the world[42]. LBANN is a deep learning HPC framework that utilizes all 4 GPUs available on the Sierra nodes. The dataset shows power consumption of LBANN over a 6-hour time period for the application's run, where samples were collected every 3 minutes. Power swings of greater than 200 kW were observed during the run, attributed to the application's phase and synchronization behavior.

Figure 11: Power swings of >200 kW were observed on the Sierra supercomputer when executing a parallel ML application called LBANN at full scale.

Such power swings are common for many HPC applications and could worsen at exascale and beyond, leading to electrical supply grid disruptions and concerns being raised by the electricity provider. Preventing these fluctuations is important, especially as the scale of supercomputing increases. Such swings are common for many HPC applications and could worsen at exascale and beyond. Dynamic power management allows for solutions where power can be ramped down slowly, or power caps can be enforced to limit the amplitude of the swings. Our Variorum and Flux integration is being used to determine how to detect such swings at scale, and how resource managers can proactively prevent them.

Addressing Manufacturing Variability

Another less understood aspect for power and energy efficiency is that of processor manufacturing variability, wherein processors with the same microarchitecture can exhibit inhomogeneous power and performance characteristics, both with and without a power cap. This is attributed to the chip fabrication and lithography process and can result in over 20% run-

to-run variation in application performance without any power constraints as well as cause over 4x variation in power-constrained scenarios[43, 44, 45]. Most vendors, including Intel and IBM, have confirmed that processor manufacturing variability is expected to worsen in the future and at larger scales, making application-level power steering in system software absolutely necessary on future systems.

Figure 12 shows an example of manufacturing variability under power capping for two generations of Intel architectures: Sandy Bridge and Broadwell across three common HPC benchmarks: NAS multigrid (MG), CG, and STREAM[46]. NAS MG approximates the solution to a 3D discrete Poisson equation using the V-cycle MG method. NAS CG estimates the smallest eigenvalue of a large, sparse, symmetric positive-definite matrix using the inverse iteration with the CG method as a subroutine for solving systems of linear equations. The STREAM benchmarks tests for memory intensity and measures memory bandwidth using simple array manipulation operations. The y-axis shows Instructions per cycle (higher is better), and the x-axis shows the power cap. Data from 64 homogeneous processors (exactly same microarchitecture) is shown, and as can be observed rather counterintuitively, there is significant variation in IPC (instructions per clock) from one processor to another. In some situations, performance can vary by as much as 4x.

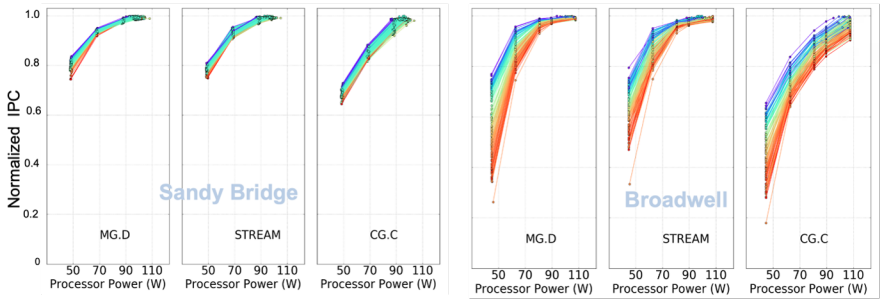

Figure 12: Comparing manufacturing variability on Intel generations.

Such variability in processor performance can significantly impact the applications running on them, especially in power-constrained scenarios. Using Variorum's vendor-neutral APIs integrated with runtime systems such as GEOPM or power-aware policies from Flux, we can significantly reduce these variations and their impact on critical applications.

Consideration for Special Recognition: Green Tech

Environmental factors are playing a key role in the modern society, with many industries embracing goals towards sustainability and energy efficiency. Reducing carbon emissions and charting a path to net zero for large-scale computing systems, including supercomputers and data centers, is an active area of research in the HPC and cloud computing communities—

especially in the new exascale era and with ever-evolving HPC architecture innovations[47, 48, 49, 50, 51, 52, 53]. Current research has focused on aspects of efficient machine room design, water usage, and cooling, but not enough research has been conducted in the area of increasing overall system power utilization and energy efficiency with dynamic power management. One of the main reasons for this dearth of research is the lack of standardized and homogeneous low-level interfaces and the sheer complexity of the vendor-specific configurations and details—i.e., dials, as discussed above—that need to be understood and optimized. The complex dials change often from one generation to another within the same vendor, and even between vendors, making it extremely challenging to develop portable and lasting power management solutions. General and advanced users alike do not have access to the low-level tools that are necessary to study the carbon footprint of their applications or to systematically explore and evaluate the impact of tuning the available vendor-specific dials on their platforms.

Additionally, little research has been conducted in the area of extending the lifetime of existing supercomputers and determining opportunities for their reuse and ideal replacement cycles. Surprisingly, supercomputers are not always retired due to increased hardware failure rates or limited performance on applications, but rather due to the people cost of maintenance and the inability of existing tools to manage mixed-hardware well[52]. Heterogenous hardware is hard to manage from the perspective of portability and developer effort, and as such, results in lack of productivity for system administrators.

Additionally, it has been shown that running large-scale centers at lower temperatures is beneficial from the perspective of CPU and GPU aging and extending their lifetimes. Being able to monitor the temperature of machine rooms and proactively (not reactively) lower the thermal limits by setting power caps is essential to extend the life of large-scale computing systems.

The vendor-neutral Variorum library offers a crucial solution for each of the aforementioned challenges. First, Variorum reduces the complexity associated with vendor-specific and ever-changing low-level dials by providing a vendor-neutral front-facing API for a diverse set of users, making it easy to integrate the tuning mechanism in their application codes. It also supports mixed hardware setups and builds easily, as it is designed to be primarily vendor-neutral. Furthermore, Variorum has the ability to provide temperature monitoring as well as power capping interfaces regardless of vendor platform, and it can be integrated with higher-level system software policies that can dynamically adjust power in response to higher ambient temperatures. Accordingly, Variorum is an essential tool for solving the broader power and energy management problem in large-scale computing.

As shown in the previous sections, supercomputing systems have tremendous opportunities for utilizing power better (e.g., the above example of LLNL's Quartz supercomputer), for power-

aware job scheduling (e.g., Flux integrated with Variorum), runtime configuration section (e.g., Conductor, Kokkos, and GEOPM integrated with Variorum), as well as saving overall energy costs. A vendor-neutral and interoperable PowerStack like the one described earlier, and which relies on Variorum, is critical for making these options possible.

B. How does the product operate?

Under the hood, Variorum relies on the specific vendor's optimization dials and firmware to accomplish its front-facing goals of simple interfaces. Each vendor has its own mechanism to expose both measurement and control dials, and each vendor has a different set of dials available. These dials allow access for power, energy, temperature, clock frequency, and several performance counter values such as instructions per cycle or last-level cache misses. Definitions of these features vary across vendors and even generations of processors for the same vendor. Variorum relies on vendor documentation as well as specific underlying kernel and file system interfaces to make the dials accessible.

For server processors commonly used in large-scale distributed computing setups, monitoring or measurement is typically accomplished through either MSRs which report values measured from hardware, physical sensors on the chip, or file system interfaces that connect to on-chip baseboard controller devices that provide relevant data. Measurement granularity (i.e., fidelity of data when collected at a high frequency) also varies across vendors, but ranges from 1 millisecond to 100 millisecond time scales. Intel CPUs have the most sophisticated and fine granularities, and most GPUs have coarser granularities for monitoring values.

The history of controlling power and energy consumption in processors dates back to early efforts to increase battery life in laptop computers. Those efforts were codified in 1992 with the Advanced Power Management standard (APM), which was quickly supplanted by the Advanced Configuration and Power Interface (ACPI) in 1996. When not plugged in, mobile devices could be instructed by the operating system to run more slowly, thus consuming less energy. When not in use, the devices could enter one of several idle or sleep states, saving even more power at the cost of some latency when returning to a useable state.

The next leap forward came for multicore processors with the Intel Nehalem (2008) architecture. Until Nehalem, processor power had been limited to what could be safely managed assuming all cores were running at the highest CPU clock frequency. Nehalem broke that assumption by allowing processor firmware to make opportunistic decisions about maximum frequency. If fewer cores were in use, they could run faster (and hotter). If all cores were in use, the maximum speed would be set substantially lower. This feature, known as frequency scaling, was mostly invisible to the average user. Advanced users with root privileges, however, could now determine whether their best performance would come about by effectively scheduling more power on fewer cores

or less power on more cores. Enhancements to frequency scaling include dynamic voltage and frequency scaling, or DVFS, which is now commonly available via the Linux cpufreq userspace governor on all architectures.

Intel's Sandy Bridge (2011) architecture marks the beginning of modern server processor power management. For the first time, users were able to set a specific number of watts as the processor power, rather than rely on capping clock frequencies and limiting power as a second-order effect. This power-capping approach is now standard in every server processor architecture, including GPUs. While a few additional power control approaches have been developed on the margins (e.g., throttling memory accesses to reduce dynamic random access memory [DRAM] power), the underlying mechanism is the same: allow the user to set a combination of power and frequency caps, then allow the processor to make rapid changes to the clock frequency so that, over a time window, the average power does not exceed the specified bound.

For controlling power and frequency dials, the following primary mechanisms exist in hardware:

- DVFS (available on all CPU architectures), which we described earlier
- Power capping, through hardware mechanisms such as Running Average Power Limit (RAPL, supported on Intel and AMD) or Open Power Abstraction Layer (OPAL, on IBM)
- Automatic hardware tuning of frequencies through Intel Turbo Boost or IBM UltraTurbo

On GPUs, the underlying power capping mechanism is not public but potentially utilizes some form of DVFS, and is made available through vendor frameworks and libraries such as NVIDIA Management Library (NVML), AMD ROCm, or Intel One API.

Examples of Vendor-Specific Dials

To illustrate the complexity of vendor dials and how they differ from internal interfaces, we consider two examples in this paper. Detailed documentation for each vendor exists on our ReadTheDocs page.

Each Intel architecture has several MSRs for power, temperature, frequency, and performance tuning. Some of these MSRs vary from one generation of Intel processors to another. Some key RAPL registers for power management are listed below, along with their hardware addresses and read/write or read-only permissions:

- `MSR_PKG_POWER_LIMIT, 0x610h, rw:` Allows software to set package power limits for a given time window
- `MSR_PKG_ENERGY_STATUS, 0x611h, ro:` Reports measured actual energy usage for processor

- `MSR _ PKG _ POWER _ INFO, 0x614h, ro`: Reports meta information for processor, such as range of valid values
- `MSR _ DRAM _ POWER _ LIMIT, 0x618h, rw`: Allows software to set memory power limits
- `MSR _ DRAM _ ENERGY _ STATUS, 0x619h, ro`: Reports measured actual energy usage for memory
- `MSR _ DRAM _ POWER _ INFO, 0x61Ch, ro`: Reports the DRAM domain meta information
- `IA32 _ APERF _ MSR and IA32 _ MPERF _ MSR, 0xE7h,0xE8h,ro`: Define a ratio that depicts instantaneous processor frequency

Figure 13 shows the internals of the MSR_PKG_POWER_LIMIT. To set a power cap on an Intel CPU architecture, specific bit values corresponding to that power value need to be written to the address in this MSR, along with an associated time window, which is also specified in bits. Two power limit ranges exist, as shown below, and although the first limit (Pkg Power Limit #1) is commonly used to set power caps and is best practice, little to no documentation exists from Intel on how the second power limit should be used. Each MSR listed above, and many others, have similar details that need to be understood and meticulously calibrated on our testbed systems before Variorum can safely deploy them to the general user in a vendor-neutral manner with a front-facing API—and this example is limited to just the Intel dials.

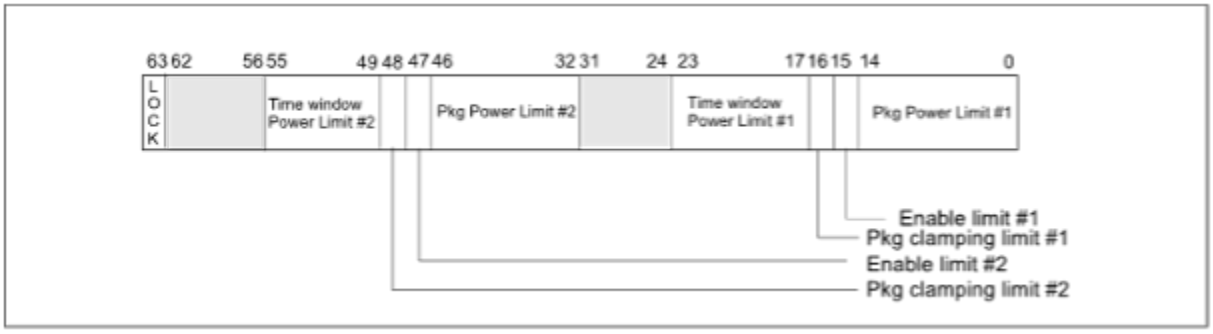

Figure 13: One of the many power management MSRs available on Intel Broadwell architecture.

Figure 14 shows a similar example from the IBM Power9 architecture, which has a primary-secondary processor chip design (represented by the two On Chip Controller, or OCCs, in the top row which connect to the sets of GPUs). This architecture also has a baseboard management controller (BMC) in addition to its two compute CPU processors. The "OCC control" resides on the BMC (shown in green in the bottom half) and connects to both the primary and secondary chips. The OCC has direct access to the system bus and memory, and collects power and thermal data every 250 microseconds through sensors. It exposes this through both lm-sensors as well as in-band sensor channels that Variorum leverages to obtain this information. Power control dials are exposed with OPAL, which allows Variorum to write values to sysfs interfaces that interact with the OCC controller to enforce power limits on the compute-node level. Variorum abstracts such

complex and tedious details away from the users for each platform supported, truly achieving ease of access and portability.

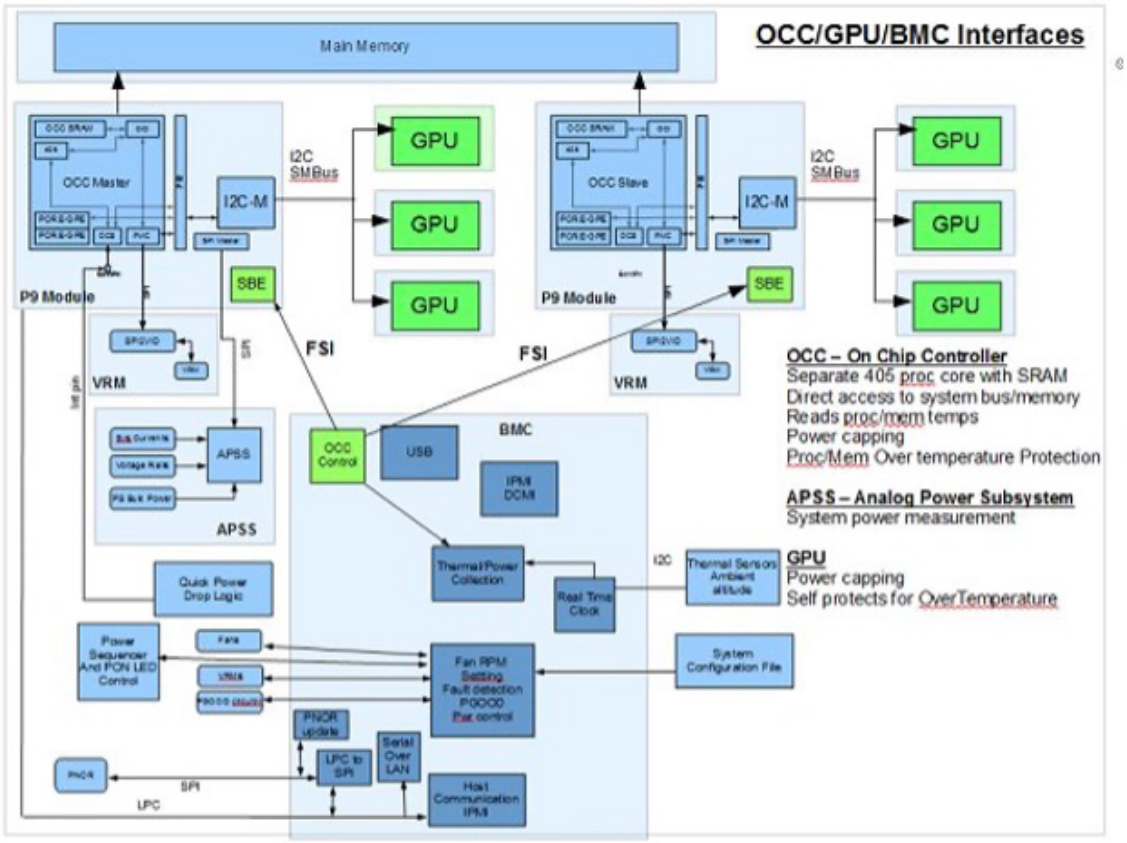

Figure 14: Internals of an IBM Power9 Witherspoon node, along with the details of the OCC that allows for power and thermal management.

Variorum APIs

Through its rich suite of APIs, Variorum abstracts the complexity of the underlying support for vendor-neutral power and energy management. The top-level API for Variorum is in the variorum.h header file. Key categories of vendor-neutral APIs are provided through Variorum, which we describe below:

- Variorum Print Functions
- Variorum Cap Functions, including Best Effort Power Capping
- Variorum JSON-Support Functions
- Variorum Enable/Disable Functions
- Variorum Topology Functions

For each feature on each architecture, there is a print and print_verbose API, which will print the metrics in different output formats. The print API prints the output in tabular format that can be filtered and parsed by a data analysis framework, such as R or Python. The print_verbose API prints the output in verbose format that is more human-readable (e.g., with units, titles). Some examples of supported APIs include:

- `variorum_print_power`
- `variorum_print_thermals`
- `variorum_print_counters`
- `variorum_print_frequency`
- `variorum_print_features`
- `variorum_print_gpu_utilization`
- `variorum_print_energy`

Variorum Cap APIs are designed to tune dials and set various power and frequency options available on the architecture. Some examples of capping APIs include the following. Note that input values have to be provided by the users or the system software that is integrating with Variorum. Variorum checks for valid inputs in all cases and generates a meaningful error message in case the input values are incorrect.

- `int variorum_cap_each_socket_power_limit(int socket_power_limit)`
- `int variorum_cap_gpu_power_ratio(int gpu_power_ratio)`
- `int variorum_cap_each_core_frequency_limit(int cpu_freq_mhz)`
- `int variorum_cap_best_effort_node_power_limit(int node_power_limit)`

The capping API, `variorum_cap_best_effort_node_power_limit`, is a special API. Because vendor dials for power capping vary and can cause portability issues, this API is designed to support setting of best-effort node power limits in a vendor-neutral manner. This interface has been developed for higher-level tools that utilize Variorum on diverse architectures and need to make node-level decisions. When the underlying hardware does not directly support a node-level power cap, a best-effort power cap is determined in software to provide an easier interface for higher level tools (e.g., Flux, Kokkos). For example, while IBM Witherspoon inherently provides the ability to set a node-level power cap in watts through its OPAL infrastructure, Intel architectures currently do not support a direct node-level power cap through MSRs. Instead, on Intel architectures, fine-grained CPU and DRAM level power caps can be dialed in using MSRs. Note that IBM Witherspoon does not provide fine-grained capping for CPU and DRAM level, but allows for a power-shifting ratio between the CPU and GPU components on a processor. Our

API, `variorum_cap_best_effort_node_power_limit()`, allows us to set a best effort power cap on Intel architectures by uniformly distributing the input power cap value across sockets as CPU power caps.

Variorum's JSON APIs are designed for interfacing with system software and user workflows in a vendor-neutral manner and enable language-independent bindings. JSON is a widely-used, open standard text-based format for storing and transporting data. It is human-readable, low overhead, and language-independent, allowing it to be portable across C, C++, Python and other languages. JSON also allows for easy integration with system software such as Flux, Caliper, Kokkos, and LDMS. Having a JSON API is a key feature of Variorum, as not many low-level libraries provide such an interface. Currently, two significant JSON APIs are supported, which have been crucial for all the integrations we discussed in the previous sections (e.g., Flux, Kokkos, Caliper, LDMS):

- `int variorum_get_node_power_json(char **get_power_obj_str)`
- `int variorum_get_node_power_domain_info_json(char **get_domain_obj_str)`

The first JSON API to obtain node power uses a string (char**) by reference as input, then populates this string with a JSON object with CPU, memory, GPU (when available), and total node power. The total node power is estimated as a summation of available domains if it is not directly reported by the underlying architecture (e.g., as is the case with Intel).

The variorum_get_node_power_json(char**) includes a string-type JSON object with the following keys, which make it vendor-neutral:

- `hostname (string value)`
- `timestamp (integer value)`
- `power_node (real value)`
- `power_cpu_watts_socket* (real value)`
- `power_mem_watts_socket* (real value)`
- `power_gpu_watts_socket* (real value)`

The asterisk refers to Socket ID. On Intel microarchitectures, total node power is not reported by hardware. As a result, total node power is estimated by adding CPU and DRAM power on both sockets. On systems without GPUs, or systems without memory power information, the value of the JSON fields is currently set to -1.0 to indicate that the GPU power or memory power cannot be measured directly. This has been done to ensure that the JSON object itself stays vendor-neutral.

The second JSON API for querying power domains allows users to query Variorum to obtain information about domains that can be measured and controlled on a certain architecture. This API also includes information about the units of measurement and control, as well as the minimum and maximum values for setting the controls. If a certain domain is unsupported, it is marked as such.

The query API, `variorum_get_node_power_domain_info_json(char**)`, accepts a string by reference and includes the following vendor-neutral keys:

- `hostname (string value)`
- `timestamp (integer value)`
- `measurement (comma-separated string value)`
- `control (comma-separated string value)`
- `unsupported (comma-separated string value)`
- `measurement_units (comma-separated string value)`
- `control_units (comma-separated string value)`
- `control_range (comma-separated string value)`

The Variorum Enable/Disable APIs support enabling and disabling of features such as TurboBoost, which are frequency boosting options that are available on selected devices. The Topology APIs allow users to query basic information about the compute node, such as its total cores or sockets. All APIs are well documented on our ReadTheDocs page.

Automatic Detection of Topology and Mapping of Power Domains

As system heterogeneity continues to be the de-facto mechanism to achieve Exascale performance, the task of enumerating the power domains on the system and breaking down power information on each device and component on a compute node is becoming non-trivial. Variorum simplifies this process of discovering, mapping and translating the high-level power control and telemetry information for the target system. For each supported API call that the user invokes, Variorum first performs an enumeration of the underlying supported devices on the target system. For this task, Variorum leverages existing device enumeration library `hwloc`[54] to collect device-level topology including CPU cores, GPU, memory and other peripheral devices. For each available device, Variorum uses the low-level domain discovery APIs to enumerate the power domain information. For high-level power control APIs, Variorum then uses the input control information to derive and apply control information for each enumerated device on the system using their respective low-level control interfaces. For high-level telemetry APIs, Variorum aggregates the telemetry collected on the individual devices using low-level telemetry interfaces on the supported devices.

Non-Intrusive Monitoring

Although the Variorum API allows for detailed critical path analysis of an application's power profile through source-code annotations as well as for integration with system software (e.g., Kokkos, Caliper, LDMS, Flux), in some scenarios such code annotations or integrations are not possible because the underlying codebase cannot be modified by the user due to limited permissions and users may not have permissions to link an external library. Accordingly, we provide the `powmon` monitoring tool along with the regular Variorum build. This tool can monitor an application externally and non-intrusively without requiring any code changes or annotations and collect power and performance data in a vendor-neutral manner. While a target application executable is running, `powmon` collects time samples of power usage, power limits, energy, thermals, and other performance counters at a regular interval.

For example, the command below will sample the power usage every 50 milliseconds for the application `userapp`, an executable application file specified by the user. The tool can also be used for collecting data on multiple nodes as well. It generates a standard comma-separated-value (CSV) file with the collected power and performance data as its output, that the user can utilize for analyzing the application.

```
$ powmon -a ./userapp
```

Language Bindings and Wrappers

In addition to C/C++ annotations, Variorum also supports Fortran and Python APIs. The Fortran wrapper is only built and installed if Fortran is found and enabled. For the Python module (called pyVariorum), a simple `pip` based install or setting of `PYTHONPATH` is needed. Examples of the usage of these wrappers are well documented so users can easily add these to their codebases. These bindings allow users to integrate Variorum's API into their Fortran and Python applications easily.

Continuous Integration and Testing Framework

Variorum code is regularly tested on a diverse set of architectures, including several Intel platforms (e.g., Broadwell, Haswell, Skylake, Ice Lake), IBM Power 9, ARM Juno r2 and Neoverse, NVIDIA Volta GPUs, and AMD Instinct GPUs. Variorum's unit tests are run externally on GitHub, as well as internally on LLNL's Livermore Computing clusters through GitLab. Within one of our GitLab continuous integration (CI) processes, we are also leveraging Ansible to expand our testing across an additional set of hardware architectures. These systems require specific requests and permissions to gain access. As part of Variorum's CI testing, we cover the following to ensure that Variorum source code stays error-free and compatible across various platforms:

- Verifying compliance of code format for C/C++, RST, and Python files
- Building Variorum with different versions of `gcc` compiler
- Building Variorum with different CMake build options and the Spack package manager
- Running Variorum's unit tests and examples (e.g., `make test` and `variorum-print-power-example`)

Applicability, User Base, and Contributors

Actively developed since 2017, Variorum is licensed as open-source software and accessible through LLNL's GitHub organization. Since the first production release (v0.1) in November 2019, Variorum has come a long way with new features and expanded vendor accommodations. The next version (v0.7, to be released in June 2023) is ready for production use on existing supercomputers, and will support all three upcoming U.S. exascale supercomputers (El Capitan at LLNL, Aurora at Argonne National Laboratory, and Frontier at Oak Ridge National Laboratory) and many other HPC systems. This version has also been integrated into the Tri-Lab Operating System Stack (TOSS), which is installed on many of the supercomputers at LLNL, Sandia, and Los Alamos national laboratories.

Variorum fills a gap in HPC users' and system software developers' access to power and performance management dials across vendors. It is the only vendor-neutral framework that does so while ensuring safe access in a production environment. Variorum covers a broad swath of users and use cases with its current support for multiple CPU microarchitectures and GPUs across AMD, ARM, IBM, Intel, and NVIDIA platforms, and multiple generations within each platform (see Variorum's documentation for the complete list).

Variorum appeals to a wide user base beyond researchers and application code teams, including system administrators of large-scale HPC clusters and system software developers, with potential to also benefit cloud computing and edge computing users. Variorum is designed for writing portable power management code in a vendor-neutral manner, and its user-friendly interface allows for easy integration for many use cases (e.g., power-aware resource management, application performance optimization under power bounds, energy accounting). Variorum users need not know any of the complex, low-level details about power or energy management across different vendor platforms.

Variorum benefits from contributors across industry, national laboratories, and academia, such as Argonne National Laboratory, AMD, IBM, Intel, ARM, NVIDIA, among others. An active list of Variorum contributors is maintained on a ReadTheDocs page. To actively support the open-source community, the Variorum development team (see Figure 15) regularly organizes workshops, tutorials, and presentations. Our documentation website also provides users with instructions and examples for easy access.

Figure 15: The Variorum development team is based at LLNL and regularly engages with the project's user community.

C. Product comparison

A wealth of performance monitoring and collection tools have been developed in the HPC community to provide insight into application execution behaviors. Performance tools cover a wide range of granularities in the data they collect and present to the user, and gathering execution behaviors enables us to optimize the application for the target architecture and achieve the best performance. Performance tools cover a wide range of granularities in the data they collect and present to the user. Users can manually instrument specific regions of interests in their applications with tools such as Performance API (PAPI)[1], Perf[2], TAU[3, 4, 5], and Caliper[6]. Tools such as Vampir[7], Jumpshot[8], and Score-P[9] enable post-execution analyses of per-thread traces. Following execution, a trace detailing per-thread execution behaviors is produced for post-hoc analyses using one of the analysis tools integrated into TAU, such as Vampir[7], Jumpshot[8] or Score-P[9].

On the other hand, tools such as LDMS[10] and Redfish[11] gather information about system resource utilization. Users can specify metrics and events to collect and the frequency at which to collect them. System administrators use the system-wide performance metrics to avoid security issues. Information gathered here is limited to the metrics configured by the tool itself—meaning, these tools do not provide a mechanism for users to self-configure counters to collect different metrics.

TAU[3,4,5], is a large-scale project for profiling applications in myriad ways, such as direct instrumentation of user functions and event-based sampling. It offers several ways of presenting the collected data, which can be very good for ttributing data to code regions. TAU has been ported to many other existing performance tools, which provide specific information about the application's behavior. However, TAU's capability is limited to the extent that these ported tools support the underlying features and architectures, and TAU does not support any control operations (such as setting power or frequency) or higher-level integration APIs.

`Libpfm`[55] is an open-source helper library used by applications to obtain specific performance monitoring events that are provided by the hardware or the OS kernel, including hardware performance counters such as elapsed cycles or cache misses. The library provides two key features: first, it provides an event listing and query interface that can be used to discover the events available on a specific hardware platform. It also provides a set of functions to obtain event data and a generic API to specific kernel APIs such as `perf _ events`. It supports a rich set of architectures, platforms and hardware features, but does not have a user-friendly interface or any software tool integration APIs.

`PlatformIO`[56] is an open-source low-level library that has been designed by Intel Corporation primarily to support its GEOPM runtime system, and is intended to be vendor agnostic. Currently, it supports a multitude of low-level dials from Intel architectures, and also some from IBM and NVIDIA architectures. It has however not been designed from a general user's perspective and requires an expert user for its interfaces.

`Libmsr`[11] and `msr-safe`[12,13,14] synergistically provide user space access to traditionally privileged-only hardware features and controls on Intel systems. The `msr-safe` kernel module provides a mechanism for users to access MSRs without exposing security risks. System administrators select which registers to whitelist and the specific bits that are safe to access, if applicable (some registers are read-only). The `libmsr` API calls check for the presence of `msr-safe` before performing read/write operations on registers, but the `libmsr` library is Intel-specific and not vendor-neutral. Variorum internally makes use of `msr-safe` kernel module for providing safe access to MSRs that require elevated permissions for its implementation relevant to the Intel platform.

Both GEOPM[15], a production-grade runtime framework developed at Intel, and LIKWID[16], a lightweight set of command-line tools targeting x86 architectures, have a dependency on the `msr-safe` kernel module. Recently, PAPI added support for power and energy usage through Intel's RAPL[17] and NVIDIA's NVML[18]. As PAPI is integrated into other tools, users will also have access to the additional data that PAPI provides. However, accessing power and energy information requires a specific install of PAPI with elevated access, which is not configured on every system.

Power API[19] detects the system hierarchy and provides standard interfaces for hardware vendors to comply with when providing monitoring and control interfaces specifically for power and energy. Current support is available for very limited devices such as a single generation of Intel and AMD processors, Power Insight, WattsUp, generic CPU registers, and Cray's XTPM. Additionally, in order to use the Intel RAPL plugin, the user must have privileged access to the stock MSR driver, which Variorum resolves.

D. Comparison summary

As mentioned in the previous section, while many performance monitoring and tracing tools exist, they do not offer a rich set of APIs for control and capping of power or frequencies, or APIs for thermal information, or a set of JSON APIs for integrating with higher levels in the stack. Some of them are not vendor-neutral. For our comparison table, we compare only with the most sophisticated and mature node-level libraries/paradigms that are available, to allow for meaningful comparison.

We depict two comparison matrices, one for hardware feature support on various vendor architectures and associated upcoming exascale systems, and another for software features. Both these feature sets are important from a user's perspective. The hardware features allow users to port their code to a diverse set of CPUs, GPUs and upcoming exascale systems. The software features make this access easy-to-use, portable, and allow for critical integrations such as those with monitoring frameworks and resource managers. Table 1 shows the Hardware Feature Comparison Matrix and Table 2 shows the Software Feature Comparison Matrix.

For Table 1, we consider three other major low-level libraries that are widely used in the community: PAPI, PlatformIO and `Libpfm`, which we have described previously. PAPI and `Libpfm` support many vendor architectures and have a robust set of monitoring APIs, and have been integrated into monitoring tools such as Caliper. However, they both have very limited support for any capping APIs (such as power or frequency capping) or enable/disable APIs. PAPI only supports these on Intel CPUs, and Libpfm does not support them at all. These APIs are extremely important for dynamic power management, without them, power cannot be redistributed or

reallocated. PlatformIO on the other hand has support for both power capping and frequency capping, but it supports fewer vendors as it is designed by Intel. We summarize these differences in Table 1, where we show each vendor platform, their CPU/GPU options, as well as upcoming exascale system readiness. The Aurora system at Argonne National Laboratory is an Intel-architecture based system, and the Frontier and El Capitan systems at Oak Ridge National Laboratory and Lawrence Livermore National Laboratory are AMD-architecture based systems. Overall, Variorum supports many hardware features and offers a wider coverage to the users.

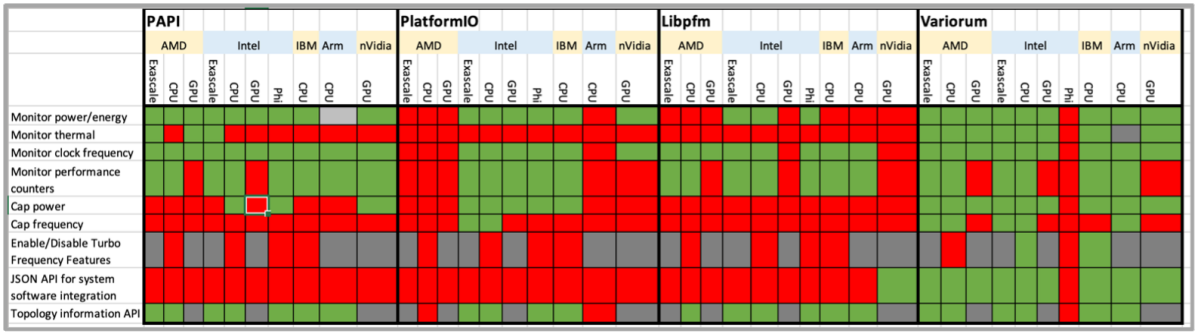

Table 1: Hardware Feature Comparison Matrix

In Table 2, we consider the above three libraries, and we also consider PowerAPI here for completeness. PowerAPI has been designed to be a user-friendly and end-to-end system API, and is an attempt to standardize low-level interface operations across vendors. Its hardware feature set is extremely limited, and only supports previous generations (from 3–5 years ago) of vendor architectures, as we discussed in the related work section, even though there are plans for adding more hardware features. Its software feature set is worth considering in our comparison due to its rich set of user-friendly APIs. In our software feature set, we consider user-friendliness, language bindings, and integrations with system software such as resource managers, runtime systems and monitoring frameworks. Another important feature we consider is support for Message Passing Interface (MPI) versus non-API science applications. Traditional HPC applications use MPI, but this trend has been rapidly changing with ML workflows and simulations that accomplish science without relying on MPI. Being able to support both types of applications is crucial for upcoming supercomputers as well as data centers.

PAPI and PlatformIO support a wide set of software features. While PAPI is user friendly, it cannot be integrated into dynamic power management runtime systems due to the lack of rich capping APIs. It is typically used at the application level, and has not been integrated into resource managers. Platform IO provides both runtime system and resource management integrations (the latter is a bit limited) but it does not have a set of user-friendly APIs and needs an expert user. Overall, we find that Variorum offers the best set of software features among these.

Category	Software or Framework	PAPI	PlatformIO	PowerAPI	Libpfm	Variorum
User-friendly and high-level APIs						
Language Bindings						
	C/C++					
	Python					
	Fortran					
Non-intrusive monitoring for applications						
Support for traditional MPI/OpenMP applications						
Support for non-MPI workflows (e.g.,MuMMI)						
Support for non-HPC Environments						
	Containers					
	Bare-metal cloud					
Resource manager integration						
	Slurm					
	Flux					
Runtime system integration						
	GEOPM					
	Kokkos					
Monitoring Frameworks						
	LDMS					
	DCDB					
	Caliper					

Table 2: Software Feature Comparison Matrix

Product Limitations

Variorum's current version (v7.0) has a few limitations. One limitation comes from the challenge of estimating node-level power usage and enforcing vendor-neutral, node-level power caps on platforms that do not support such capping in hardware. To address this challenge, Variorum has introduced the vendor-neutral Best Effort Node Power Capping API (discussed above), which allows for an estimate-based power capping as opposed to the one inherently supported by the underlying hardware. This Best Effort Node Power Capping API works well in practice and can be easily integrated into system software, but its current capability is limited on platforms that lack native hardware-based node-level capping support (e.g., Intel). This is because enforcing a software-level best-effort power cap requires us to estimate how much power is being consumed by the application, necessitating a sophisticated mathematical and analytical model of the underlying architecture that accommodates for components such as individual cores, memory, connectors, and those in other areas outside the processing core. Such models are complex to develop and are unlikely to be portable, so we make simplifying assumptions in our API implementation. For example, we currently do not use memory power caps through

the best-effort node power capping mechanism on Intel systems. We plan to develop better techniques for best-effort software capping in the future.

Another limitation is in the `powmon` tool, which, in its current form, supports two modes: one that exports instantaneous power usage values (through the JSON APIs), and another that exports data from all possible counters (selected through MSRs on Intel) or sensors (on IBM) or dials. The latter mode generates a comprehensive and sophisticated dataset, but results in a verbose trace file. Additionally, this second mode is targeted at the advanced user who understands the nuances of each of the values in that dataset. Future work involves adding another novel mode to `powmon` that allows users to specify which counters or sensors they want to collect data from. Accordingly, the general user would be able to select the data of interest to their use case, while the advanced user would still access the comprehensive and sophisticated dataset. We expect to address both of these enhancements in upcoming Variorum releases.

A third limitation occurs when some features are not supported by the underlying vendor, requiring Variorum to exit with a graceful error message about the unavailable feature. This occurs due to physical hardware limitations and lack of support for certain features. For example, boost frequency options such as TurboBoost are not available on any server (non-gaming) GPUs, and are not exposed easily by GPU manufacturers. If a new user unaware of this detail tries to access an API such as `variorum_enable_turbo` on a GPU device, an error message tells them that the feature is not supported on the device. This type of limitation is beyond the control of the Variorum's development team, and we mitigate these situations by providing thorough documentation on dials, as well as by answering queries users submit through GitHub issues.

4. SUMMARY

Power and energy efficiency are key design constraints for large-scale distributed computing in terms of both monetary and environmental cost. Optimizing power and energy at large scale requires fine tuning of obscure, complex, and often poorly documented low-level options, or "dials"—an extreme challenge for users. The open-source and extensible Variorum software library provides a rich suite of APIs that deliver portable, homogeneous, easy-to-use, and vendor-neutral interfaces for these low-level dials that are critical for HPC power and performance optimization. With Variorum, application scientists, system administrators, and researchers alike can tune these dials through a common, user-friendly interface without needing to know a vendor's specific implementation details. Variorum provides robust interfaces that allow measurement and optimization of computation at the physical level: temperature, cycles, energy, and power. With that foundation, scientists and HPC centers can make the best possible use of their computing resources—from purchasing to runtime systems to job scheduling, and ranging from bare-metal cloud instances to the some of the world's fastest supercomputers.

5. REFERENCES

1. Mucci, P.J., Browne, S., Deane, C. and Ho, G., 1999, June. PAPI: A portable interface to hardware performance counters. In *Proceedings of the department of defense HPCMP users group conference* (Vol. 710).

2. Shende, S.S. and Malony, A.D., 2006. The TAU parallel performance system. *The International Journal of High Performance Computing Applications*, 20(2), pp.287–311.

3. Lindlan, K.A., Cuny, J., Malony, A.D., Shende, S., Mohr, B., Rivenburgh, R. and Rasmussen, C., 2000, November. A tool framework for static and dynamic analysis of object-oriented software with templates. In *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing* (pp. 49–49). IEEE.

4. Malony, A., Shende, S., Trebon, N., Ray, J., Armstrong, R., Rasmussen, C. and Sottile, M., 2005. Performance technology for parallel and distributed component software. *Concurrency and Computation: Practice and Experience*, 17(2–4), pp.117–141.

5. Boehme, D., Gamblin, T., Beckingsale, D., Bremer, P.T., Gimenez, A., LeGendre, M., Pearce, O. and Schulz, M., 2016, November. Caliper: performance introspection for HPC software stacks. In SC'16: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 550–560). IEEE.

6. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S. and Nagel, W.E., 2008. The Vampir performance analysis tool-set. In *Tools for High Performance Computing: Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008*, HLRS, Stuttgart (pp. 139–155). Springer Berlin Heidelberg.

7. Zaki, O., Lusk, E., Gropp, W. and Swider, D., 1999. Toward scalable performance visualization with Jumpshot. *The International Journal of High Performance Computing Applications*, 13(3), pp.277–288.

8. Knüpfer, A., Rössel, C., Mey, D.A., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A. and Nagel, W.E., 2012. Score-p: A joint performance measurement run-time infrastructure for Periscope, Scalasca, Tau, and Vampir. In *Tools for High Performance Computing 2011: Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing, September 2011*, ZIH, Dresden (pp. 79–91). Springer Berlin Heidelberg.

9. Agelastos, A., Allan, B., Brandt, J., Cassella, P., Enos, J., Fullop, J., Gentile, A., Monk, S., Naksinehaboon, N., Ogden, J. and Rajan, M., 2014, November. The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In S*C'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 154–165). IEEE.

10. www.dmtf.org. (n.d.). REDFISH | DMTF. [online] Available at: https://www.dmtf.org/standards/redfish [Accessed 23 May 2023].

11. software.llnl.gov. (n.d.). libmsr. [online] Available at: https://software.llnl.gov/libmsr/ [Accessed 23 May 2023].

12. Shoga, K., Rountree, B., Schulz, M. and Shafer, J., 2014, November. Whitelisting MSRs with msr-safe. In *3rd Workshop on Exascale Systems Programming Tools*, in conjunction with SC14.

13. Walker, S. and McFadden, M., 2016, May. Best practices for scalable power measurement and control. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops* (IPDPSW) (pp. 1122–1131). IEEE.

14. GitHub. (2023). NAME. [online] Available at: https://github.com/LLNL/msr-safe/ [Accessed 23 May 2023].

15. Eastep, J., Sylvester, S., Cantalupo, C., Geltz, B., Ardanaz, F., Al-Rawi, A., Livingston, K., Keceli, F., Maiterth, M. and Jana, S., 2017. Global extensible open power manager: a vehicle for HPC community collaboration on co-designed energy management solutions. In *High Performance Computing: 32nd International Conference, ISC High Performance 2017*, Frankfurt, Germany, June 18–22, 2017, Proceedings 32 (pp. 394–412). Springer International Publishing.

16. Treibig, J., Hager, G. and Wellein, G., 2010, September. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In 2*010 39th international conference on parallel processing workshops* (pp. 207–216). IEEE.

17. Guide, P., 2011. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide*, Part, 2(11).

18. docs.nvidia.com. (n.d.). NVML API Reference. [online] Available at: https://docs.nvidia.com/deploy/nvml-api/nvml-api-reference.html [Accessed 23 May 2023].

19. Grant, R.E., Levenhagen, M., Olivier, S.L., DeBonis, D., Pedretti, K.T. and Laros III, J.H., 2016. Standardizing power monitoring and control at exascale. *Computer,* 49(10), pp.38–46.

20. Ahn, D.H., Garlick, J., Grondona, M., Lipari, D., Springmeyer, B. and Schulz, M., 2014, September. Flux: A next-generation resource management framework for large HPC centers. In *2014 43rd International Conference on Parallel Processing Workshops* (pp. 9-17). IEEE.

21. Exascale Computing Project. (n.d.). Home Page. [online] Available at: https://www.exascaleproject.org/.

22. The Argo Project. (n.d.). Argo. [online] Available at: https://web.cels.anl.gov/projects/argo/ [Accessed 24 May 2023].

23. Perarnau, S., Van Essen, B.C., Gioiosa, R., Iskra, K., Gokhale, M.B., Yoshii, K. and Beckman, P., 2019. Argo. *Operating Systems for Supercomputers and High Performance Computing*, pp.199–220.

24. The HPC PowerStack. (n.d.). The HPC PowerStack. [online] Available at: https://hpcpowerstack.github.io/ [Accessed 24 May 2023].

25. Marathe, A., Bailey, P.E., Lowenthal, D.K., Rountree, B., Schulz, M. and de Supinski, B.R., 2015. 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 (pp. 394–408). Springer International Publishing.

26. GitHub. (n.d.). Home. [online] Available at: https://github.com/ovis-hpc/ovis-wiki/wiki [Accessed 24 May 2023].

27. Gamblin, T., LeGendre, M., Collette, M.R., Lee, G.L., Moody, A., De Supinski, B.R. and Futral, S., 2015, November. The Spack package manager: bringing order to HPC software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–12).

28. Edwards, H.C. and Trott, C.R., 2013, August. Kokkos: Enabling performance portability across manycore architectures. In *2013 Extreme Scaling Workshop* (xsw 2013) (pp. 18–24). IEEE.

29. www.llnl.gov. (n.d.). LLNL-led team awarded Best Paper at SC19 for modeling cancer-causing protein interactions | Lawrence Livermore National Laboratory. [online] Available at: https://www.llnl.gov/news/llnl-led-team-awarded-best-paper-sc19-modeling-cancer-causing-protein-interactions [Accessed 25 May 2023].

30. GitHub. (2023). skiboot. [online] Available at: https://github.com/open-power/skiboot [Accessed 25 May 2023].

31. Ellsworth, D.A., Malony, A.D., Rountree, B. and Schulz, M., 2015, November. Dynamic power sharing for higher job throughput. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–11).

32. GitHub. (2023). LLNL/ddcMD. [online] Available at: https://github.com/LLNL/ddcMD [Accessed 25 May 2023].

33. Bailey, P.E., Marathe, A., Lowenthal, D.K., Rountree, B. and Schulz, M., 2015, November. Finding the limits of power-constrained application performance. In P*roceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–12).

34. Rountree, B., Lownenthal, D.K., De Supinski, B.R., Schulz, M., Freeh, V.W. and Bletsch, T., 2009, June. Adagio: making DVS practical for complex HPC applications. In *Proceedings of the 23rd international conference on Supercomputing* (pp. 460–469).

35. Karlin, I., 2012. Lulesh programming model and performance ports overview (No. LLNL-TR-608824). Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

36. Dongarra, J.J., Luszczek, P. and Petitet, A., 2003. The LINPACK benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9), pp.803–820.

37. Patki, T., Lowenthal, D.K., Rountree, B., Schulz, M. and De Supinski, B.R., 2013, June. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th international ACM conference on International conference on supercomputing* (pp. 173–182).

38. Patki, T., Lowenthal, D.K., Sasidharan, A., Maiterth, M., Rountree, B.L., Schulz, M. and De Supinski, B.R., 2015, June. Practical resource management in power-constrained, high performance computing. In *Proceedings of the 24th international symposium on high-performance parallel and distributed computing* (pp. 121–132).

39. Gholkar, N., Mueller, F. and Rountree, B., 2016, September. Power tuning HPC jobs on power-constrained systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation* (pp. 179–191).

40. Sarood, O., Langer, A., Gupta, A. and Kale, L., 2014, November. Maximizing throughput of overprovisioned hpc data centers under a strict power budget. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 807–818). IEEE.

41. Essen, B.V., Jacobs, S., Kim, H., Dryden, N. and Moon, T., 2016. Livermore Big Artificial Neural Network Toolkit (No. LBANN V. 0.9; 004857MLTPL00). Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

42. top500.org. (n.d.). Home - | TOP500. [online] Available at: https://top500.org [Accessed 25 May 2023].

43. Marathe, A., Zhang, Y., Blanks, G., Kumbhare, N., Abdulla, G. and Rountree, B., 2017, November. An empirical survey of performance and energy efficiency variation on intel processors. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing* (pp. 1–8).

44. Inadomi, Y., Patki, T., Inoue, K., Aoyagi, M., Rountree, B., Schulz, M., Lowenthal, D., Wada, Y., Fukazawa, K., Ueda, M. and Kondo, M., 2015, November. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1–12).

45. Rountree, B., Ahn, D.H., De Supinski, B.R., Lowenthal, D.K. and Schulz, M., 2012, May. Beyond DVFS: A first look at performance under a hardware-enforced power bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum* (pp. 947–953). IEEE.

46. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S. and Simon, H.D., 1991. The NAS parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3), pp.63–73.

47. Amazon (2022). The Climate Pledge. [online] About Amazon. Available at: https://www.aboutamazon.com/planet/climate-pledge.

48. Meta Sustainability. (n.d.). Data Centers. [online] Available at: https://sustainability.fb.com/data-centers/.

49. Walsh, N. (2021). Supporting our customers on the path to net zero: The Microsoft cloud and decarbonization. [online] The Official Microsoft Blog. Available at: https://blogs.microsoft.com/blog/2021/10/27/supporting-our-customers-on-the-path-to-net-zero-the-microsoft-cloud-and-decarbonization/.

50. carbon-economy.llnl.gov. (n.d.). Engineering the Carbon Economy. [online] Available at: https://carbon-economy.llnl.gov [Accessed 25 May 2023].

51. Radovanović, A., Koningstein, R., Schneider, I., Chen, B., Duarte, A., Roy, B., Xiao, D., Haridasan, M., Hung, P., Care, N. and Talukdar, S., 2022. Carbon-aware computing for datacenters. IEEE Transactions on Power Systems, 38(2), pp.1270–1280.

52. Barroso, L.A., Hölzle, U. and Ranganathan, P., 2019. *The datacenter as a computer: Designing warehouse-scale machines* (p. 189). Springer Nature.

53. Andrae, A.S. and Edler, T., 2015. *On global electricity usage of communication technology: trends to 2030*. Challenges, 6(1), pp.117–157.

54. Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S. and Namyst, R., 2010, February. hwloc: A generic framework for managing hardware affinities in HPC applications. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing* (pp. 180–186). IEEE.

55. Cohen, W. (2023). wcohen/libpfm4. [online] GitHub. Available at: https://github.com/wcohen/libpfm4 [Accessed 25 May 2023].

56. geopm.github.io. (n.d.). geopm::PlatformIO(3) -- GEOPM platform interface — GEOPM documentation. [online] Available at: https://geopm.github.io/GEOPM_CXX_MAN_PlatformIO.3.html [Accessed 25 May 2023].

6. ADDITIONAL SUPPORTING INFORMATION

Logo:

Promo video: https://youtu.be/rgJGgPERBao

Documentation (up to three):

- **Website:** https://computing.llnl.gov/projects/variorum
- **User documentation**: https://variorum.readthedocs.io/en/latest/
- **Open-source code:** https://github.com/LLNL/variorum

Support letters are combined in a separate PDF file:

- **Brad McCredie,** Advanced Micro Devices, Inc.
- **Michael A. Heroux,** U.S. DOE Exascale Computing Project
- **Natalie Bates,** Energy Efficient HPC Working Group
- **Martin Schulz,** Leibniz Supercomputing Centre

7. AFFIRMATION

I/we certify that all of the information within this submission entry is accurate and represents the most up-to-date information available for this entry.

Tapasya Patki, LLNL May 30, 2023

8. CONTACTS

Principal investigator from each of the submitting organizations:

Tapasya Patki, *Computer Scientist*
Lawrence Livermore National Laboratory
925.423.3632
patki1@llnl.gov

Development team:

Stephanie Brink, *Computer Scientist*
Lawrence Livermore National Laboratory
925.423.6131
brink2@llnl.gov

Eric Green, *Computer Scientist*
Lawrence Livermore National Laboratory
925.424.6975
green77@llnl.gov

Aniruddha Marathe, *Computer* Scientist
Lawrence Livermore National Laboratory
925.422.1909
marathe1@llnl.gov

Barry Rountree, *Computer Scientist*
Lawrence Livermore National Laboratory
925.422.3520
rountree4@llnl.gov

Kathleen Shoga, *Computer Scientist*
Lawrence Livermore National Laboratory
925.422.5850
shoga1@llnl.gov

Media or marketing contact:

Mary Holden-Sanchez, *Business Development and Marketing Associate*
Lawrence Livermore National Laboratory
925.422.4614
holdensanchez2@llnl.gov

Person who will handle R&D 100 Awards Event arrangements:

Tapasya Patki, *Computer Scientist*
Lawrence Livermore National Laboratory
925.423.3632
patki1@llnl.gov

Organization's LinkedIn profile URL: *http://www.linkedin.com/company/4994?trk=tyah*

Organization's Twitter handle: *https://twitter.com/Livermore_Lab and https://twitter.com/Livermore_Comp*

Organization's Facebook page URL: *https://www.facebook.com/livermore.lab*

Additional social media URLs for your organization: *https://www.instagram.com/livermore_lab/*