Power API: Standardizing Power Monitoring and Control at Exascale

Ryan E. Grant, Michael Levenhagen, Stephen L. Olivier, David DeBonis, Kevin T. Pedretti, James H. Laros III
Center for Computational Research
Sandia National Laboratories*
P.O. Box 5800, MS-1110
Albuquerque, NM 87185-1110
Email: regrant,mjleven,slolivi,ddeboni,ktpedre,jhlaros@sandia.gov

***Abstract*—As power and energy become dominant constraints on large-scale platforms, software support is necessary to measure and control energy and power usage. Current generation systems rely on vendor-specific measurement and control interfaces, making portable power-aware computing techniques a major concern for future systems. To address this portability challenge, Sandia National Laboratories has collaborated with other laboratories, universities and major vendors including Intel, IBM, AMD, Cray, HP, Adaptive Computing, and Penguin Computing to develop a Power API for High Performance Computing. The API standardizes measurement and control of power and energy for large-scale systems. It provides several interfaces targeting diverse requirements, from application-level fine-grained control and measurement to facility-level accounting. It enables high-frequency measurement and exposes valuable metadata to assess the utility of the observed values. The API also includes a rich statistics-gathering interface that scales from individual component-level measurements up to and including whole-system statistics in both real-time (active measurements) and historical (database logging) contexts. This article details the design of the Power API and presents a case study using an implementation of the API.**

I. Introduction

Exascale computing denotes the use of supercomputer platforms capable of operating at speeds exceeding 1 ExaFLOP/s. This goal requires very large clusters of high performance compute nodes combined with the fastest network connections available. Unlike typical large commercial data centers, Exascale scientific computing requires that only a few very large jobs (potentially a single job) occupy the entire system at any one time. Compared to systems designed to support many small jobs, managing the overall power consumption of the system can be more difficult since jobs and resources cannot be reallocated on a fine-grained basis. These leadership class systems, whether deployed for open science or classified processing, will soon require tens of megawatts of power. Power has become a primary design constraint pushing the limits of commercial power delivery and greatly increasing the cost of facility infrastructure to support these platforms. The cost of powering these platforms over their typical useful lifetime, three to five years, may soon rival their acquisition cost.

There are various power measurement and control devices available today. The most ubiquitous are those included on modern CPUs, namely Intel's running average power limit (RAPL) controls and AMD's similar TDP Power Cap. These mechanisms provide capabilities to control power usage as well as measure power and energy consumption on the CPU. These mechanisms are accessed through machine specific registers (MSRs) on the chip. As these MSRs are protected resources in Linux, methods for reading and writing to them are necessary to enable user-level measurement and control. Some vendors, such as Cray, provide proprietary solutions, while open-source solutions such as libMSRsafe [1] are also available. Less common, but more encompassing solutions are node-level devices that measure power using expansion cards or internally resident stand-alone measurement devices. Examples include PowerInsight [2], Wattprof [3] and PowerMon [4]. These devices collect telemetry out of band, typically employing in-line hall effect chips and/or shunt resistors. They often have the capability of measuring more than just CPU power, sometimes

*Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

providing whole node and individual component level measurements. The methods by which they interface with the node vary. PowerInsight can communicate by USB port or through Ethernet connections. WattProf uses the PCIe bus and future versions will offer Ethernet support. PowerMon works through a USB interface. All three devices offer sampling rates in the hundreds to thousands of Hertz range. Stand alone external devices are also common for measuring power and energy, such as WattsUp devices. They are installed in between the wall plate and the node power supply external to the machine. Like some of the internal devices, WattsUp meters use a USB interface, with the possibility of using Ethernet as well. Other manufacturer-specific solutions are also available, such as iLO from HP.

Along with the increasing number of measurement and control mechanisms available come an equally diverse and numerous set of interfaces. These interfaces can vary greatly from one manufacturer to another, and as such much of the power control software in use today is vendor or device specific. The development of tools for specific platforms is costly and inefficient. A single power measurement and control API for systems is highly desirable to reduce investment costs in tools and power-aware runtimes for each successive machine generation. The US Department of Energy (DOE) and similar agencies throughout the world are major consumers of supercomputer platforms of a variety of architectures from many vendors. A single standard API for controlling power consumption and collecting power measurements on these different systems is essential to ensure that future supercomputers provide the mechanisms needed to control power, for example, to avoid breaching contracts with power utility providers and exceeding physical power limits onsite.

The Power API [5] provides a common, cross-vendor interface to measure and control the power usage of hardware, including support for many of the existing solutions on the market. The overall scope of the Power API is broad, encompassing API function calls, with interfaces for applications to measure and react to their own execution, high level interfaces to support tools for administrative level whole system accounting tasks, and scripting interfaces for system administrators to enact quick custom power control. The Power API solves many of the issues that exist in in today's power measurement and control system environment and facilitates evolving requirements. However, providing a portable interface alone is not enough to provide a truly useful power measurement and control interface. A key element to measurement is understanding the accuracy and frequency of the measurements. Power measurement can have multiple different layers of measurements, with the highest level measurements exposed to the user. For example, a measurement may be taken at 10 kHz at a low-level hardware sampling, but the aggregated average power may be exposed only at a rate of 1 Hz. As such, a measurement sampled at 1 Hz may not be fast enough to capture shorter duration power fluctuations that are important to the real power usage of the system, but if the measurements are the average of 10 kHz samples the resulting measurements may be accurate enough for some use cases. The Power API has associated metadata with each measurement point to inform the user about the underlying sampling methodology and accuracy. This allows the user to determine whether the measurement capability can be used effectively for their particular use case.

II. The Power API

The *"High Performance Computing - Power Application Programming Interface Specification"* [5] was developed at Sandia National Laboratories in collaboration with major vendors, laboratory and academic partners. The organizational structure behind the API's development follows those of several other very successful standards, with a vendor-neutral national laboratory funded through the federal government of the United States leading an effort that has community input and public review feedback with the goal of becoming a public community-led standard. The goal of developing a common API for power measurement and control was realized with the first specification release in

2014. The API continues to evolve and grow as further capabilities are added and new language bindings are supported. While most existing interfaces are mature, work continues on some of the highest level interfaces as research and the development of future systems better informs the high level reporting requirements. The Power API is specified primarily as a C API, as C is the preferred language for low-level software on HPC systems and is universally supported, however, alternative language bindings such as Python are provided. Implementations of the Power API may internally use whatever language is most convenient. For example the Power API reference implementation developed by Sandia is primarily written in C++ with user-visible C interfaces provided externally.

A. High-Level Design

The design philosophy behind the Power API is to allow for flexibility in future system architectures and power measurement and control capabilities. As such, it is designed to allow great freedom in describing system architectures and handling requests to many devices, including requests for information or capabilities that may not be supported in today's systems but are expected to be available in future Exascale class machines.

The Power API creates a system description in a hierarchical form, with basic supported, but not mandatory, objects starting with a platform and descending through the system to cabinets, boards, nodes, sockets, and core object types. Additional devices can be inserted where applicable in the system description, including memories, network interfaces, accelerators and more generic power plane objects. Power planes provide a useful control/measurement point for cases where power measurements or controls are aggregated amongst underlying objects. An example of this would be a power plane for a CPU where the individual cores do not have individual measurements available, but an aggregate measurement is available (two cores per power plane, for example. This hierarchical form can be expressed statically or can be built dynamically through a system description tool. The design of the hierarchy allows for current tools such as hwloc [6] to accurately describe current systems while still allowing for possible future system architecture changes that depart radically from contemporary systems. An example of this hierarchy in Figure 1 shows a simple small scale system using core Power API object types.

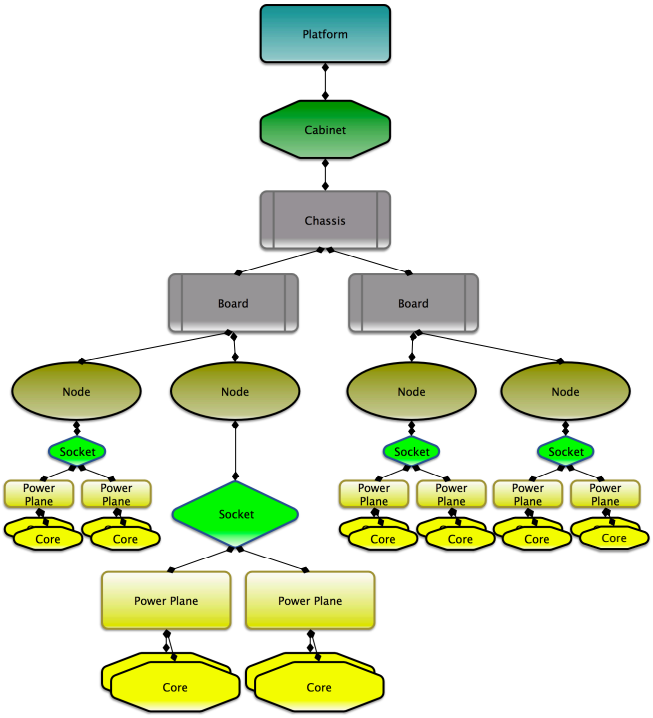

Fig. 1. Example of a simple machine hierarchy for the Power API.

User interaction with the API follows the philosophy of making hard tasks possible to accomplish and easy tasks easy to accomplish. The Power API design also chooses potential complexity in implementation rather than in the user interface whenever possible. The rationale behind this approach is that the implementation can better deal with complexity once, where experts can be utilized more easily, and avoid complexity in the code that will be written more often, the Power API calls themselves at the user level.

B. Roles

The roles that users of the Power API can assume best illustrate the encompassing nature of the API.

A diagram showing all of the roles and how they interact with different levels of the interface is shown in Figure 2. In the figure, role names are often proceeded with "HPCS" to indicate the primary focus of the specification, High Performance Computing Systems. One of the high level roles, Accounting, provides an interface for generating reports and metrics of the system at different levels of granularity, from the whole system down to individual components. The System Manager role is provided to represent the responsibility of dictating overall system-level policies, such as scheduling priorities and facility limitations. The Administrator role represents the traditional IT system administrator function, managing day to day operation of the system, but from a power and energy perspective. This interface is aimed at providing easy access to control power throughout the system on both a coarse and fine-grained basis as well as providing useful information on power measurements to better understand immediate and long term system needs. The administrator can choose between C and Python interfaces for these tasks, where Python scripts are desirable for quick unique scripting requirements, the C interface is useful for building command line tools requiring high-performance for frequently used operations. The Resource Manager role is oriented toward resource managers and job schedulers. Policy decisions communicated by the System Manager are translated into job policy on the running system, such as power caps that represent time of day differences in power costs. Interfaces are available for the Resource Manager to mine information or leverage information provided by the system from the Monitor and Control role (for example). The next role is the generic User role. This interface provides all of the capabilities potentially exposed to end-users of an HPC system, primarily taking measurements and potentially controlling power within bounds enforced by the system administrators or resource manager. The Application role is the first-person interface for user applications running on the system. In many ways this is similar to the User role, but with lower-level requirements where necessary for describing the needs of HPC applications. The last two roles directly interact with hardware and expose the fundamental measurement and control capabilities of the system. While user-space level hardware interaction may be possible on some systems and therefore enable other roles to interact with hardware directly, the Operating System (OS) and Monitor and Control roles are required to interact with hardware on all systems. This is due to the high level of privileges required to interact with most hardware. The OS role is primarily a node centric role while the Monitor and Control role is a broader focused system level management role. The Monitor and Control role is largely analogous to traditional Reliability Availability and Serviceability (RAS) systems.

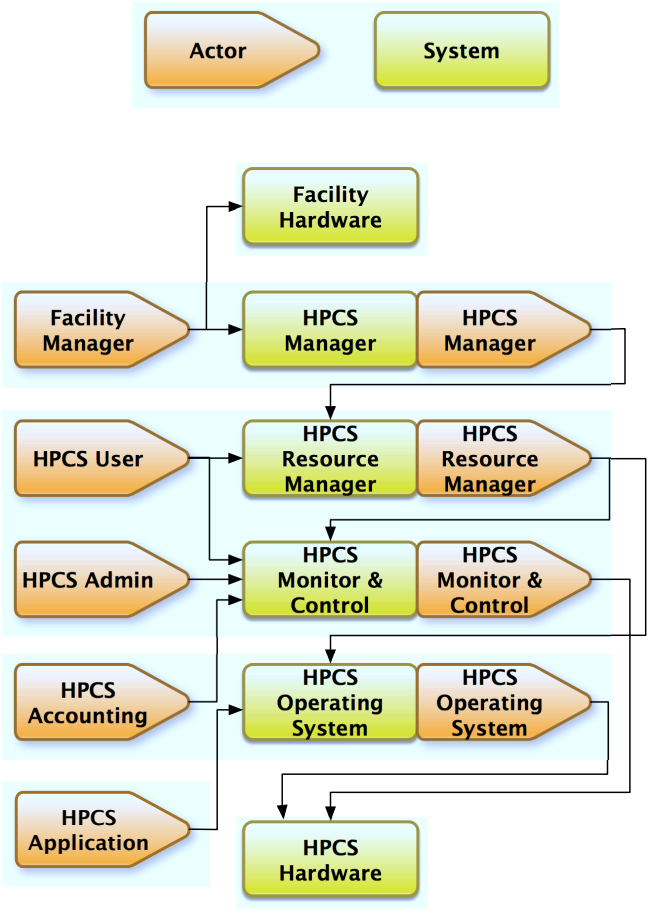

Fig. 2. Top Level Conceptual Diagram representing the interaction of roles with different levels of the Power API interfaces.

C. Using the Power API

The Power API provides many core functions shared by the different interfaces offered. Upon initialization, the user is presented with a context, basically the user's window into the functionality available to their role/user combination by the implementation. The system view exposed likewise depends on the combination of the role and the individual user. For example, an application may only have access to the hardware (the node) that it is currently executing on. A system administrator would commonly have access to all platform resources. Navigation functions allow any user (or role) to navigate to the device (object) in the system hierarchy with which the user seeks to interact. The API provides functions for creating groups of objects, which can then operated upon using group functions that mirror the capabilities of functions used to interact with individual system objects. Groups can also be combined using several different functions to create unions or intersections and differences of the two groups.

Each object in the system hierarchy has attributes associated with it, which correspond to measurement or control interfaces available, and exposed, for that individual object. For example, for a CPU core object, valid attributes may include power, energy, performance state, sleep state and low level measurements such as voltage and current.

The metadata about object-attribute pairs can be easily fetched using the Power API metadata interface. Metadata is particularly important for determining the utility of data obtained using the Power API interfaces, such as the frequency or accuracy of measured values.

Figure 3 demonstrates an example of using the Power API metadata and attribute interfaces. After initializing a Power API context, the `PWR_CntxtGetEntryPoint()` interface is used to get the object representing the caller's entry point for navigating the machine hierarchy. In the interest of space, this example assumes the entry point returned is the local node's object but in general the Power API's navigation interfaces would be used to find the desired object. Next, the `PWR_ObjAttrGetMeta()` metadata interface is used to retrieve the expected accuracy of energy measurements obtained from the local node's `PWR_ATTR_ENERGY` attribute. Finally, the `PWR_ObjAttrGetValue()` attribute interface is used to measure the energy consumed by the `do_work()` function. Since `PWR_ATTR_ENERGY` is an energy counter, the difference of its value between calls is used to calculate the energy consumed.

```
PWR_Cntxt context;
PWR_Obj   my_node;
PWR_Time  timestamp1, timestamp2;
double    energy1, energy2, accuracy;

// Initialize and get my node object
PWR_CntxtInit(PWR_CNTXT_DEFAULT,
    PWR_ROLE_APP, "MyContext", &context);
PWR_CntxtGetEntryPoint(context, &my_node);

// Get accuracy of energy counter for my node
PWR_ObjAttrGetMeta(my_node, PWR_ATTR_ENERGY,
    PWR_MD_ACCURACY, &accuracy);

printf("Accuracy +/- %f percent\n", accuracy);

// Measure energy consumed by do_work()
PWR_ObjAttrGetValue(my_node, PWR_ATTR_ENERGY,
    &energy1, &timestamp1);
do_work();
PWR_ObjAttrGetValue(my_node, PWR_ATTR_ENERGY,
    &energy2, &timestamp2);

printf("do_work() consumed %f J in %f ns\n",
    energy2-energy1, timestamp2-timestamp1);
```

Fig. 3. Example of using Power API to measure energy usage of a function.

Another powerful use case for the Power API is the collection of statistics. The API provides a statistics interface that allows the user to gather statistics on individual objects or groups of objects for individual attributes. These statistics, such as sum, max, min, and average, can then be further reduced if desired to provide averages of sums on multiple objects or find a maximum of maximums and the object that it occurred on.

High-level application interfaces are provided to allow the application to communicate to the system (the OS or potentially an intelligent run-time layer). These "hints" include informing the system about application phases such as serial or parallel regions

that can be exploited at the node level to potentially deliver more performance and power savings. The application could also hint that it is in a communication phase on a particular node which would allow node level alterations but also allow an intelligent runtime system to coordinate between the nodes allocated to shift additional power to nodes which remain in computation phases, for example.

D. Implementation

While commercial vendor implementations of the Power API are in development, an open source reference implementation is available for early adopters. The Power API reference implementation is architected to support the core functions of the API in a single implementation, with multiple measurement device support implemented through a plugin architecture. This allows for rapid integration of new measurement devices as well as power control points. The current implementation supports many low-level hardware power measurement devices, from common off-the-shelf solutions such as WattsUp meters, to device/vendor specific methods such as Intel's RAPL. Support for more comprehensive out-of-band power measurement devices, such as PowerInsight from Penguin Computing, is also provided.

The reference implementation is mostly complete. Core functions, aside from historical data collection and a subset of statistics functions for certain objects, are implemented. The reference implementation is currently integrating and optimizing large-scale collection methods. The current functionality in the reference implementation is sufficient for most real-time data measurement and control use cases. The reference implementation is currently deployed as part of the Tri-lab operating system (TOSS) and is running on several test and production platforms at DOE laboratories. Reference implementation development and research is conducted at small scale on many of Sandia's Advanced Architecture Test Bed clusters [7]. Large scale testing and research is being accomplished on the production Skybridge cluster at Sandia National Laboratories.

The reference implementation incorporates a scalable framework for collecting measurements from many different objects in a group at one time. Although aggregation of results is implicitly embedded in the object hierarchy of the Power API, distributing the aggregation at multiple points instead of a single aggregation point is an implementation optimization. The scalable distributed aggregation method for the implementation has shown good results at this early stage, before significant performance optimization has been completed. Figure 4 shows the initial scaling of collecting basic energy samples from a number of nodes in a large system. The microbenchmark used for the results in Figure 4 measures the time that 1000 `PWR_ObjAttrGetValue()` requests take to complete and divides by 1000. The test was performed on Chama, a production supercomputer at Sandia National Laboratories.

The Power API Reference Implementation was developed alongside the specification and is publicly available at http://powerapi.sandia.gov.

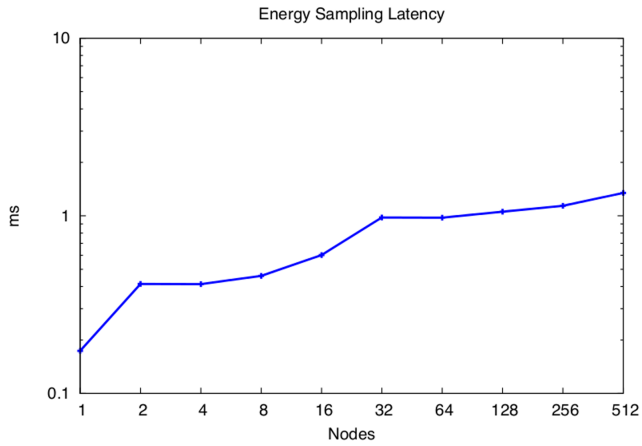

Fig. 4. Power API Energy data collection latency using multiple nodes.

E. Enabling Understanding

We have already put the Power API to use in studies focusing on understanding the power consumption of systems as well as understanding the effectiveness of power control methods on platforms.

Power management solutions are increasingly available on large systems. One of the newest control mechanisms on Cray Inc. supercomputers is node-level power capping. Understanding how

the power capping mechanism impacts performance and how much power is consumed by state-of-the-art production applications during execution are important questions to answer. The Power API is currently installed on a small Cray XC40 system at Sandia National Laboratories that has the new node-level power capping capability. Testing the power capping mechanism on this system with CTH, a widely-used solid mechanics application, has shown the distribution of power samples under different power caps. Figure 5 shows the cumulative distribution function for power samples for several different node-level power caps for 96 nodes on the test system. The Power API data show that the power capping mechanism allows limited time periods where the power draw can exceed the power cap. These measurements reveal the consequences of the power cap mechanism's enforcement that is based only on an average of samples in a given time window [8]. The Power API has enabled collection of all measurements on this system and enables portable testing on other systems and with other applications.

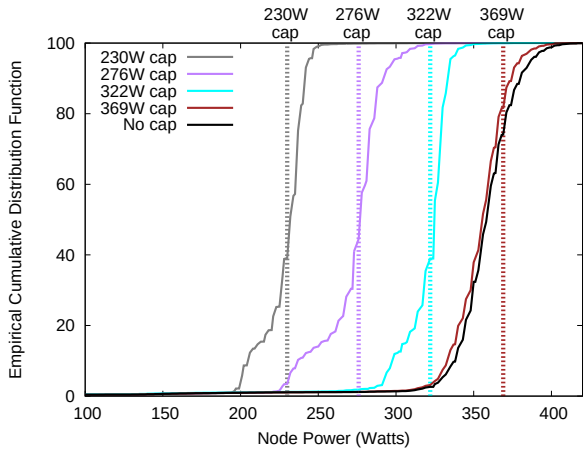

Fig. 5. Power API power measurements used to understand node-level power capping behavior with production application, CTH. CTH is a strong shock wave, multi-material solid mechanics code

III. Related Work

Scalable collection mechanisms have been developed for a variety of use cases for other metrics. The Lightweight Distributed Metric Service (LDMS) [9] is a data collection and aggregation system used in large systems. It works on a variety of hardware including Cray systems and is complimentary to the Power API as it collects data on a configurable set of diverse metrics. Overlay networks are well-adopted methods of collecting and aggregating data on large systems. Approaches like MRNet [10] introduce a second virtual network hierarchy on top of existing networks that can be used for collection aggregation purposes.

Many power measurement solutions (e.g., PowerInsight [2], WattProf [11], Powermon [4], Powerpack [12]) include APIs that are specific to their particular measurement solution. Power control APIs have been developed that are specific to particular systems and used as part of energy-saving techniques like Oscar API [13].

Global Energy Optimization (GEO) is a energy optimization framework developed by Intel [14]. It manages job power bounds in a cluster while also attempting to increase performance by tuning the power consumption of systems involved in a job. GEO has a scalable collection mechanism that is based on MPI communication for individual job measurement collection. Unlike the Power API, GEO's external interfaces are not proposed as a standard [14], though they are open-source. Like the Power API it can work in a distributed manner [15].

IV. Expanding Adoption

As part of a collaboration with Sandia and Los Alamos national laboratories, Cray is implementing portions of the Power API specification for deployment on the Trinity supercomputer, ranked #7 on the June 2016 TOP500 list [16]. Additionally, these partners are working with Adaptive Computing, makers of the Moab/Torque resource manager, to enable intelligent power-aware job scheduling decisions. These capabilities will leverage the Power API implementation available on Trinity to demonstrate that power budgets of large-scale supercomputers can be effectively managed. We intend to explore power prediction on large systems such that power schedules can be created and relayed to the

power utility provider to anticipate demand from such systems and therefore be proactive in power generation based on short term future needs. These efforts are intended to be forward thinking and benefit future systems both within the DOE national laboratories and the wider HPC community.

Many vendor partners have been involved with the Power API specification and many plugins for the reference implementation have been completed for a variety of hardware. Current support is available for Intel and AMD CPUs, Power Insight, Watt Prof, WattsUp, Power Gadget, generic CPU registers, and Cray's XTPM measurement devices. We intend to further expand this list of supported devices in collaboration with vendors of power measurement capable computing components.

Like any proposed standard, the Power API depends on community interest to drive adoption and implementation by HPC technology providers. The Power API specification is presented as a starting point. As this field evolves, the Power API specification must necessarily evolve to include support for future capabilities that will allow the entire community to field HPC platforms in power and energy constrained environments.

REFERENCES

[1] K. Shoga, B. Rountree, M. Schulz, and J. Shafer, "Whitelisting MSRs with msr-safe," in *3rd Workshop on Extreme-Scale Programming Tools*, 2014.

[2] J. H. Laros, P. Pokorny, and D. DeBonis, "PowerInsight - a commodity power measurement capability," in *2013 International Green Computing Conference (IGCC)*. IEEE, 2013, pp. 1–6.

[3] M. Rashti, G. Sabin, D. Vansickle, and B. Norris, "WattProf: A flexible platform for fine-grained HPC power profiling," in *2015 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2015, pp. 698–705.

[4] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "Powermon: Fine-grained and integrated power monitoring for commodity computer systems," in *Proceedings of the IEEE Region 3 Southeast Conference 2010 (SoutheastCon)*. IEEE, 2010, pp. 479–484.

[5] J. H. Laros, D. Debonis, R. E. Grant, S. M. Kelly, M. Levenhagen, S. Olivier, and K. T. Pedretti, "High performance computing power application programming interface specification version 1.1," 2015. [Online]. Available: http://powerapi.sandia.gov

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

[7] "Sandia national laboratories advanced architecture test beds." [Online]. Available: http://www.sandia.gov/asc/computational_systems/HAAPS.html

[8] K. Pedretti, S. L. Olivier, K. B. Ferreira, G. Shipman, and W. Shu, "Early experiences with node-level power capping on the Cray XC40 platform," in *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*. ACM, 2015, p. 1.

[9] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden *et al.*, "The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 154–165.

[10] P. C. Roth, D. C. Arnold, and B. P. Miller, "MRNet: A software-based multicast/reduction network for scalable tools," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. ACM, 2003, p. 21.

[11] M. Rashti, G. Sabin, D. Vansickle, and B. Norris, "WattProf: A flexible platform for fine-grained HPC power profiling," in *International Conference on Cluster Computing*. IEEE, 2015, pp. 698–705.

[12] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, 2010.

[13] K. Kimura, M. Mase, H. Mikami, T. Miyamoto, J. Shirako, and H. Kasahara, "Oscar API for real-time low-power multicores and its performance on multicores and SMP servers," in *International Workshop on Languages and Compilers for Parallel Computing*. Springer, 2009, pp. 188–202.

[14] J. Eastep, "An overview of GEO (global energy optimization)," 2015. [Online]. Available: https://eehpcwg.llnl.gov/documents/webinars/systems/120915_eastep-geo.pdf

[15] R. E. Grant, M. Levehagen, S. Olivier, D. DeBonis, K. Pedretti, and J. H. Laros, "Overcoming challenges in scalable power monitoring with the power api," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, Workshop on High-Performance Power-Aware Computing (HPPAC)*. IEEE, 2016.

[16] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, "Top500 supercomputing sites," 2016. [Online]. Available: http://www.top500.org/

Ryan E. Grant is a Senior Member of Technical Staff in the Center for Computing Research at Sandia National Laboratories in Albuqueruque NM USA. He holds a PhD in computer engineering from Queen's University at Kingston, Ontario, Canada.

Michael Levenhagen is a Senior Member of Technical Staff in the Center for Computing Research at Sandia National Laboratories in Albuqueruque NM USA.

Stephen L. Olivier is a Senior Member of Technical Staff in the Center for Computing Research at Sandia National Laboratories in Albuqueruque NM USA. He holds a PhD in computer science from the University of North Carolina at Chapel Hill.

David DeBonis is a HP contractor at the Center for Computing Research at Sandia National Laboratories in Albuqueruque NM USA. He is a PhD student at the Univeristy of New Mexico.

Kevin T. Pedretti is a Prinicipal Member of Technical Staff in the Center for Computing Research at

Sandia National Laboratories in Albuqueruque NM USA. He is a co-project lead for the Power API.

James H. Laros III is a Prinicipal Member of Technical Staff in the Center for Computing Research at Sandia National Laboratories in Albuqueruque NM USA. He is a co-project lead for the Power API.