Standardizing Power Monitoring and Control at Exascale

Ryan E. Grant, Michael Levenhagen, Stephen L. Olivier, David DeBonis, Kevin T. Pedretti, and James H. Laros III, Sandia National Laboratories

Power API—the result of collaboration among national laboratories, universities, and major vendors—provides a range of standardized power management functions, from application-level control and measurement to facility-level accounting, including real-time and historical statistics gathering. Support is already available for Intel and AMD CPUs and standalone measurement devices.

In exascale computing, which is likely to arrive within the next decade, supercomputer platforms will operate at speeds exceeding 10^{18} floating-point operations per second, or 1 exaflops, and require very large clusters of high-performance computational nodes combined with the fastest network connections available. Unlike large commercial datacenters, exascale systems—which are primarily scientific in nature—are expected to run only a few very large jobs at a time and sometimes only one job.

Because of this limitation, obviously, job and resource reallocation cannot be fine-tuned. This characteristic plus exascale computing's power requirement of tens of megawatts makes it much harder to manage power consumption, relative to systems that support many small jobs. In fact, 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 useful lifetime—usually three to five years—might soon rival their acquisition cost.

As power and energy become dominant constraints, vendors have developed a variety of packages to measure and control energy and power use. However, these mechanisms come with widely varying capabilities and interfaces, making the development of supporting tools costly and inefficient. In contrast, a single power measurement and control API for exascale systems can reduce investment costs in tools and power-aware runtime system software for each successive machine generation. The US Department of Energy (DOE) and similar agencies throughout the world are the major users

 0018-9162/16/$33.00 © 2016 IEEE

of supercomputer platforms, which reflect a wide 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 on-site physical power limits.

To address the challenge of developing a standard API, in 2014, Sandia National Laboratories entered into a collaboration with other laboratories, universities, and major vendors—including Intel, IBM, AMD, Cray, Hewlett Packard, Adaptive Computing, and Penguin Computing—to develop a power API for high-performance computing (HPC) systems. The result is Power API, an interface for standardizing power and energy measurement and control for large-scale and eventually exascale systems, which are expected to emerge in the 2020–2024 timeframe. Power API, which is also suitable for smaller-scale systems, from commercial datacenters to individual desktops, has several key features:

- It targets 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 observed values.
- It 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.

To better illustrate Power API's functionality, we present its design details and sample implementations.

EXISTING SOLUTIONS

A variety of power measurement and control devices are available, the most ubiquitous of which are included as part of Intel's running average power limit (RAPL) controls and AMD's TDP Power Cap. Both these mechanisms are accessed through on-chip machine-specific registers that are protected resources in Linux. Consequently, methods for reading and writing to them are required for user-level measurement and control.

Measurement devices

Some vendors, such as Cray, provide proprietary solutions, and open source solutions, such as libMSRsafe,[1] are also available. Less common are node-level devices that use expansion cards or internally resident standalone mechanisms to measure power. These devices, including Penguin Computing's PowerInsight,[2] RNet's WattProf,[3] and RENCI's PowerMon,[4] typically use in-line hall-effect chips or shunt resistors to collect telemetry out of band. They often measure more than CPU power, sometimes providing measurements for the whole node or individual components. Interfaces for node communication employ different mechanisms. PowerInsight uses a USB port or Ethernet connections, while WattProf communicates through a Peripheral Component Interconnect Express (PCIe) bus, with future versions aiming to provide Ethernet support. PowerMon works through a USB interface. All three devices can take samples at rates in the hundreds to thousands of hertz.

Standalone external devices, such as WattsUp, are also popular for measuring power and energy. WattsUp meters, installed between the wall plate and external node power supply, now use a USB interface, or a webserver interface through Ethernet. Other manufacturer-specific solutions are also available, such as iLO from Hewlett Packard.

Measurement data collection and aggregation

Scalable measurement-collection mechanisms have been developed for metrics that are not power based, such as performance, which can complement Power API. The Lightweight Distributed Metric Service (LDMS)[5] collects and aggregates data for large systems that works on a variety of hardware, including Cray systems, collecting data on a configurable set of diverse metrics such as performance and system events.

Frameworks also enable power measurement, and overlay networks are well-established methods of collecting and aggregating data on large systems. Approaches like MRNet[6] introduce a second virtual network hierarchy on top of existing networks, which can be used for collection aggregation. Power-control APIs such as Oscar,[7] use compiler and runtime commands to control power on node as an energy-saving technique. Frameworks such as PowerPack[8] provide APIs to enable power measurement on data acquisition and Advanced Configuration and Power Interface (ACPI) power supplies.

Intel's Global Energy Optimization (GEO) framework[9] 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 based on MPI communication that is used to collect measurements for individual jobs. Unlike Power API, GEO's external interfaces are not a proposed standard, although they are open source. However, GEO is akin to Power API because it can work in a distributed manner.[10]

STANDARD API REQUIREMENTS

Power API provides a common, cross-vendor interface to measure and control the power use of hardware, including support for many of the tools just described and other commercially available solutions. Power API's overall scope is broad, including

- API function calls, with interfaces for applications to measure and react to their own execution;
- high-level interfaces to support tools for whole-system accounting tasks; and
- scripting interfaces for system administrators to quickly enact custom power control.

With these features, Power API solves many of the issues that exist in today's power measurement and control systems' environment and facilitates evolving requirements.[11]

However, it is not enough to make an API portable. A truly useful power measurement and control interface must be based on a deep understanding of measurement accuracy and frequency.

Measurement accuracy stems from two reporting frequencies. One is on the actual measurement hardware itself. The other is the output frequency of those measurements that is visible to the host system. Some systems gather samples at high frequency and average them to report them back to the host. Although this reporting comes with a much higher cost than the measurements themselves, the higher sample rate makes the measurements more reliable because it captures behavior in power consumption that a slower sample rate would not.

Power measurement has multiple measurement layers, with the top layers exposed to the user. For example, a device might sample the hardware and take measurements at 10 kHz, but the aggregated average power exposed to the user might be only 1 Hz. A measurement natively sampled at 1 Hz might not be fast enough to capture the shorter power fluctuations that are important to understanding the system's actual power use. However, if the measurements are the average of the 10-kHz samples collected and aggregated by the measurement hardware, the resulting measurements might be accurate enough for some use cases. To accommodate these differences, Power API has associated metadata with each measurement point to inform the user about the underlying sampling methodology and accuracy. The user can then determine whether the measurement capability can be used effectively for a particular use case.

Power API is specified primarily as a C API because 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. Power API implementations can use whatever language is most convenient; for example, Sandia's reference implementation is written primarily in C++, with user-visible C interfaces provided externally.

API DESIGN

Power API's design aims for flexibility in future system architectures and power measurement and control capabilities. As such, it allows considerable freedom in describing system architectures and handling requests to many devices, including requests for information or capabilities that might be available only in future exascale-class machines. Two of its main design characteristics are hierarchical system description and user roles that correspond to interfaces.

Hierarchical description

Power API creates a system description in hierarchical form, with basic supported objects that start with a platform and descend through the system to cabinets, boards, nodes, sockets, and core object types. Figure 1 shows a simple small-scale system using basic object types.

A particular Power API implementation does not have to include all basic objects in a system description, and it can insert additional objects as needed, including memories, network interfaces, accelerators, and more generic power-plane objects. Power planes provide a useful control and measurement point when power measurements or controls are aggregated among underlying objects. An example is a power plane for a CPU, in which individual measurements are not available for cores, but an aggregate measurement is possible, as in two cores per power plane.

Power API implementations can express this hierarchical form statically through a fixed system-description file or build it dynamically through a system

description tool. The hierarchy is designed to allow current tools such as `hwloc`[12] to accurately describe current systems but still leave room for future system architectural changes that radically depart from contemporary systems.

Power API aims to make hard tasks possible to accomplish and easy tasks straightforward to complete. Whenever possible, complexity is moved to the implementation in the interest of keeping the interface simple. The rationale is that implementations are developed by subject-area experts who can better deal with complex cases, and this specialized code needs to be written and verified only once. Complexity in code that must be written more often is therefore avoided.

User roles

As Figure 2 shows, Power API users can have many and varied roles at different levels, and each role requires multiple interfaces to interact with because system components can both issue and receive commands.

In Figure 2, the Platform Manager role reflects the responsibility of dictating overall system-level policies, such as scheduling priorities and facility limitations. The User role provides all the capabilities potentially exposed to end users; in an HPC system, for example, primary capabilities would be taking measurements and potentially controlling power within bounds enforced by system administrators or the resource manager. The Application role is the interface for user applications. It is similar to the User role, but with the ability to describe lower-level requirements.

The Administrator role represents the traditional IT system administrator function, managing day-to-day system operation, but from a power and energy perspective. This interface aims to provide 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. An administrator can choose between C and Python interfaces for these tasks; Python scripts are desirable for quick unique scripting requirements, whereas the C interface is useful for building command-line tools that require high performance in frequently used operations. The Accounting role is an interface for generating reports and system metrics at different levels of granularity, from system to individual components.

The Resource Manager role is oriented toward resource managers and job schedulers. Policy decisions

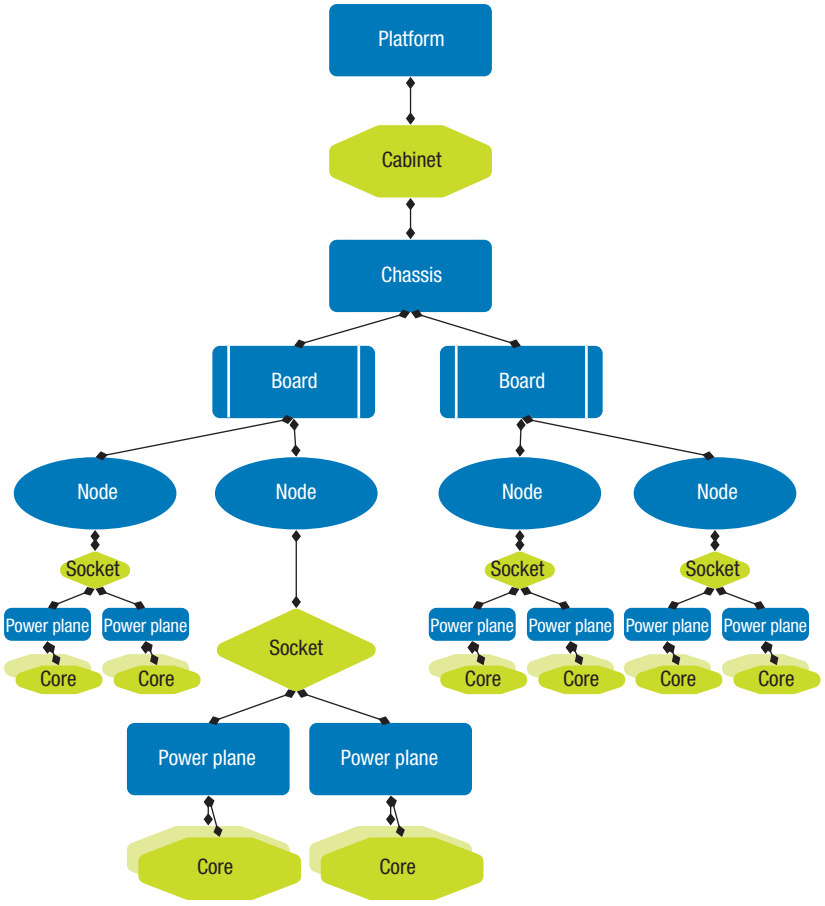

FIGURE 1. Example of a simple machine hierarchy for Power API. Power API provides a set of basic object types in a hierarchical system description. Implementations provide standard objects relevant to a particular use case and can include nonstandard objects to represent additional devices as needed.

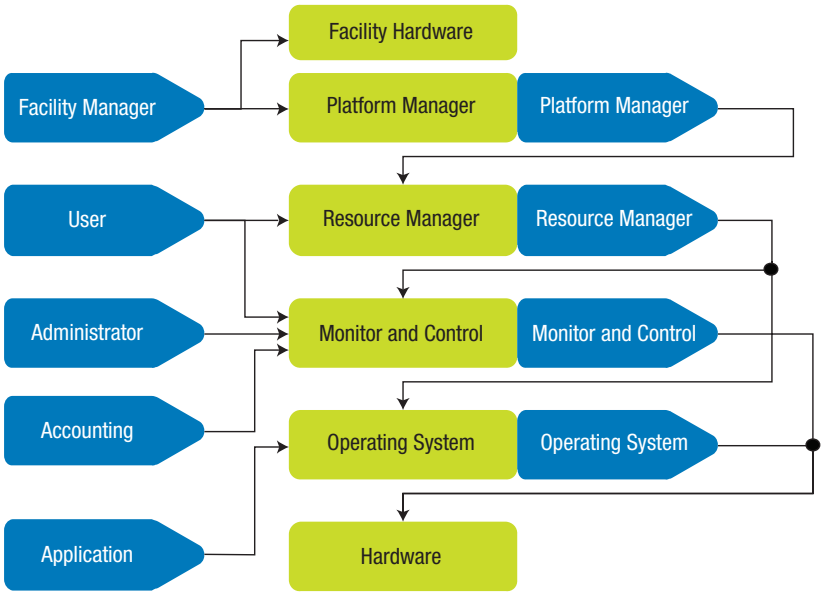

FIGURE 2. User roles in Power API. This conceptual diagram of generic user roles exposes two fundamental user types. The system (green) and actor (blue) boxes are meant to show the progression from a command's origin through the hierarchy of subsystems that the command might have to navigate to retrieve the requested data. Some systems cache data and can return it; others need to go to hardware to satisfy the request. The Facility Manager, Platform Manager, User, and Administrator roles map to humans who interact with the system and set policies. The Accounting and Application roles are computing system interfaces accessed through the API.

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 other roles, for example, the Monitor and Control role.

The last two roles directly interact with hardware and expose the system's fundamental measurement and control capabilities. Some systems allow user-level hardware interaction, which therefore enables other roles to interact with hardware directly. However, most such hardware interaction requires a high level of user privilege, so Power API provides the OS and Monitor and Control roles to interact with hardware on all systems. The OS role is primarily node centric, whereas the Monitor and Control role is more broadly focused on system-level management. The Monitor and Control role is largely analogous to traditional reliability, availability, and serviceability systems.

SAMPLE USE CASE

Power API provides many central functions that the interfaces just described share. When the API is initialized, the user is presented with a context, which is basically the user's window into the available functions specific to their implementation. The system view likewise depends on the combination of the role and individual user. For example, an application might have access only to the hardware (the node) it is currently executing on. A system administrator would commonly have access to all platform resources.

Navigation functions allow any user to navigate to a device in the system hierarchy. The API provides functions for creating object groups, which can then operate by using group functions that mirror the capabilities of functions used to interact with individual system objects. Groups can also be combined using several functions to create unions or intersections and differences of the two groups.

Each object in the system hierarchy has associated attributes that correspond to measurement or control interfaces available and exposed for that individual object. For example, in a CPU core object, valid attributes might 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 through other interfaces, such as the frequency or accuracy of measured values.

Figure 3 shows an example of using the Power API metadata and attribute interfaces. After initializing a Power API context, the `PWR_CntxtGetEntryPoint()` interface gets the object that represents the caller's entry point for navigating the machine hierarchy. For simplicity, the example assumes the entry point returned is the local node's object, but generally the Power API's navigation interfaces would find the desired object. Next, the `PWR_ObjAttrGetMeta()` metadata interface retrieves 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. Because `PWR_ATTR_ENERGY` is an energy counter, the difference of its value between calls is used to calculate the energy consumed.

Another powerful use case is the collection of statistics, which Power API supports through a statistics interface that enables the user to gather statistics on individual objects or object groups for individual attributes. If desired, statistics such as sum, max, min, and average can be further reduced to provide averages of sums on multiple objects or to find a maximum of maximums as well as the object where the maximum value occurred.

The application communicates with the system (the OS or possibly an intelligent runtime layer) through high-level interfaces. Communication might include informing the system about application phases in which serial or parallel regions can be exploited at the node level to potentially deliver more performance and power savings. The application might also reveal that it is in a communication phase on a particular node, which would enable node-level alterations and could allow an intelligent runtime system to coordinate allocated nodes, for example, to shift additional power to nodes that remain in computation phases.

```
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);
```

FIGURE 3. Using the Power API metadata and attribute interfaces. In this example, Power API is being applied to measure the energy use of the `do_work()` function. The metadata interface provides valuable information about measurement accuracy, whereas the attribute interface retrieves the `PWR_ATTR_ENERGY` attribute to measure the energy consumed by the `do_work()` function.

REFERENCE IMPLEMENTATION

Commercial implementations of Power API are in development, but an open source reference implementation is available for early adopters (http://powerapi.sandia.gov). The reference implementation is architected to support Power API's core functions in a single implementation, with support for multiple measurement devices implemented through plugins. In this way, developers and users can rapidly integrate new measurement devices and power-control points.

All core functions have been implemented in the reference implementation, except for historical data collection and a subset of statistics functions for certain objects, and the implementation is sufficient for most real-time data measurement and control use cases. The implementation has been deployed as part of the Tri-lab OS (TOSS) and is running on several test and production platforms at DOE laboratories. The Sandia National Laboratories Power API team is conducting small-scale development and research on many of Sandia's Advanced Architecture Test Bed clusters,[13] and large-scale testing and research on the production Skybridge and Chama clusters at Sandia National Laboratories. The current focus is on integrating and optimizing large-scale collection methods.

Current support

The reference implementation supports low-level hardware measurement devices—from common off-the-shelf solutions such as WattsUp meters to device- and vendor-specific methods such as RAPL—as well as more comprehensive out-of-band power-measurement devices, such as PowerInsight. Its scalable framework enables

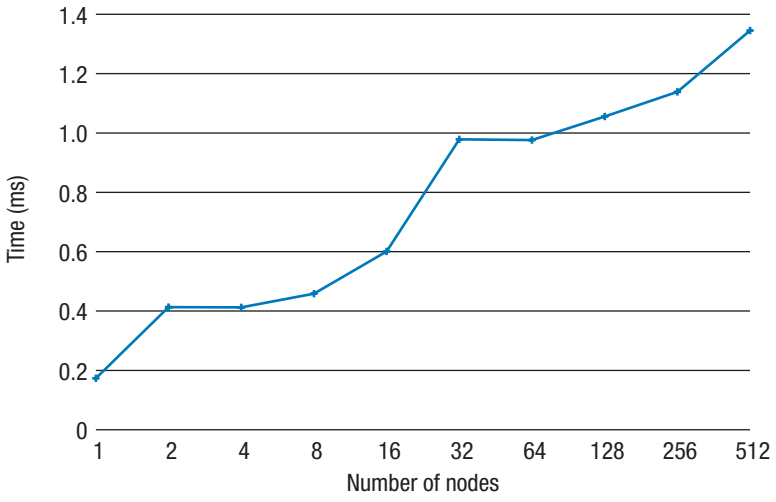

FIGURE 4. Results of evaluating latency in collecting energy data from multiple nodes. The microbenchmark used in the test measures the time that 1,000 `PWR_ObjAttrGetValue()` requests take to complete and divides that time by 1,000. The test was performed on Chama, a production supercomputer at Sandia National Laboratories.

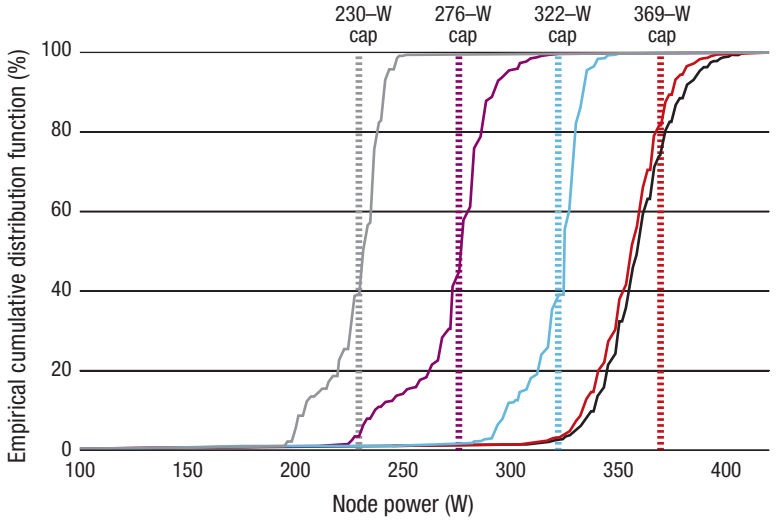

FIGURE 5. Power API power measurements used to understand node-level power-capping behavior with the CTH production application, a popular solid mechanics code. The graph depicts the cumulative distribution function for power samples for four node-level power caps on 96 Cray XC40 nodes. The vertical dashed lines are visual aids for each power cap level to illustrate samples that exceed the given power cap. The black curve denotes power with no cap. Power API data shows that the power-capping mechanism permits time periods during which the power draw can exceed the power cap.

the collection of measurements from many objects in a group simultaneously. Although results aggregation is implicitly embedded in the object hierarchy, work is underway to enable distributed results aggregation at multiple points instead of only at a single point. This aggregation has already shown good results even before developers have completed planned performance enhancements of the distributed collection techniques. Figure 4 shows the initial scaling of basic energy sampling from nodes in a large system, using Power API's distributed collection mechanism in the Power API.

Power-capping studies

Power API has already been used in studies to understand system power consumption and the effectiveness of power-control methods on large-system platforms. Among the increasingly available power management solutions for large systems is the node-level power-capping mechanism on Cray supercomputers. How this mechanism impacts performance and how much power state-of-the-art production applications consume during execution are important questions. To answer them, the project team installed Power API on a small Cray XC40 system with node-level power capping, which is housed at Sandia National Laboratories and currently running tests with CTH, a widely used solid mechanics application. Testing has revealed the distribution of power samples under different power caps. Figure 5 shows one such distribution. 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.[14] Power API has enabled collection of

all measurements on this system and enables portable testing on other systems and with other applications.

Power API's development structure follows that of several highly successful standards: a vendor-neutral national laboratory funded through the US federal government leads an effort with community input and public review, and the goal is to become a community-led public standard. Although the objective of developing a common API for power measurement and control was realized with the first specification release in 2014, Power API continues to evolve and grow as capabilities are added and new language bindings are supported. Most of Power API's interfaces are mature, but work continues on some of the highest-level interfaces as research and the development of future systems better inform high-level reporting requirements.

As part of a collaboration with the Sandia and Los Alamos National Laboratories, Cray is implementing portions of Power API on the Trinity supercomputer, which was ranked 7th on the June 2016 TOP500 list (www.top500.org/lists/2016/06). Additionally, these partners are working with Adaptive Computing, creators of the Moab/Torque resource manager, to enable intelligent power-aware job-scheduling decisions. These capabilities will leverage the Power API implementation on Trinity to demonstrate that power budgets of large-scale supercomputers can be effectively managed.

The project team intends to explore power prediction on large systems. Their objective is to allow power schedules to be created and relayed to the power-utility provider, who can then anticipate demand from such systems and be proactive in generating power according to short-term future needs. The intent is to think about what will benefit future systems within 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, PowerInsight, WattProf, WattsUp, Power Gadget, generic CPU registers, and Cray's XTPM measurement devices. The list of supported devices will grow in collaboration with vendors of components capable of power measurement.

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

ACKNOWLEDGMENTS

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

REFERENCES

1. K. Shoga et al., "Whitelisting MSRs with msr-safe," presentation, 3rd Workshop Extreme-Scale Programming Tools, *Proc. ACM/IEEE Int'l Conf. High-Performance Computing, Networking, Storage, and Analysis* (SC 14), 2014; www.vi-hps.org/upload/program/espt-sc14/vi-hps-ESPT14-Shoga.pdf.
2. J.H. Laros, P. Pokorny, and D. DeBonis, "PowerInsight—A Commodity Power Measurement Capability," *Proc. IEEE Int'l Green Computing Conf.* (IGCC 13), 2013; doi: 10.1109/IGCC.2013.6604485.
3. M. Rashti et al., "WattProf: A Flexible Platform for Fine-Grained HPC Power Profiling," *Proc. IEEE Int'l Conf. Cluster Computing* (Cluster 15), 2015, pp. 698–705.
4. D. Bedard et al., "PowerMon: Fine-Grained and Integrated Power Monitoring for Commodity Computer Systems," *Proc. IEEE Region 3 Southeast Conf.* (SoutheastCon 10), 2010, pp. 479–484.
5. A. Agelastos et al., "The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications," *Proc. ACM/IEEE Int'l Conf. High-Performance Computing, Networking, Storage, and Analysis* (SC 14), 2014, pp. 154–165.
6. P.C. Roth, D.C. Arnold, and B.P. Miller, "MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools," *Proc. ACM/IEEE Int'l Conf. High-Performance Computing, Networking, Storage, and Analysis* (SC 03), 2003, pp. 21–37.
7. T. Hirano et al., "Evaluation of Automatic Power Reduction with Oscar Compiler on Intel Haswell and ARM Cortex-a9 Multicores," *Proc. Int'l Workshop Languages and Compilers for Parallel Computing* (LCPC 14), 2014, pp. 239–252.
8. R. Ge et al., "Powerpack: Energy Profiling and Analysis of High-Performance

ABOUT THE AUTHORS

RYAN E. GRANT is a senior member of the technical staff in the Center for Computing Research at Sandia National Laboratories. His research interests include networking, systems software, and power management and control for high-performance computing (HPC) systems. Grant received a PhD in computer engineering from Queen's University at Kingston. He is a member of the IEEE Computer Society, IEEE, and ACM. Contact him at regrant@sandia.gov.

MICHAEL LEVENHAGEN is a senior member of the technical staff in the Center for Computing Research at Sandia National Laboratories. His research interests include enabling the codesign of exascale-class computing systems and enabling software control of energy use on HPC systems. Levenhagen received an MS in computer science from the University of New Mexico. Contact him at mjleven@sandia.gov.

STEPHEN L. OLIVIER is a senior member of the technical staff in the Center for Computing Research at Sandia National Laboratories. His research interests include parallel programming models, runtime systems, and power-aware HPC systems. Olivier received a PhD in computer science from the University of North Carolina at Chapel Hill. He is a member of ACM. Contact him at slolivi@sandia.gov.

DAVID DEBONIS is a Hewlett Packard Enterprise contractor at the Center for Computing Research at Sandia National Laboratories and a doctoral student in computer science at the University of New Mexico. His research interests include power-aware scheduling, distributed power monitoring, and scalable power-adaptive system software. DeBonis received an MS in computer science from the University of New Mexico. He is a member of the IEEE Computer Society and ACM. Contact him at ddeboni@sandia.gov.

KEVIN T. PEDRETTI is a principal member of the technical staff in the Center for Computing Research at Sandia National Laboratories and a co–project lead for Power API. His research interests include lightweight OSs, HPC networks, and power management in supercomputers. Pedretti received an MS in computer engineering from the University of Iowa. He is a member of IEEE. Contact him at ktpedre@sandia.gov.

JAMES H. LAROS III is a principal member of the technical staff in the Center for Computing Research at Sandia National Laboratories and a co–project lead for Power API. His research interests include systems architectures, systems software, and measurement and control of power and energy for HPC systems. Laros received an MS in computer engineering from the University of New Mexico. He is a member of IEEE. Contact him at jhlaros@sandia.gov.

Systems and Applications," *IEEE Trans. Parallel and Distributed Systems*, vol. 21, no. 5, 2010, pp. 658–671.
9. J. Eastep, "An Overview of GEO (Global Energy Optimization)," presentation, Intel, 2015; https://eehpcwg.llnl.gov/documents/webinars/systems/120915_eastep-geo.pdf.
10. R.E. Grant et al., "Overcoming Challenges in Scalable Power Monitoring with the Power API," *Proc. 20th IEEE Int'l Parallel & Distributed Processing Symp., Workshop on High-Performance Power-Aware Computing* (HPPAC 16), 2016, pp. 1094–1097.
11. J.H. Laros et al., *High Performance Computing Power Application Programming Interface Specification: Version 1.3*, report SAND2016-4446, Sandia National Labs., 2016; http://powerapi.sandia.gov.
12. F. Broquedis et al., "Hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications," *Proc. 18th IEEE Euromicro Conf. Parallel, Distributed, and Network-Based Processing* (PDP 10), 2010, pp. 180–186.
13. "Sandia National Laboratories Advanced Architecture Test Beds," 2016; www.sandia.gov/asc/computational_systems/HAAPS.html.
14. K. Pedretti et al., "Early Experiences with Node-Level Power Capping on the Cray XC40 Platform," *Proc. 3rd ACM Int'l Workshop Energy-Efficient Supercomputing* (E2SC 15), 2015; doi: 10.1145/2834800.2834801.

Selected CS articles and columns are also available for free at **http://ComputingNow.computer.org**.