A Power API for the HPC Community

David DeBonis, Ryan E. Grant, Stephen L. Olivier, Michael Levenhagen, Suzanne M. Kelly, Kevin T. Pedretti, James H. Laros III
Sandia National Laboratories, Albuquerque, New Mexico
{ddeboni, regrant, slolivi, mjleven, smkelly, ktpedre, jhlaros}@sandia.gov

http://powerapi.sandia.gov

Overview of the Power API

Motivation

Extreme scale computing requires that power and energy budgets be carefully managed. While much of the work of achieving the power targets for Exascale computing will have to be addressed through hardware improvements, the job of utilizing the tools made available through hardware will ultimately fall on system software and even applications.

Power caps are likely to be a reality on future top supercomputers. Practical power delivery concerns will impose some limits while operational costs and power utility generation capabilities will further influence caps that could be dependent on time of day use. Providing power capping capabilities on a system wide basis is not trivial and individual device power caps must be kept consistent with the overall power allocations.

With power becoming a first-class resource in future systems, applications will have to adapt to this new constraint and balance it with performance. Whenever possible power efficient algorithms can be adopted to increase performance for expected power budgets. The first step to understanding the power/performance tradeoffs for different algorithms is the observation and study of their current characteristics. The Power API enables both the measurement and control of power at both basic and advanced levels throughout an entire system, depending on the level of support available in hardware. As power constraints begin to impact components other than CPUs, such flexibility will be a key factor in providing a large-scale whole system solution to power management.

Prior Experience with Top Supercomputers

When DOE/NNSA and Sandia deployed the Red Storm platform in 2005, it ranked number 6 in the Top 500 list, and few if any other machines offered even the most foundational of power measurement and control capabilities. Starting with a RAS-level interface that Cray had exposed, Sandia developed energy-saving methods for large-scale executions of production codes. These efforts were recognized with the NNSA Environmental Stewardship Award and influenced Cray toward subsequent power/energy interfaces in future systems. However, these interfaces have been to-date vendor proprietary.

Sandia operates a commodity test-bed cluster [1] which each node is equipped by Penguin Computing with out-of-band measurement hardware called PowerInsight [2], which is described later in more detail. This cluster allows us to do research, to prototype power-aware code, and to emulate capabilities only now emerging in large-scale systems such as the future ACES Trinity platform. Based on our experiences with large systems, relationships with vendors, and NNSA/ASC support, we propose the Power API as a starting point for common, vendor-neutral power measurement and control in HPC.

"Cray has a long history working with Sandia National Laboratories and a strong partnership around developing new technologies and advanced solutions. The foundation for Cray's roadmap in Advanced Power Management was built on the pioneering research jointly conducted at Sandia Laboratories on the first Cray XT platform, Red Storm. In addition, Cray wants to acknowledge Sandia's leadership role in driving vendors to use common API's for power management and control." **-Peter Ungaro, President and CEO, Cray Inc.**

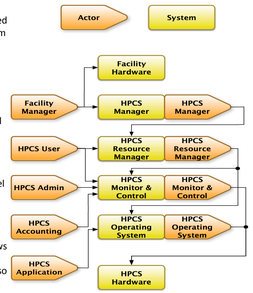
Example application on Red Storm: SAGE. Running at P-state 2 using 4096 cores **decreased energy usage by almost 50% while increasing execution time by less than 8%** compared to the default (P-state 0). Similar results observed on XNobel and AMG2006 using 6144 cores.

Comprehensive Interface Support

A key goal of the Power API is to support all layers of the HPC software stack. To identify key requirements, an intensive use case study considered the interactions between the system layers and between users and system layers, as shown in the diagram on the right. The use case document was reviewed by many community partners (as was the subsequent API) [3].

Informed by the use case study, the Power API defines a set of interfaces. Each interface expresses interactions between two system layers (e.g., operating system / monitor & control) or between a system layer and a person or entity (e.g., resource manager / user). The capabilities and level of abstraction vary by interface. For example, while both the hardware / operating system interface and the operating system / application interface expose power and energy readings, voltage and current are not exposed at the operating system / application interface. The low-level current and voltage measurements are not needed at the application level as energy and power measurements can be provided by the operating system at the interface.

The structure of each interface is the same, comprising the supported attributes and functions for that interface. This uniformity of design allows shared specification of shared core functionality, in addition to the individual specifications of functionality particular to each interface. It also enables the vendors working at multiple layers to maintain consistency in their implementations of the API.



The Power API Interfaces and System Actors

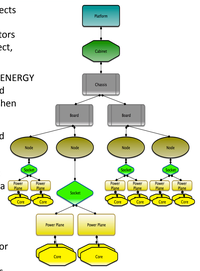
API Design

The Power API provides a method of describing a given system through a collection of objects and a discoverable hierarchy. The hierarchy of objects illustrates the basic objects in a system, which may be heterogeneous. Other objects, such as memory or NICs or accelerators can be added to the object hierarchy. For example, a NIC could be attached to a node object, being accessible to all other objects beneath the node in the hierarchy.

Each object in the Power API hierarchy has a set of associated attributes, e.g. PWR_ATTR_ENERGY and PWR_ATTR_MAX_POWER, which allow for measurement and control of power related capabilities of individual components or groups of components throughout the system. When measuring, the applicable attributes are read when controlling they are written.

In order to provide a comprehensive measurement ability, statistics gathering is integrated into the Power API. Statistics may be gathered on any object or attribute providing measurements. Statistics can also be gathered over groups of objects, for example one could gather the average power of a group of nodes over a given time. Statistics have two interfaces, one for real-time measurement and another for gathering historical data from a data store.

In order to be able to utilize the measurements and statistics from the Power API, a metadata interface is provided that allows for more detailed information on object attributes. Such information may contain how frequently internal sampling is performed for each measurement or how accurate the measurements taken are expected to be. This enables easy, comprehensive use of the Power API for measurement and control purposes.



Power API Object Hierarchy

Interface Example

The following code example presents several fundamental concepts of the API: context, user role, object hierarchy, hierarchy traversal, groups of objects, time stamps, object attributes, measurement and control. This code traverses the object hierarchy to cabinet 0 board 4, reads the current power cap for the board and sets the power cap to 500 watts if the current cap is greater than 500 watts. Note that the implementation must determine how to split the 500 watt budget among the objects in the hierarchy under the board. One could envision similar code in a utility program used by a system administrator.

```
PWR_Cntxt   cntxt = PWR_CntxtInit( PWR_CNTXT_DEFAULT, PWR_ROLE_ADMIN, "Admin CTX" );
PWR_Obj     platform_obj = PWR_GetSelf( cntxt );
PWR_Grp     cabinet_grp = PWR_ObjectGetChildren( platformObj );
PWR_Obj     cabinet0_obj = PWR_GrpGetObjByIndex( 0 );
PWR_Grp     cab0_board_grp = PWR_ObjectGetChildren( cabinet0_obj );
PWR_Obj     cab0_board4 = PWR_ObjectGetObjByIndex( 4 );
double value;
PWR_Time time;
PWR_ObjAttrGetValue( cab0_brd4, PWR_MAX_PCAP, &value, &time );
if ( value > 500.0 ) {
    PWR_ObjAttrSetValue( cab0_brd4, PWR_MAX_PCAP, 500.0 );
}
```

Interface code example illustrating the use of the Power API for a simple grouped power capping

Progress, Partners, and Extreme Scale Deployment

Partner organizations reviewed the API document [4] in July 2014 and subsequently the following September. Prototyping on test-bed platforms is an-going effort [5]. Implementation of the API will be included in the deployment of the $174 million dollar ASC/NNSA Trinity platform. Moreover, we anticipate and encourage continuing community feedback to drive refinement of the API. It serves as a first step toward a vendor-neutral standardization of power measurement and control, which is sorely needed as the community grapples with the power and energy challenges of extreme scale HPC.

Associated R &D Efforts

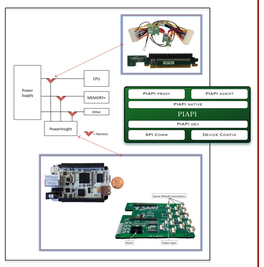
PowerInsight – Power Measurement at Scale

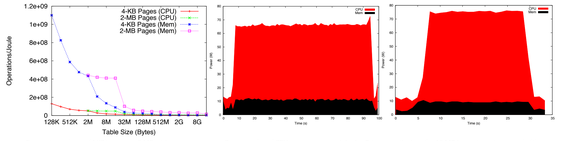
Emulation and testing of energy features of hardware is important for understanding the viability and usability of attributes exposed through the Power API. An early mechanism to exercise these features is through the Power API prototype [5].

Each node of a 102-node cluster has been instrumented with PowerInsight allowing researchers to collect component level power and energy measurements [6]. Additionally, collections are offloaded to the embedded ARM microprocessor allowing for out-of-band analysis and data reduction without impacting the node(s) under test.

An in-house software stack (PIAPI) allows us to control and communicate the collection rate and direct or intercept sample reports. The PIAPI also provides an internal framework for emulating future hardware power features on the embedded device and accommodates either push or pull modes of operation; subscribe to a collection request or poll instantaneous or hardware emulated counter data.

The PIAPI stack is presented through the plugin interface of the Power API prototype, allowing access through attributes. An experiment where we compare the energy efficiency of memory operations using small vs. large pages is shown below.





Comparison of small (4 KB) vs. large (2 MB) virtual memory page size for a random access micro-benchmark

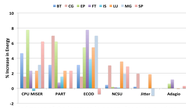
Power profile of CPU and memory on Intel Ivy Bridge architecture for 16 MB table size using small (left) and large (right) page sizes sampling at 10 Hz using out-of-band collection with PowerInsight

Power and Reliability

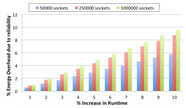
The intersection of power consumption and system reliability is an area of great interest for future extreme scale systems. The power consumption of reliability mechanisms is an important factor in considering overall system efficiency. Traditional checkpointing with local SSD checkpoints that are written out to a persistent store is a method that has been proposed for extreme scale systems.

Reliability can impact the gains made through runtime energy savings methods as well. Due to failure and recovery, the increased runtime that energy saving methods typically incur can result in resilience events like a failure occurring during the application runtime that without the increasing the runtime would not have been seen. Alternatively, extended runtime may result in more checkpoints being taken, also impacting the total energy usage of the application [7].

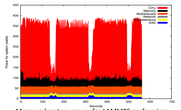
The Power API enables research in the intersecting areas of power, reliability and performance tradeoffs by providing a comprehensive method of gathering power related data as well as providing control capabilities to investigate possible methods of solving this difficult problem. The estimated increase in energy due to reliability of several runtime energy savings methods [7] based on their published performance and energy consumption results is illustrated below. Finally, real power measurements through the Power API compatible PowerInsight devices demonstrate the related power measurements during a checkpoint in a 4-node system during a store to topologically close network mounted SSDs with differing CPU frequencies [8].



Estimates of the increase in energy for given runtime power saving methods due to resilience



Energy overhead due to reliability when runtime is increased in large socket count systems



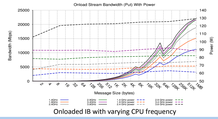
Measured system power for LAMMPS performing three checkpoints to local SSD

Network Power Management

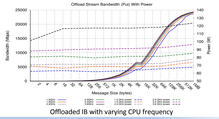
The performance of high-speed networks on power capped systems, and those with large numbers of smaller slower compute cores is a topic of interest. Exploring alternative networking approaches, like onloaded networks versus offloaded networks in the context of light-weight cores can be of use in deciding which approaches are suitable for next generation systems. The difference in network bandwidth provided by two different InfiniBand network approaches has been examined using an onloading approach and an offloading approach with different CPU frequencies on an Intel Xeon Ivy Bridge processor server.

Power management of networks may become a reality in future systems, which will require advanced techniques to ensure that latency and bandwidth requirements are met for applications while attempting to reduce network power consumption in underutilized network links.

The Power API enables network power research by allowing for easy access to per-component measurement and control on systems with appropriate hardware sampling support. Through the use of the Power API not only can methods be easily researched, but they can be easily deployed as the research development environment and production environment are similar.



Onloaded IB with varying CPU frequency

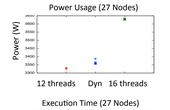


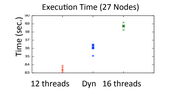
Offloaded IB with varying CPU frequency

Concurrency Throttling

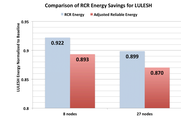
Dynamic run time adaptation is one anticipated system software approach to power management. In this study [9], we consider the use of the Qthreads lightweight threading library as a power-aware OpenMP run time. Qthreads transforms work (e.g., loop iterations) into tasks and distributes those tasks among long-running pthreads (one per core) for execution. Cores residing on the same chip contend for resources, e.g., cache and memory bandwidth. Memory bandwidth may become saturated. In that case, throttling some cores into a low power regime may save power, and sometimes even improve performance by easing memory contention.

The goal of the Maestro project is to enable Qthreads to make throttling decisions *online* using power and performance counter data. A daemon, RCRTool, gathers data from the hardware counters and posts it to a blackboard. Qthreads checks the data at intervals. If it detects that memory is saturated and power usage is high, it applies clock modulation to one or more cores on each chip. Power usage of those cores is reduced, and the run time scheduler stops assigning them work. At a later time, the RCRTool data may show a drop in the memory and power usage. In that case, the cores may be returned to full speed and assigned work once again.





An evaluation of the LULESH hydrodynamics mini-application running hybrid MPI with the Maestro OpenMP on each node showed both power/energy savings and improved performance [7]. Projected execution times factoring in reliability are even lower since fewer failures are predicted when programs finish quicker. Maestro required privileged access to machine-specific registers for counter data and clock modulation. The Power API presents such measurement and control mechanisms in a vendor-neutral interface, and the implementation would manage access to them.



References

1. Advanced Systems Technology Test Beds at SNL (http://www.sandia.gov/asc/computational_systems/HAAPS.html)
2. PowerInsight (http://www.penguincomputing.com/resources/press-releases/penguin-computing-releases-new-power-monitoring-device)
3. James H Laros, Suzanne M Kelly, Steven Hammond, Ryan Elmore, and Kristin Munch. Power/Energy Use Cases for High Performance Computing. Internal SAND Report SAND2013-10789. https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/ UseCase-powapi.pdf.
4. J. H. Laros, P. Pokorny, D. DeBonis, "PowerInsight – A Commodity Power Measurement Capability," *The Third International Workshop on Power Measurement and Profiling in conjunction with IEEE IGCC 2013, Arlington VA*, June 2013.
5. J. H. Laros, D. DeBonis, R. Grant, S. M. Kelly, M. Levenhagen, S. Olivier, K. Pedretti, "High Performance Computing – Power Application Programming Interface Specification," Version 1.0 – SAND2014-17061, Sept 2014
6. D. DeBonis, M. Levenhagen, J. H. Laros, "A Prototype of a Power API Framework," at the 3rd Extreme-Scale Programming Tools workshop in conjunction with SC14, New Orleans, LA, Nov 2014
7. R. E. Grant, S. L. Olivier, J. H. Laros, A. Porterfield, and R. Brightwell, "Metrics for evaluating energy saving techniques for resilient HPC systems," in *Proceedings of the 28th International Parallel and Distributed Processing Symposium Workshops*, IEEE, 2014, p. 8.
8. B. Mills, R. E. Grant, K. B. Ferreira and R. Riesen, "Evaluating energy savings for checkpoint/restart," in *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, ACM, 2013, p. 8.
9. Allan Porterfield, Stephen Olivier, Sridutt Bhalachandra, Jan Prins. Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs. Proc. of 8th IEEE Workshop on High-Performance, Power-Aware Computing (HP-PAC 2013). IEEE: Boston, MA, May 2013.

U.S. DEPARTMENT OF ENERGY | National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
SAND No. 2014-16629D