ACES and Cray Collaborate on Advanced Power Management for Trinity

James H. Laros III, Kevin Pedretti,
Ryan E. Grant, Stephen L. Olivier,
Michael Levenhagen, David DeBonis
Center for Computational Research
Sandia National Laboratories
Email: jhlaros,kepedre,regrant,
slolivi,mjleven,ddeboni@sandia.gov

Scott Pakin,
Computer, Computational, and
Statistical Sciences Division
Los Alamos Laboratory
Email: pakin@lanl.gov

Steven Martin, Matthew Kappel,
Paul Falde
Research and Development
Cray Inc.
Email: stevem,mkappel,
falde@cray.com

***Abstract*—The motivation for power and energy measurement and control capabilities for High Performance Computing (HPC) systems is now well accepted by the community. While technology providers have begun to deliver some capabilities in this area, interfaces to expose these features are vendor specific. The need for a standard way to leverage these emerging capabilities, now and in the future is clear. To address this need, the Department of Energy funded an effort to produce a Power application programming interface (API) specification for High Performance Computing systems with the goal of contributing this API to the community as a proposed standard for power measurement and control. In addition to the open publication of this standard an Advanced Power Management Non-recurring Engineering project has been initiated with Cray Inc. with the intention of advancing capabilities in this area and delivering them on a leadership class platform. We will detail the collaboration established between the Alliance for Computing at Extreme Scale (Sandia Laboratories and Los Alamos Laboratory) and Cray and the portions of the Power API that have been selected for the first production implementation of the standard.**

***Keywords*-power monitoring; power control; energy efficiency; power measurement;**

I. Introduction and Background

As the need to include power as a first-class consideration in every aspect of High Performance Computing (HPC) became clear, the lack of standard interfaces in this area became more evident. Sandia National Laboratories (Sandia) began investigating how to address this gap in 2012 by evaluating use cases revealed by early research in this area. The result of this effort was a document [1] that outlined the scope and interfaces that a power application programming interface should address if it were to meet the demanding needs of HPC. Immediately following this effort, in January 2014, a team at Sandia formally began creating the *High Performance Computing - Power Application Programming Interface* specification [2] (Power API).

The Power API targets a broad range of interfaces ranging from low level capabilities exposed by technology providers to higher level interfaces that address use cases involving end users, applications and work-load managers, for example. Six months after focusing primarily on the core interfaces of the specification an early draft was vetted by a range of technology providers (Adaptive Computing, Cray, AMD, Penguin Computing, Intel, and IBM), laboratory (National Renewable Energy Laboratory, Oak Ridge National Laboratory) and university (University of New Mexico) representatives. The technology providers were specifically targeted since the success of any proposed standard depends on it being implemented. However, community involvement is just as critical to drive the development of the specification in an unbiased manner and ensure that it remains vendor-neutral. One of the primary goals of the Power API is to present a set of portable interfaces, shielding the end user, no matter what role they serve, from vendor specific implementation details.

During the same time that Sandia was preparing to begin development of the Power API, the Alliance for Computing at Extreme Scale (ACES), a collaboration between Sandia and Los Alamos National Laboratory (Los Alamos), was preparing to release a Request for Proposal (RFP) for Trinity, the DOE's National Nuclear Security Administrations (NNSA) first Advanced Technology System (ATS-1). An important aspect of this new effort by the DOE is the investment in advanced technologies in the form of non-recurring engineering (NRE). A portion of funding for each platform in the ATS line is invested in advanced technologies selected for their potential impact to the DOE/NNSA mission. For the Trinity (ATS-1) procurement, Burst-Buffer and Power were selected as the two focuses of NRE investment. This paper will focus on the Trinity Advanced Power Management (APM) NRE program.

In 2014 the contract to deliver Trinity was awarded to Cray Inc. The Trinity APM NRE with Cray Inc. focuses on two general areas that leverage technologies of interest that Cray exposes in the Trinity time-frame. An additional collaboration has recently been established with Adaptive Computing (Adaptive). Adaptive is the provider of the production resource manager that will be used on Trinity. The bulk of this paper will discuss the NRE efforts that are currently in progress with both Cray and Adaptive (Section IV). We will present some essential background

*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 SAND2016-3329 C.

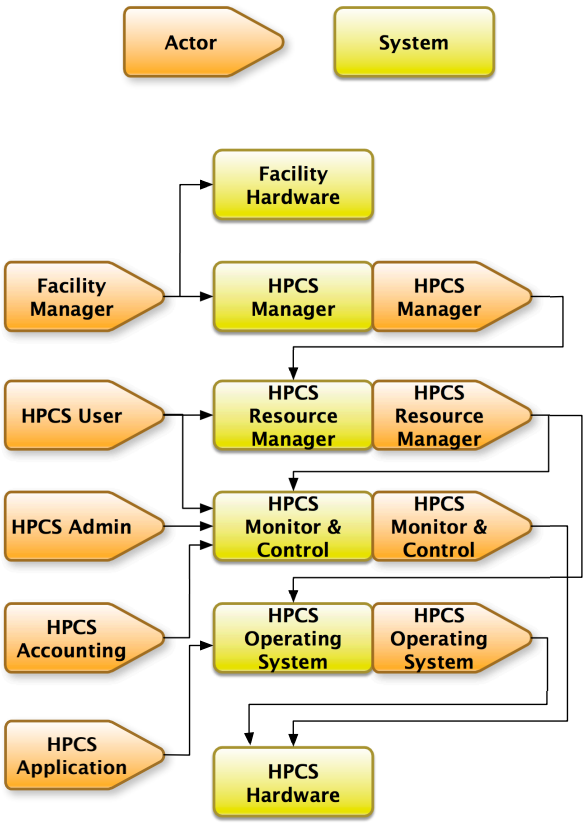


Figure 1. Power API, Roles and Interfaces

regarding Power API concepts (Section II) and Crays system management infrastructure (Section III). This paper will also briefly discuss some related works (Section V) and will provide some concluding thoughts and our intentions for future work in this area (Section VI).

II. Power API

As previously mentioned, the scope of the Power API is broad. While a complete description of the specification is beyond the scope of this paper, some fundamental concepts are important to understand the areas of focus that were selected for the ACES/Cray and ACES/Adaptive collaborations. The following is a very high-level description of topics included in the Power API specification. For a complete description please refer to the specification itself [2]. Figures 1 and 2 will help illustrate some fundamental Power API concepts.

The diagram in Figure 1 was developed during the use case exercise [1] preceding the development of the Power API specification. The key at the top lists the two named shapes that are used in the diagram, Actor and System. These names are a result of the Unified Modeling Language (UML) approach that was applied during our use case investigation. The diagram depicts two important concepts that are used in the specification and are important for the material in

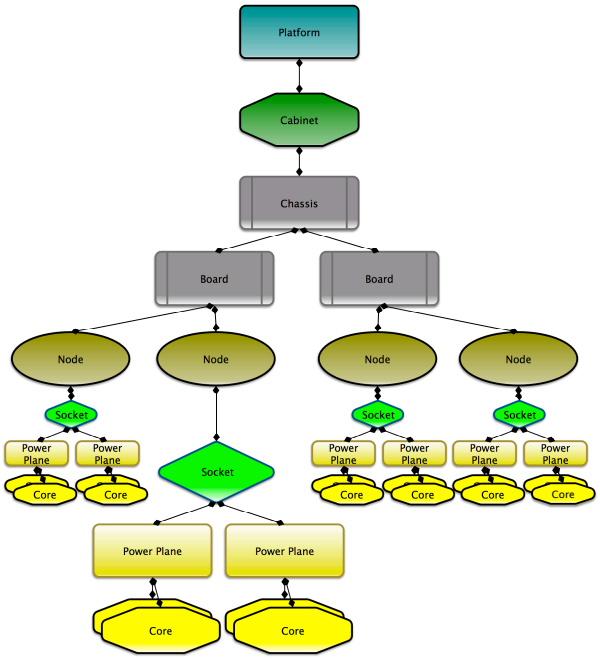


Figure 2. Example System Description

this paper. The Actors in the diagram are nearly directly translated into the `Roles` covered in the specification. A `Role`, as it relates to the specification and this paper, is the entity interacting with the HPC system whether it is a physical person or a program (application for example). A `System` is an entity that is being interacted with, provides a service, instrumentation (like sensors), anything that enables measurement and control of power and energy. The lines in Figure 1 represent the interfaces between a `Role` and the `System` that it interacts with. Note, that in many cases a `Role` acts as the `System` for another `Role`. [1]

Figure 2 depicts an example of a system description comprised of basic objects supported, but not required to be used, arranged hierarchically that represent a description of a hypothetical system (note, not a `System`) presented to the user of the Power API. As you can see in figure 2, objects represent very familiar types of components found in most HPC systems like nodes, sockets and cores and higher level physical organization constructs like boards, cabinets and platform. The system description is exposed to the user of the Power API as part of the context returned upon initialization. Figure 2 represents a symmetric homogeneous example but a system description can be constructed to represent any system, or portion of a system. In fact, while not required, it is suggested that an implementation limit the view of the system that a user of the API is exposed to based on their

[1] Throughout this paper we will highlight the words `Role` and `System` with a different font when they are used in a Power API specific context for clarification.

typical use case. For example, an application will likely only need to see a portion of the system description hierarchy, the node that it is executing on and below, while a system administrator probably has a need to interact with the entire system. Navigation functionality is provided by the Power API to traverse the system description hierarchy as desired.

The Power API specification is roughly organized into common core functionality, i.e. a superset of basic functionality that is potentially useful to any `Role`, and more specific high-level interfaces. The specification also recognizes commonality for high-level functionality among a sub-set of the `Roles` covered. Common core functionality covers a wide range of measurement and control functionality using an attribute interface where setting an attribute is equivalent to control and getting an attribute equates to measurement. Attributes are associated with individual objects (the objects that comprise the system description) so individual instrumentation points can be measured or controlled (if supported by hardware). For example, Power is an available attribute in the Power API specification and can be associated with any object that has the ability to provide a power measurement, direct or derived. Frequency is an example of an attribute that is available to set (control) and or get (measure or more accurately retrieve). While is it probably clear that the Frequency attribute was inspired by processors, there is no reason that it could not be associated with any current or future object type that provides an interface to set or get its frequency. The attribute interfaces included in the specification also allow for interaction with multiple objects with a single call or multiple attributes of a single or multiple objects. The specification also contains the concept of groups. A group is a collection of one or more objects. Operating on a group is a convenience and efficiency mechanism supported by most of the capabilities in the specification, e.g. the attribute interface operates on a group to get or set an attribute, or attributes, on a group of one or more objects.

Statistics are useful for most if not all `Roles` that interact with an HPC platform. The Power API includes statistics interfaces as part of the core functionality that allow for the real-time or historic collection of statistics like average, minimum, maximum and standard deviation, for example. The specification also includes a metadata interface that allows the user of an implementation of the Power API to get important information regarding the measurement and control interfaces that are present on that particular system. Metadata is specific to an object attribute pair. This is significant, for example, since one sensor may provide less accurate information than another. It should be noted that most aspects of the Power API are extensible, in fact the specification was designed to allow for the evolution of the specification itself and for an individual implementation to extend beyond capabilities specified when desired.

When deciding which areas of the Power API would be implemented as part of the Trinity NRE, the existing Cray management infrastructure was examined to determine prime areas of collaboration.

III. Cray Systems Management Infrastructure

To facilitate goals of Reliability, Availability, and Serviceability (RAS), Cray HPC systems dating from the XT-series systems to the current XC-series systems utilize a separate, out-of-band management network in addition to the in-band high-speed network used by compute resources. Over this out-of-band network, a head node known as the System Management Workstation (SMW) is connected in a tree structure descending to embedded cabinet controllers (CCs) and from CCs to embedded blade controllers (BCs). This, along with the software that it supports, is known as the Hardware Supervisory System (HSS). HSS orchestrates power, booting, environmental monitoring, hardware health monitoring and logging, and response to hardware failures among other RAS-focused duties.

For power monitoring and management, Cray XC-series systems leverage the HSS infrastructure to:

- Monitor and store node-, blade-, and cabinet-level power, energy, and environmental telemetry,
- Set power "knobs" on sets of nodes including P- and C-states and setting power caps,
- Enable in-band monitoring on compute nodes using the PM counters interface,
- Support queries of historical power, energy, and environmental telemetry coupled with job and application data with a powerful PostgreSQL-based, time-series database, and
- Provide a backend system to support a RESTful interface for platform and power monitoring and control.

In the following two sections, we will overview the database and the RESTful interface for monitoring and control.

A. Power Management Database Overview

The Power Management Database (PMDB) is a round-robin, time-series database implemented leveraging PostgreSQL alongside Cray-custom software [3]. The PMDB was first released with SMW 7.0.UP02 in July 2013. Broadly, it stores node-, blade- and cabinet-level power and energy telemetry, job- and APID-level information and timings, and System Environmental Data Collections (SEDC) data, including thermals and hardware health data. Power and energy telemetry is captured system-wide by default at 1 Hz (i.e., one observation per second), but for a subset of the system, this frequency can be increased to 5 Hz. SEDC has long been part of the HSS infrastructure, existing prior to Cray's power management efforts but, targeting narrower hardware debugging use cases, had previously only been available in flatfile-form.

Because storage is unfortunately a finite resource, the PMDB is necessarily a round-robin database. That is, once a defined storage threshold is exceeded, the oldest data are dropped to make room for the newest data. These thresholds

are defined on a per-table basis using an SMW-resident utility called *xtpmdbconfig*. Customers may use the *xtpmd hooks* interface to execute commands on rotation of old data, such as archiving the old data to a remote server [4].

B. Cray Advanced Platform Monitoring and Control

With an eye toward allowing workload managers to actively manage power and node configuration, Cray released the Cray Advanced Platform Monitoring Control (CAPMC) with SMW 7.2.UP02 and CLE 5.2.UP02 in the fall of 2014. With CAPMC, a remote (and authenticated) user may control the system by booting and shutting down nodes, setting P- and C-states (i.e., frequency and sleep-state limits), setting power caps. etc. This remote user may also monitor the system, by getting node state information, energy statistics about sets of nodes, system- and cabinet-power information, etc. The CAPMC infrastructure implements a RESTful interface using *nginx* in one of its common deployment roles. It provides encryption and user authorization capabilities to an independent, application-specific server. In this case, that application-specific server is called *xtremoted*, a Cray-specific daemon residing on the SMW. This provides bridge between the external world and HSS using industry-standard security.

A full description of CAPMC functions and its API is documented in [5]. Some technical and use-case details about CAPMC are given in [6].

IV. Trinity NRE

The decision of which area of the Power API to focus on involved many considerations. A complete implementation of the Power API would likely require more time and funding than available for the Trinity APM project. Since the team had to be more selective, we focused on high priority areas that aligned well with capabilities that appeared in Cray's roadmap in the Trinity time-frame, even if these capabilities required modification or acceleration to meet our combined goals. We also considered areas of the existing Cray systems management infrastructure that we could leverage and align with, see Section III.

An important `System` for the purposes of this paper (and the Trinity Power NRE project) is the Monitor and Control system. The Monitor and Control system encapsulates the concepts of systems management or RAS systems (Reliability, Availability and Serviceability). Cray has been introducing measurement and control capabilities important for this topic for a number of years. Cray's Power Management Database (PMDB) is a collection point for a wide range of power and energy related information, along with data important to correlate this information with jobs that are and have executed on the platform (see Section III-A). Exposing the information contained in the PMDB to the Admin Role is the first focus area of the Trinity APM NRE collaboration with Cray that will be covered in Section IV-A.

The second area of focus for the ACES/Cray collaboration is a compute node implementation. The `Systems` (Figure 1) relative to this area are Hardware and Operating System. In general, the focus is exposing power and energy relevant measurement and control knobs to `Roles` such as the Application and the Resource Manager. The compute node implementation is covered in Section IV-B.

The third focus area is power aware scheduling, a very broad topic. Adaptive and ACES are actively working towards finalizing the goals for this project. In Section IV-C we will discuss some of the use cases that we hope to enable with this effort and some of the capabilities implemented as part of the ACES/Cray collaboration that will be exercised.

A. Power Management Database Implementation

Cray has recently introduced a capability to retain historic information related to power and energy called the Power Management Database (PMDB). See section III for a description of the PMDB and type of information retained in the database. One of most important aspects in any effort to modify a characteristic is to first understand the current condition of that characteristic. Trying to affect power on an HPC platform is no different. Cray's PMDB provides a repository of information that allows a user (some Power API `Role`) to mine power and energy relevant data (measure). The `Role` that Cray is initially implementing to interface with the PMDB (essentially part of the Monitor and Control `System`) is the systems administrator (Admin) `Role`. As mentioned previously, a system administrator typically has the need to understand the entire HPC system. In the PMDB implementation, the Admin `Role` will have a view of the entire HPC platform.

While the initial versions of the Power API were specified in the C language, systems administrators more commonly use scripting or interpretive languages to do their jobs. Python was selected due to its popularity for `Roles` like system administrators (Admin) and Resource Managers, for example. The PMDB implementation will include most of the core functionality of the specification. This includes the attribute interface which allow the user of the Power API to get (measure) information about specific objects or groups of objects. For example, the administrator may desire to get a point in time power measurement from a node (object) or a group of nodes (group of objects). Possibly more useful would be to monitor the energy use of a node or group of nodes. Using the attribute interface the administrator could request the energy reading from a node or nodes, wait a period of time and repeat the call to determine the energy used by that node or group of nodes over that period of time. Note the specification states that the time-stamp related to the sample returned be temporally as near as possible to when the sample was measured.

These low-level interfaces, while useful, are probably not as powerful when interacting with a database as they are at lower levels, like interfacing in real-time directly with the hardware. The PMDB implementation will additionally

include the historic statistics interface of the Power API. This interface will allow the user to obtain information like the minimum and/or maximum of a power reading across a number of nodes (all of the nodes assigned to a particular job) over a period of time. The average power of the same group of nodes could be requested.

Probably the most common interaction with a data-base is generating a report. While the Power API specification has the beginnings of some high-level interfaces for this purpose, Cray and ACES are in the process of defining a flexible report interface that will enable the user to request reports for a range of information available in the PMDB. This information will contain job and system related information that the Power API does not currently address but is clearly closely related and necessary for many reasons. For example, Cray is working with ACES to develop two Python report programs that produce text output. The first uses some combination of job ID, application ID, and user ID as input to generate report output that includes data that are of general interest to the systems administrator (Admin) `Role`. These data include: job ID, application ID, user ID, total energy, start time, end time, and node count. Verbose detailed data for this type of report may include per-node power and energy statistics. The second report type will deliver useful system- and cabinet-level power and energy information, perhaps over a 24-hour window. This report will include data targeted for data-center managers and site planning personnel. It will include statistics like daily and hourly minimum, average, median, and maximum power usage for each compute cabinet. Recall that the PMDB information is stored round-robin and information expires dependent on space available. Generating reports withing the bounds of data expiration is one way to retain important information on a more permanent basis.

An important value already realized by the ACES/Cray collaboration on the Trinity APM NRE is the improvement of the Power API specification. Cray has been instrumental in vetting the Power API from the implementation perspective. In the short time we have been collaboration we have discovered multiple opportunities for improvement that have been included in the latest three point releases of the Power API specification. The Python implementation of the Power API, when complete, will be included in the Power API specification as the first alternative language binding. We anticipate release later in 2016.

B. Compute Node Implementation

Any effort to understand (measure) or control power and energy for HPC platforms almost necessarily considers node level measurement and control. Early (and on-going) research in this area focused on the potential of manipulating CPU frequencies to reduce power or energy use, for example. For HPC, this is complicated by the need to maintain performance, or minimally affect it. ACES and Cray consider this area of focus to be of great importance in demonstrating and investigating advanced capabilities. Cray will be delivering a C based compute node implementation of the Power API as part of the Trinity APM NRE project. While we cannot cover every capability that will be implemented we will discuss some common and high value characteristics of the implementation in this section.

The `Roles` that will be initially developed are the Application and Resource Manger `Roles`. While these `Roles` could potentially have different needs from the perspective of how much of the system description is exposed upon initialization, the initial efforts will limit exposure to the node level, and below, to both `Roles`. As the collaboration proceeds an expansion of the use cases addressed related to the Resource Manager `Role` may be considered.

As with the Python PMDB implementation, the core functionality of the Power API will be implemented as part of this effort. For the compute node implementation, the core functionality has the potential of being of great value. For example, obtaining power or energy information for the node, or specific component of the node like the CPU, is something that any power-aware application or resource manager would require. The attribute interface (part of the core functionality) allows the user (the Application or Resource Manager `Role` in this case) to obtain point in time power samples or energy over a given time period. With the exception of energy, measurement attributes, like power, are individual point in time samples. The user will have access to attributes representing power, temperature, frequency, and power cap, for example. The specification requires that the time-stamp returned with any sample accurately represents the time the sample was measured. In the case of energy, power over a period of time, two or more calls using the attribute interface are required. For example, an application that is interested in the energy used over a certain phase can make a call using the attribute interface to get the energy at the beginning of the phase. This value will typically be an accumulator with an associated time-stamp. At the end of the phase being examined the application can make a second call. In the typical use case, the energy over the period of time between the first and second call is represented by the difference between the first and second values returned.

One of the primary additions to typical production functionality that will be enabled by this effort is the ability to control CPU frequency from a user space process. This capability opens up a wide range of potential use cases. Currently, Cray, through the CAPMC interface enables the user (via the Resource Manager) to set CPU P-states that will remain constant through the life of the job execution. Through the Power API implementation, the ability to dynamically change CPU frequency will be exposed to user space processes. This will enable more granular dynamic control during the entire application execution. A power-aware application could, for example, run at a lower frequency P-state when it is in a communication phase. Figure 3 shows that lower frequency

P-states can be used with little to no performance impact during communication phases [7] (given the network supports offloaded processing of network packets). While on-loaded networking solutions see significant impact from P-state changes, as the CPU is used to process network traffic, such systems can still benefit from P-state changes if applications use small messages that are latency sensitive as shown in Figure 4. These P-state changes allow significant power savings from a node perspective and in some cases may minimally impact application performance, making power saving P-states feasible to exploit for some applications. Capabilities like dynamic frequency/P-state and C-state control, implemented on the compute node can be made available to the Resource manager and/or the Application.

The compute node implementation will also include the statistics interface. In this case the real-time interfaces will be implemented (recall that the historic interfaces are being implemented to interact with the PMDB (see Section IV-A)). The user will be able to create statistics objects that represent a tuple of object, attribute and statistic that they wish to collect. Statistics objects can be started, polled while active and stopped to mine the particular time range of interest. The statistics interface is a very powerful tool that can leverage lower level telemetry capabilities like those provided by Crays HSS (see Section III). For a complete description of the interface capabilities see the Power API specification.

The Power API specification contains the concept of application hints. The idea behind this is that the application best understands what it is doing at any point in time, or will be doing at some point in the future. While there are many open questions in implementing this type of capability, this is one of the areas that ACES and Cray are interested in investigating as part of this collaboration. Given the capability of dynamically controlling CPU frequency is available, the application is in a good position to provide hints regarding how some underlying layer might manage the CPU, or other components, to obtain an optimal balance between performance and power efficiency. Some of the hints available to the user via the high-level application interface are: serial, parallel, compute, and communicate[2]. As previously described we have shown potential power savings when using lower CPU P-states during an application communication phase (Figure 3). The application hints interface would be a convenient way for the application to communicate this to an intelligent run-time layer. Once the communication phase is complete, the application could again hint that it is about to enter a compute intensive region, for example. As applications adapt to evolving node architectures it has become increasingly important to exploit parallelism to take advantage of larger numbers of cores or accelerators. However, portions of current, and likely future, applications still have serial phases. If the application can provide hints that indicate a serial versus a parallel phase, the run-time could potentially deliver both greater performance and power efficiency. For example, when an application is entering a serial phase an intelligent run time could proactively shut down cores that will not be in use, enabling the core executing the serial portion of the application to run at the highest frequency available. This could result in greater performance for the application (serial phase is accelerated) while potentially saving power (cores not in use are put in a minimal power state). Once the serial phase is complete, the application can hint the end of the phase by sending the default hint or some other hint that could help optimize performance, power or both. The use case that was just described suggests that the application will provide appropriate hints. It is also possible that another layer, like the Message Passing Interface (MPI) layer, could send similar hints. This would allow this type of optimization without requiring modification of the application.

[2]For a complete list please refer to the Power API specification.

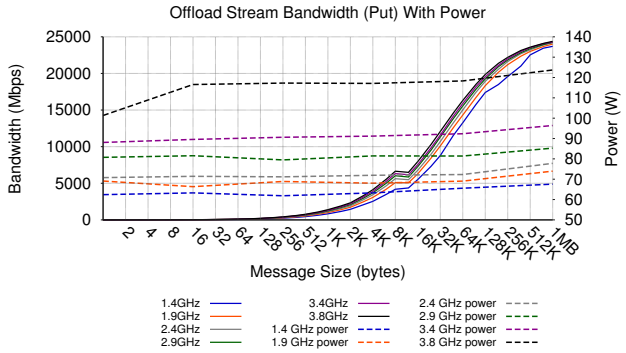


Figure 3. Offloaded network traffic stream bandwidth with varying CPU frequencies

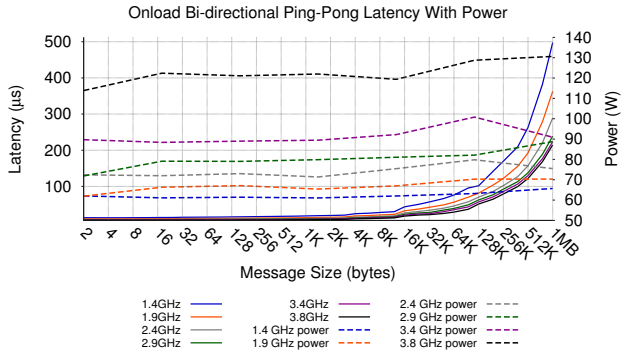


Figure 4. Onloaded network traffic stream latency with varying CPU frequencies

C. Power Aware Scheduling

ACES recently began a collaboration with Adaptive Computing as part of the Trinity APM NRE project. We will only discuss this briefly since we are in the very early stages of this work. Power ramp control, the ability to control

the rate at which the system increases its power use, is being implemented by Cray as part of the compute node implementation (see Section IV-B) Trinity APM NRE. ACES will be working with Adaptive computing to develop, test and implement this capability which is controlled by the resource manager through the CAPMC interface and will leverage the Power API interface on each compute node. One of the challenges we see in the future is very large swings in power draw for our largest platforms. While this may not present a problem for the facility in the Trinity time-frame, it is likely to for the next ACES ATS platform, Crossroads. Refining this capability on Trinity will allow ACES to be prepared for platforms that can experience multiple megawatt swings in power in very short periods of time. Likewise, managing platform power within pre-determined, or pre-negotiated lower and upper limits can prove to be a huge cost savings for the facility. Using more or less of the pre-negotiated power results in much higher costs for a facility. Similar to the power ramping effort, ACES will be working with Adaptive to exploit capabilities that Cray will expose, developed as part of the ACES/Cray Trinity APM NRE, to operate the platform within pre-determined upper and lower bounds, even bounds that differ throughout the day. In addition, we will be working with adaptive to execute individual applications within power constraints to maximize the science output within a given platform power constraint. The combination of the ACES/Adaptive and ACES/Cray collaborations should result in power-aware scheduling and management capabilities that have never been possible on a leader-ship class HPC platform.

D. Power Capping

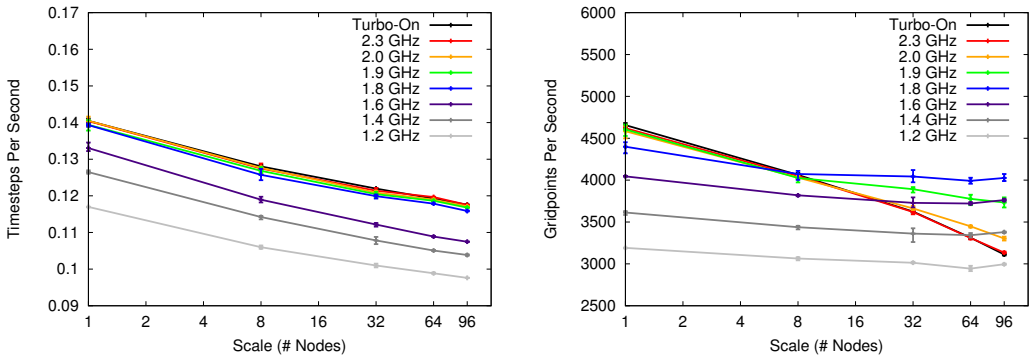
Related to power-aware scheduling, we have begun to explore the use of power capping as a mechanism for enforcing power budgeting decisions made by the workload manager. Cray's CAPMC infrastructure enables workload managers to set a desired power budget for each compute node in the system, which is then enforced by firmware running on each compute node. For example, a workload manager may decide to power cap a given job's compute nodes to 200 W of the maximum 400 W per node and shift the 200 W difference elsewhere in the system where it can be better utilized. Our initial finding is that while node-level power capping on Trinity is effective at maintaining the desired average power usage over multi-second time windows, the scalability of some workloads is significantly impacted by the performance variability introduced by the power capping mechanism [8]. As an example, Figure 5 shows the performance of the CTH and S3D applications running under a 230 W per-node power cap setting on Mutrino, a small-scale Trinity testbed at Sandia. CTH performance behaves as expected, with performance degrading gracefully under the power cap. S3D, on the other hand, experiences significant performance degradation with scale when running at the default Turbo-On p-state (2.3 GHz base with dynamic scaling up to 3.6 GHz). In this case it is better to run S3D at a fixed 1.8 GHz p-state because it results in average power usage being below the 230 W cap, which avoids the power capping mechanism from being triggered. We are currently working to better understand this behavior and find ways to mitigate it, as well as examining how node-level power capping affects power usage at the facility level [9].

V. Related Work

Power measurement/monitoring APIs have been developed in the past. Most of these APIs are device specific, such as Powermon [10] and Powerpack [11]. Also, more recent out-of-band measurement devices have had specific targeted APIs, like PowerInsight [12] and WattProf [13]. Out-of-band measurement devices have provided fine-grained measurement and control via their device APIs, and have allowed remote measurement to occur, such as the PowerInsight [12] specific piapi. Such APIs allow for collection from their respective devices in an efficient manner, and as such are used in the Power API implementation through device-specific plugins. In addition to external or dedicated monitoring devices, some hardware has built in power/energy monitoring and management functionality. Measurement directly from Running Average Power Limit (RAPL) control mechanisms on Intel processors have been introduced through MSR-safe [14]. Such user-level access to machine specific registers (MSRs) is critical to allow in-band energy measurement. The Power API provides similar functionality and can utilize mechanisms like MSR-safe as plugins to allow for user-level access to privileged information. Cray's CAPMC allows for power monitoring and control capabilities on Cray systems, it is a RESTful interface that uses JSON for issuing and interpreting commands [3], [6], [15]. Another RESTful JSON interface named Redfish [16] can be used for management of generic cloud infrastructures and provides basic support for managing power reserouces in a cloud environment. HP's iLO [17] is a proprietary out-of-band interface for taking measurements, including power/energy on HP clusters.

Global Energy Optimization (GEO) is a energy optimization framework developed by Intel [18]. 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 [18], though they are open-source.

PAPI [19] is an example of a high-level, portable API for performance monitoring that seeks to solve similar problems to the PowerAPI, but in the performance realm instead of power. Scalable measurement collection for performance-related data is available in tools like the Lightweight Distributed Metric Service (LDMS) [20]. Many scalable collection tools use overlay networks to aggregate



(a) CTH Performance with 230 W Power Cap

(b) S3D Performance with 230 W Power Cap

Figure 5. CTH and S3D application scalability when running under a node-level power cap.

data such as MRNet [21]. Large scale collection of data using methods like these has also been applied to other performance and correctness data collection like the Stack Trace Analysis Tool (STAT) [22].

VI. Conclusions and Future Work

We are in the beginning phases of the Trinity NRE collaborations with both Cray and Adaptive. The ultimate goal of this work is to deliver measurement and control capabilities that form the basis of all use cases related to power and energy for HPC systems. With both the Cray and Adaptive efforts ACES has also endeavored to provide these capabilities using platform independent interfaces (documented in the Power API specification). ACES intends to use many of these capabilities on the production Trinity platform. We also intend to use the capabilities exposed as part of both NRE collaborations to research emerging use cases to determine how best to maximize productivity on Trinity (and follow on platforms) given power and energy constraints.

Acknowledgment

The authors would like to thank the Trinity and Cray project leadership, Doug Doerfler, Manuel Vigil and Scott Hemmert (ACES) and Tim Ingebritson (Cray) for supporting our efforts on this project and having the vision to support this important area of collaboration. Jonathan Woodring from Los Alamos Laboratory provided some much needed Python expertise for the ACES team. The Cray team responsible for the implementations described in this paper are: Steve Martin, Dave Rush, Matthew Kappel, Josh Williams, Leo Maurer, Sam Watters, Rick Ward, Jim Robanske, Elwira Karwowski, Greg Koprowski, Larry Babb and Paul Falde, some of whom are also recognized as authors. The funding for this project was provided by the Department of Energy Nuclear Security Administration Advanced Simulation and Computing Program.

References

[1] J. H. Laros III, , S. M. Kelly, S. Hammond, R. Elmore, and K. Munch, "Power/energy use cases for high performance computing," *Sandia National Laboratories, Tech. Rep. SAND2013-10789*, 2013.

[2] J. H. Laros III, D. DeBonis, R. Grant, S. M. Kelly, M. Levenhagen, S. Olivier, and K. Pedretti, "High performance computing-power application programming interface specification version 1.0," *Sandia National Laboratories, Tech. Rep. SAND2014-17061*, 2014.

[3] S. Martin and M. Kappel, "Cray XC30 power monitoring and management," *Proceedings of CUG*, 2014.

[4] "Monitoring and managing power consumption on the Cray XC system," 2015. [Online]. Available: http://docs.cray.com/books/S-0043-7204/S-0043-7204.pdf

[5] "CAPMC API documentation release 1.1," 2015. [Online]. Available: http://docs.cray.com/books/S-2553-11/S-2553-11.pdf

[6] S. Martin, D. Rush, and M. Kappel, "Cray advanced platform monitoring and control (CAPMC)," *Proceedings of CUG*, 2015.

[7] M. G. Dosanjh, R. E. Grant, P. G. Bridges, and R. Brightwell, "Re-evaluating network onload vs. offload for the many-core era," in *2015 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2015, pp. 342–350.

[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 (E2SC)*, 2015.

[9] J. Brandt, D. DeBonis, A. Gentile, J. Lujan, C. Martin, D. Martinez, S. Olivier, K. Pedretti, N. Taerat, and R. Velarde, "Enabling advanced operational analysis through multi-subsystem data integration on trinity," *Proceedings of the Cray User Group (CUG)*, 2015.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] A. Hart, H. Richardson, J. Doleschal, T. Ilsche, M. Bielert, and M. Kappel, "User-level power monitoring and application performance on Cray XC30 supercomputers," *Proceedings of the Cray User Group (CUG)*, 2014.

[16] D. M. T. Force, "Redfish - simple, modern and secure management for multi-vendor cloud and web-based infrastructures," SPMF, Tech. Rep., 2015. [Online]. Available: https://www.dmtf.org/sites/default/files/standards/documents/RedfishTechNote.pdf

[17] A. R. White, "Methods and apparatus for diagnosing and correcting faults in computers by a support agent at a remote location," Apr. 2 2002, uS Patent 6,367,035.

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

[19] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci, "A scalable cross-platform infrastructure for application performance tuning using hardware counters," in *Supercomputing, ACM/IEEE 2000 Conference*. IEEE, 2000, pp. 42–42.

[20] 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.

[21] 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.

[22] D. C. Arnold, D. H. Ahn, B. R. De Supinski, G. L. Lee, B. P. Miller, and M. Schulz, "Stack trace analysis for large scale debugging," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–10.