Topics on Measuring Real Power Usage on High Performance Computing Platforms

James H. Laros III [#1], Kevin T. Pedretti [#]
Suzanne M. Kelly [#], John P. Vandyke [#], Kurt B. Ferreira [#]
Courtenay T. Vaughan [#], Mark Swan [*]

[#] *Sandia National Laboratories*
[1] jhlaros@sandia.gov

[*] *Cray Inc.*

***Abstract*—Power has recently been recognized as one of the major obstacles in fielding a Peta-FLOPs class system. To reach Exa-FLOPs, the challenge will certainly be compounded. In this paper we will discuss a number of High Performance Computing power related topics. We first describe our implementation of a scalable power measurement framework that has enabled us to examine real power use (current draw). [Using this framework, samples were obtained at a per-node (socket) granularity, at frequencies of up to 100 samples per second.] Additionally, we describe how we applied this capability to implement power conserving measures on our Catamount Light Weight Kernel, where we achieved an 80% improvement. This ability has enabled us to quantify the amount of energy used by applications and to contrast application energy use between a Light Weight and General Purpose operating system. Finally, we show application energy use increases proportionally with the increase in run-time due to operating system noise. Areas of future interest will also be discussed.**

I. Introduction

Power has become, and will likely remain, one of the primary considerations in architecting High Performance Computing (HPC) platforms. The following is a selected list of recent capability class platforms[1] and corresponding power requirements.

- Red Storm, Sandia National Laboratories 2.506 Mega-Watts[1], [2]
- Blue Gene/L Lawrence Livermore National Laboratories 2.330 Mega-Watts[3], [2]
- Jaguar, Oak Ridge National Laboratories 6.951 Mega-Watts[4], [2]
- Road Runner, Los Alamos National Laboratories 2.483 Mega-Watts[5], [2]

While these platforms represent a range of theoretical peak performance per watt, it is clear the amount of power required to achieve capability class performance is substantial and growing. Projections for multi-PetaFLOPs platforms can exceed tens of Mega-Watts. These facts provide clear motivation for investigation in this area.

Power is proportional to the product of the Capacitance, Frequency and Voltage squared. Manipulated individually or in combination, these factors affect power usage. Over time, increasing CPU frequencies have necessitated voltage increases. Leakage currents have grown in proportion to the increase in processor frequencies, number of transistors, and decrease in die size. To address these issues, industry has made a conscious decision to continue to increase the number of transistors (in accordance with Moore's Law) by delivering multi-core sockets while keeping frequencies static, or in many cases lowering them (on a per core basis). While the sum performance continues to increase, exploiting the performance requires greater levels of parallelism. This direction has presented challenges to the HPC community in many areas but provides some opportunity for power savings as we will discuss.

The first step in addressing any issue is the ability to quantify the problem. The Cray XT line of hardware provides a rare opportunity for instrumentation and measurement that will be discussed in Section II. Once able to see effect, we set out to affect a change. The results of our modification of the Catamount[6] Light Weight Kernel (LWK) operating system and comparisons to Compute Node Linux (CNL)[2] are outlined in Section III. In addition to observing operating system power draw, this work has allowed us to characterize application power use. In Section IV we will discuss our observations related to HPC applications. In Section V we apply these methods in an effort to analyze the effects of operating system noise on application power usage.

We feel this work has provided a new capability and perspective on a field that is being actively researched. Section VI compares and contrasts related research efforts. This new window has afforded a unique view that has answered questions and posed many more. Sections VII and VIII will present some concluding remarks and outline future work in

[1]Systems designed to support applications that use a significant fraction of the total resource in support of a single cooperating application.

[2]Cray Inc custom port of Linux, part of the Cray Linux Environment (CLE)™

this area.

The main contributions of this work are:

1) A scalable mechanism for fine-grained, high frequency measurement of power on a Peta-FLOPs class system.
2) A description of how this capability was used to significantly reduce idle power in a LWK operating system.
3) The ability to quantify and visualize the energy usage of individual applications (on a per node, or per job, basis).
4) A demonstration of how power usage scales with the effects of operating system noise.

II. Instrumentation

A. Hardware

Unlike typical commodity hardware, the Cray XT3/4/5 node boards provide interfaces that can be exploited to measure real power usage (current draw) over time. Each node board has an embedded processor called an L0. The L0 has the ability to interface with many on board components. For this effort, we will specifically leverage the i2c serial bus interface from the L0 to the Voltage Regulator Modules (VRM) (Each processor socket has an associated VRM). In addition to an L0 on each node board, every Cray XT cabinet has an embedded processor at the cabinet level called the L1. Each L1 acts as a parent responsible for all the L0 components in the cabinet. At the top of the Reliability Availability and Serviceability (RAS) hardware hierarchy is the System Management Workstation (SMW) which in turn provides the parent role for all the L1's in the system. Our goal is to collect per socket current draw measurements from each associated VRM. This foundation should provide sufficient scalability for the collection of power data.

The following figure (Figure 1) is a depiction of the CRMS hardware hierarchy. A single SMW appears at the top of the hierarchy. The 2nd level of the hierarchy depicts the L1 controllers at the cabinet level.[3] The 3rd level depicts the L0 controllers. Each L0 controller physically exists on a node board. The figure depicts an L0 on a compute node board responsible for four nodes and the associated VRM's.

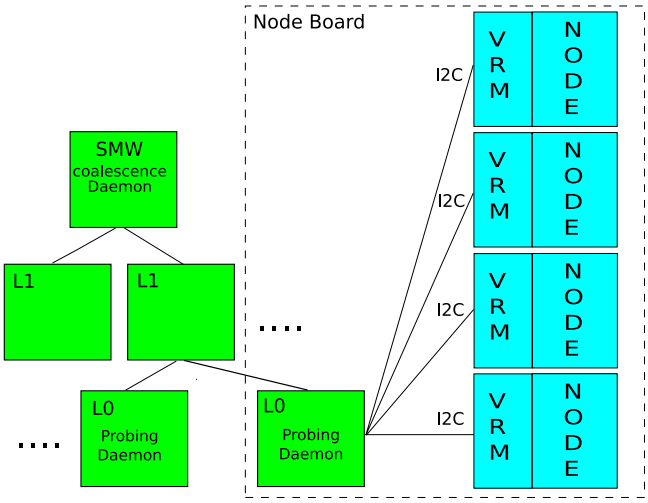

Fig. 1. CRMS Hardware Hierarchy

B. Software

While the ability to exploit the hardware (collect current draw data) is not currently a feature provided by the **C**ray **R**eliability Availability and Serviceability **M**anagement **S**ystem (CRMS), the existing software infrastructure can be leveraged to provide the desired instrumentation.

The CRMS consists of a number of persistent daemons which communicate in a hierarchical manner to provide a wide range of control and monitoring capabilities. We have augmented the base CRMS software with a *probing* daemon that runs on each L0 and a single *coalescence* daemon that runs on the SMW. (See Figure 1) The *probing* daemon registers a callback with the event loop executing in the main L0 daemon process (part of the standard CRMS) to interrogate the VRM at a specific bus:device location (corresponding to each individual processor socket). The results of a series of timed probes are combined and communicated through the event routers to the *coalescence* daemon on the SMW, which outputs the results. The output is a formatted flat file with timestamped hexadecimal current and voltage values for each CPU socket monitored (results are per socket not per core).

By leveraging the existing hardware and software foundation of the CRMS in this way we have been able to achieve a per socket collection granularity at a frequency of up to 100 samples per second[4]. The accuracy of each sample is approximately +/-2 amperes. While the accuracy of the sample is not as precise as we would like, the data remains extremely valuable for general magnitude observations and has proven to be quite valuable for relative comparisons. This is in contrast with most other platforms where measuring current draw is typically limited to insetting a meter between a power cable and energy source which results in a very coarse measurement capability at best. (It should be noted, we can also obtain current draw for the Network Interface Controller (Seastar) but we have observed a constant current draw for this device. This information is useful to quantify the total power used but does not vary depending on usage. The current draw measurements include memory controller activity but not power used by the memory banks themselves.) The granularity and frequency of this sampling capability has enabled us to observe real power usage in new and powerful ways.

We have closely monitored the impact our instrumentation has had on CRMS. At 100 samples per second we have seen little impact on the L0. Likewise, no adverse impact on communication between the L0's and L1 controllers, or between the L1's and the SMW has been observed. We have tested this instrumentation on up to 5 XT cabinets (480 nodes) and have observed no scaling issues. With this said, until tested at larger scale we cannot confirm functionality beyond what

[3]The ellipses indicates additional devices at this level and at the L0 level

[4]data included in this paper reflects a sampling frequency range from 1 to 100 samples per second

we have reported.

III. Idle Power Draw

The Linux™community has long been concerned with power saving measures particularly in the mobile computing sector. Linux has been quick to leverage architectural features of microprocessors to reduce power consumption during idle cycles. The HPC community makes great use of Linux on many of their platforms but LWK's are often used to deliver the maximum amount of performance at extreme scale (Red Storm and Blue Gene/L, for example). To achieve greater performance at scale, LWK's often have a selective feature set when compared to general purpose operating systems like Linux. As a result, LWK's are a prime area for investigating opportunities for power savings as long as performance is not affected. In the area of idle power usage Linux serves as an established benchmark. Our first goal will be to match or beat the idle current draw of Linux.

Once in place, we applied the previously described instrumentation to examine the current draw of our Catamount LWK. Catamount is the most recent generation of a long line of LWK operating systems designed and developed at Sandia National Labs (Performance at scale, a key design point from the start). Our initial findings were not surprising. As we suspected, but could not previously quantify, idle cycles were consuming current as Catamount busily awaits new work.

One of the advantages of most LWK's (Catamount is not an exception) is the relative simplicity of the operating system. The last two versions of Catamount (Catamount Virtual Node (CVN) and Catamount N-Way(CNW))[5] have supported multi-core sockets. The architecture of Catamount is such that there are only two regions the operating system enters during idle cycles. We first addressed the region where cores greater than 0 (in a zero based numbering scheme) enter during idle. (We will call core 0 the *master* core and cores greater than 0 *slave* cores.) We modified Catamount to individually halt *slave* cores when idle and awaken immediately when signaled by the *master* core. The result was a significant savings in current draw. As the number of cores per socket increase the savings will likely increase on Capability platforms. Capability class applications are typically memory and/or communication bound. Adding more cores, generally, provides little benefit and applications often run on one or two of the available cores. It should be emphasized that each *slave* core enters and returns from the halt state independently, resulting in very granular control on multi and many core architectures. After these very positive results, we then modified the region of the operating system the master core enters during idle. While the master core is interrupted on every timer tick (the slaves are not) we still observed significant additional power savings during idle periods.

[5]The name Catamount will generally be used throughout this document unless the more specific names CVN and CNW are nessary to point out and important distinction. Further specific information about Catamount can be found in [6]

Figure 2 depicts measurements obtained running three applications (HPL[7], PALLAS[8] and HPCC[9]) on a Dual Core AMD Opteron Processor[6] using CNL. Figure 3, in contrast, illustrates the results obtained when executing the same three applications on the same CPU using Catamount. (In our testing we typically compare results using the same exact hardware in an attempt to limit variability of measured results.)

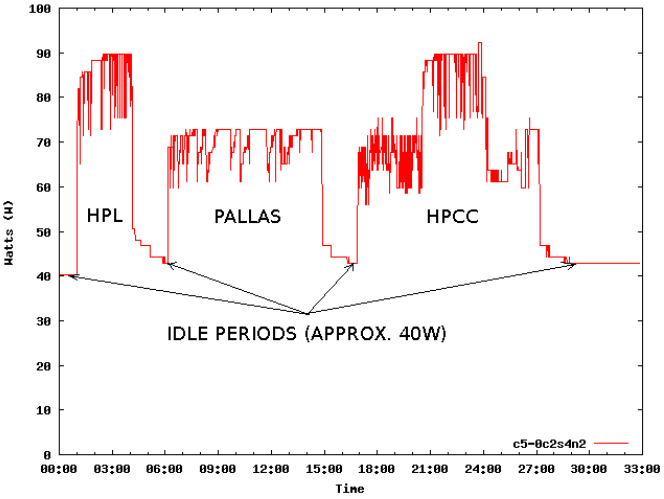

Fig. 2. Compute Node Linux (CNL)

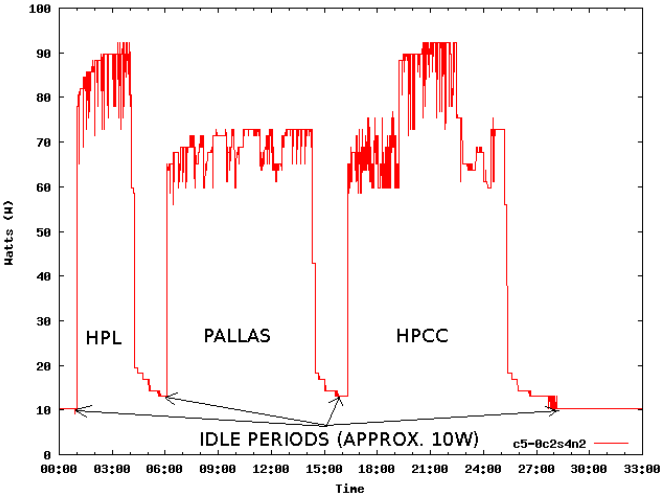

Fig. 3. Catamount Virtual Node (CVN)

The most noticeable difference between the two graphs is the idle power wattage. CNL uses approximately 40W when idle in contrast to Catamount which uses approximately 10W (prior to our operating system modifications Catamount used approximately 60W). Later results obtained on quad core AMD Opteron[7] sockets showed nearly identical idle power

[6]AMD Opteron 280 AMD Dual-Core Opteron 2.4GHz 2M Cache Socket 940 OSA280FAA6CB

[7]AMD Opteron Budapest 2.2 GHz socket AM2

wattage use for both CNL and Catamount[8](delta within accuracy of measurement). On this particular dual core architecture the instructions `MONITOR` and `MWAIT` are not supported. Both instructions are supported on the quad core architecture used in subsequent testing. Linux can be configured to poll, halt or use MONITOR/MWAIT during idle. It is possible that what we are observing in Figure 2 is a polling loop which in Linux is optimized to conserve power. Later observations on the quad core architecture were likely the result of CNL exploiting MONITOR/MWAIT. Regardless, these results are intended to show the ability to observe and contrast. These measurements have demonstrated our first goal of equaling the idle power savings of Linux.

These results also provided our first look at what we have termed Application Power Signatures (see Section IV). Each application has a characteristic signature. While small differences in the signature can be observed (even when running the same application on a different operating system) the signature is easily recognized.

A few more subtle points should be made. Without the ability to examine power usage at this level we could only guess that Catamount was inefficient during idle periods and we could not quantify the efficiency. Additionally, we would not have been able to easily measure the effect of our modifications and determine, definitively, when or if we reached our goal. Likewise, when using CNL, we could make the assumption that CNL benefits from power saving features of Linux but without this capability we would not have recognized the difference in power use between the two CPU architectures.

Using the information obtained we can make some simple calculations for a hypothetical system. For the purposes of this calculation we make the following assumptions: a 13,000 node (dual core), 80% utilized, 20% idle, ignoring downtime. The idle node hours for this system over a year would be:

$$(1300\,nodes * 0.2) * (365\,days/year * 24\,hours/day) = 22.776 * 10^6\,node\,hours/year \quad (1)$$

If we calculate the idle Kilo-Watt hours saved based on 50W per node (the delta between the pre-modified Catamount idle wattage and the modified Catamount idle wattage) we get:

$$(22.776 * 10^6\,node\,hours/year * 50\,Watts/node) \div 1000 = 1.1388 * 10^6\,KW\,hours/year \quad (2)$$

Assuming 10 cents per Kilo-Watt hour based on Department of Energy averages for 2008[10] we can calculate real dollar savings for this hypothetical system.

$$(1.1388 * 10^6\,KW\,hours/year * 10\,cents/KW\,hour) \div 100\,cents/dollar = \mathbf{113880\,dollars/year} \quad (3)$$

For a capability system using a figure of 80% utilization in the way we have characterized is probably very optimistic. Capability systems are typically intended to support one to several large applications at one time which tends to drive the total resource utilization numbers down. Additionally, this calculation does not consider idle cores resulting from applications that use less than the maximum cores available per node (as previously discussed). In the case of dual core sockets half of the resource could remain idle (in power saving mode) when the system is considered to be 100% utilized. In the case of quad core sockets three fourths of the resource could potentially remain idle. Figure 4 illustrates incremental power usage on a quad core socket when a short HPL job is executed on one, two, three and four cores of a quad core node.

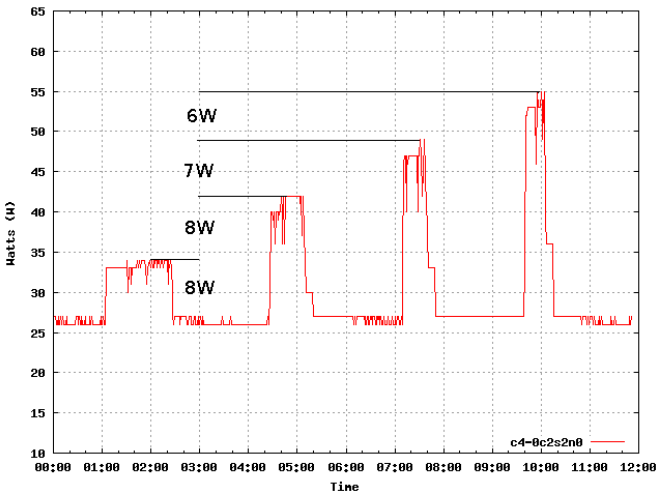

Fig. 4. Catamount N-Way Per Core Power Utilization

Even though our measurements are on a per node basis we can see the incremental rise in power usage when additional cores are enlisted. These results provide both a nice illustration of per core savings and a confirmation that our operating system modifications properly handle per core idle states.

In addition, we have not considered the 30-40% additional power savings as a result of not having to remove the additional heat generated by higher idle wattages. By exploiting power saving measures, as we have illustrated, significant savings can be realized by targeting idle cores alone.

IV. Application Power Signatures

Application Power Signature is a term we have applied to the measured power usage of an application over the duration of that application. The term signature is used since each application exhibits a repeatable and somewhat distinct shape when graphed. We have found that a user knowledgeable

[8]CVN was enhanced to support more than two cores, the resulting Catamount version was named CNW. Unless otherwise specified all results shown after Figure3 were obtained running on CNW

of the application flow can easily distinguish phases of the application simply by viewing the signature. While simply graphing the resulting data can be useful, we have extended this by calculating the energy used over the duration of the application. We call this application energy. To calculate this metric we simply calculate the area under the curve. To accomplish this we enhanced our post processing code to approximate the definite integral using the trapezoidal rule. The following graphs (Figures 5 and 6) depict the data collected while running HPCC on Catamount and CNL. HPCC was executed using the same input file on the same physical hardware. Each run used 16 processors (four nodes, four cores per node).

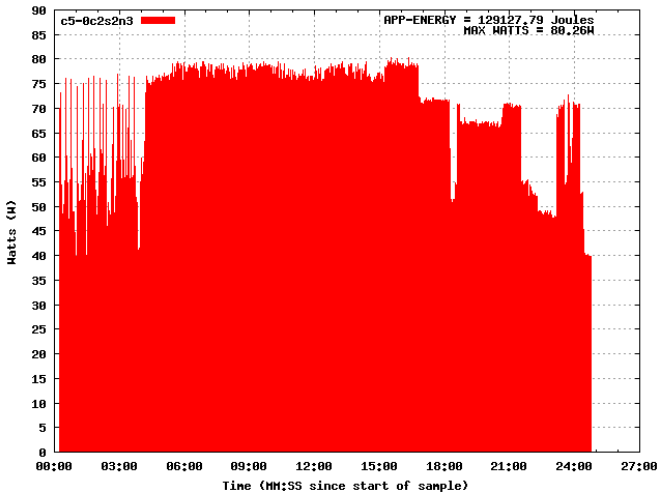

Fig. 5. HPCC on Catamount, Application Power Signature and Application Energy

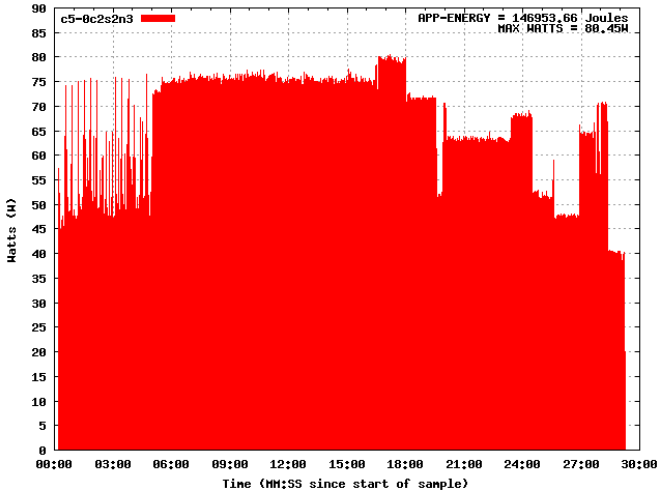

Fig. 6. HPCC on CNL, Application Power Signature and Application Energy

In the upper right hand corner of each graph is the energy used by the application (on a single node, all four cores). Again, notice the similarity of the signatures regardless of the underlying operating system. In this case HPCC finished more quickly on Catamount than CNL. HPCC and other applications have been shown to execute more quickly on Catamount[11]. It is not surprising that an application that takes longer to execute, given similar power draw during execution, will consume more power. In this case HPCC ran 16% faster on Catamount. The amount of energy used by HPCC is 13% less using Catamount than CNL. We also tested HPCC on quad core nodes using two cores per node (HPCC ran 15% faster on Catamount and used 13% less power) and on dual core nodes using two cores per node (HPCC ran 10% faster on Catamount and used 10% less power). The salient point is that performance is not only important in reducing the run time of an application but also in increasing the power efficiency of that application. Additionally, without the ability to examine real power use at this granularity the power efficiency of an application could not be as precisely quantified.

In the next Section (V) we will further analyze application energy by introducing operating system noise. Additionally, we will discuss plans for applying these concepts in Section VIII.

V. Power and Noise

Operating system interference, also referred to as noise or jitter, is caused by asynchronous interruption of the application by the system software on the node. This interruption can occur for a variety of reasons from the periodic timer "tick" commonly used by many commodity operating systems to keep track of time to the scheduling points used to replace the currently running process with another task or kernel daemon.

The detrimental side effects of operating system interference on HPC systems have been known and studied, primarily qualitatively, for nearly two decades [12], [13]. Previous investigations have suggested the global performance cost of noise is due to the variance in the time it takes processes to participate in collective operations, such as `MPI_Allreduce`. LWK's, like Catamount, are essentially noise-less in comparison to general-purpose operating systems like Linux. Previous work has shown that operating system noise can have substantial impact on the performance of HPC applications [14]. In addition, this work shows the impact varies by application, some showing relatively no impact in noisy environments while others exhibit exponential slowdowns. While many aspects of the impact of noise on run time performance are well understood, the impact of noise in terms of power usage is not. Specifically, we set out to answer if power usage in noisy environments scales linearly (or otherwise) with the increase in application run time.

To evaluate the impact of noise we use the kernel-level noise injection framework built into the Catamount LWK [14]. This framework provides the ability to direct the operating system to inject various per-job noise patterns during application execution. The available parameters for generating the noise patten include: the frequency of the noise (in Hz), the duration of each individual noise event (in us), the set of participating nodes, and a randomization method for noise patterns across

nodes (not employed for this analysis). The noise is generated (simulated) using a timer interrupt on core 0 of the participating nodes. When the interrupt is generated, Catamount interrupts the application and spins in a tight busy-wait loop for the specified duration. The purpose of separately specifying the frequency and duration of each noise event is to simulate various types of noise that occur on general purpose operating systems. Catamount provides an ideal environment for these studies due to its extremely low native noise signature.

In the following analysis we focused on a single application (SAGE). We chose SAGE based on our initial studies and the previous analysis done in [14]. SAGE, SAIC's Adaptive Grid Eulerian hydro-code, is a multi-dimensional, multi-material, Eulerian hydrodynamics code with adaptive mesh refinement that uses second-order accurate numerical techniques [15]. SAGE represents a large class of production applications at Los Alamos National Laboratory. A large-scale parallel code written in Fortran 90, SAGE uses MPI for inter-processor communications and routinely runs on thousands of processors for months at a time. Applications like SAGE have the potential to be significantly impacted by noise and any proportional increase in energy.

Table I is a representative sample of our results.

TABLE I
POWER IMPACT OF NOISE

Noise	Freq	Duration	Diff Runtime	Diff App Energy (AVG)
2.5%	10Hz	2500us	4.0%	4.0 %
1%	10Hz	1000us	1.7%	1.9%
2.5%	100Hz	250us	2.6%	2.5%
2.5%	1000Hz	25us	2.6%	2.5%
1%	1000Hz	10us	0.1%	0.1%
10%	**10Hz**	**10000us**	**21.6%**	**21.0%**

We injected a number of different noise patterns varying the frequency and duration of the noise. The Noise percentage (column one) is determined using the following calculation.

$$((Frequency(Hz) * Duration(us)) \div (1 * 10^6)) * 100 \quad (4)$$

The frequency of the noise (column two) is how often a noise event occurs. The duration (column three) is how long each noise event lasts. The difference in runtime is shown in column four and is relative to the runtime of the application with no noise injected. Likewise, the difference in application energy (column five) is relative to the energy used by the application without noise injected. The results, with the exception of row six, are representative of multiple runs on the same equipment using the same parameters. In addition, we varied the runtime of the application with consistent results. The results were obtained using 16 quad core nodes. The application utilized core 0 only since noise can only be injected on core 0 using this framework. What we observed is that the difference in application energy used by applications when noise is injected is linearly proportional to the difference in runtime. If we normalize the impact of the injected noise, even in the most extreme example (again excluding row six) the impact of noise on both the runtime and the application energy is approximately 1.5%. We found these results to be very consistent. We repeated our tests at a larger scale (48 nodes, again utilizing only core 0) and observed results consistent with Table I. In an effort to simulate effects seen at larger scale we introduced a large amount of noise (10%) while running the same application. The results (row six of table I) show a larger impact to both runtime and application energy (approximately 11% when normalized). These results are significant in the fact that they show the same linearly proportional increase in application energy for applications effected by noise. Though Table I shows small percentage increases in runtime for various noise patters, accompanied by proportional increases in the percentage of energy used, these results were obtained at a relatively small scale. The run time of some applications can increase dramatically at larger scale in noisy environments.

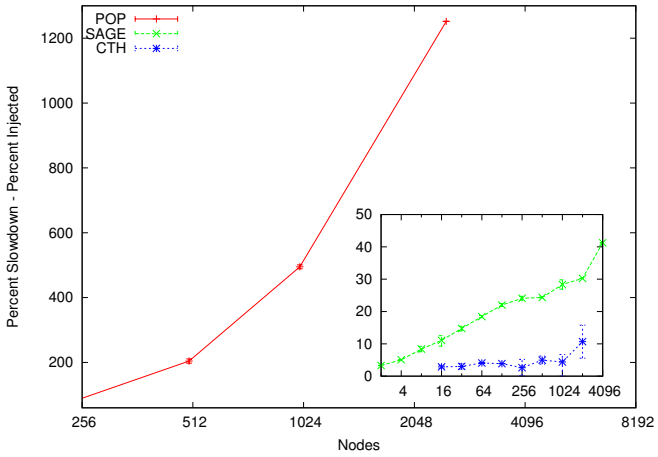

Fig. 7. Slowdown at Scale

In Figure 7, we see the measured slowdown of POP, CTH, and SAGE, at scale. In this figure the Y-axis is the global accumulation of noise for the application. We compute this global accumulation by taking the slowdown of the app in a noisy environment versus a baseline with no OS noise and subtract the amount of locally injected noise. For example, if we inject a 2.5% net processor noise signature, as we did for this figure, and measure a 20% slowdown the global accumulation of noise would be 17.5%. We see in this figure that with only 2.5% net processor noise injected the slowdown for POP exceeds 1200% at scale, and therefore we can project a proportional increase of 1200% application energy at scale. The inset of Figure 7 also shows considerable slowdowns for SAGE and CTH due to noise. While not as dramatic as POP, the additional impact on application energy projected by our analysis is proportionally as significant. Further analysis will need to be accomplished to verify that these results are truly representative at scale.

VI. Related Work

Power, as it relates to computers and computation, has been researched from many perspectives. The use of performance counters to estimate power efficiency has been researched from a micro [16], [17] and macro [18] perspective. While estimates based on performance counters have shown to be useful, we have shown that scalable fine-grained measurements can be leveraged at both the micro and macro level to analyze HPC operating system and application power use. Our work would strengthen these modeling efforts.

Previous work done in [19] and [20] had similar motivations to our efforts and targeted similar scale systems. In [20], they nicely evaluate various methods of measuring power. They conclude that measuring power at system scale is problematic. Nodal and very small scale measurements can be accomplished using power meters and cabinet level measurements (very coarse) can be accomplished, but the scalable collection of samples is not feasible. Both [19] and [20] extrapolate nodal level measurement to produce larger system power estimates. Our work has shown that it is possible to collect scalable, granular, high-frequency power measurements on HPC platforms. Additionally, our work shows that nodal and system level measurements can be obtained in-situ. Further, by enabling collection from all nodes used in an application we can observe and quantify the additional power effects of parallel applications rather than extrapolate based on nodal measurements. By leveraging what will hopefully become a ubiquitous capability in the future power analysis of both operating system and applications on HPC systems would be greatly enhanced.

VII. Conclusions

Our results have shown that once observation is enabled, beneficial effects can be achieved with relatively little effort and subsequently quantified. This capability has provided new insight into operating system and application analysis. It is our intent to employ this capability in future efforts (Section VIII) to increase the efficiency of current and future platforms.

We feel the most important aspect presented in this paper is the ability to measure the actual current draw at a high level of granularity and frequency. Without this ability, the work described would have to be done without the ability to see effect or quantify results. It is also important to note that without the underlying hardware capability to measure current draw the instrumentation would not be possible. It is our hope that this capability, both hardware and software, will be found on more platforms in the future.

VIII. Future Work

This initial work has inspired additional efforts in a number of related areas. In addition to reducing idle power consumption, reductions during application execution might prove valuable. Even on the most well balanced system, capability class applications experience periods where nodes are waiting for information from cooperating nodes of the same application. We are investigating ways for applications to signal the operating system to enter power saving states (or lower frequency levels), while blocked, and quickly return to a running state when prepared to continue. The ability to measure power draw during these periods will help us implement, subsequently test and quantify this ability.

We also plan on experimenting with frequency scaling during application execution. Our primary goal here is to reduce frequency such that application performance remains unaffected. If this is not possible, a small impact on application performance may be acceptable given a large increase in application power efficiency. Again, the ability to measure our impact during implementation and testing will be critical to success in this area.

Finally, we plan to apply our ability to calculate application energy to areas such as resource scheduling. For example, as stated previously, capability class systems are destined to require huge amounts of power. While running High Performance Linpack requires a large percentage of the maximum CPU power, typical applications require less than 75% of maximum power (our estimates are as low as 60% supported by [21]). A platform that requires a peak power of 10 Mega-Watts could be scheduled in such a way as to maintain a maximum power draw of 7.5 Mega-Watts, for example, with no impact on application performance or runtime. We could likely maintain an even lower percentage of peak. Related work has been done in this area for real-time and embedded systems [22], [23]. Other work [24] targets similar efforts using dynamic voltage and frequency scaling (previously mentioned as another area of future interest).

Acknowledgements

We would like to thank Sudip Dosanjh, James Ang and Doug Doerfler for their support of this research. Additionally, the local Cray staff (Dick Dimock, Jason Repik, Victor Kuhns, Barry Oliphant, Bob Purdy) and Jeff Sampson, provided valuable help and hardware insight that assisted these efforts.

References

[1] RedStorm. Sandia National Labs. [Online]. Available: http://www.cs.sandia.gov/platforms/RedStorm.html

[2] (2008, Nov) Top500. [Online]. Available: http://www.top500.org/list/2008/11/100

[3] Blue-Gene/L. Lawrence Livermore National Labs. [Online]. Available: https://asc.llnl.gov/computing_resources/bluegenel

[4] Jaguar. Oak Ridge National Laboratories. [Online]. Available: http://www.nccs.gov/jaguar/

[5] RoadRunner. Los Alamos National Laboratories. [Online]. Available: http://www.lanl.gov/roadrunner/

[6] S. M. Kelly and R. B. Brightwell, "Software Architecture of the Light Weight Kernel, Catamount," in *Cray User Group*, May 2005.

[7] J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart, "High Performance Linpack HPL," 1989. [Online]. Available: http://www.netlib.org/benchmark/hpl/references.html

[8] PALLAS. [Online]. Available: http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/219848.htm

[9] HPCC. [Online]. Available: http://icl.cs.utk.edu/hpcc/

[10] DOE Energy Statistics. Department of Energy. [Online]. Available: http://www.eia.doe.gov/cneaf/electricity/epm/table5_6_a.html

[11] C. T. Vaughan, J. P. VanDyke, and S. M. Kelly, "Application Performance under Different XT Operating Systems," in *Cray User Group*, May 2008.

[12] R. Zajcew, P. Roy, D. Black, C. Peak, P. Guedes, B. Kemp, J. LoVerso, M. Leibensperger, M. Barnett, F. Rabii, and D. Netterwala, "An OSF/1 UNIX for Massively Parallel Multicomputers," in *Proceedings of the USENIX Technical Conference*, Winter 1993.

[13] F. Petrini, D. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q," in *Proceedings of SC*, 2003.

[14] K. B. Ferreira, R. Brightwell, and P. G. Bridges, "Characterizing Application Sensitivity to OS Interference Using Kernel-Level Noise Injection," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (Supercomputing'08)*, November 2008.

[15] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proceedings of the ACM IEEE conference on Supercomputing, Denver CO*, 2001.

[16] F. Bellosa, "The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems." in *ACM SIGOPS European Workshop*, September 2000.

[17] W. L. Bircher, M. Valluri, J. Law, and L. John, "Runtime Identification of Microprocessor Energy Saving Opportunities." in *International Symposium on Low Power Electronics and Design*, pp. 275–280.

[18] W. L. Bircher and L. K. John, "Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events," in *International Symposium on Performance Analysis of Systems & Software*. University of Texas at Austin, April 2007.

[19] X. Feng, R. Ge, and K. W. Cameron, "Power and Energy Profiling on Scientific Applications on Distributed Systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium IPDPS*. University of South Carolina, Columbia SC, 2005.

[20] S. Kamil, J. Shalf, and E. Strohmaier, "Power Efficiency in High Performance Computing," in *IEEE International Symposium on Parallel and Distributed Processing*, 2008.

[21] X. Fan, W.-D. Weber, and L. A. Barroso, "Power Provisioning for a Warehouse-sized Computer," in *The 34th ACM International Symposium on Computer Architecture*, 2007.

[22] H. hung Lin and C.-W. Hsueh, "Power-Aware real-Time Scheduling using Pinwheel Model and Profiling Technique," in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2005.

[23] J. Liu, P. H. Chou, N. Bagherzardeh, and F. Kurdahi, "Power-Aware Scheduling under Timing Constraints for Mission-Critical Embedded Systems," in *Proceedings of the 38th conference on Design automation*. Dept. of Electrical & Computer Engineering, University of California at Irvine, Irvine, CA, 2001.

[24] C. hsing Hsu and W. chun Feng, "Power-Aware Run-Time System for High-Performance Computing," in *Conference on High Performance Networking and Computing*, 2005.