Investigating Real Power Usage on Red Storm

James H. Laros III, Kevin T. Pedretti,
Suzanne M. Kelly, John P. Vandyke, Courtenay T. Vaughan
Sandia National Laboratories

Mark Swan - Cray Inc.

***ABSTRACT*: High Performance Computing (HPC) has historically been the impetus for new technologies. However, in dealing with power related issues HPC has lagged behind. Power has recently been recognized as one of the major obstacles to fielding a Peta-Flop class system. In this paper we will discuss power related topics by leveraging what is currently a rare capability of examining real power usage at a very granular level on an HPC platform. We will discuss what we have observed, initial efforts to implement power conserving measures, quantifications and comparisons between Light Weight and General Purpose Operating Systems and, finally, areas of investigation that have been enabled by this novel capability.**

***KEYWORDS*: High Performance Computing (HPC), Power, Power Efficiency, Cray XT3/XT4/XT5, Multi-Core**

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[1] class platforms and corresponding power requirements.

- Red Storm, Sandia National Laboratories 2.506 Mega-Watts[1], [2]
- Blue Gene/P 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-PetaFLOP 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 has necessitated voltage increases. Leakage currents have grown proportionally 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] LWK OS and comparisons to Compute Node Linux (CNL)[2] are outlined in Section III. In addition to observing OS power draw, this work has allowed us to characterize application power use. In Section IV we will discuss our observations related to HPC applications.

This new window has afforded a unique view that has answered questions and posed many more. Sections V and VI will present some concluding remarks and outline future work in this area.

II. Instrumentation

A. Hardware

Unlike typical commodity hardware, the Cray XT4/5 node board provides hardware 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 (node) current draw measurements from each associated VRM. This foundation should provide sufficient scalability for the collection of power data.

[1]Capability Class Platform - 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

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.

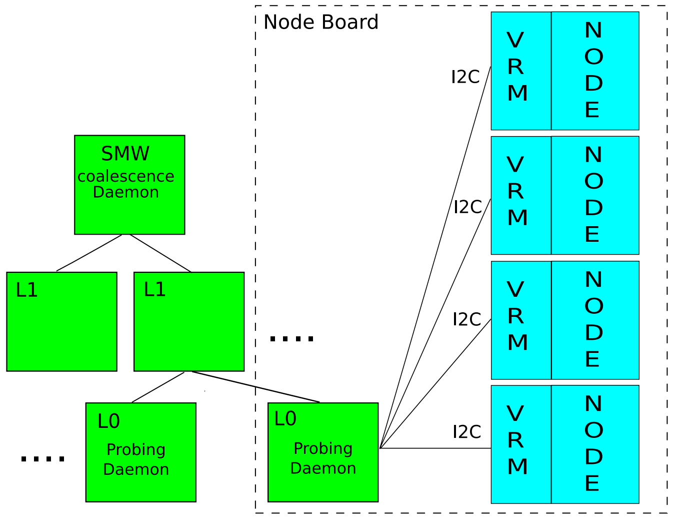

Fig. 1. CRMS Hardware Hierarchy

B. Software

Unfortunately, the ability to exploit the hardware (collect current draw data) is not presently a feature provided by the **C**ray **R**eliability Availability and Serviceability **M**anagement **S**ystem (CRMS). While not currently a feature of the 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 formated 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 20 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. In contrast with most other platforms, measuring current draw is typically limited to inserting a meter between a power cable and energy source, resulting in a very coarse measurement capability at best. The granularity and frequency of this sampling capability has enabled us to observe real power usage in new and powerful ways.

III. Idle Power Draw

The Linux^{TM}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. HPC makes great use of Linux on many of their platforms but Light Weight Kernels (LWK) are often used to deliver the maximum amount of performance at extreme scale (Red Storm and Blue Gene/P for example). To achieve greater performance at scale, LWK's often have a selective feature set when compared to general purpose Operating Systems (OS) 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 effort 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 OS's 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 is busily awaiting new work.

One of the advantages of most LWK's (Catamount is not an exception) is the relative simplicity of the OS. The last two versions of Catamount (Catamount Virtual Node (CVN) and Catamount N-Way(CNW)) have supported multi-core sockets. The architecture of Catamount is such that there are only two regions the OS enters during idle cycles. We first addressed the region that 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

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

[4]data included in this paper reflects a sampling frequency of one sample per second

emphasized that each *slave* core enters and returns from the halted stated independently, resulting in very granular control on multi and many core architectures. After these very positive results we then modified the region of the OS 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 savings during idle periods.

Figure 2 depicts measurements obtained running three applications (HPL[7], PALLAS[8] and HPCC[9]) on a Dual Core AMD Opteron Processor[5] 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 from using the same exact hardware in an attempt to limit variability of measure results)

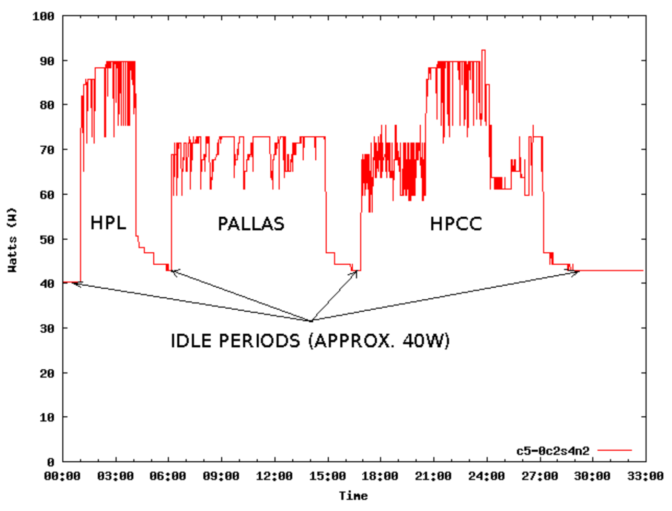

Fig. 2. Compute Node Linux (CNL)

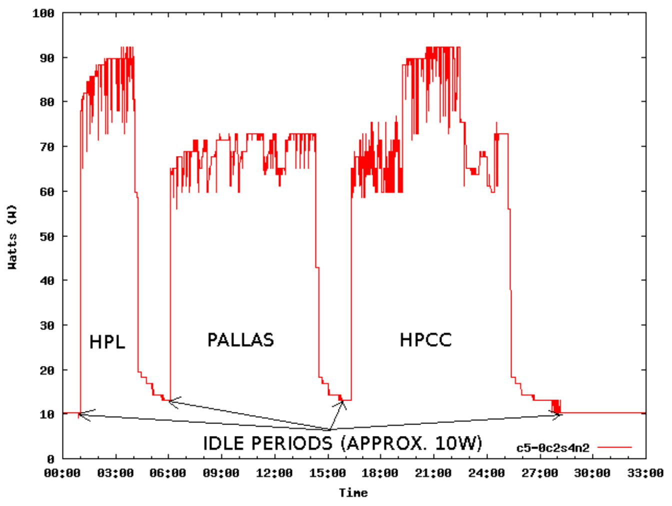

Fig. 3. Catamount Virtual Node (CVN)

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

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 OS modifications Catamount used approximately 60W). Later results obtained on quad core AMD Opteron[6] sockets showed nearly identical idle power wattage use for both CNL and Catamount[7](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 OS 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, we could not quantify the efficiency. Additionally, we would not have been able to so 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 (based on the delta between the pre-modified Catamount idle wattage and the modified Catamount idle wattage) we get:

[6]AMD Opteron Budapest 2.2 GHz socket AM2

[7]CVN was enhanced to support more than two cores, the resulting Catamount version was named CNW. Unless otherwise specfied all results shown after Figure 3 were obtained running CNW.

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

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.

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

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 a few 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 remains 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 remains idle. Figure 4 illustrates incremental power usage on a quad core socket.

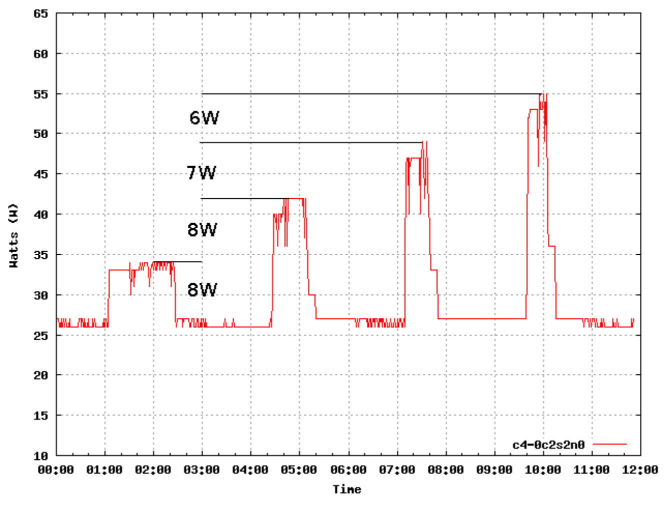

Fig. 4. Catamount N-Way Per Core Power Utilization

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 OS 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 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 CNL and Catamount. HPCC was executed using the same input file on the same physical hardware. Each run used 16 processors (four nodes, four cores per node).

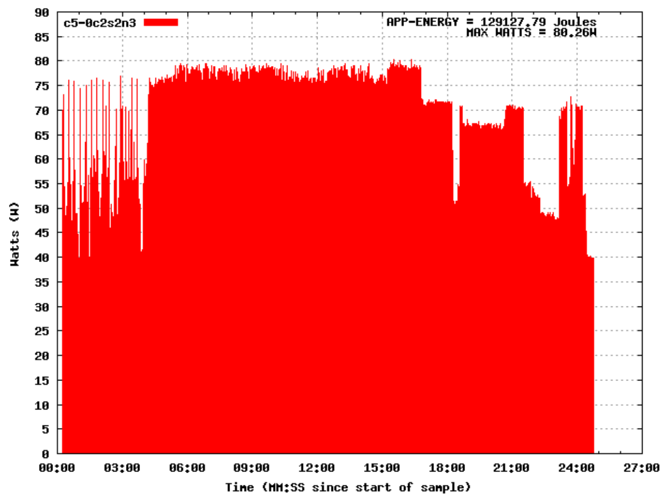

Fig. 5. HPCC on Catamount, Application Power Signature and Application Energy

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 OS. 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

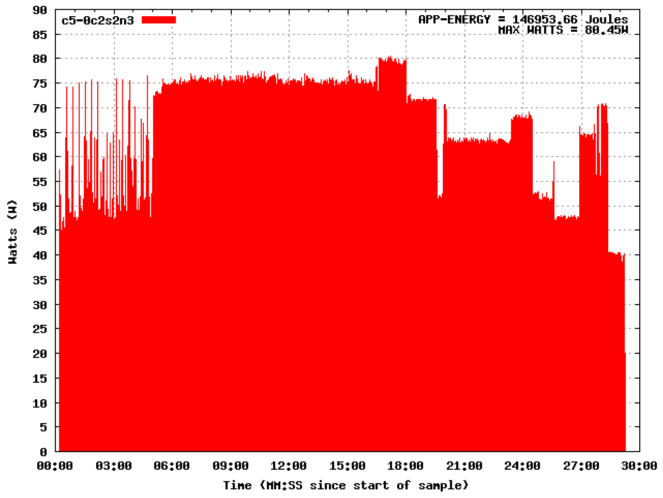

Fig. 6. HPCC on CNL, Application Power Signature and Application Energy

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 sufficiently quantified.

In Section VI we will discuss our plans for applying some of these concepts.

V. 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 VI) 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.

VI. 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 [12]). 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 run-time. We could likely maintain an even lower percentage of peak. Related work has been done in this area for real-time and embedded systems [13], [14]. Other work [15] 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.

About the Author

James H. Laros III is a Principle Member of the Technical staff at Sandia National Laboratories. His current interests include research and development in operating systems and systems software.

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] 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.
[13] 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.
[14] 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.
[15] 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.