7-2-2012

Measuring and tuning energy efficiency on large scale high performance computing platforms

James Howard Laros III

James Howard Laros III

Candidate

Electrical and Computer Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Wei Shu , Chairperson

Dr. Howard Pollard

Dr. James Ang

Measuring and Tuning Energy Efficiency on Large Scale High Performance Computing Platforms

by

James H. Laros III

A.A.S., Drafting and Design Technology,
Northampton County Area Community College, 1982
B.S., Computer Science, Chapman University, 1998

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Engineering

The University of New Mexico

Albuquerque, New Mexico

May, 2012

Dedication

This work is dedicated to my wife Janet. Your love and support is the only reason this journey of so many years has become a reality.
To my daughter Nicolette, nothing I have experienced in this life has compared to the privilege of watching you grow into the person you have become.

Acknowledgments

I would like to sincerely thank and acknowledge the following people who have contributed in various ways, all of which significant, to the successful completion of this Thesis: Dr. Wei Shu - Thesis advisor, committee chair and collaborator, University of New Mexico Electrical and Computer Engineering Department; Dr. Howard Pollard - Committee member, University of New Mexico Electrical and Computer Engineering Department; Dr. James A. Ang - Committee member and Manager of the Scalable Computer Architectures Department at Sandia National Laboratories; Kevin Pedretti, Sue Kelly, John Vandyke, Kurt Ferreira and Courtenay Vaughan - Collaborators, Technical Staff Sandia National Laboratories and Mark Swan - Collaborator, Cray Inc.

The following agencies have provided funding directly or indirectly to this research: National Nuclear Security Agency (NNSA) Advanced Simulation and Computing (ASC) program and the Department of Energy's (DOE) Innovative and Novel Computational Impact on Theory and Experiment (INSITE) program. Sandia National Laboratories Center 1420 Sudip Dosanjh Senior Manager, Department 1422, James Ang manager and Department 1423 Ronald Brightwell manager.

Measuring and Tuning Energy Efficiency on Large Scale High Performance Computing Platforms

by

James H. Laros III

A.A.S., Drafting and Design Technology,

Northampton County Area Community College, 1982

B.S., Computer Science, Chapman University, 1998

M.S., Computer Engineering, University of New Mexico, 2012

Abstract

Recognition of the importance of power in the field of High Performance Computing, whether it be as an obstacle, expense or design consideration, has never been greater and more pervasive. Research has been conducted in a number of areas related to power. Little, if any, existing research has focused on large scale High Performance Computing. Part of the reason is the lack of measurement capability currently available on small or large platforms. Typically, research is conducted using coarse methods of measurement such as inserting a power meter between the power source and the platform, or fine grained measurements using custom instrumented boards (with obvious limitations in scale). To collect the measurements necessary to analyze real scientific computing applications at large scale, an in-situ measurement capability must exist on a large scale capability class platform.

In response to this challenge, the unique power measurement capabilities of the Cray XT architecture were exploited to gain an understanding of power use and the effects of tuning both CPU and network bandwidth. Modifications were made at the operating system level to deterministically halt cores when idle. Additionally, capabilities to alter operating P-state were added. At the application level, an understanding of the power requirements of a range of important DOE/NNSA production scientific computing applications running at large scale (thousands of nodes) is gained, by simultaneously collecting current and voltage measurements on the hosting nodes. The effects of both CPU and network bandwidth tuning are examined and energy savings opportunities of up to 39% with little or no impact on run-time performance is demonstrated. Capturing scale effects was key. This thesis provides strong evidence that next generation large-scale platforms should not only approach CPU frequency scaling differently, but could also benefit from the capability to tune other platform components, such as the network, to achieve energy efficient performance.

Contents

List of Figures

List of Figures

List of Tables

Glossary

Capacity As in Capacity platform, refers to a platform that is primarily designed to execute a number of jobs concurrently that are individually much smaller than the total number of nodes the platform is comprised of.

Capability As in Capability platform, refers to a platform that is primarily designed to execute a single job on all the nodes the platform is comprised of.

Light Weight Kernels (LWK) Kernels that are generally designed to optimize performance on capability class platforms. They typically focus on providing the resources necessary for application execution with a focus on performance and scalability. Light weight kernels typically provide a small subset of the services provided by a general purpose operating system such as Linux.

Dynamic Voltage and Frequency Scaling (DVFS) In this context refers to the ability for a component, typically a CPU, to provide an automatic or directed frequency scaling capability that results in the potential for a reduced input voltage.

Dynamic Voltage Scaling (DVS) Refers to the platform or infrastructure ability to reduce the input voltage to a component in response to a

component frequency change. See also, Dynamic Voltage and Frequency Scaling.

Energy Delay Product (EDP) One method of providing a *unified* metric which combines performance and energy.

Advanced Power Management (APM) is an API developed by Intel and Microsoft which enables an operating system in conjunction with the BIOS to achieve power management. We will generally use this term to refer to the ability of a CPU to support frequency or P-state changes.

Chapter 1

Introduction

1.1 Motivation

There are three *walls* in CPU chip architecture design, memory[1], instruction level parallelism[2] and power. Power is widely accepted as the tallest of these walls. Currently, the power used by a CMOS circuit is dominated by dynamic power. As feature sizes shrink, static power (power used due to leakage current) will increasingly become an important factor. In addition, the vast majority of power introduced into the chip must be removed in the form of heat. As feature sizes shrink, this becomes more difficult. More heat must be removed from a smaller area. Addressing these issues (and others) for mobile and server CPU chip architectures has long established motivations.

The High Performance Computing (HPC) community has for many years ignored power as a major factor in favor of increased performance. With annual power costs on track to rival acquisition costs of next generation platforms, power has increasingly been identified by the HPC community as the *tall pole* in the path to Exascale systems. Ubiquitous in the top three considerations of virtually every report on

next generation or Exascale platforms, power has been recognized across the board by government agencies and commercial enterprises alike as possibly the greatest challenge in fielding future HPC platforms.

Existing hardware has been successfully leveraged by present day operating systems to conserve energy whenever possible but these approaches have proved ineffective and even detrimental at large scale. While hardware must provide part of the solution, how these solutions are leveraged on large scale platforms requires a new and flexible approach. It is particularly important that any approach taken has a system-level, rather than node-level, view of these issues.

1.2 Thesis Overview

In response to this challenge this research has leveraged a thus far unique ability to measure current draw and voltage, in-situ, on a large HPC platform at a very fine granularity and high frequency. The first experiment begins with the goal of reducing power consumption during idle cycles. This concept is extended to multi-core architectures by ensuring cores not in use remain in low power states during application execution. It is common for HPC applications to use fewer than the number of available cores per node. For many scientific applications, the memory wall in the form of capacity, capability or both force users to limit the number of cores per node to better balance their application memory requirements at scale. While conserving power during idle cycles can produce large energy and related cost savings,[1] this research has additionally endeavored to explore possible energy efficiencies during application run-time on active CPU cores.

[1]These changes were applied to the production Red Storm capability class platform at Sandia National Laboratories and the Cray XT4 platform at Pittsburgh Supercomputer Center and have reduced power related facility charges by one million dollars to date.

HPC applications are typically bulk synchronous. In [3], bulk synchronous programs are described to have three execution phases; computation, communication and synchronization. In [4] and [5], the authors describe how operating system noise can effectively slow overall computation of bulk synchronous programs. In short, the runtime of an application will be throttled by the slowest MPI rank involved in a bulk synchronous computation. Allowing frequency changes that are dictated locally, rather than from the systems perspective, can cause the equivalent of operating system noise (or jitter) adversely affecting application performance.

To avoid this potential performance impact, the second experiment modifies CPU frequency during application execution, but with a system level perspective. Initially, it was assumed to achieve significant savings, without unacceptable impacts in performance, frequency scaling would have to be performed during natural wait states in application execution. For example, during communication wait phases. This research concludes a less aggressive, and likely more stable, approach of static frequency scaling can produce large energy savings with little to no performance penalty.

The third and final experiment included in this thesis takes advantage of the ability to tune the performance of the network components of a large scale HPC platform. This research demonstrates a *sweet spot* exists for most, if not all, HPC applications run at large scale where maximum energy efficiency can be achieved without unacceptable performance trade-offs. A large amount of empirical data is provided to support this claim.

Evaluating acceptable trade-offs between energy efficiency and run-time performance is, of course, somewhat subjective. This research indicates that the parameters of these trade-offs are application dependent. While the HPC community has traditionally prioritized performance above all metrics, future per-processor or per-platform power requirements will likely alter priorities and place more importance on energy efficiency metrics like FLOPS/Watt, Energy Delay Product (EDP) and

energy or cost to solution. While this research is directed at exploring power savings potential, performance remains a critical parameter of evaluation.

All experiments were conducted on two Cray XT class platforms; Red Storm located at Sandia National Laboratories and Jaguar hosted by Oak Ridge National Laboratory. The results of these experiments clearly indicate that opportunities exist to save energy by tuning platform components, individually or together, while maintaining application performance. The ultimate goal is to reduce energy consumption of real applications run at very large scale while minimizing the impact on run-time performance (defined for our purposes as wall-clock execution time).

1.3 Related Work

Power, as it relates to computers and computation, has been researched from many perspectives. Certainly the largest body of research has been done in collaboration by Ge, Feng and Cameron (in some cases in association with other researchers). In [6] and [7] the authors introduce PowerPack. PowerPack is a framework for profiling and analyzing scientific applications on distributed systems. The authors frame the problem well, warning of future costs and stressing the need for component level analysis. In their research the importance of fine granularity component level measurement is recognized. It is clear that direct measurements are taken at node level. It is less clear that some individual components were directly measured. It appears the authors took great care in attempts to isolate the power draw for individual components from the overall nodal measurements. The frequency of sample collection in this research is good. While this research does rely on a direct measurement technique, only one node, of a 32 node cluster, was instrumented. The authors use a technique called "node remapping" in which they emulate measuring a single application run by executing an application that runs on M nodes M number of times

ensuring that node assignments are rotated for each execution. There are obvious scalability limitations to this technique but their analysis techniques are rigorous and their results valuable at small scale for similar clusters built with commodity components. The authors conclude that power has a direct relationship on reliability as stated by Arrhenius' Law[8] but there is evidence to the contrary in an extensive study done at Google[9]. This research shows there is not a strong correlation between temperature and reliability of disks. Intuitively, it would seem that components with moving parts would be more likely affected by temperature than a CPU, for example. Hsu in [10], however, claims informal empirical data supports Arrenhius' equation as applied to cluster systems. This remains an open question, but certainly lowering power, and therefore heat, is unlikely to increase failure rates based on currently published research. At a minimum, there is a resulting savings in heat removal.

In [11] the authors improve on the single node sampling technique described in [6] and [7] by expanding their collection capability to 16 nodes (this remains the limit on node count in the remaining publications). The NEMO power aware cluster is comprised of laptops which allow the necessary measurements to be taken. These measurements are supplemented and verified by contrasting them with readings obtained from an external power monitoring device. On the downside, total node power is the granularity for this cluster unless they additionally instrument a single node as described in [6] and extrapolate. The frequency of the power samples drops from four per second to one sample every 15 to 20 seconds. Regardless, their configuration provides good small scale information. The authors nicely define three strategies of scheduling using Dynamic Voltage Scaling (DVS). The approach taken in this thesis equates to what they define as *external*, scheduling from the command line. They also evaluate *automatic* (scheduling accomplished by a daemon such as the Linux cpuspeed daemon) and *internal* (scheduling initiated by the application) scheduling approaches.

Another important aspect of the approach taken by these researchers is their attempts to provide a useful single or fused metric to gauge success. Energy Delay Product (EDP), initially proposed by Horowitz[12] to evaluate trade-offs between circuit level power saving techniques for digital designs, has been applied by Brooks[13] to more heavily, and some would argue more appropriately, weigh delay by squaring or even cubing the delay factor in the calculation. Cameron, in a poster presented at SC04 [14], and in later papers, suggests a weighted approach where the delay factor can be weighted based on the priority of performance. The decision of how to weight performance versus power is largely a policy decision. These metrics can be useful in evaluating individual application efficiency on a platform. In this thesis performance versus energy use is contrasted in various ways including EDP and weighted EDP.

In [15] the authors focus on exploiting parallel processing inefficiencies to achieve savings in power with little performance impact. The authors use micro-benchmarks on what appears to be the aforementioned NEMO 16 node cluster or a close equivalent. With micro-benchmarks, the authors achieve a respectable 25% energy savings with only a 2% performance impact. The results presented in this thesis for real scientific applications are better which could suggest that potential savings increase with scale.

Ge, Feng and Cameron (et al.) make an important contribution with their research. The research presented in this thesis has improved on their work significantly by analyzing real applications on a large scale HPC platform (the majority of their research is accomplished analyzing synthetic benchmarks). The instrumentation approach presented in this thesis allows the collection of data from thousands of nodes, in-situ, at a higher frequency without interrupting the operating system to obtain the data. Additionally, this thesis contains research on the effects of network bandwidth tuning on both power and performance.

Li et al. in [16] models hybrid MPI/OpenMP from a performance and energy perspective examining both dynamic concurrency throttling (DCT) and DVS. In [17] Li takes a modeling approach to investigate task aggregation to reduce energy consumption by reducing the number of nodes. Li uses AMG along with some NPB MPI benchmarks, one of the few efforts that use a real HPC applications in their analysis. Li et al. seem to suggest that in some cases the benefits of DVS diminish with scale. The results for AMG in this thesis differ Li uses a hybrid implementation which is likely quite different. Their research predicts the performance energy trade-off up to 1024 cores (128 nodes). In both [16] and [17] the System G supercomputer at Virginia Tech is used. System G consists of 324 nodes. The power measurements appear to be done in a similar manner to the NEMO cluster (assumed to be the predecessor to System G). Their research takes a different approach; more model and analytical based versus our empirical approach. The investigation of hybrid approaches is timely considering the likelihood of more cores per node, both homogeneous and heterogeneous, in the future. The research provided in this thesis would be an excellent validation platform for their research to a much higher scale than they have currently investigated.

Hsu and Kremer in [18] investigate a compiler based approach to leverage DVS in efforts to reduce energy consumption during times where the CPU can be slowed without greatly affecting performance; for example, during memory stalls. Hsu's later research [10] investigates a run-time approach which does not require source code modification, compiler based or otherwise. These efforts are orthogonal to research included in this thesis but many of the motivations are similar.

The use of performance counters to estimate power efficiency has been researched from a micro [19, 20] and macro [21] perspective. While estimates based on performance counters have shown to be useful, this research will show that scalable fine-grained measurements can be leveraged at both the micro and macro level to ana-

lyze HPC operating system and application power use, with and without frequency scaling, while employing network bandwidth tuning. While the approach taken is orthogonal, clearly, modeling efforts would benefit by combining these approaches if only for validation purposes.

In [22], the authors evaluate various methods of measuring power including cabinet level collection on the Cray XT architecture. The method is clearly much more coarse and their methods proved to not be scalable. Using cabinet level data as a verification of the data collection mechanisms used in our research was considered. This approach, if it can be implemented in a scalable way, may be leveraged in the future. The authors concluded that measuring power at system scale is problematic and while nodal and small scale measurements can be accomplished the scalable, large scale, collection of samples is not feasible. The research provided in this thesis shows that it is possible to collect scalable, fine granularity, high-frequency power measurements on HPC platforms given the necessary hardware and software infrastructure. Further, by enabling collection from all nodes used in an application (during a single application run) the additional power effects of parallel applications, at scale, can be observed and quantified rather than extrapolated 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 can be greatly enhanced.

Kodi et al. [23] discusses the ability to tune network bandwidth using techniques similar to DVS (but for network components) to dynamically reconfigure optical interconnects with the goal of increasing energy efficiency. The authors make many of the same assertions made in this thesis. Kodi focuses on an analysis of the specific optical interconnect technology. The research presented in this thesis provides evidence confirming large benefits are achievable by using tunable network components on HPC platforms.

Research was found involving DVS on network links as early as 2003 (Shang et al. [24]). No evidence of an HPC platform that currently has this capability could be found. Shang uses a modeling approach to project energy savings while applying a history based policy. Shang's research illustrates that implementing DVS requires consideration of many trade-offs.

Brightwell et al. [25], [26] and [27] and Pedretti et al. [28] analyze many aspects of network performance but do not evaluate power as part of their experiment. No research has been found that measures energy in-situ at any scale as it relates to network tuning.

While the research presented in this thesis suggest it is possible, there are significant challenges in developing network hardware to support the equivalent of DVS. One goal of this research is to provide strong motivation for tunable network technologies.

Probably the greatest difference and contribution of this thesis is the sheer scale of the experiments involving the largest set of real HPC scientific applications. This thesis is clearly focused on empirical analysis. The necessity of extrapolation to large scale is removed and true scale effects for scientific applications are demonstrated. No related work was found to compare to the research contained in this thesis regarding tuning network bandwidth and evaluating the impact on power efficiency.

Chapter 2

Research Platform

The only difference between men and boys is the cost of their toys. – Author Unknown

2.1 Hardware Architecture

The experiments conducted as part of this research were all accomplished on some variant of the Cray XT architecture. To our knowledge, this is the only platform that exposes the ability to measure current draw and voltage, in situ, as described in Chapter 3. Both the idle cycle and frequency scaling experiments required specific operating system modifications to the Catamount light-weight kernel[29] (LWK). Catamount support is currently limited to the Cray XT architecture. The Cray XT architecture also affords the rare ability to tune performance parameters of other components. This capability is exploited to tune network bandwidth while measuring the effect on application energy in Chapter 6. The following sections will describe specific test platforms and configurations in more detail as they pertain to this research. It is important to note, obtaining dedicated time on production HPC platforms is difficult, and very expensive.

2.1.1 Red Storm

Red Storm, the first of the Cray XT architecture line, was developed jointly by Cray Inc., and Sandia National Laboratories. The Cray XT architecture has been installed at numerous government and commercial sites including Oak Ridge National Laboratories. Red Storm is currently a heterogeneous architecture containing both dual and quad core processors. Both variants are used in the experiments discussed in this thesis. All nodes, dual and quad, are connected via a Seastar 2.1 network interface controller/router (Seastar NIC) in a modified mesh (mesh in X and Y directions and a torus in the Z direction).

Dual Core Nodes The network bandwidth experiments described in Chapter 6 were accomplished on the dual core (XT3) partition of Red Storm. The XT3 Partition contains 3,360 AMD 64 bit 2.4 GHz dual-core processors with 4GB of DDR memory (2 GB per compute core). Each XT3 node is connected to the network via a Seastar NIC. The ability to manipulate the network bandwidth of the platform is equivalent on both the XT3 and XT4 partitions. The primary driver of using the XT3 partition for the network bandwidth experiments was simply the availability of this partition. The idle experiments were conducted on both the dual and quad core partitions of Red Storm.

Quad Core Nodes Red Storm's XT4 partition utilizes AMD 64 bit 2.2 GHz quad-core processors with 8 GB of DDR2 memory (2 GB per compute core). Red Storm has 6,240 quad-core compute nodes, each connected to the network via a Seastar NIC. The frequency scaling experiments described in this thesis were conducted solely on the quad-core processors of either Jaguar or Red Storm due to the Advanced Power Management (APM) architectural requirements. The method of exploiting APM features will be discussed in Chapter 6. Some of the applications used in this research

are export controlled and could not be executed on Jaguar (an open platform). Since the architectures and software stacks used were identical, we simply maximized our use of each platform based on application requirements and test platform availability.

2.1.2 Jaguar

Use of Jaguar was granted through the Department of Energy's Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. Jaguar, located at Oak Ridge Leadership Computing Facility (OLCF), was used for a portion of the frequency scaling experiments outlined in Chapter 6. The XT4 partition of Jaguar was specifically employed since it was both easier to gain dedicated access to and the architecture supported Catamount with much less up front effort than the XT5 partition would have required. Dedicated access was necessary for a number of reasons, primarily driven by our requirement to run Catamount (no longer a Cray supported operating system for the XT architecture). The XT4 partition of Jaguar contains 7,832 64 bit quad-core AMD Opteron processors (or nodes). Each core executes at 2.2 GHz and has access to 8 GB of DDR2 memory (2 GB per compute core). Each node on Jaguar is connected to the network via a Seastar NIC. The network topology of the Jaguar XT4 partition is a 3D torus[1]. Jaguar's network topology differs somewhat from Red Storm's. These differences are not significant to the experiments conducted for this research and had no affect on the results.

2.2 Operating System

Serial number one of the Cray XT architecture employed a light-weight kernel operating system named Catamount. For approximately four years, Catamount was

[1]The XT4 partition of Jaguar was recently decommissioned.

delivered by Cray Inc. as the production operating system for the XT3 platform line. Catamount, at the time, was the latest in the lineage of light-weight operating systems authored, or co-authored, by Sandia National Laboratories[2]. Catamount was designed to get out of the way of the application. Important hardware abstractions and memory management are all provided with performance being the primary design consideration. When a parallel application is launched, Catamount provides the initial set up for the application, including contiguous memory allocation, and basically suspends itself other than handling interrupt driven tasks such as those from network devices (Seastar NIC). This is a simplistic description but sufficient for the purposes of this thesis. The basic point is, Catamount is a small deterministic operating system in contrast with general purpose operating systems such as Linux. While it has proven to be a very successful production operating system, Catamount has also proved invaluable for operating system research at Sandia National Laboratories.

The first experiment conducted as part of this research was directed at saving energy during idle cycles since early versions of Catamount ignored this as a design consideration. The design of Catamount preceded many of the APM capabilities found on recent processor architectures. Once successful, later experiments explored further power efficiencies by leveraging more advanced APM features like frequency scaling. The deterministic nature of Catamount greatly aided in conducting this research[3]. More detail on specific modifications will be included in our coverage as necessary.

[2]Sandia's most recent effort is the Kitten light-weight kernel[30].

[3]Much of our research has been accepted into the production environment. To date, Sandia has saved more than one million dollars in facility power costs directly attributed to this work

2.3 Reliability Availability and Serviceability System

Historically, Reliability Availability and Serviceability (RAS) systems were commonly provided by vendors on mainframe class systems. Today, RAS systems are mostly unique to very high end custom HPC class architectures (Cray XT/XE and IBM BlueGene L/P and Q, for example). Its hard to define a clear line where cluster management systems become RAS systems. Generally, cluster management systems consist of a loose collection of open source utilities that are each individually designed for a narrow purpose. They are seldom integrated in any way and can often be intrusive to the primary purpose of the platform, computation. At small scale, the level of interference is often acceptable. RAS systems, in general, are typically more intentionally designed and integrated, often specific to a single architecture (in my opinion one of the failures of RAS system designs historically[31]).

The following are excerpts from the requirements for a recent capability class procurement by the Alliance for Computing at Extreme Scale (ACES), a collaboration between Sandia National Laboratories and Los Alamos National Laboratory[4].

- *To achieve delivery of the maximum continuous system resource availability, the RAS system must be a well engineered, implemented and integrated part of the proposed platform.*
- *There shall be a separate and fully independent and coherent RAS system.*
- *The RAS system shall be a systematic union of software and hardware for the purpose of managing and monitoring all hardware and software components of the system to their individual potential.*
- *Failure of the RAS system (software or hardware) shall not cause a system or*

[4]The requirements listed were contributed to the Cielo RFP by the author of this thesis.

job interrupt or necessitate system reboot.

While this is only a small portion of the requirements that described and specified the RAS system for Cielo[32], it suggests some differentiating characteristics between a generic cluster management system and what is considered a RAS system. For this research, one of the most important characteristics is the separation but close integration of the RAS system in relationship to the capability platform. This allows for the out of band[5] scalable collection of current and voltage data necessary for all experiments included in this research. It is very important that experimental methods do not, or minimally, affect the normal activity of the system. In related works, other research efforts that employ laptops and measure power using the ACPI interface are described. This method causes an operating system interrupt each time a measurement is requested. While the interruption is minimal, at large scale this type of measurement could introduce the equivalent of operating system noise[5],[4]. There is no such interruption during the measurements used in in our experiments. The separate RAS network additionally allows the collection of these measurements in a scalable manner.

[5]Out of band, in this context, means that control and monitoring of the platform, in general, is accomplished without affecting the platform or the software running on the platform. Measuring current and voltage data, for example, does not require an operating system interrupt using our methodology.

Chapter 3

Measuring Power

To understand the affect we must be able to measure the effect. – Unattributed

3.1 Overview

The effort expended to measure current draw and voltage are enabling technologies supporting this research. To date there has been no other work published that has been based on such large scale fine grained in-situ measurements of energy on a High Performance Computing (HPC) platform. This chapter contains a discussion of both the hardware and software infrastructure used to support this research.

3.2 Hardware

The Cray XT architecture contains an integrated Reliability Availability and Serviceability (RAS) system comprised of both hardware and software with the goal of increasing the reliability of the overall platform (see Section 2.3). At a very high

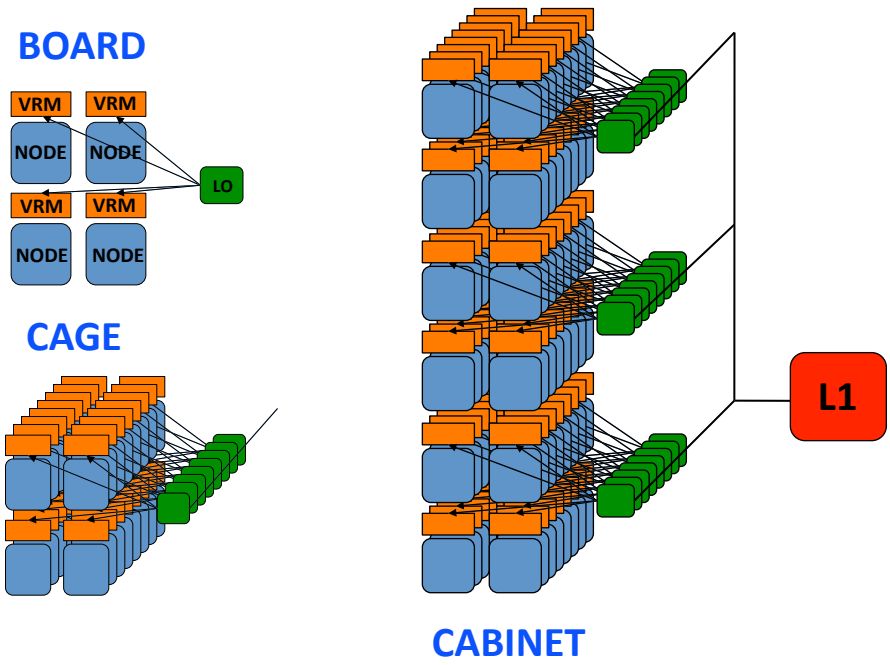

Figure 3.1: RAS: Board, Cage and Cabinet Connectivity

level, the RAS system is responsible for the control and monitoring of the underlying platform. The separate hardware allocated for the RAS system is intended to ensure the primary purpose of the underlying platform - computation - is affected as little as possible. The distinct, but closely integrated, nature of the RAS system provides an out-of-band path that allows a large variety of monitoring data to be collected.

Unlike typical commodity hardware, the Cray XT3/4/5 node boards provide interfaces that can be exploited to measure component level current draw and voltage. Figure 3.1, depicts the logical board, cage and cabinet connectivity of important components of the RAS system involved in collecting current and voltage measurements. Each node board has an embedded processor called an L0 or *level 0* (depicted in green). The L0 has the ability (and responsibility) to interface with many on board

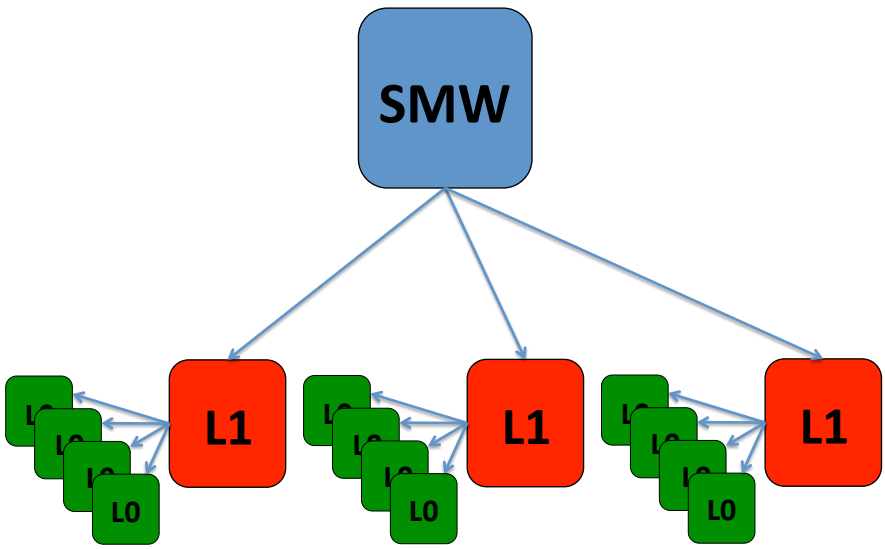

Figure 3.2: RAS: Hierarchical Connectivity

components. To obtain current and voltage measurements the i2c[1] link between the L0 and the Voltage Regulator Modules (VRM) is enlisted. Note, each node (depicted in blue) has an associated VRM (depicted in orange). The black lines connecting the L0 to each node's VRM represent the i2c interface.[2] The current and voltage measurements are collected by the L0 on each board. In the Cray XT architecture there are eight boards in a cage. Three cages comprise a cabinet. At the cabinet level there is an additional embedded processor called an L1 or *level 1* (depicted in red in Figures 3.1 and 3.2). Each of the 24 L0s in a cabinet are connected to the cabinet L1 via Ethernet, also depicted by black lines. The L1 acts as a parent for each of the L0s in a cabinet.

[1]The i2c is one of several two wire serial bus protocols commonly used for this type of component control and monitoring. SMBus, for example, is another common standard.

[2]In reality, the connectivity is more complicated, involving a number of Peripheral Interface Controllers (PIC micro-controllers) and Field Programmable Gate Arrays (FPGA).

Figure 3.2 represents the overall hardware RAS hierarchical topology down to the L0 level. Each cabinet level L1 in the platform connects to the top level System Management Workstation (SMW) (depicted in blue in Figure 3.2) via Ethernet. Similar to how the L1 acts as a parent for each underlying L0, the SMW acts as a parent for all L1s in the platform. This configuration forms the RAS hardware hierarchy for the Cray XT architecture. This hierarchical configuration provides sufficient scalability for the control and monitoring of very large platforms[3]. While not infinitely scalable in its current configuration, no scalability issues were encountered.

3.3 Software

While the ability to exploit the hardware (collect current and voltage 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 was leveraged in the following way to conduct this research.

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. The base CRMS software was augmented with a *probing* daemon that runs on each L0 and a single *coalescence* daemon that runs on the top level SMW. 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 node or processor socket). In the standard CRMS, the L0 daemon processes communicate to their parent L1 daemon process (also part of the standard CRMS, executing on the cabinet level L1) through an event router daemon (additionally part of the standard CRMS). In turn, each L1

[3]Systems of more than 200 cabinets have been supported by this configuration. A 200 cabinet system would be comprised of one top level SMW, 200 L1s and 6400 L0s. The RAS system alone is larger than most commodity clusters.

communicates to the top-level SMW through an event router daemon. The results of a series of timed probes, requested by the probing daemon, are combined at the L0 level and communicated through the standard event router daemon hierarchy to the *coalescence* daemon executing on the SMW, which outputs the results.

```
c14-2c0s2,1300491313,n0=0x17,n0_V=0x4fa,n0_W=0x1d, ....,mezz=0x46
c14-2c0s3,1300491313,n0=0x1a,n0_V=0x500,n0_W=0x21, ....,mezz=0x46
c14-2c0s4,1300491313,n0=0x15,n0_V=0x4e3,n0_W=0x1a, ....,mezz=0x46
```

Figure 3.3: Sample Raw Current, Voltage and Calculated Wattage Output Data

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). Figure 3.3 depicts a few sample output lines from an actual experiment. The cname (c14-2c0s3), for example, corresponds to the L0 that resides in the cabinet with X coordinate 14 Y coordinate 2 cage 0 slot 3. The 2nd field is the time-stamp relative to the L0 that collected the data. (Some challenges were encountered when we collected data from L0s with poorly synchronized date and time.) The remaining fields are the current and voltage measurements for this board and a calculated value for power in watts. For example, the entry **n0=0x1a** specifies a current measurement of 26 amps. The entry **n0_V=0x500** specifies a voltage measurement of 1280 millivolts or 1.28 volts. Finally, the calculated wattage value is provided (**n0_W=0x21** or 33 Watts). Typically, the wattage is calculated using the current and voltage values in the post processing step rather than using the value specified in this output to retain as much precision as possible. The entries omit the values for nodes 1, 2 and 3 in the interest of space. Note, the last field labeled *mezz* represents the Seastar NIC. As can be seen in the samples provided, the current draw is very consistent for the Seastar and does not change in response to load. The power draw for most network interfaces is dominated by the SerDes. The SerDes typically operate at a constant rate independent of network traffic. While the current draw of the network does not

vary over time, it is useful in quantifying the total power used by the node. The measured value from the NIC (Seastar) is used to establish a baseline in the network bandwidth experiments (see Section 7.2).

A per socket collection granularity at a frequency of up to 100 samples per second[4] was achieved by leveraging the existing hardware and software foundation of the CRMS. The accuracy of each sample is approximately +/-2 amperes. While the samples are not as accurate as would be optimal, the data remains extremely valuable for comparing deltas. Since the values reported by a single node are consistent, the deltas are reliable for comparison purposes. This is in contrast with most other platforms where measuring current draw is typically limited to inserting a meter between a power cable and energy source, which results in a very coarse measurement capability at best. The current draw measurements include memory controller activity (since the processors used have an on-board memory controller) but not power used by the memory banks themselves. The granularity and frequency of this sampling capability has enabled real power usage to be observed and used in new and powerful ways.

The impact of the instrumentation on the CRMS has been closely monitored. Even at 100 samples per second little impact on the L0 was observed. Likewise, no adverse impact to communication between the L0s and L1 devices, or between the L1s and the SMW has been observed. The instrumentation has been tested on up to 15 Cray XT cabinets (1440 nodes) and no scaling issues have been identified. It should be noted that this was not intended to be a production implementation. Cray Inc. has recently become interested in replicating our approach and this measurement capability is expected to be a standard option of the CRMS in the near future.

[4]The data used in our research is generally at a one sample per second frequency. We found little benefit in higher frequency collection for the purposes of this analysis. In addition, we do not have sufficient information concerning the ability of the low level device to collect discrete values at higher rates.

3.4 Post Processing Measurement Data

In all of the following experiments, current and voltage measurements are collected, simultaneously, from 15 cabinets (1440 nodes), more specifically each node's VRM, at a frequency of one sample per second over the duration of the entire test period to avoid start up and tear down overhead of the collection process. Since a range of applications were executed at various scales, cabinets were targeted in a distributed manner throughout the platform to achieve consistent collection coverage for the applications tested. For example, the CTH application run on 4096 nodes intersected with 960 of the collection nodes (23% coverage). Similarly, the AMG application run on 1536 nodes intersected with 480 of the collection nodes (31% coverage). The number of nodes sampled was not limited by the scalability of the collection mechanism but by the available test time on these large scale platforms.

Post processing begins with ensuring all data samples used are from nodes involved in the specific application run. The data samples are then synchronized with the application execution start and finish timestamps. The resulting file is used as input for a Perl script which accomplishes the vast majority of post processing automatically. The post processing script has a wide range of capabilities used for analyzing input data for a number of purposes. The following describes a typical analysis.

In Figure 3.3 notice that a single line of data contains information for all four nodes on a board (and the Seastar or mezz). Each line is first parsed into individual node data and stored in a data-structure which allows great flexibility with post-processing options. Each line also contains a time-stamp. Each data sample for each node is stored with an associated time-stamp value. The trapezoidal rule is used to integrate these values over time, which approximates the energy used over the duration of the application. This value is expressed in Joules and calculated for

each individual node. In addition to this calculation, a graph is produced for each node over the duration of the data sample. Graphs are typically produced (using gnuplot) of the absolute measurement values in Watts on the X axis and time on the Y axis. Alternatively, plots can be produced using the measured values relative to the measured idle current by sampling the current for a small duration of time before launching the application. A number of output graph types and formats can be specified. For the purposes of displaying energy used over the duration of an application run, the gnuplot filled-curves format is most useful for analysis (example graph can be found in Section 6.3, Figure 6.2).

In addition to calculating an energy value and producing a graph for each node represented in the data file a statistics file is also generated that is used as the basis for cumulative, per application analysis. The statistics file contains the energy for each node along with a number of additional statistics including: total energy (sum), the average or mean, median, mode and the coefficient of variation (CV). This analysis allows huge amounts of data to be processed while ensuring the results are valid. The CV is leveraged to ensure the measurements are dependable since the CV is expressed as a percentage, independent of the magnitude of the data samples. For the purposes of this analysis, the differences between complete samples or deltas are the primary focus. Previous experiments have proven this approach to be very reliable and provide a solid foundation for comparison. This process is used to produce the data and graphs that appear throughout this document.

Chapter 4

Applications

The applications used in this research were primarily selected based on their importance to the three Department of Energy (DOE) National Nuclear Security Administration (NNSA) nuclear weapons laboratories (Sandia, Los Alamos and Lawrence Livermore). As part of the procurement of Cielo, (DOE/NNSA's most recent HPC capability platform (2010)) each laboratory in the Tri-Lab complex specified two production scientific computing applications that would be used in the acceptance phase of the procurement of Cielo. These applications are herein referred to as the *6X* applications (due to the requirement they, on average, must perform six times faster on Cielo, not that there are six applications). The 6X applications include: SAGE, CTH, AMG2006, xNOBEL, UMT and Charon. In addition to the 6X applications LAMMPS, another production DOE application, and two synthetic benchmarks, HPL and Pallas were used. The following sections provide short descriptions of each application.

4.1 High Performance Computing Applications

SAGE[33, 34] 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. SAGE represents a large class of production applications at Los Alamos National Laboratory. Both strong and weak scaling inputs were used in our experiments.

CTH[35] is a multi-material, large deformation, strong shock wave, solid mechanics code developed at Sandia National Laboratories. It has models for multi-phase, elastic viscoplastic, porous and explosive materials. Three dimensional rectangular meshes; two-dimensional rectangular, and cylindrical meshes; and one-dimensional rectilinear, cylindrical, and spherical meshes are available. CTH has adaptive mesh refinement and uses second-order accurate numerical methods to reduce dispersion and dissipation and produce accurate, efficient results. For our experiments the test problem used was a 3-D shaped charge simulation discretized to a rectangular mesh.

AMG2006[36], developed at the Center for Applied Scientific Computing, at Lawrence Livermore National Laboratory, AMG is a parallel algebraic multi-grid solver for linear systems arising from problems on unstructured grids. Based on Hypre[37] library functionality, the benchmark, configured for weak scaling on a logical three dimensional processor grid $px \times py \times pz$, solves the Laplace equations on a global grid of dimension $(px \times 220) \times (py \times 220) \times (pz \times 220)$.

xNOBEL[38], developed at Los Alamos Laboratory, is a one, two, or three dimensional multi-material Eulerian hydrodynamics code used for solving a variety of high deformation flow of materials problems. The problem used for this study was the *sc301p* shape charge problem in two dimensions in a weakly-scaled configuration.

UMT[39] is a 3D, deterministic, multigroup, photon transport code for unstructured meshes. The transport code solves the first-order form of the steady-state Boltzmann transport equation.

Charon[40], developed at Sandia National Laboratories, is a semiconductor device simulation code. Charon employs the drift-diffusion model, a coupled system of nonlinear partial differential equations that relate the electric potential to the electron and hole concentrations. A fully-implicit, fully-coupled solution approach is utilized where Newton's method is used to linearize the discretized equations and a multigrid preconditioned iterative solver is used for the sparse linear systems. Charon uses the solvers from the Sandia National Laboratories Trilinos[41] project. The problem used for this study is a 2D steady-state drift-diffusion simulation for a bipolar junction transistor with approximately 31,000 degrees of freedom per MPI rank.

LAMMPS[42, 43] is a classical molecular dynamics code, and an acronym for **L**arge scale **A**tomic/**M**olecular **M**assively **P**arallel **S**imulator. LAMMPS has potentials for soft materials (biomolecules, polymers) and solid state materials (metals, semiconductors) and coarse grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale. When run in parallel, LAMMPS uses message passing techniques and a spatial decomposition of the simulation domain.

4.2 Synthetic Benchmarks

Highly Parallel Computing Benchmark **(HPL)**[44] is the third benchmark in the Linpack Benchmark Report, used as the benchmark for the bi-annual Top500 report[45]. While the value of the benchmark in measuring how a system will perform for *real* applications can be debated, its pervasiveness is unquestionable. The benchmark solves a random dense linear system in double precision arithmetic on a distributed

memory system. The HPL benchmark is well understood and recognized as a compute intensive application.

Pallas[46], now called the **I**ntel **M**PI **B**enchmark (IMB), successor to Pallas GmbH, is a suite of benchmarks designed to measure the performance of a wide range of important MPI routines. Pallas is communication intensive.

Chapter 5

Affecting Power During Idle Cycles

5.1 Motivation and Goals

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 (and under load as we will discuss in Chapter 6) as these features have become commercially available. The HPC community makes great use of Linux on many of their platforms, but light weight kernels (LWKs) are often used to deliver the maximum amount of performance at extreme scale (Red Storm and Blue Gene, for example). To achieve greater performance at scale, LWKs have a selective feature set when compared to general purpose operating systems like Linux. As a result, LWKs 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. The goal of the first experiment is to match or beat the idle current draw of Linux.

5.2 Operating System Modifications

The measuring capabilities described in Chapter 3 were first leveraged to examine the current draw of the Catamount LWK. Initial findings were not surprising. As suspected idle cycles were consuming current as Catamount busily awaits new work.

One of the advantages of most LWKs (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))[1] support 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. (core 0 will be referred to as the *master* core and cores greater than 0 *slave* cores.) Catamount was modified to individually halt *slave* cores when idle and awaken immediately when signaled by the *master* core. Slave cores are signaled prior to an application launch. If no application is executing on a slave core, the core remains idle without interruption. The result of this modification was a significant savings in current draw when slave cores are idle.

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 (these characteristics are leveraged in later experiments to show energy savings during application run-time). Adding more cores, generally, decreases the balance of the platform, aggravating application bottlenecks. For this reason, many scientific HPC applications utilize less than the total number of 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

[1]The name Catamount will generally be used throughout this document unless the more specific names CVN and CNW are necessary to point out an important distinction. Further specific information about Catamount can be found in [29]

architectures.

After these very positive results, the region of the operating system the master core enters during idle was targeted. While the master core is interrupted (awakened from halt) on every timer tick (the slaves are not) significant additional energy savings during idle periods were still observed.[2] Note, if only one core is used during an application run it executes on the master core. After extensive testing, these modifications were accepted into the production release of Catamount for Red Storm and also delivered to the Pittsburgh Computing Center.

5.3 Results and Analysis

5.3.1 Idle Power: Before and After

Figure 5.1 depicts measurements obtained running three applications (HPL[44], PALLAS[46] and HPCC[47]) on a Dual Core AMD Opteron Processor[3] using Compute Node Linux (CNL). Figure 5.2, in contrast, illustrates the results obtained when executing the same three applications on the same physical CPU using Catamount. (Experiments compare results using the same exact hardware to limit variability of measured results.)

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[4] sockets showed nearly identical idle power wattage measurements for both CNL and Cata-

[2]Timer ticks are configured to occur 10 times per second on Catamount

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

[4]AMD Opteron Budapest 2.2 GHz socket AM2

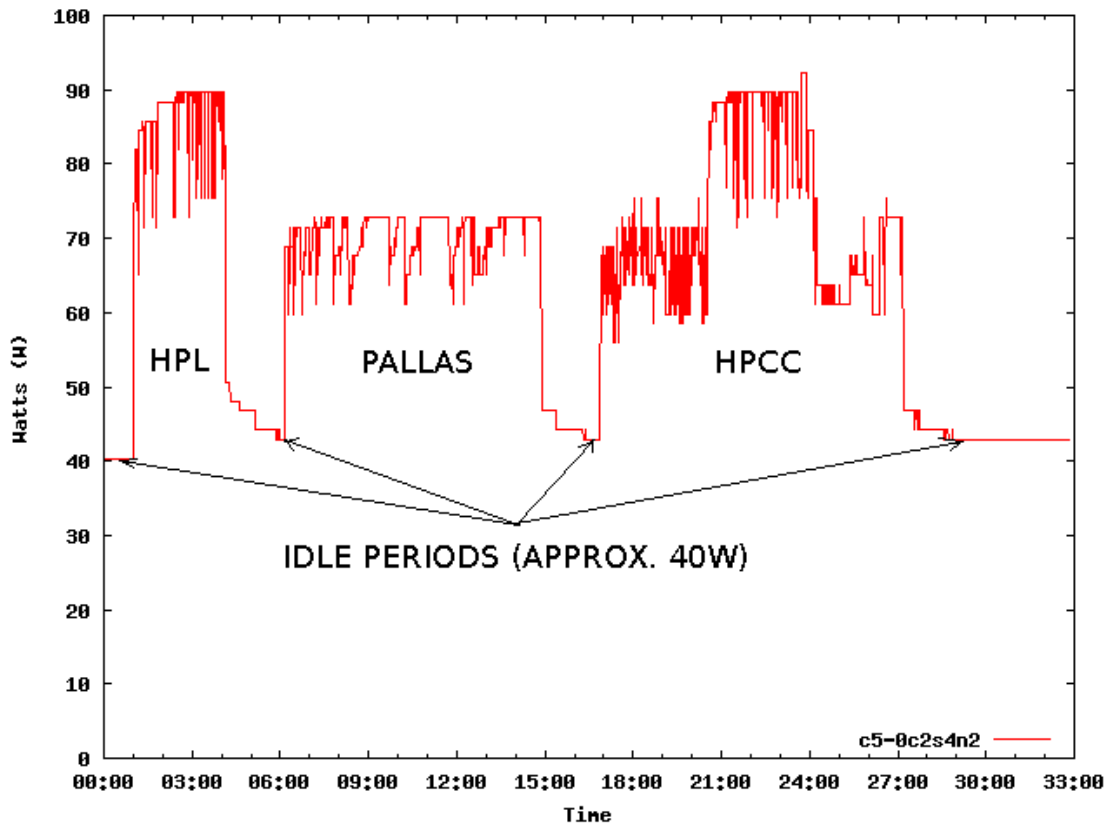

Figure 5.1: Compute Node Linux (CNL)

mount[5] (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 is observed in Figure 5.1 is a polling loop which in Linux is optimized to conserve power. While CNL draws less power than our unmodified Catamount at idle (40W for CNL vs. 60W for unmodified Catamount) it is not as optimal as the modified Catamount implementation (10W). Later observations on the quad

[5]CVN was enhanced to support more than two cores, the resulting Catamount version was named CNW. Unless otherwise specified all results shown after Figure 5.2 were obtained running on CNW

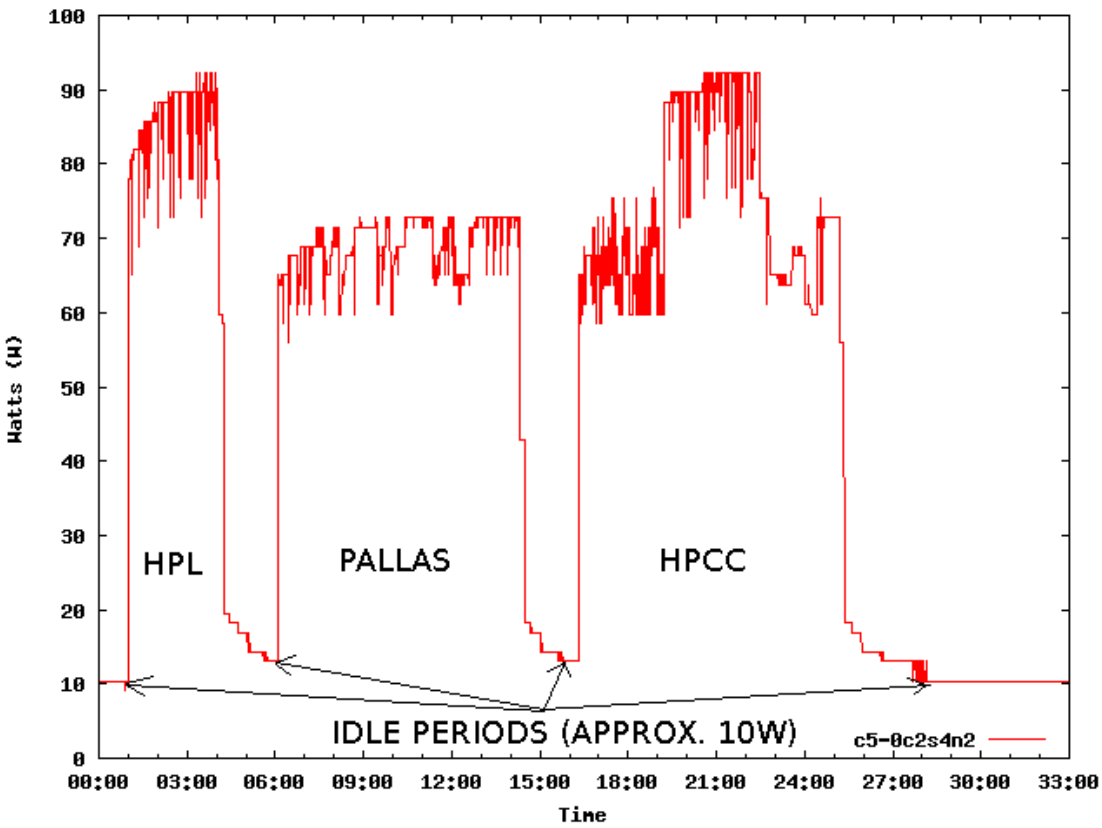

Figure 5.2: Catamount Virtual Node (CVN)

core architecture, where CNL and Catamount draw equivalent amounts of power during idle, are likely the result of CNL exploiting MONITOR/MWAIT. The use of MONITOR/MWAIT should produce similar results to using HALT. Regardless, these results are intended to show the ability to observe and contrast. These measurements have demonstrated the first goal of equaling (and very possibly beating) the idle power savings of Linux.

These results also provide a first look at *Application Power Signatures* (see Section 5.3.2). 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 (as can be observed in

Figures 5.1 and 5.2 between CNL and Catamount).

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

$$(13000\,nodes * 0.2) * (365\,days/year\ *\ 24\,hours/day) = 22.776 * 10^6\,node\,hours/year \tag{5.1}$$

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:

$$(22.776 * 10^6\,node\,hours/year * 50\,Watts/node) \div 1000 = 1.1388 * 10^6\,KW\,hours/year \tag{5.2}$$

Assuming 10 cents per Kilo-Watt hour, based on Department of Energy averages for 2008[48], we can calculate real dollar savings for this hypothetical system.

$$(1.1388 * 10^6\,KW\,hours/year * 10\,cents/KW\,hour) \div 100\,cents/dollar = \mathbf{113{,}880\,dollars/year} \tag{5.3}$$

For a capability system using a figure of 80% utilization as characterized is optimistic. Capability systems are typically intended to support one or more 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.

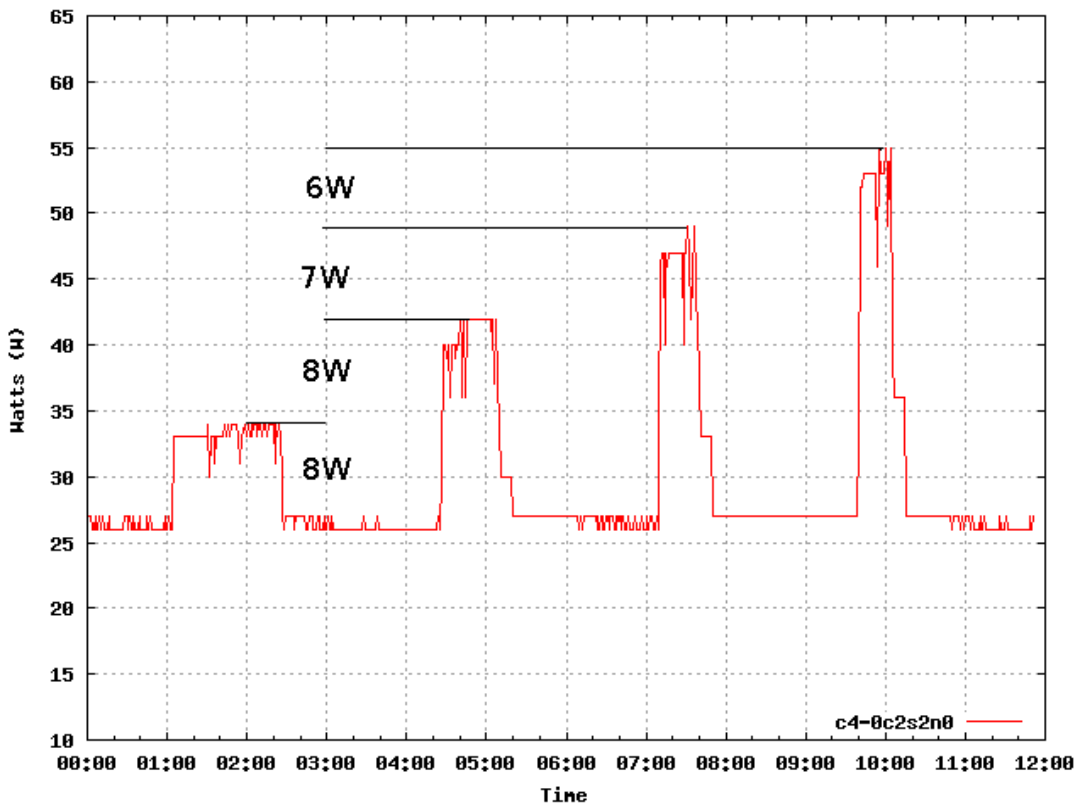

Figure 5.3: Catamount N-Way Per Core Power Utilization

Figure 5.3 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. Even

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

In addition to the previous calculated savings, a 30-40% additional power savings as a result of not having to remove the additional heat generated by higher idle wattages would be realized. If cooling were included, the cost savings would range from between $148,044 and $159,432 per year. By exploiting these power saving measures, significant savings can be realized by targeting idle cores alone.

5.3.2 Application Power Signatures

Application Power Signature is the term 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. A user knowledgeable of the application flow can easily distinguish phases of the application simply by viewing the graphed application 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. The result is termed *application energy*. This metric is derrived by calculating the area under the application signature curve. To accomplish this the post processing code was enhanced to approximate the definite integral using the trapezoidal rule. The following graphs (Figures 5.4 and 5.5) 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).

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

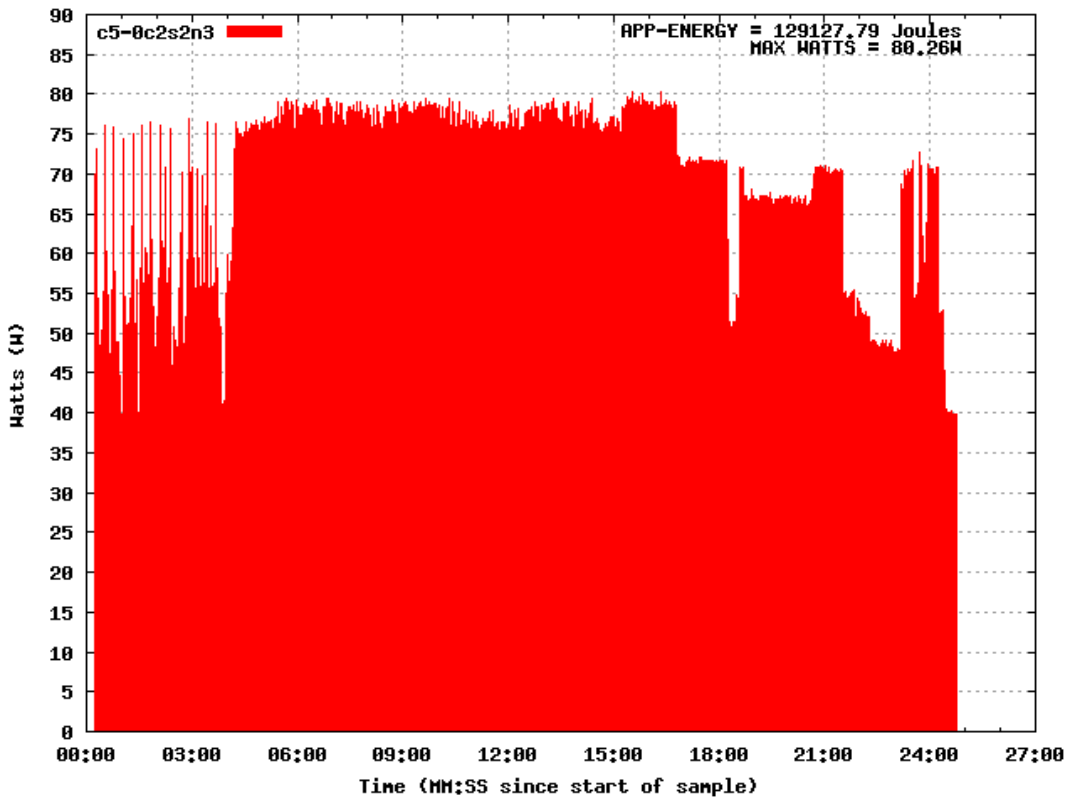

Figure 5.4: HPCC on Catamount

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[49]. It is not surprising that an application that takes longer to execute, given similar power draw during execution, will consume more energy. In this case HPCC ran 16% faster on Catamount. The amount of energy used by HPCC is 13% less using Catamount than CNL. HPCC was also tested on quad core nodes using two cores per node (HPCC ran 15% faster on Catamount and used 13% less energy) and on dual core nodes using two cores per node (HPCC ran 10% faster on Catamount and used 10% less energy). The salient point is that performance is not only important in reducing the run time of an application but also in increasing the energy efficiency of that application. Additionally, without

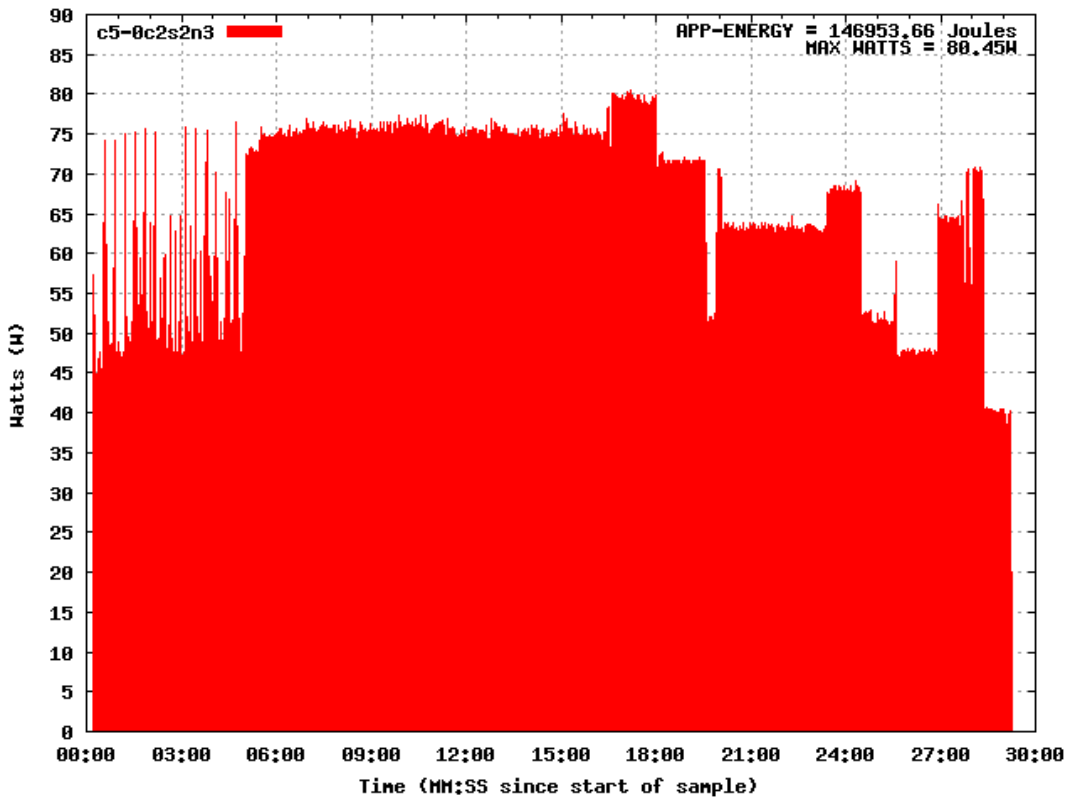

Figure 5.5: HPCC on CNL

the ability to examine real power use at this granularity, the energy efficiency of an application could not be as precisely quantified.

5.3.3 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 [50, 4]. 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`. LWKs, 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 [5]. 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 energy usage is not. Specifically, the goal of this experiment is to determine if energy usage in noisy environments scales linearly (or otherwise) with the increase in application run time.

To evaluate the impact of noise the kernel-level noise injection framework built into the Catamount LWK [5] was used. 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 μs), 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 specifying the frequency and duration of each noise event separately 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.

The following analysis focuses on a single application (SAGE). SAGE was chosen based on initial studies and previous analysis done in [5]. Applications like SAGE have the potential to be significantly impacted by noise and any proportional increase in energy.

Table 5.1 is a representative sample of our results.

Table 5.1: 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%**

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

$$((Frequency(Hz) * Duration(us)) \div (1 * 10^6)) * 100 \tag{5.4}$$

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, the runtime of the application was varied 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 was observed is that the difference in application energy used by applications when noise is injected is linearly proportional to the difference in runtime. If the impact of the injected noise is normalized, 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%. The results were found to be very consistent. The experiment was repeated at a larger scale (48 nodes, again utilizing only core 0) and observed results were consistent with Table 5.1. In an effort to simulate effects seen at larger scale a large amount of noise (10%) was introduced while running the same application. The results (row six of Table 5.1) 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 5.1 shows small percentage increases in runtime for various noise patterns, 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.

In Figure 5.6[5], the measured slowdown of POP, CTH, and SAGE, at scale, is observed. In this figure the Y-axis is the global accumulation of noise for the application. The global accumulation is computed 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 a 2.5% net processor noise signature is injected, a 20% slowdown is measured and the global accumulation of noise would be 17.5%. As see in Figure 5.6 with only 2.5% net processor noise injected the slowdown for POP exceeds 1200% at scale. A proportional increase of 1200% application energy at scale can therefore be projected. The inset of Figure 5.6 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

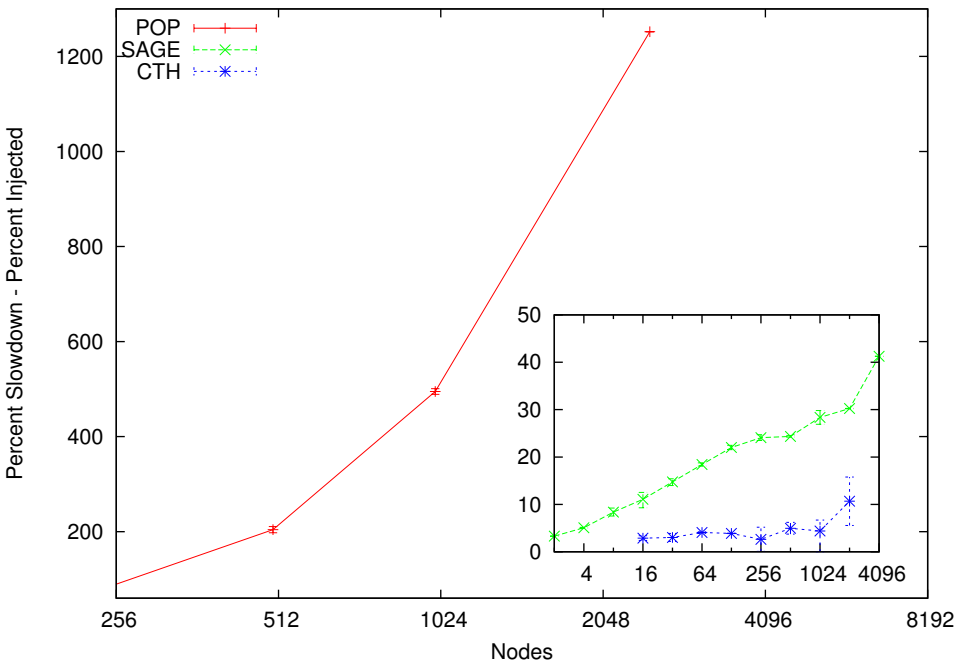

Figure 5.6: Slowdown at Scale

as significant. Further analysis is necessary to verify that these results are truly representative at scale.

Chapter 6

Tuning CPU Power During Application Run-time

Science progresses best when observations force us to alter our preconceptions. - Vera Rubin

6.1 Motivation and Goals

In Chapter 5, the focus was reducing power by exploiting idle cycles. In the experiments outlined in this chapter and in Chapter 7 opportunities to reduce the energy use of a running application without affecting performance are researched. As mentioned previously, determining what is, and is not, an acceptable trade-off between energy and performance is somewhat subjective. The motivation of this research is to show that significant energy savings can be achieved by tuning architectural components on a per application basis. The focus of the following experiment is CPU frequency scaling during application execution.

Typical approaches employed by operating systems, such as Linux, while efficient for single server or laptop implementations, have proven to be detrimental when used at scale causing the equivalent of operating system jitter[4]. For this reason, most, if not all, sites that deploy clusters at medium to large scale disable these features while running applications (some sites enable these features during idle cycles). It is clear that techniques designed for laptop energy efficiency are not directly applicable to large scale High Performance Computing (HPC) platforms. In the following CPU frequency scaling experiments, a more deterministic approach is taken which ensures all cores participating in an application are operating at the target frequency in lock step. This is termed static frequency modification. First, the runtime impact is contrasted with the energy savings on a per application basis. This analysis focuses on CPU energy. In (Chapter 7), the runtime impact is contrasted with total system energy while tuning the network interface, again on a per application basis. Both approaches provide valuable insight.

Feng reports in [6] that the CPU is the largest single component consumer of energy on a node. While Feng's analysis is for a commodity board that contains disk and other components, a similar analysis can be performed on the hardware used for these experiments. A Cray XT architecture node board contains only CPU, memory and a Network Interface Controller (NIC)[1]. If 20W is allowed for memory, 25W for the NIC (based on the measured value) and a measured value for CPU based on the data collected for these experiments, the CPU ranges from 44-57% of the total node power. Clearly, since the CPU is the largest consumer of power, it is productive to analyze it both in isolation and as a component of total system power. In the following experiment CPU data is measured in isolation.

[1]the small amount of power used by the embedded controller is not included since it would be amortized across the four nodes present on the board and be largely insignificant for this purpose.

6.2 Static CPU Frequency Tuning

6.2.1 Operating System Modifications

A number of targeted modifications were made to Catamount[2] to accomplish this research. First, it is necessary to interrogate chip architecture capabilities to determine if Advanced Power Management (APM) is supported. More specifically, the CPU is interrogated to determine if hardware P-state[3] control is supported. Changing P-states requires writing to P-state related Memory Status Registers (MSR). If APM is not supported, writing to P-state MSRs is fatal. Even if APM is enabled, however, only a single P-state is required to be defined. In addition, even if multiple P-states (up to 5) are defined, they may have identical definitions. This is typically not the case but enforces the importance of closely interrogating specific hardware capabilities, and dynamically adjusting to them. From this point forward it is assumed APM is supported, multiple P-states are defined, and at least P-states define different operating frequencies.

The method of frequency scaling used in these experiments is currently limited to frequencies defined in the P-state table, although most processors support frequency stepping in 100MHz increments. The impetus of changing frequency is ultimately to lower the input voltage to the processor. Power is proportional to the frequency, capacitance and voltage squared. By this definition the largest impact to power can

[2]Additional detailed information specific to the AMD architecture family discussed here can be found in the BIOS and Kernel Developers guide (BKDG)[51]

[3]From BKDG: P-states are operational performance states (states in which the processor is executing instructions, running software) characterized by a unique frequency and voltage. The processor supports up to 5 P-states called P-states 0 through 4 or P0 though P4. P0 is the highest power, highest performance P-state; each ascending P-state number represents a lower-power, lower performance P-state than the prior P-state number. As P-state numbers increase, the operating frequency and voltage for a given P-state must be less than or equal to the frequency and voltage of the prior P-state. At least one enabled P-state (P0) is specified for all processors.

be obtained by lowering input voltage. Both the processor and the infrastructure must support dynamic voltage transitions to take advantage of this potential power savings. While it is unlikely that future architectures will support independent per core power planes, there will likely be multiple power domains per processor chip. Understanding how these power planes are partitioned (which cores are on which planes) will be important to achieve maximum energy savings while maintaining performance. On the test platforms used for these experiments all cores were required to be in the same *higher*[3] P-state before a lower input voltage could be achieved. Basically, if one core is operating at a higher frequency (which requires a higher input voltage) the input voltage to the processor remains at the voltage necessary to support the highest active frequency (current lowest active core P-state or current operating frequency (COF)). While describing the subtleties of serial and parallel voltage planes is beyond the scope of this thesis, they are very important architectural details and cannot be overlooked in practice.

At a very early stage in the boot process the default P-state and supported P-states of each core are collected. This information is stored and used by a trap function added to handle a variety of P-state related functionality. Since changing P-states is a privileged call (writing to MSRs) the ability to change P-states was added in two parts; an operating system trap and a user level library interface. The trap implements query functionality to determine what P-states are available, what P-state the core is presently in and of course the ability to transition from the current P-state to an alternate supported P-state. The trap also reports the final P-state achieved and in debug mode the number of nanoseconds the P-state transition took. The amount of time necessary to transition between P-states is not important for the experiments covered in this thesis since a single static change prior to application execution is accomplished. Transition time becomes a critical consideration when more dynamic methods of CPU frequency scaling are employed.

P-state	CPU frequency		Input Voltage	
	Red Storm	**Jaguar**	**Red Storm**	**Jaguar**
0	2.2 GHz	2.1 GHz	1.200 V	1.200 V
1	2.0 GHz	2.1 GHz	1.200 V	1.200 V
2	1.7 GHz	1.7 GHz	1.150 V	1.150 V
3	1.4 GHz	1.4 GHz	1.075 V	1.075 V
4	1.1 GHz	1.1 GHz	1.050 V	1.050 V

Table 6.1: Test Platform P-states, CPU Frequencies and Input Voltages

Table 6.1 lists the supported P-states, corresponding CPU frequencies and required input voltages from Red Storm and Jaguar. Note that the default P-state on Red Storm is P-state 0. Testing was conducted on P-states 0, 2, 3 and 4 on Red Storm. On Jaguar the default P-state is P-state 1. Testing on Jaguar was conducted using P-states 1, 2, 3 and 4. Some inconsistency was observed in the reported P-state *vids* (core voltage ID) on Jaguar. Since both the current and the voltage were directly measured, these inconsistencies had no affect on the results (actual measured values are used).

The frequency in MHz at the end of each entry is calculated using the CPU *fid* (core frequency ID) and the CPU *did* (core divisor ID) (see BKDG[51] for additional information). The *vid* specifies the necessary input voltage to support the operating frequency of each P-state. Notice also the *nbvid* (north-bridge voltage ID) in each case is the same as the vid. On the test platform the input voltage for both the CPU and the north-bridge must be the same.

The input voltage, represented in hexadecimal, can be found in the BKDG for the processor family. The 10h family supports both a parallel and a serial voltage interface. The platforms employed in these experiments use the serial voltage interface infrastructure. A vid of 0x1c corresponds to 1.2 volts[4]. By selecting P-state 2 the in-

[4]To determine the voltage the hexadecimal value must first be converted to its binary representation then referenced in the appropriate table for either serial or parallel power planes in the BKDG[51]

put voltage can be lowered to vid 0x20 (1.150 volts). In this example notice the *vids* for P-state 0 and P-state 1 are identical. Using P-state 1 offers little advantage since the operating frequency (fid) is lowered with no accompanying reduction in voltage. Figure 6.1 displays the measured voltage changes (on Jaguar) when the P-state is altered through the following states: P-state 0, P-state 2, P-state 3, P-state 1. A difference was observed in the input voltage between P-state 0 and P-state 1. Based on the defined P-state (Table 6.1) there should be no input voltage difference. It is for this reason, and others, that it is important to measure actual values.

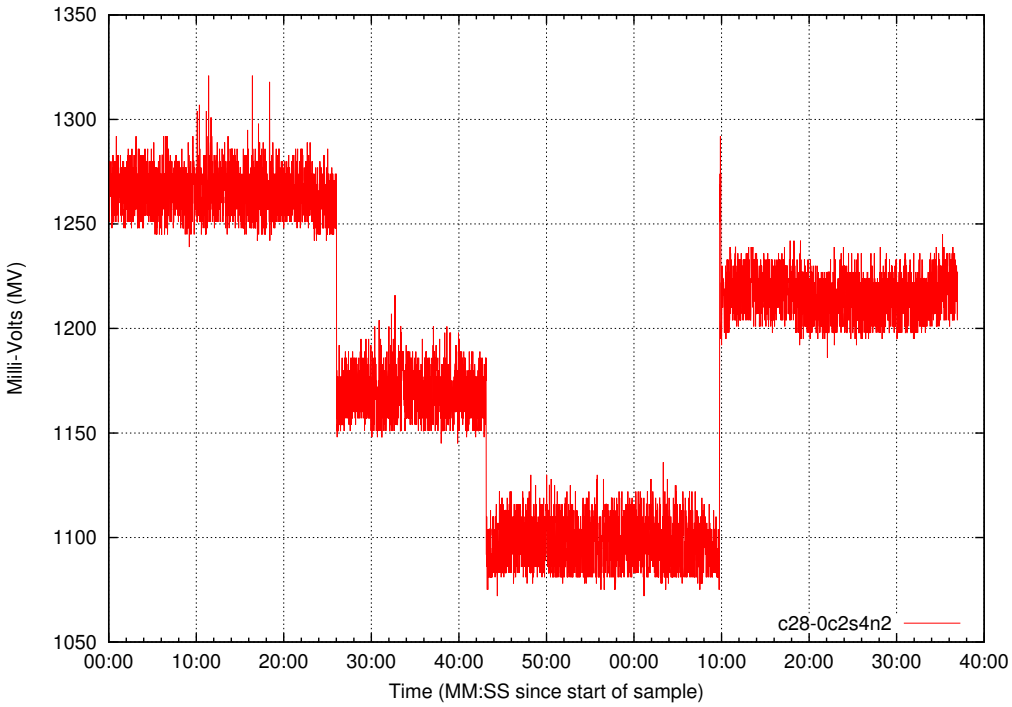

Figure 6.1: Input Voltage Drop in Response to P-state Frequency Changes

The current implementation the operating system trap has proved to be extremely stable. Transition between P-states is done in a step-wise fashion. If, for example, P-state 4 is requested and the core is currently in P-state 0, the transition occurs between P-states 0 and 4 one by one. This approach was found to be more reliable

than directly requesting cores to transition to specific P-states. Additionally, the transition time does not impact the results, at least in the case of static P-state operation. This method would possibly have to be revisited if it is found that it adversely impacts dynamic frequency scaling in future experiments.

To confirm that P-state transitions are successful the ability to monitor both current draw and voltage on a per socket granularity is employed. After requesting a change in P-state a corresponding drop in input voltage can be observed (see Figure 6.1). Note, in some cases, even when the transition to a new lower voltage P-state has been confirmed (recorded in the MSR), a drop in input voltage does not occur (verified through measurement). While this is sub-optimal, it has the effect of understating the results of these experiments since energy is calculated based on measured voltages. In other words, if all voltages behaved ideally, the energy savings would be greater than reported.

More issues were observed, in general, while running on Red Storm than on Jaguar. Further investigation led to the discovery that while Jaguar has very recent Peripheral Interface Controller (PIC) revisions, Red Storm's PIC revisions are quite dated. This is partially due to the age of Red Storm. Since Red Storm was serial number one of this architecture, PIC updates were expected to be infrequent therefore remote update capability was not incorporated into the platform. Benefiting from the lessons learned from earlier installations, the hardware used on Jaguar does support remote PIC updates and therefore it is much less intrusive to keep PIC firmware up to date. Obtaining results for these experiments proved much less problematic on Jaguar likely due to the up-to-date PIC firmware which affects many of the hardware characteristics we manipulate.

6.2.2 Library Interface

Since changing P-states (changing COF) is a privileged operation, the trap is accessed through a variety of functions provided by a user level library. While a single trap function implements all of the frequency change functionality, for ease of use and clarity a library function interface is implemented to exploit each capability individually.

- **cpu_pstates(void)** - Returns detailed processor P-state information
- **cpu_freq_step(P-state)** - Request a P-state transition (up or down)
- **cpu_freq_default(void)** - Returns default processor P-state

In this experiment, the affects of CPU frequency modification are quantified. Prior to executing the user application a *control* application is executed. The control application simply changes the CPU's COF by changing the CPU's P-state to the desired level (cpu_freq_step(P-state)). The control application is launched on every core of each CPU that will be used in the test application. Note, while convenient for these experiments, a separate control application would not be required in a production environment. For example, a user environment variable could be defined and used by the runtime, or a flag could be specified at launch time. Clearly, there are many ways to enable this capability. P-state changes are accessible from any portion of the software stack using this library interface.

Following execution of the control application the HPC application under test is executed on the same nodes. The HPC application will run at a lower frequency defined by the P-state selected by the control program. During the execution of the HPC application data is collected for both current draw and voltage at one second intervals. Once the HPC application is completed all nodes are returned to the default P-state or a new P-state is selected for a subsequent experiment with

cpu_freq_step(P-state).

The trap and library interface was designed to support both static and dynamic frequency scaling. Initially, it was assumed that it would be necessary to change frequency often during application execution to achieve an acceptable trade-off between performance and energy. During testing, it was discovered that large benefits exist for static frequency scaling. Static frequency scaling has many benefits including simplicity and stability.

6.3 Results and Analysis: CPU Frequency Tuning

The following discussion refers to Table 6.2 and Figures 6.2 and 6.3. Increases in run-time or energy percentage in Tables 6.2 are indicated by positive numbers. Negative values or decreases are indicated by parenthesized numbers (all relative to the baseline values listed). In all cases, great care was taken to use the same nodes for each application execution. Figures 6.2 and 6.3 were created by overlaying the energy results from an individual node for P-states 1-4 running the specified application.

Table 6.2: Experiment 1: CPU Frequency Scaling: Run-time and CPU Energy %Difference vs. Baseline

		Baseline Frequency		P-2 - 1.7 GHz %Diff		P-3 - 1.4 GHz %Diff		P-4 - 1.1 GHz %Diff	
	Nodes/Cores	Run-time (s)	Energy (J)	Run-time	Energy	Run-time	Energy	Run-time	Energy
HPL	6000/24000	1571	4.49×10^8	21.1	(26.4)				
Pallas	1024/1024	6816	1.72×10^8	2.30	(43.6)				
AMG2006	1536/6144	174	9.49×10^6	7.47	(32.0)	18.4	(57.1)	39.1	(78.0)
LAMMPS	4096/16384	172	2.79×10^7	16.3	(22.9)	36.0	(48.4)	69.8	(72.2)
SAGE	4096/16384	249	4.85×10^7	0.402	(39.5)				
(weak)	1024/4096	337	1.51×10^7	3.86	(38.9)	7.72	(49.9)		
CTH	4096/16384	1753	3.60×10^8	14.4	(28.2)	29.0	(38.9)		
xNOBEL	1536/6144	542	4.96×10^7	6.09	(35.5)	11.8	(50.3)		
UMT	4096/16384	1831	3.48×10^8	18.0	(26.5)				
Charon	1024/4096	879	4.47×10^7	19.1	(27.8)				

The frequency scaling experiments were conducted during five separate dedicated systems times. Four of the experiments were conducted on Jaguar during eight to twelve hour sessions. The final experiments were conducted on Red Storm during a three day dedicated system time. Over this period a range of experiments were conducted using both real production scientific applications and synthetic benchmarks listed and described in Chapter 4. As can be seen in Table 6.2 some applications were tested in all available P-states. Other applications exhibited clear results in early testing and did not warrant further experiments. In some cases, results at higher P-states (lower frequencies) were not obtained primarily due to hardware issues (namely PIC revision issues as previously described). However, as can be seen in Table 6.2 the results are nonetheless extensive.

Decreasing CPU frequency, in general, will slow active computation. If applications were solely gated by computation this approach would be entirely detrimental. However, applications exhibit a range of characteristics. In this experiment, the CPU frequency is altered and the impact on energy and run-time is measured (other platform parameters are left unchanged). The extremes, or bounds, are represented by two synthetic benchmarks, HPL and Pallas. Note, for all experiments both run-time and energy are contrasted to the baseline runs conducted at P-states 0 or 1 (depending on the platform used) and reported as percent increase or decrease from the baseline value in Table 6.2. For the baseline runs the execution time in seconds (s) and the energy used in Joules (J) is recorded. The CPU frequency experiments focused on the affect that CPU frequency modifications had on CPU energy alone. In [6] Feng evaluates CPU power as a percentage of total system power both during idle cycles and under application load. Even at idle, the CPU accounts for 14% of the total power draw according to their measurements. More significant to this research, Feng's measurements suggest that during load the CPU accounts for more than a third (35%) of the total node power draw. As previously stated, on the platforms used in these experiments the CPU accounts for between 44% and 57% of total node

energy. We recognize that additional run-time will be accompanied by additional energy use from other node components, such as memory and network. These additional factors will be accounted for in our next study that takes a broader look at total system energy. Evaluating both CPU power in isolation and as part of total system power provides important insights.

HPL is largely a compute intensive application. HPL was chosen to demonstrate an application that would potentially be highly impacted by reducing CPU frequency. HPL results were as expected. In Table 6.2 it is observed that a change to P-state 2 causes a 21.1% increase in run-time and a 26.4% decrease in energy used. This would likely not be an acceptable trade-off for a real application unless the priority was energy savings.

In contrast to HPL, Pallas (IMB) is a highly communication intensive benchmark. Pallas was chosen to demonstrate an application that would be, potentially, less affected by reductions in CPU frequency. Again, as expected, Pallas demonstrates only a 2.30% increase in run-time and a 43.6% reduction in energy used when run in P-state 2. This would certainly be a favorable trade-off for most, if not all, applications. Given the results from these synthetic benchmarks, it is expected that real applications will fall somewhere in between these two extremes. Applications are addressed in table order.

Results for AMG2006 at P-states 1-4 at a scale of 6K cores were obtained. At P-state 2 an increase in run-time of 7.47% was observed, accompanied by an energy savings of 32.0%. Note, the longest of three run-times was used in each case for the final measurements. AMG2006 had a short run-time and it was observed that the shortest run-time in P-state 2 was actually faster than the longest run-time in P-state 1. It is, therefore, clear that AMG2006 could benefit from a reduction in frequency to, at a minimum, P-state 2. The trade-off at P-state 3 is not as clear. The runtime impact is proportionally greater than the energy savings at P-states 3

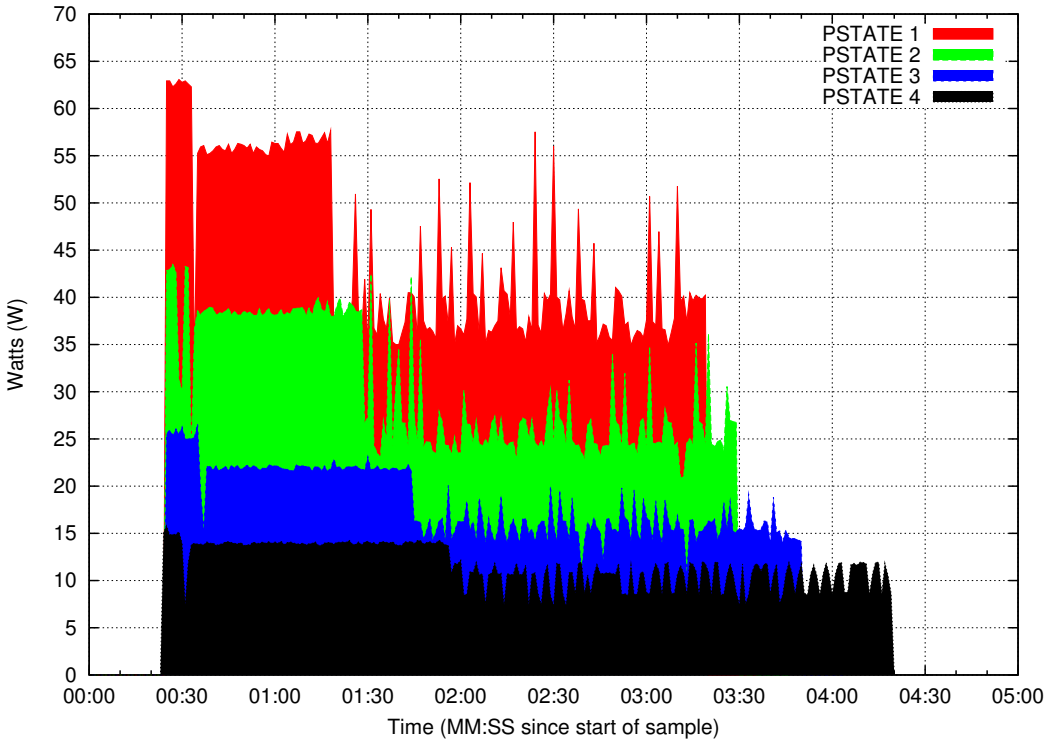

Figure 6.2: AMG P-states 1-4

and 4. Note, while P-state 4 exhibits a significant hit in run-time the largest savings in energy recorded in these experiments was observed. Depending on policies and/or priorities AMG2006 might be able to take advantage of any of the available P-states to produce significant savings in energy.

LAMMPS (tested at 16K cores), in contrast to AMG2006, does not display a clear win when run at lower frequencies. Resuts at P-state 2 show a 16.3% increase in run-time and a 22.9% decrease in energy. Not a clear win but in some cases this might be an acceptable trade-off. The results for P-states 3 and 4 demonstrate a very significant hit in run-time. Increases in run-time of this magnitude might not be acceptable in an HPC environment, but LAMMPS does, however, show a correspondingly large savings in energy at P-states 3 and 4. Again, this may be acceptable in some circumstances where energy consumption is the primary consideration or policy

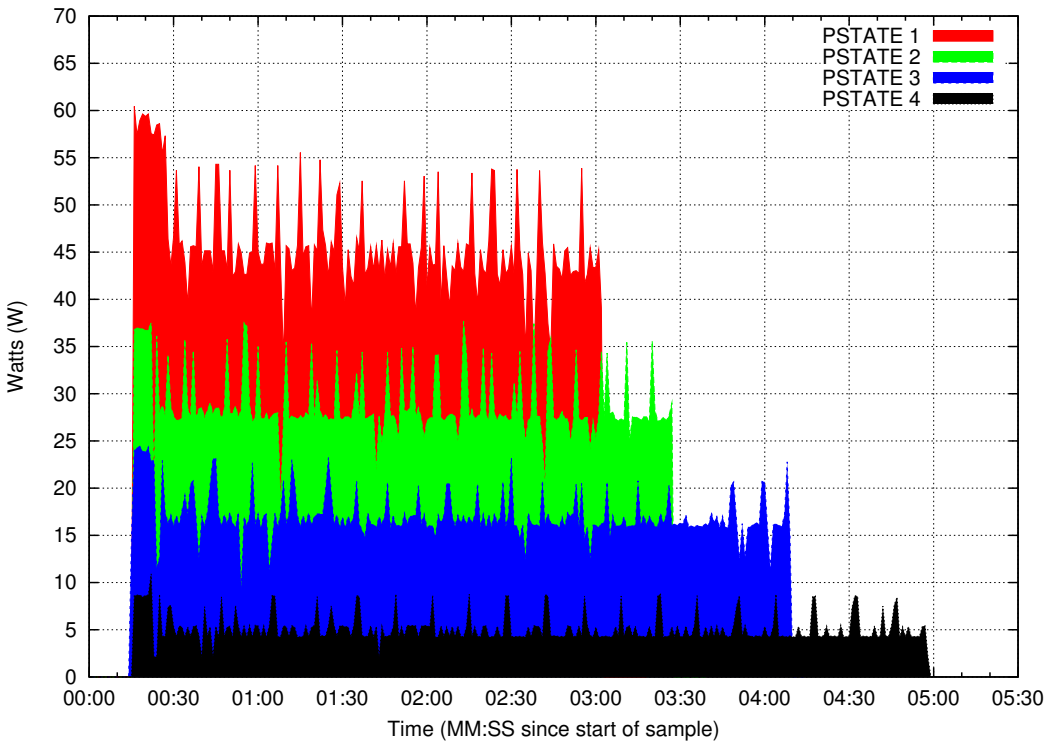

Figure 6.3: LAMMPS P-states 1-4

decisions enforce energy limitations.

Figures 6.2 and 6.3 graphically depict each of the four executions of AMG2006 and LAMMPS at P-states 1-4. The shaded area under each curve represents the energy used over the duration of the application. Figure 6.2 clearly depicts the positive run-time vs. energy trade-off (for AMG2006) indicated in Table 6.2 especially between P-states 1 and 2. In contrast, more dramatic increases in run-time can be seen in Figure 6.3 for LAMMPS. While AMG2006 showed a favorable trade-off between run-time and energy when run at lower frequencies there might be even more benefit to obtain. Notice the compute intensive phase of AMG2006 early in the application execution. If CPU frequency remained high during this phase but transitioned to a lower frequency for the remainder of the application it is likely that the benefits would be even greater. While LAMMPS did not show a clear win when lowering CPU

frequency, notice the very regular compute phases throughout the entire application execution (indicated by peaks in the graph). Chapter 9 contains a short discussion of how these application characteristics, once understood, could be leveraged to obtain power savings even in applications that do not present a clear choice when we statically modify the CPU frequency.

Results for SAGE were obtained using a weak scaling problem at two different scales (4K and 16K cores) at P-state 2. In both cases, a small increase in run-time (0.402% at 16K and 3.86 % at 4K) is observed with a very significant reduction in energy (39.5% at 16K and 38.9% at 4K). Results for a 4k core run of SAGE at P-state 3 were also obtained. The impact on run-time almost doubled but remains low while some additional energy savings were recorded. Based on these observations, it is possible that the 16k core SAGE would also demonstrate a favorable trade-off at P-state 3.

CTH was executed at P-states 0, 2 and 3 at a scale of 16K cores. Similar to LAMMPS, there is no clear win with CTH when the CPU frequency is lowered. Also, like LAMMPS, CTH has very regular compute and communication phases. LAMMPS and CTH will likely be the targets of future experiments in dynamic CPU frequency scaling at large scale.

Results for xNOBEL at 6K cores at P-states 0, 2 and 3 were obtained. The results indicate that xNOBEL, like AMG2006, is a good candidate for CPU frequency reduction, even using this static method. Having the ability to tune CPU frequency at large scale for this application would be a clear win.

UMT and Charon behaved in a very similar manner. Since UMT was run at a much larger scale than Charon (16K cores vs. 4K cores) the results obtained for UMT are more meaningful and more accurately represent what could be expected at large scale. Charon may act differently when run at larger scale but these results

indicate that both UMT and Charon are sensitive to CPU frequency changes. It is possible that further analysis will reveal opportunities to dynamically scale frequency during the execution of these applications.

The CPU is only one component that affects application performance. In the following chapter, network bandwidth tuning is applied and the resulting performance vs. total system energy trade-off is evaluated.

Chapter 7

Network Bandwidth Tuning During Application Run-time

7.1 Enabling Bandwidth Tuning

The goal of the following experiment was to determine the affect on run-time performance and energy of production scientific applications run at very large scale while tuning the network bandwidth of an otherwise balanced platform[27]. To accomplish network bandwidth scaling two different tunable characteristics of the Cray XT platform were leveraged. First, the interconnect bandwidth of the Seastar was tuned to reduce the network bandwidth in stages to 1/2 and $1/4^{th}$ of full bandwidth. Since the network bandwidth could not be reduced further by tuning the Seastar, the injection bandwidth was tuned, effectively reducing the network bandwidth to $1/8^{th}$. This allowed for the most accurate stepwise reduction in overall network bandwidth that could be achieved, using this architecture, and a more complete analysis of the affects of network bandwidth tuning on energy.

Modifying the network interconnect bandwidth on the Cray XT3 (or any XT platform using Seastar) requires a fairly simple change to the router configuration file which is consulted (if present) during the routing process of the boot sequence. This unfortunately necessitates a full system reboot for every alteration of the interconnect bandwidth. Typically, all four rails of the Seastar are configured on. This is the default behavior but the number of rails can also be specified in the *rail_enable* field of the router configuration file by specifying a single hex number (representing the four configuration bits[1]). In the following experiments, the interconnect bandwidth of the Seastar was configured to effectively tune the network bandwidth to full (baseline), 1/2 and $1/4^{th}$.

Since the interconnect bandwidth on the XT architecture is far greater than the injection bandwidth of an individual node, the interconnect bandwidth had to be reduced to 1/2 before it produced a measurable effect. It is important to note, the interconnect topology of this platform (Red Storm) is a modified mesh (partial torus). Multiple nodes may route through an individual Seastar depending on communication patterns and where they logically reside in the network topology. For this reason, the experiments were limited to one application execution at a time. This allowed for the nearest estimation of the impact of network bandwidth tuning on an individual application. Running other applications concurrently would be an interesting experiment but would greatly complicate analysis and was beyond the scope of this experiment.

[1] 4 rails = 1111 = 0xF, 3 rails (not used) = 0111 = 0x7, 2 rails = 0011 = 0x3, 1 rail = 0001 - 0x1

Tuning the node injection bandwidth, to further reduce the network bandwidth, requires a small modification to the Cray XT bootstrap source code. Cray provided access to this source code under a proprietary license. The portion of the code that required modification (*coldstart*) serves an equivalent purpose to the BIOS on a personal computer or server. Early in the power-on sequence, coldstart initializes the HyperTransport (HT) link that connects each node to its dedicated SeaStar network interface. The speed of this link is determined by its operating frequency (S) and width in bits (B):

$$\mathbf{S}\, MHz \times 2bits/clock/link \times \mathbf{B}\, bits/link \times 1Byte/8bits = \mathbf{BW} \tag{7.1}$$

In normal operation, the injection bandwidth is determined by the maximum negotiated rate between the node and the Seastar. Similar to modification of the interconnect bandwidth, a reboot is required to configure the injection bandwidth to the desired setting. Normally the links operate at S = 800 MHz and utilize the full B = 16 bits of each link resulting in an injection bandwidth of 3.2GB/sec (equation 7.1). To achieve $1/8^{th}$ injection bandwidth each link is configured to run at S = 200 MHz with an B = 8 bit per link width reducing the injection bandwidth to 400MB/sec (equation 7.1). This injection bandwidth rate was selected since it further reduced the overall network bandwidth beyond what was possible by reducing the interconnect bandwidth of the Seastar. It should also be noted that when the injection bandwidth is reduced, only the individual node is impacted. While all nodes ingress into the network are equally impacted, the network bandwidth available for routing between nodes once on the network is not reduced. A single set of baseline runs will be used and compared against identical runs while tuning the network bandwidth using the previously described methods in sequence.

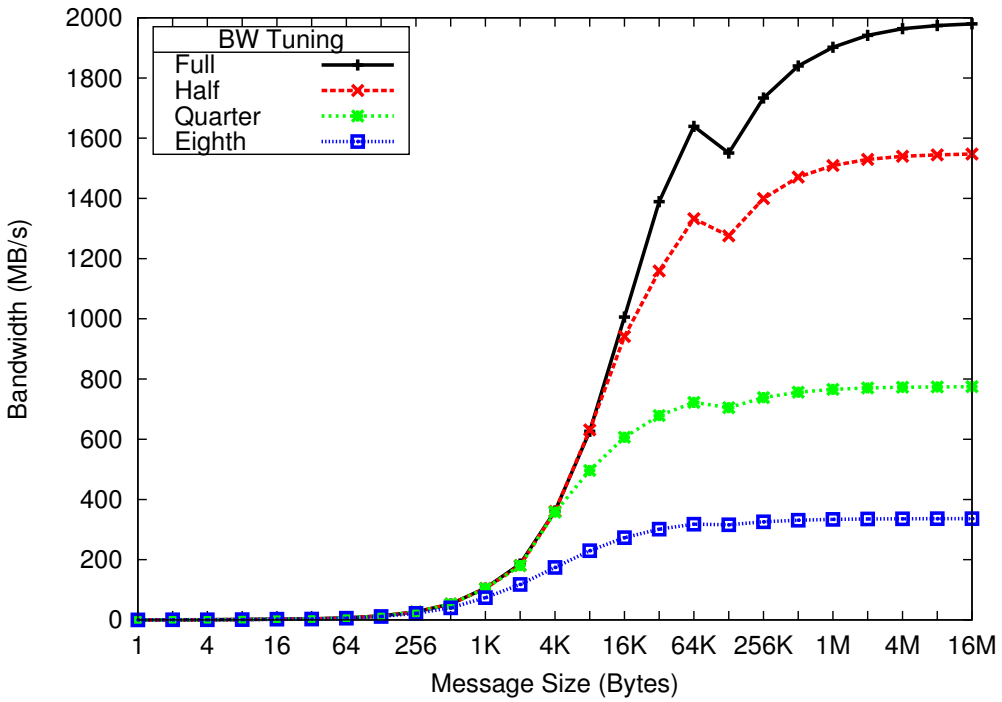

Figure 7.1: Pallas PingPong Bandwidth for All Levels of Network Bandwidth Tuning

Figure 7.1 depicts the four network bandwidth rates as measured by Pallas PingPong between two dual core nodes, one core per node. As can be seen, the maximum bandwidth observed at 1/2 using these benchmarks is not 1/2 of full bandwidth. As previously mentioned, this is due to the larger interconnect bandwidth capacity as it relates to injection bandwidth. Injection bandwidth was not altered other than to achieve the $1/8^{th}$ bandwidth configuration. Below 1/2 bandwidth, the steps become regular. While not perfect, using the configuration techniques available this is the best approximation that could be achieved. It should be noted that the goal was to test at reduced network bandwidths and measure impact on performance and energy. These configurations clearly achieve this goal and represent a tunable network bandwidth capability.

The following is the sequence of experiments for the network bandwidth study:

1. Run each application at full interconnect and injection bandwidth to establish a benchmark (run-time and energy use)
2. Reboot into a 1/2 interconnect bandwidth configuration and run each application
3. Reboot into a $1/4^{th}$ interconnect bandwidth configuration and run each application
4. Reboot into full interconnect bandwidth and $1/8^{th}$ injection bandwidth configuration (to accomplish $1/8^{th}$ network bandwidth) and run each application

For each phase, power samples are collected (current draw and voltage) as described in Chapter 3. The scale used for each application, number of nodes and cores, is listed in Table 7.1. The results will be discussed in the following section. No operating system modifications were necessary for either the interconnect or injection bandwidth experiments.

7.2 Results and Analysis: Network Bandwidth Tuning

The following discussion will reference Table 7.1. As in Table 6.2, increases in run-time or energy percentage are represented by positive numbers. Negative values or decreases are indicated by parenthesized numbers (all relative to the baseline values listed). Again, great care was taken to use the same nodes for each application execution. Total energy includes the measured energy from the CPU, a measured energy from the Seastar and an estimated energy from the memory subsystem. Since the memory sub-system could not be measured in isolation, the energy used by this

component was calculated using a fixed wattage over time. The current draw of the entire mezzanine (the mezzanine contains four Seastar network chips, one per node) was measured but as previously stated, is constant over time. This is typical of network interface controllers since the SerDes do not throttle up and down based on network traffic or on demand in current network chips. Since there are four Seastars in a single mezzanine the current reading was multiplied by the input voltage. The total was then divided by four and used as the baseline network power value for a single NIC. For 1/2, $1/4^{th}$ and $1/8^{th}$ network bandwidth calculations a linear reduction in power was assumed, which had a proportional affect on total energy. For each energy value a per node use was calculated and multiplied by the number of nodes to produce the final value. The calculation is as follows (where E = Energy):

$$(E_{cpu} + E_{network} + E_{memory}) \times \text{number of nodes} = \textbf{Total Energy} \tag{7.2}$$

The calculations use 25 W for the full network bandwidth value, 12.5 W for 1/2, 6.25 W for $1/4^{th}$ and 3.125 W for $1/8^{th}$ network bandwidth. A value of 20 W was used for the memory value in all calculations primarily to avoid the network energy having a disproportional affect on the total energy calculation. While the E_{CPU} value fluctuated based on the CPU usage of the application and was measured over time, the network and memory values assumed a constant value over time. As mentioned previously, the CPU energy was measured on a large subset of the total nodes involved in the experiment. An average per node energy is calculated based on the samples from all nodes and used as the E_{CPU} value. Again note that a decrease, or savings, in energy or run-time is indicated by a parenthesized value.

Table 7.1: Experiment 2: Network Bandwidth: Run-time and Total Energy %Difference vs. Baseline

		Baseline Bandwidth (BW)		1/2 BW %Diff		$1/4^{th}$ BW %Diff		$1/8^{th}$ BW %Diff	
	Nodes/Cores	Run-time (s)	Energy (J)	Run-time	Energy	Run-time	Energy	Run-time	Energy
SAGE_strong	2048/4096	337	5.79×10^7	(0.593)	(15.3)	8.90	(15.5)	20.2	(11.4)
SAGE_weak	2048/4096	328	5.64×10^7	0.609	(14.3)	8.23	(15.8)	22.6	(9.63)
CTH	2048/4096	1519	2.58×10^8	9.81	(7.09)	30.2	1.04	40.4	3.50
AMG2006	2048/4096	859	1.45×10^7	(0.815)	(15.8)	(0.116)	(22.7)	0.931	(25.9)
xNOBEL	1536/3072	533	7.01×10^7	(0.938)	(15.4)	(0.375)	(22.2)	(0.375)	(25.9)
UMT	512/1024	838	3.57×10^7	0.357	(14.7)	1.07	(21.7)	6.32	(21.8)
Charon	1024/2048	1162	9.96×10^7	1.55	(13.7)	2.15	(20.8)	2.67	(24.5)

Addressing each application in table order (see Table 7.1) it can be seen that the strong and weak scaling versions of SAGE have very similar characteristics. Reducing the network bandwidth by 1/2 had little affect on the run-time of both SAGE_strong (decreased by 0.593%) and SAGE_weak (increased by increased by 0.609%). In the same test, a significant savings in energy was observed for SAGE_strong (a decrease of 15.3%) and SAGE_weak (decrease of 14.3%). The impact on, or increase in, run-time is larger by more than 8X when the network bandwidth is reduced to $1/4^{th}$ for both SAGE_strong and SAGE_weak. Little additional energy savings were observed for this test. As might be expected, further reductions to $1/8^{th}$ network bandwidth for both strong and weak scaling modes of SAGE produce significant impacts in the run-time of SAGE (in excess of 20% in both modes). The accompanied energy savings using $1/8^{th}$ network bandwidth is actually smaller than the $1/4^{th}$ network bandwidth experiment. The difference in both run-time and energy savings between strong and weak scaling at $1/8^{th}$ network bandwidth might be an indicator that additional divergence might been seen at higher scale. Based on this data, reducing network bandwidth by 1/2, if the corresponding energy consumption of the network could be reduced by half, would be advantageous for this application. Considering the run-time energy trade-off, further reductions in the network bandwidth would not be productive based on the available data.

CTH was more dramatically affected by changes in the network bandwidth than any other *real* application tested. Even at 1/2 bandwidth, CTH experiences a greater percent increase in run-time (9.81%) than is saved by reducing network energy (7.09% decrease in total energy). At $1/4^{th}$ bandwidth, CTH experiences a very large increase in run-time (30.2%) accompanied by an actual increase in energy used of 1.04%. Clearly, reducing network bandwidth further is highly detrimental to both run-time and energy as can be seen from $1/8^{th}$ network bandwidth results. Even at this moderately large scale CTH requires a high performance network to execute efficiently.

AMG2006 and xNOBEL, in contrast with CTH, were insensitive to the network bandwidth changes made from the run-time perspective, which yields an opportunity for large savings in energy. Reductions down to $1/8^{th}$ network bandwidth cause virtually no impact in run-time for both AMG2006 and xNOBEL while a 25.9% savings in energy can be achieved for both. (AMG2006 executed 0.931% slower while xNOBEL actually ran slightly faster, 0.375%). The savings in energy seems to be flattening by the time the network bandwidth is reduced to $1/8^{th}$. While further reductions in network bandwidth may or may not increase run-time there is likely little additional energy savings available.

UMT produced similar results to AMG2006 and xNOBEL when the network bandwidth was reduced up to $1/4^{th}$, little to no impact in run-time accompanied by a large energy savings. At $1/8^{th}$ network bandwidth different characteristics were observed. UMT has a much higher impact to run-time at $1/8^{th}$ network bandwidth (6.32%) than at $1/4^{th}$ (1.07%) with virtually no additional energy savings (21.7% at $1/4^{th}$ and 21.8% at $1/8^{th}$) . The limit of network bandwidth tuning that should be applied to UMT, at least at this scale and on this platform, seems to have been located. Note, UMT was run at a smaller scale relative to the other applications. It is possible that at larger scale the results might differ.

Charon showed small, but increasing, impact on run-time as network bandwidth was reduced. At this scale it is clear that the network bandwidth could be reduced down to $1/4^{th}$ with an acceptable impact in run-time (increase of 2.15%) accompanied by a very significant savings in energy (decrease of 20.8)%). Moving from $1/4^{th}$ to $1/8^{th}$ network bandwidth shows some signs of a flattening of energy savings but results are not conclusive. Experiments with Charon at larger scale are also warranted.

Overall, a large amount of evidence was observed to support the conjecture that a tunable NIC would be highly beneficial if corresponding energy savings resulted. In

all cases but CTH, virtually no impact to run-time would be experienced by tuning the network bandwidth to 1/2. The result would be significant energy savings with little to no performance impact. In the case of AMG2006, xNOBEL and UMT the network bandwidth could be reduced to $1/4^{th}$ full bandwidth with little run-time impact, allowing for even larger energy savings. These observations indicate that a tunable NIC would be beneficial but they also indicate a high performance network is critical for some applications. An ability to tune the NIC, similar to how frequency is tunable on a CPU, would be an important characteristic on next generation Exascale platforms.

It should be stressed that this data represents a single application running at a time. One of the reasons the interconnect bandwidth of the Seastar was designed to be greater than the injection bandwidth of a single node is that communications on networks, like the ones used on Red Storm and Jaguar, are not point to point. Often, many hops are required for a messages to travel from source to destination. Having a greater interconnect bandwidth is essential for a platform that supports a range of applications, often sharing the interconnect bandwidth. The ability to tune this component could not be exploited without considering the possible impact on other applications running on the platform, at least for network topologies like meshes and 3D-toruses. Network topologies with fewer hops on average could benefit more easily from a tunable network since less consideration would be necessary regarding the impact on other applications co-existing on the platform.

Chapter 8

Energy Delay Product

In this chapter previous data are analyzed using a range of fused metrics based on Energy Delay Product (EDP). Many people find it useful to have a single metric to analyze data or make policy decisions. The following discussion will reference Figures 8.1 and 8.2.

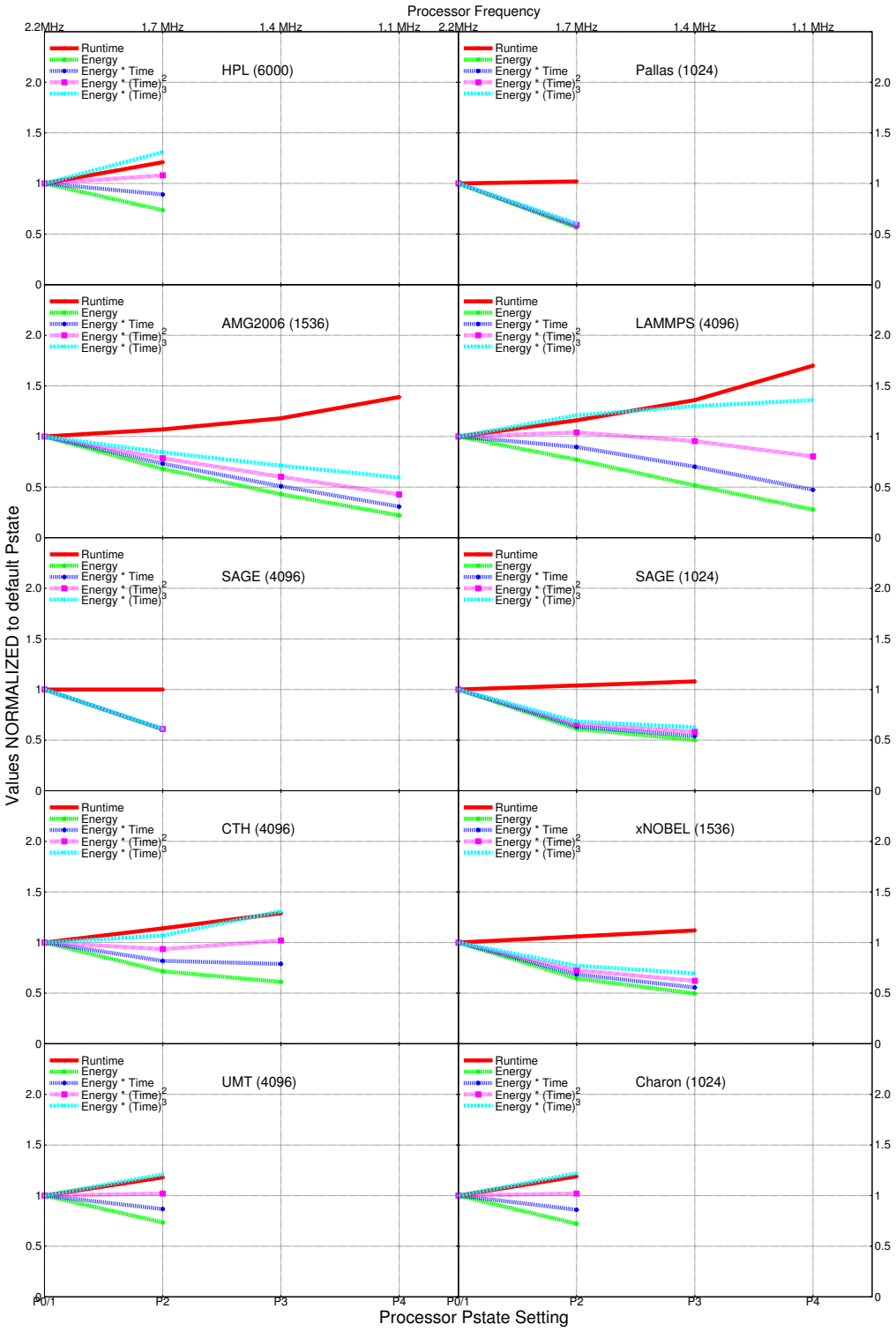

Figure 8.1: Experiment 1: Normalized Energy, Run-time and $E * T^w$ where $w = 1, 2,$ *or* 3

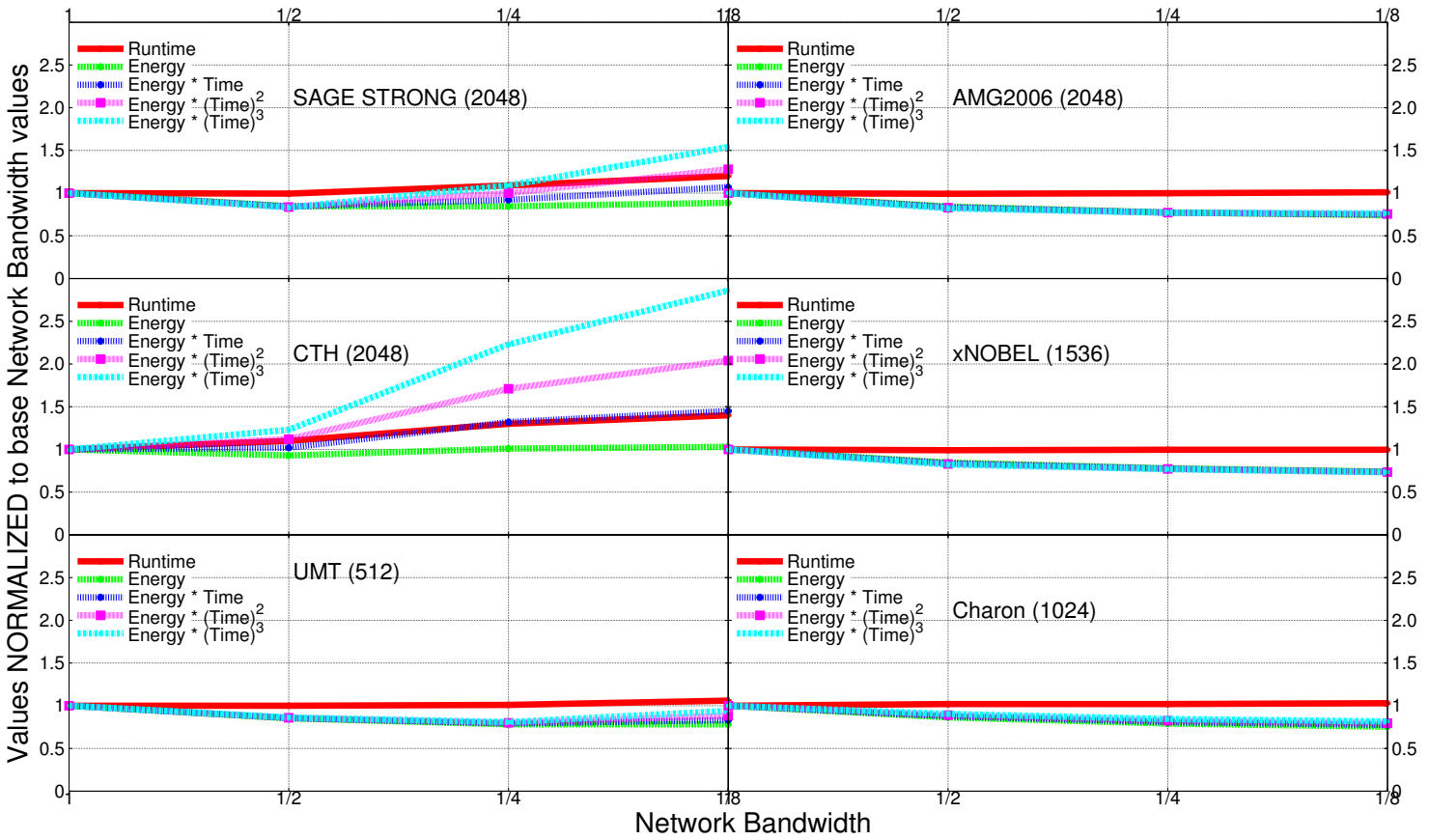

Figure 8.2: Experiment 2: Normalized Total Energy, Run-time and $E * T^w$ where $w = 1, 2, or\ 3$

The graphs in Figures 8.1 and 8.2 were produced using the same data used to create Tables 6.2 and 7.1. In all graphs included in Figures 8.1 and 8.2, the *Runtime* curve is produced by normalizing the runtime measured for each individual application at every P-state tested to the baseline run measured at the default P-state (P0 or P1). In Figure 8.1 the lower X axis lists test points by P-state, the upper X axis lists test points by CPU frequency. The *Energy* curve is produced identically to the *Runtime* curve using measured CPU energy instead of measured runtime. Three EDP curves are present using the following equation:

$$E * T^w \ - \ where: \ E = Energy, \ T = Runtime \, and \ w = 1, 2 \, or \, 3 \tag{8.1}$$

All three curves are included to represent how the metric might differ depending on the weight given to time or performance. Weighting the time factor in the EDP equation seems in-line with the existing priorities of HPC. As previously discussed, these priorities might change in the near future.

The HPL results nicely show how weighting run-time (performance) moves the curve upward, indicating a less favorable trade-off as performance is more highly prioritized. Squaring the delay produces an EDP greater than one which would typically be interpreted as detrimental. It should be noted that the unweighted EDP curve trends below one. Using this metric alone HPL might productively be run at a lower P-state. This should be an indication that if performance is a priority the unweighted EDP metric might not be appropriate. The Pallas graph, however, shows that even if the delay is cubed this metric indicates a benefit running at P-state 2.

While the results for HPL and Pallas are mostly in-line with previous observations, some of the real application results could be interpreted differently. Even based on the EDP cubed metric, AMG2006, for example, appears to benefit when executed at any P-state including P-state 4. This differs from the analysis in Chapter 6 based

on separate runtime and energy differences listed in Table 6.2. The previous conclusion was that a 39.1% hit in runtime would not be acceptable. Considering the percentage hit in runtime in isolation 39.1% does sound extreme, but considering the actual runtime value the EDP metric exposes a dimension that might otherwise have been ignored. The runtime for AMG2006 is very short in these experiments. Recall, it was noted that the fastest runs in P-state 2 actually took less time than the slowest runs measured in P-state 1. Since the runtime for AMG2006 is so short, a 39.1% hit in runtime only increases the runtime by approximately 68 seconds. When dealing with very short run-times even cubing the delay in the EDP equation might not be enough.

If the cubed EDP is used as a metric for the remaining applications the resulting analysis closely resembles the initial analysis based on the separate energy and runtime differences (Table 6.2) . In conclusion, lowering the CPU frequency for CTH, UMT and Charon is detrimental, while SAGE and xNOBEL are less sensitive to CPU frequency changes.

The graphs in Figure 8.2 represent the runtime and total energy measurements produced from the data used in Chapter 7. The *Runtime* and *Energy* curves are normalized in the same manner as described for Figure 8.1 with the exception that total energy is measured as described in equation 7.2. EDP is calculated using equation 8.1 using total energy. The X axis lists the steps of network bandwidth reduction. The EDP curves, again, generally represent the previous analysis. SAGE is not sensitive to the initial network bandwidth reduction to 1/2 but quickly trends negative as bandwidth is reduced further. CTH is very sensitive to all network bandwidth changes. AMG2006, xNOBEL, UMT and Charon in contrast are generally insensitive to network bandwidth changes. Recall, however, some finer judgments were previously stated based on the analysis of the separate energy and runtime metrics listed in Table 8.2. In the case of UMT, diminishing returns were noted as network

bandwidth was decreased. The curves trend upward as $1/8^{th}$ network bandwidth is approached. This seemed easier to identify when analyzing the tabularized data. Regardless, this method of analyzing data can help develop a quick impression of the trends. If further analysis is warranted the raw data can be consulted. For HPC, the EDP cubed equation is most appropriate. Bounding an equation like EDP could be productive. For example, if a upper limit on the wall-clock time for an application is desired, the EDP equation might indicate what level of tuning can be applied while meeting the runtime requirements. Likewise, if the amount of energy used during a certain period is limited, it would be possible to calculate the expected turnaround time for a particular application.

Chapter 9

Conclusions

. . . not everything that can be counted counts, and not everything that counts can be counted. – William Bruce Cameron

9.1 Overall

One of the conclusions drawn from this research is the importance of the ability to measure power at large scale without affecting the experiment being conducted. Previously, results such as the ones presented in this thesis have not been possible. This research began by targeting low hanging fruit in the form of power savings during idle cycles. The measuring capability was fine tuned and the effects that the operating system modifications had were easily observed. Additionally, a capability to characterize application energy use was developed. This initial work yielded significant rewards and prompted further research as outlined in this thesis.

While it was initially assumed that a dynamic approach to tuning platform components would be necessary to achieve a positive trade-off between performance and energy, it was discovered that great initial gains could result from a simpler static ap-

proach. Static tuning has many advantages including stability. Dynamic tuning, at scale, has the potential to be difficult to manage. If not done properly dynamic tuning could introduce reliability issues or diminishing results for performance and/or energy. Dynamic frequency scaling, of any component, also requires consideration of how long it takes to accomplish the desired frequency changes. If transitions are too frequent the resulting overhead could negate any potential gain. It is likely that many applications will require dynamic tuning to achieve energy savings while maintaining acceptable levels of performance. Whether static or dynamic, a system level approach is essential for scientific application, such as those used in this research, run at large scale.

In these experiments, applications like AMG2006 and xNOBEL were shown to be fairly insensitive to CPU frequency reductions. Both AMG2006 and xNOBEL were also tolerant of network bandwidth reductions. It is possible that if both the CPU frequency and network bandwidth were tuned at the same time even greater energy savings could be realized. As an example, using the per node energy value (CPU) for xNOBEL when executed at P-state 2 in the total energy calculation for the $1/8^{th}$ network bandwidth experiment, xNOBEL would experience a total of 56.4% increase in energy savings with a 6% impact in runtime. Numbers could be projected for every application but experimental results are the most meaningful.

xNOBEL was specifically selected in the previous example since it was the least impacted by any tuning applied. CTH, however, was significantly affected by CPU frequency adjustments. If CPU frequency was reduced while executing CTH could the network bandwidth be reduced without further impact on run-time? Would this result in additional energy savings making the run-time impact more palatable? Possibly, if memory or some other component is the bottleneck, but this cannot be definitively stated.

This research indicates each application has a *sweet spot* based on its computation and communication requirements. Additionally, it has been observed that energy savings and run-time are impacted, sometimes significantly, by scale. The trade-offs are platform and application specific – when one bottleneck is removed, another will appear, and the order the bottlenecks appear will depend on the platform. Conducting these experiments on a platform like the Cray XT/XE/XK architecture is valuable since it is well-balanced in its default configuration. As a result of these experiments, we can conclude that components on future HPC platforms (exascale and beyond) should be as tunable as possible under software control so that end-users or system software can optimize the energy/performance tradeoff. Additionally, a systems level approach is essential for success on HPC platforms. This research has and continues to influence industry in this direction.

References

[1] S. A. McKee, "Reflections on the Memory Wall," in *Proceedings of the Conference on Computing Frontiers*, ser. CF '04. ACM, 2004.

[2] D. W. Wall, "Limits of Instruction-Level Parallelism," in *SIGARCH Comput. Archit. News*, vol. 19. ACM, 1991.

[3] A. Zavanella and A. Milazzo, "Predictability of Bulk Synchronous Programs Using MPI," in *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing.* IEEE, 2000.

[4] 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 the International Conference on High Performance Computing, Networking, Storage, and Analysis (SC).* ACM/IEEE, 2003.

[5] K. B. Ferreira, R. Brightwell, and P. G. Bridges, "Characterizing Application Sensitivity to OS Interference Using Kernel-Level Noise Injection," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis (SC).* ACM/IEEE, 2008.

[6] X. Feng, R. Ge, and K. W. Cameron, "Power and Energy Profiling on Scientific Applications on Distributed Systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS).* IEEE, 2005.

[7] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, 2010.

[8] M. White, "Physics-of-Failure Based Modeling and Lifetime Evaluation," in *Microelectronics Reliability.* Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2008.

[9] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure Trends in a Large Disk Drive Population," in *Proceedings of the 5th USENIX conference on File and Storage Technologies*. USENIX, 2007.

[10] C. Hsu and W. Feng, "Power-Aware Run-Time System for High-Performance Computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis (SC)*. ACM/IEEE, 2005.

[11] R. Ge, X. Feng, and K. W. Cameron, "Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis (SC)*. ACM/IEEE, 2005.

[12] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-Power Digital Design," in *Proceedings of the Symposium on Low Power Electronics*. IEEE, 1994.

[13] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook, "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors," *Micro, IEEE*, vol. 20, no. 6, pp. 26–44, 2000.

[14] K. W. Cameron, R. Ge, X. Feng, D. Varner, and C. Jones, "POSTER: High-performance, Power-aware Distributed Computing Framework," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis (SC)*. ACM/IEEE, 2004.

[15] R. Ge, X. Feng, and K. Cameron, "Improvement of Power-Performance Efficiency for High-End Computing," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2005.

[16] D. Li, B. de Supinski, M. Schulz, K. Cameron, and D. Nikolopoulos, "Hybrid MPI/OpenMP Power-Aware Computing," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2010.

[17] D. Li, D. Nikolopoulos, K. Cameron, B. de Supinski, and M. Schulz, "Power-Aware MPI Task Aggregation Prediction for High-End Computing Systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2010.

[18] C. Hsu and U. Kremer, "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction," in *Proceedings of the Conference on Programming Language Design and Implementation, (PLDI)*. ACM, 2003.

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

[20] W. L. Bircher, M. Valluri, J. Law, and L. John, "Runtime Identification of Microprocessor Energy Saving Opportunities." in *Proceedings of the International Symposium on Low Power Electronics and Design, (ISLPED)*. ACM, 2005.

[21] W. L. Bircher and L. K. John, "Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events," in *Proceedings of the International Symposium on Performance Analysis of Systems & Software, (ISPASS)*. IEEE, 2007.

[22] S. Kamil, J. Shalf, and E. Strohmaier, "Power Efficiency in High Performance Computing," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2008.

[23] A. Kodi and A. Louri, "Performance Adaptive Power-Aware Reconfigurable Optical Interconnects for High-Performance Computing (HPC) Systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis (SC)*. ACM/IEEE, 2007.

[24] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks," in *Proceedings of the International Symposium on High-Performance Computer Architecture, (HPCA)*. IEEE, 2003.

[25] R. Brightwell, B. Barrett, K. Hemmert, and K. Underwood, "Challenges for High-Performance Networking for Exascale Computing," in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2010.

[26] R. Brightwell, K. D. Underwood, C. Vaughan, and J. Stevenson, "Performance evaluation of the Red Storm dual-core upgrade," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 175–190, 2010.

[27] R. Brightwell, K. Predretti, K. Underwood, and T. Hudson, "SeaStar Interconnect: Balanced Bandwidth for Scalable Performance," *IEEE Micro*, vol. 26, no. 3, pp. 41–57, 2006.

[28] K. Pedretti, R. Brightwell, D. Doerfler, K. Hemmert, and J. Laros, "The Impact of Injection Bandwidth Performance on Application Scalability," in *Proceedings of the European MPI Users' Group Conference on Recent Advances in the Message Passing Interface*. Springer-Verlag, 2011.

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

[30] "Kitten Light Weight Kernel," Sandia National Laboratories. [Online]. Available: https://software.sandia.gov/trac/kitten

[31] J. H. Laros III, "A Software and Hardware Architecture for a Modular, Portable, Extensible Reliability Availability and Serviceability System," in *Proceedings of the Workshop on High Performance Computing Reliability Issues*. IEEE, 2006.

[32] "Cielo," Sandia National Laboratories and Los Alamos Laboratory. [Online]. Available: http://www.lanl.gov/orgs/hpc/cielo/

[33] R. Weaver and M. Gittings, " Massively Parallel Simulations with DOEś ASCI Supercomputers: An Overview of the Los Alamos Crestone Project ," in *Adaptive Mesh Refinement - Theory and Applications*. Springer Berlin Heidelberg, 2005.

[34] 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 International Conference on High Performance Computing, Networking, Storage, and Analysis (SC)*. ACM/IEEE, 2001.

[35] E. S. Hertel, Jr., R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. Mcglaun, S. V. Petney, S. A. Silling, P. A. Taylor, and L. Yarrington, "CTH: A Software Family for Multi-Dimensional Shock Physics Analysis," in *Proceedings of the International Symposium on Shock Waves*. NTIS, 1993.

[36] R. D. Falgout, P. S. Vassilevski, Panayot, and S. Vassilevski, "On Generalizing the AMG Framework," in *Society for Industrial and Applied Mathmatics: Journal on Numererical Analysis*. SIAM, 2003.

[37] "Hypre," Lawrence Livermore National Laboratory. [Online]. Available: http://acts.nersc.gov/hypre/

[38] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, D. Ranta, and R. Stefan, "The RAGE Radiation-Hydrodynamic Code," *Journal of Computational Science & Discovery*, vol. 1, no. 1, p. 015005, 2008.

[39] "UMT2K," Lawrence Livermore National Laboratory. [Online]. Available: https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/umt/umt1.2.readme.html

[40] P. T. Lin, J. N. Shadid, M. Sala, R. S. Tuminaro, G. L. Hennigan, and R. J. Hoekstra, "Performance of a parallel algebraic multilevel preconditioner for stabilized finite element semiconductor device modeling," *Journal of Computational Physics*, vol. 228, pp. 6250–6267, 2009.

[41] "Trilinos," Sandia National Laboratories. [Online]. Available: http://trilinos.sandia.gov/

[42] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, 1995.

[43] "LAMMPS," Sandia National Laboratories. [Online]. Available: http://lammps.sandia.gov/index.html

[44] J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart, "High Performance Linpack HPL," in *Technical Report CS-89-85*. University of Tennessee, 1989.

[45] "Top 500 Supercomputer Sites." [Online]. Available: http://www.top500.org/

[46] "PALLAS," Intel. [Online]. Available: http://www.intel.com/cd/software/products/asmona/eng/cluster/mpi/219848.htm

[47] "HPCC," DARPA. [Online]. Available: http://icl.cs.utk.edu/hpcc/

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

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

[50] 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*. USENIX, 1993.

[51] "BKDG: AMD BIOS and Kernel Developers Guide for AMD Family 10h Processors Rev 3.48," Advanced Micro Devices. [Online]. Available: http://www.amd.com